
AWS OpsWorks

User Guide

API Version 2013-02-18



AWS OpsWorks: User Guide

Copyright © 2017 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 AWS OpsWorks?	1
AWS OpsWorks Services	1
AWS OpsWorks for Chef Automate	3
Region Support for AWS OpsWorks for Chef Automate	3
Getting Started	4
Prerequisites	4
Create a Chef Automate Server	6
Finish configuration and upload cookbooks	8
Add nodes to manage	11
Sign in to the Chef Automate dashboard	12
Back up and restore servers	14
Back Up an AWS OpsWorks for Chef Automate Server	15
Restore an AWS OpsWorks for Chef Automate Server	15
Add nodes automatically	16
Supported Operating Systems	16
Step 1: Create an IAM Role to Use as Your Instance Profile	17
Step 2: Install the Chef Client Cookbook	17
Step 3: Create Instances by Using an Unattended Association Script	17
Other Methods of Automating Repeated Runs of <code>chef-client</code>	19
Remove nodes	20
Delete a Chef Automate server	20
Step 1: Disassociate Managed Nodes	21
Step 2: Delete the Server	21
Reset Chef credentials	21
CloudTrail integration	22
AWS OpsWorks for Chef Automate Information in CloudTrail	22
Understanding AWS OpsWorks for Chef Automate Log File Entries	23
Troubleshooting	24
General Troubleshooting Tips	24
Troubleshooting Specific Errors	24
Additional help and support	27
AWS OpsWorks Stacks	28
Stacks	30
Layers	30
Recipes and LifeCycle Events	30
Instances	31
Apps	32
Customizing your Stack	32
Resource Management	33
Security and Permissions	33
Monitoring and Logging	33
CLI, SDK, and AWS CloudFormation Templates	33
Getting Started	34
Region Support	34
Getting Started: Sample	35
Getting Started: Linux	48
Getting Started: Windows	67
Getting Started: Cookbooks	86
Best Practices	105
Root Device Storage	105
Optimizing the Number of Servers	107
Managing Permissions	108
Managing and Deploying Apps and Cookbooks	110
Packaging Cookbook Dependencies Locally	116

Installing Security Updates	117
Using Security Groups	118
Stacks	120
Create a New Stack	120
Running a Stack in a VPC	126
Update a Stack	132
Clone a Stack	132
Run Stack Commands	133
Using Custom JSON	135
Shut Down a Stack	137
Layers	138
OpsWorks Layer Basics	139
Elastic Load Balancing Layer	147
Amazon RDS Service Layer	150
ECS Cluster Layers	153
Custom Layers	157
Per-layer Package Installations	158
Instances	158
Using AWS OpsWorks Stacks Instances	159
Using Computing Resources Created Outside of AWS OpsWorks Stacks	188
Editing the Instance Configuration	210
Deleting AWS OpsWorks Stacks Instances	211
Logging In with SSH	212
Logging In with RDP	214
Apps	216
Adding Apps	217
Deploying Apps	221
Editing Apps	224
Connecting to a Database	224
Using Environment Variables	226
Passing Data to Applications	226
Using Git Repository SSH Keys	228
Using Custom Domains	229
Using SSL	230
Cookbooks and Recipes	235
Cookbook Repositories	235
Chef Versions	237
Ruby Versions	248
Installing Custom Cookbooks	249
Updating Custom Cookbooks	251
Executing Recipes	252
Resource Management	256
Registering Resources with a Stack	257
Attaching and Moving Resources	261
Detaching Resources	265
Deregistering Resources	267
Monitoring	268
Using Amazon CloudWatch	269
Using AWS CloudTrail	276
Using Amazon CloudWatch Logs	278
Security and Permissions	281
Managing User Permissions	282
Signing in as an IAM User	297
Allowing AWS OpsWorks Stacks to Act on Your Behalf	298
Specifying Permissions for Apps Running on EC2 instances	300
Managing SSH Access	302
Managing Security Updates	306

Using Security Groups	307
Chef 12 Linux	309
Overview	309
Moving to Chef 12	310
Supported Operating Systems	311
Supported Instance Types	311
More Information	311
Moving to Data Bags	311
Previous Chef Versions	312
Chef 11.10 and Earlier Versions for Linux	313
Using AWS OpsWorks Stacks with Other AWS Services	561
Using a Back-end Data Store	562
ElastiCache Redis	566
Using an Amazon S3 Bucket	574
Using AWS CodePipeline with AWS OpsWorks Stacks	583
Using the AWS OpsWorks Stacks CLI	621
Create an Instance	623
Deploy an App	624
List Apps	625
List Commands	626
List Deployments	627
List Elastic IP Addresses	627
List Instances	628
List Stacks	629
List Layers	630
Execute a Recipe	632
Install Dependencies	633
Update the Stack Configuration	633
Debugging and Troubleshooting Guide	633
Debugging Recipes	634
Common Debugging and Troubleshooting Issues	645
AWS OpsWorks Stacks Agent CLI	651
agent_report	652
get_json	652
instance_report	655
list_commands	656
run_command	656
show_log	656
stack_state	657
AWS OpsWorks Stacks Data Bag Reference	659
App Data Bag (aws_opsworks_app)	661
Command Data Bag (aws_opsworks_command)	663
Amazon ECS Cluster Data Bag (aws_opsworks_ecs_cluster)	665
Elastic Load Balancing Data Bag (aws_opsworks_elastic_load_balancer)	665
Instance Data Bag (aws_opsworks_instance)	666
Layer Data Bag (aws_opsworks_layer)	669
Amazon RDS Data Bag (aws_opsworks_rds_db_instance)	670
Stack Data Bag (aws_opsworks_stack)	671
User Data Bag (aws_opsworks_user)	673
OpsWorks Agent Changes	673
Resources	675
Reference Guides, Tools, and Support Resources	675
AWS Software Development Kits	676
Open Source Software	676
History	677

What Is AWS OpsWorks?

AWS OpsWorks is a configuration management service that helps you configure and operate applications in a cloud enterprise by using Chef. AWS OpsWorks Stacks and AWS OpsWorks for Chef Automate let you use [Chef cookbooks](#) and solutions for configuration management.

AWS OpsWorks Services

[AWS OpsWorks for Chef Automate \(p. 3\)](#)

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-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 both the Chef server and Chef Automate 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 AWS OpsWorks-specific changes.

Because AWS OpsWorks for Chef Automate manages both Chef Automate Server and Chef Server software 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 Auto Scaling groups to associate new Amazon EC2 nodes with your server automatically.

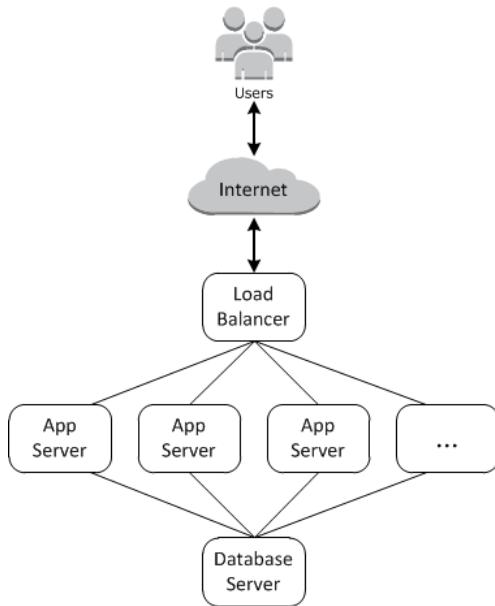
[AWS OpsWorks Stacks \(p. 28\)](#)

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*.

AWS OpsWorks Stacks, the original service, provides a simple and flexible way to create and manage stacks and applications. AWS 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, AWS OpsWorks Stacks does not require or create Chef servers; AWS OpsWorks Stacks performs some of the work of a Chef server for you. AWS 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 Chef Automate

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 Stacks 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 and premium features of Chef Automate: visibility, workflow, and compliance.

An AWS OpsWorks for Chef Automate server runs on an Amazon Elastic Compute Cloud instance. The instance is configured to run the newest version of Amazon Linux (2016.09), Chef Server 12.x, and the most current version of Chef Automate Server, version 0.6.x.

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 \(p. 3\)](#)
- [Getting Started with AWS OpsWorks for Chef Automate \(p. 4\)](#)
- [Back Up and Restore an AWS OpsWorks for Chef Automate Server \(p. 14\)](#)
- [Adding Nodes Automatically in AWS OpsWorks for Chef Automate \(p. 16\)](#)
- [Disassociate a Node from an AWS OpsWorks for Chef Automate Server \(p. 20\)](#)
- [Delete an AWS OpsWorks for Chef Automate Server \(p. 20\)](#)
- [Reset Chef Automate Dashboard Credentials \(p. 21\)](#)
- [Logging AWS OpsWorks for Chef Automate API Calls with AWS CloudTrail \(p. 22\)](#)
- [Troubleshooting AWS OpsWorks for Chef Automate \(p. 24\)](#)

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,

IAM users, and service roles, in the same regional endpoint as your Chef server. Your Chef server must be in a VPC in the same region. You can use a VPC that you create or already have, or use the default VPC.

- US East (N. Virginia) Region
- US West (Oregon) Region
- EU (Ireland) Region

Getting Started with AWS OpsWorks for Chef Automate

AWS OpsWorks for Chef Automate lets you run a [Chef Automate](#) server in AWS. You can provision a Chef server in about 15 minutes.

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

Prerequisites

First, create the resources outside of AWS OpsWorks for Chef Automate that you'll need to access and manage your Chef server. If you already have an AWS account set up, skip to [Set Up a VPC \(p. 5\)](#).

Topics

- [Get an AWS Account and Your AWS Credentials \(p. 4\)](#)
- [Set Up a VPC \(p. 5\)](#)
- [Set Up an EC2 Key Pair \(Optional\) \(p. 5\)](#)

Get an AWS Account and Your AWS Credentials

To access AWS, you will need to sign up for an AWS account.

To sign up for an AWS account

1. Open <https://aws.amazon.com/>, and then choose **Create an AWS Account**.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

AWS sends you a confirmation e-mail 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 clicking **My Account/Console**.

To get your access key ID and secret access key

Access keys consist of an access key ID and secret access key, which are used to sign programmatic requests that you make to AWS. If you don't have access keys, you can create them by using the AWS Management Console.

Note

To create access keys, you must have permissions to perform the required IAM actions. For more information, see [Granting IAM User Permission to Manage Password Policy and Credentials](#) in the [IAM User Guide](#).

1. Open the [IAM console](#).
2. In the navigation pane, choose **Users**.
3. Choose your IAM user name (not the check box).
4. Choose the **Security Credentials** tab and then choose **Create Access Key**.
5. To see your access key, choose **Show User Security Credentials**. Your credentials will look something like this:
 - Access Key ID: AKIAIOSFODNN7EXAMPLE
 - Secret Access Key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
6. Choose **Download Credentials**, and store the keys in a secure location.

Your secret key will no longer be available through the AWS Management Console; you will have the only copy. Keep it confidential in order to protect your account, and never email it. Do not share it outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon will ever ask you for your secret key.

Related topics

- [What Is IAM?](#) in the *IAM User Guide*
- [AWS Security Credentials](#) in *AWS General Reference*

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 a single, public subnet.
- **DNS resolution** should be enabled.
- On the subnet, enable **Auto-assign public IP**.

If you are unfamiliar with creating VPCs or running your instances in them, you can run the following AWS CLI command to create a VPC, by using an AWS CloudFormation template that AWS OpsWorks provides for you. If you prefer to use the AWS Management Console, you can also upload the [template](#) to the AWS CloudFormation console.

```
aws cloudformation create-stack --stack-name OpsWorksVPC --template-url https://s3.amazonaws.com/opsworks-cm-us-east-1-prod-default-assets/misc/opsworks-cm-vpc.yaml
```

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

You can create a Chef server by using the AWS OpsWorks for Chef Automate console, or the AWS CLI. This walkthrough describes how to create a Chef server in the AWS OpsWorks for Chef Automate console.

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



AWS OpsWorks provides two solutions to configure your infrastructure:



OpsWorks Stacks

Define, group, provision, deploy, and operate your applications in AWS by using Chef in local mode.

[Go to OpsWorks Stacks](#)

[Learn more about OpsWorks Stacks](#)



OpsWorks for Chef Automate

Create Chef servers that include Chef Automate premium features, and use the Chef DK or any Chef tooling to manage them.

[Go to OpsWorks for Chef Automate](#)

[Learn more about OpsWorks for Chef Automate](#)

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

Welcome to OpsWorks for Chef Automate

OpsWorks for Chef Automate helps you automate, provision, and configure your environment. The Chef Automate platform delivers DevOps workflow, automated compliance, and end-to-end pipeline visibility.

A Chef Automate server manages nodes in your environment, stores information about those nodes, and serves as a central repository for your Chef cookbooks.

[Create Chef Automate server](#)

- 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 a **t2.medium** instance type in the US West (Oregon) Region. Choose **Next**.

Set name, region, and type

Type a name for the Chef Automate server, select the region in which you want to locate the server, and select the Amazon EC2 instance type that best fits your needs.

Chef Automate server name: test-automate-server

Chef Automate server region: US West (Oregon)

EC2 instance type:

- t2.medium**: 4 GiB Memory, Supports up to 50 nodes
- m4.large: 8 GiB Memory, Supports up to 200 nodes
- m4.2xlarge: 32 GiB Memory, Supports 200+ nodes

- On the **Select an SSH key** page, leave the default selection in the **SSH key** drop-down list, unless you want to specify a key pair name. Choose **Next**.

Select an SSH key

Select the EC2 key pair. You will need this key to connect to the Chef Automate server EC2 instance by using SSH.

SSH key: I'm not connecting by SSH

You can still use Knife commands to communicate with the Chef Automate server.

- On the **Configure Advanced Settings** page, in the **Network and Security** area, choose a VPC, subnet, and security group. AWS 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. You cannot change network and security settings for the Chef server after you have left this page.

Network and security

You cannot change network and security settings after you launch your Chef Automate server.

VPC: vpc-XXXX - LinuxAMIVPC

You have selected a non-default VPC. Be sure the selected VPC has outbound network access. [Learn more](#).

Subnet: 10.XXX.us-west-2a - Public subnet

Associate Public IP Address: Yes No
Choose Yes if the selected subnet is public.

Security group: Generate a new one

By default, the new security group is open to all IP address ranges. After the server is launched, we recommend that you restrict ranges to help secure your Chef Automate server.

Service role: aws-opsworks-cm-service-role

Instance profile: aws-opsworks-cm-ec2-role

7. 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.

System maintenance

AWS OpsWorks installs updates for Chef Automate minor versions or security packages in the time range and on the weekday that you specify here. Your Chef Automate server will be offline during system maintenance.

Start day: Sunday

Start time (UTC): 2 am - 3 am

8. 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.

Automated backup

AWS OpsWorks supports two ways to back up your Chef Automate server: manual or automated. Backups are uploaded to your Amazon S3 bucket. If you ever need to restore your Chef Automate server, you can restore it by applying a backup that you choose.

Enable automated backup: Yes

Frequency: Daily

Start time (UTC): 12 am - 1 am

Number of generations to keep: 10

Specify how many automated backups to keep. Minimum: 1, maximum: 30.

9. When you are finished configuring advanced settings, choose **Next**.
10. On the **Review** page, review your choices. When you are ready to create the server, choose **Launch**.

While you are waiting for AWS OpsWorks to create your Chef server, go on to [Configure the Chef Server Using the Starter Kit \(p. 8\)](#) 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`.

Configure the Chef Server Using the Starter Kit

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.

The screenshot shows the AWS OpsWorks console for a stack named "my-chef-server". At the top, there's a progress bar indicating the status: "Creating an Elastic IP address" (in progress), "Launching an EC2 instance" (not yet available), and "Installing Chef Automate server" (not yet available). Below the progress bar, a message says "AWS OpsWorks is creating your Chef Automate server. This takes about 20 minutes." A callout box highlights two items: "Sign-in credentials for your Chef Automate dashboard" and "Starter Kit for your Chef Automate server". A red arrow points from this callout to the "Download credentials" button in the next section. Another red arrow points from the "Download Starter Kit" button in the third section back to the "Starter Kit" item in the callout.

AWS OpsWorks is creating your Chef Automate server. This takes about 20 minutes.

Creating an Elastic IP address Launching an EC2 instance Installing Chef Automate server

Make sure you download the following before your server is online.

- ① Sign-in credentials for your Chef Automate dashboard
- ② Starter Kit for your Chef Automate server

i Download the sign-in credentials for your Chef Automate 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 [Chef Automate dashboard](#).

i 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, a knife.rb configuration file, and a private key. A new key pair is generated and reset each time you download the Starter Kit.

- **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. AWS OpsWorks Stacks 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. For more information about how to add users to the Chef server, see [Manage Users](#) in the Chef Automate documentation.
3. Download and install 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 the Chef DK. For instructions, see [Install the Chef DK](#) on the Chef website.

Configure Your Server with the `knife` Utility

The `.chef` directory is hidden, and contains the following files:

- `.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-2016-root.pem` - A certification authority (CA)-signed SSL private key that is provided by AWS OpsWorks. This key allows the server to identify itself to the chef-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: `knife` commands, or Berkshelf commands. This walkthrough uses Berkshelf 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 AWS CodeCommit, Git, or Amazon S3.
2. In the `chef-repo` directory, create the following three directories, as shown in the Starter Kit:
 - `cookbooks/` - Stores cookbooks that you download or create.
 - `roles/` - Stores roles in `.rb` or `.json` formats.
 - `environments/` - Stores environments in `.rb` or `.json` formats.

Use Berkshelf to Get Cookbooks from a Remote Source

Berkshelf is a tool for managing cookbooks and their dependencies. It downloads a specified cookbook into local storage, which is called the Berkshelf. 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 lists your cookbooks. The included `Berksfile` references the chef-client cookbook that configures the Chef client agent software on each node that you connect to your Chef server. To learn more about this cookbook, see [Chef Client Cookbook](#) in the Chef Supermarket.

1. Using a text editor, append another cookbook to your `Berksfile` in which to install the web server software; for example, to install the Apache web server application. Your `Berksfile` should resemble the following.

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

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

```
berks install
```

3. Upload the cookbook to the Chef server.

On Linux, run the following.

```
SSL_CERT_FILE='~.chef/ca_certs/opsworks-cm-ca-2016-root.pem' berks upload
```

On Windows, run the following Chef DK 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 DK utility commands available to PowerShell.

```
$env:SSL_CERT_FILE="ca_certs\opsworks-cm-ca-2016-root.pem"  
chef shell-init berks upload  
Remove- Item Env :\\ SSL_CERT_FILE
```

4. Verify the installation of the cookbook by showing a list of cookbooks that are currently available on the Chef Automate server.

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

```
knife cookbook list
```

Add Nodes for the Chef Server to Manage

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.

This walkthrough demonstrates how to run a `knife` command that adds, or *bootstraps*, an EC2 instance so that the Chef server can manage it. For more information about how to add nodes automatically by using a script to perform unattended association of nodes with the Chef server, see [Adding Nodes Automatically in AWS OpsWorks for Chef Automate \(p. 16\)](#).

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 the Chef DK. If you are more familiar with `knife-ec2`, you can use it instead of `knife bootstrap` to provision and bootstrap new EC2 instances. 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 `apache2` cookbook that you installed in [Use Berkshelf to Get Cookbooks from a Remote Source \(p. 10\)](#). 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	<code>ec2-user</code>
Red Hat Enterprise Linux 5	<code>root</code> or <code>ec2-user</code>
Ubuntu	<code>ubuntu</code>
Fedora	<code>fedora</code> or <code>ec2-user</code>
SUSE Linux	<code>root</code> or <code>ec2-user</code>

```
knife bootstrap INSTANCE_IP_ADDRESS -N INSTANCE_NAME -x USER_NAME --sudo --run-list "recipe[apache2]"
```

- 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
```

More Info

Visit the [Learn Chef tutorial site](#) to learn more about using AWS OpsWorks for Chef Automate servers and Chef Automate premium features.

Sign in to the Chef Automate dashboard

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 AWS 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.

To install the AWS OpsWorks SSL certificate

- Choose the certificate that matches your system.
- For Linux or MacOS-based systems, download the file with the **PEM** filename extension from the following Amazon S3 location: <https://s3.amazonaws.com/opsworks-cm-us-east-1-prod-default-assets/misc/opsworks-cm-ca-2016-root.pem>.

For more information about how to install an SSL certificate on MacOS, see [If your certificate isn't being accepted](#) on the Apple Support website.

- For Windows-based systems, download the file with the **P7B** filename extension from the following Amazon S3 location: <https://s3.amazonaws.com/opsworks-cm-us-east-1-prod-default-assets/misc/opsworks-cm-ca-2016-root.p7b>.

For more information about how to install an SSL certificate on Windows, see [Manage Trusted Root Certificates](#) on Microsoft TechNet.

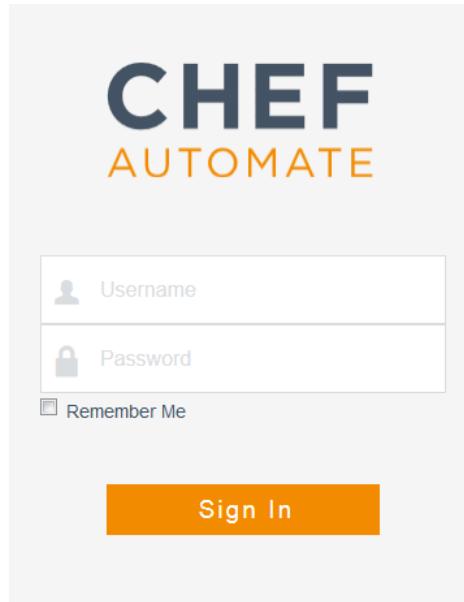
After you have installed the client-side SSL certificate, you can sign in to the Chef Automate dashboard without seeing warning messages.

Note

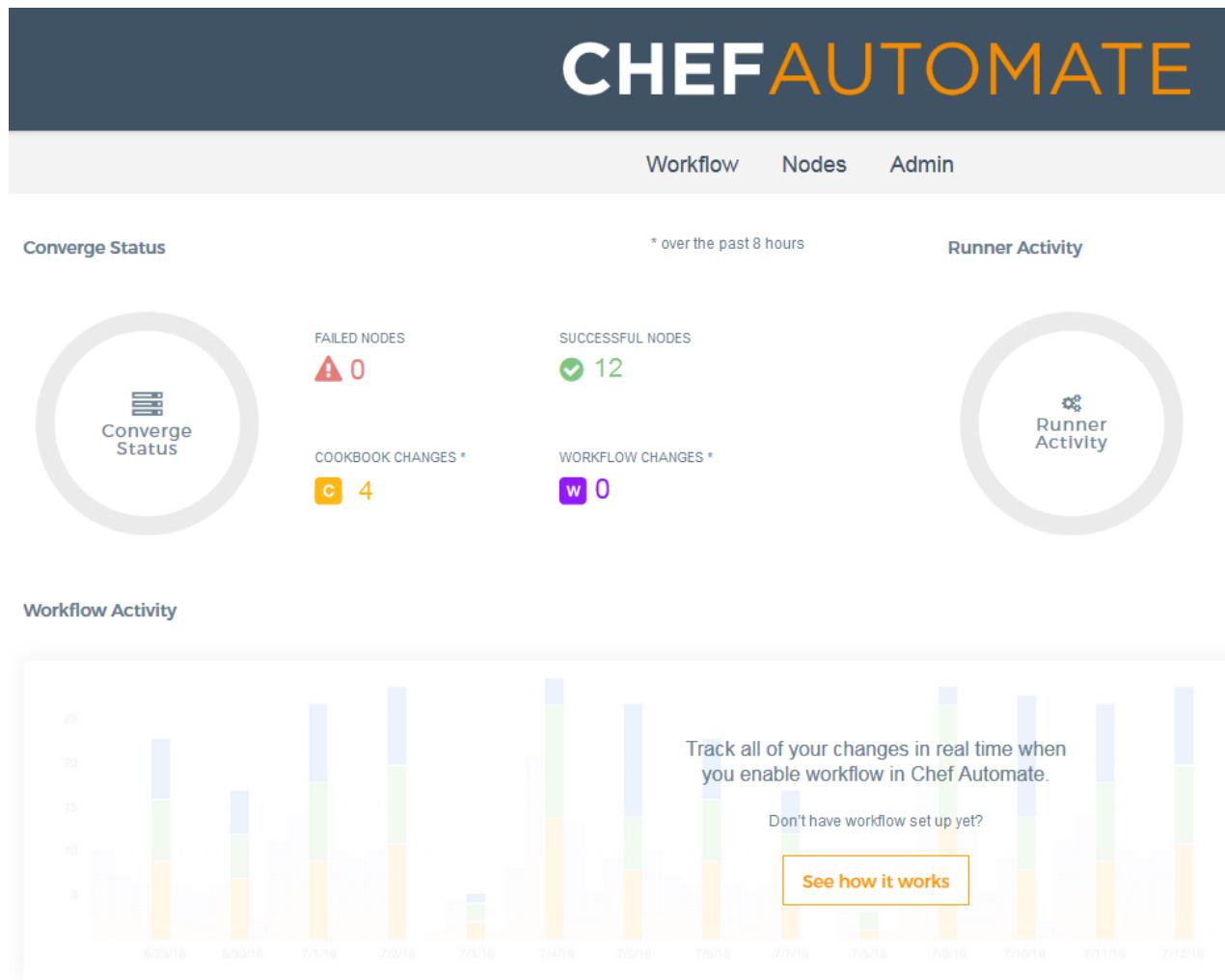
Users of Google Chrome on Ubuntu and Linux Mint operating systems can have difficulty signing in. We recommend that you use Mozilla Firefox or other browsers to sign in and use the Chef Automate dashboard on those operating systems. No issues have been found using Google Chrome on Windows or MacOS.

To sign in to the Chef Automate dashboard

1. Unzip and open the Chef Automate credentials that you downloaded in [Prerequisites \(p. 10\)](#). 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 \(p. 21\)](#).

Back Up and Restore an AWS OpsWorks for Chef Automate Server

This section describes how to back up and restore an AWS OpsWorks for Chef Automate server.

Topics

- [Back Up an AWS OpsWorks for Chef Automate Server \(p. 15\)](#)
- [Restore an AWS OpsWorks for Chef Automate Server from a Backup \(p. 15\)](#)

Back Up an AWS OpsWorks for Chef Automate Server

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.

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.

You can start a manual backup at any time 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

- 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"
```

Restore an AWS OpsWorks for Chef Automate Server from a Backup

After browsing through your available backups, you can easily 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 to the same EC2 instance) reregisters nodes that were registered at the time of the backup that you use to restore the server. Restoring to a new instance does not maintain node connections.

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-east-1 restore-server --backup-id "MyChefServer-20161120122143125" --server-name "MyChefServer"
```

3. Wait until restoration is complete.

Adding Nodes Automatically in AWS OpsWorks for Chef Automate

This topic describes how to add Amazon Elastic Compute Cloud (Amazon EC2) nodes to your Chef server automatically. In [Add Nodes for the Chef Server to Manage \(p. 11\)](#), you learned how to use the `knife bootstrap` command to add one node at a time to your Chef 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 [Chef Client Cookbook](#). Before you run the `chef-client` agent, upload the Chef Client cookbook to 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 AWS OpsWorks `associate-node` API call with the node name. The script shown in this topic, 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.

For information about how to disassociate a node, see [Disassociate a Node from an AWS OpsWorks for Chef Automate Server \(p. 20\)](#) in this guide, and `disassociate-node` in the AWS OpsWorks for Chef Automate API documentation.

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.

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

Step 2: Install the Chef Client Cookbook

If you have not done so already, follow the steps in [Use Berkshelf to Get Cookbooks from a Remote Source \(p. 10\)](#) to ensure that your Berksfile references the Chef Client cookbook and installs the cookbook.

Step 3: Create Instances by Using an Unattended Association Script

- To create EC2 instances, you can copy the following code to the `userdata` section of EC2 instance instructions, Auto Scaling group launch configurations, or an AWS CloudFormation template. You do not need to prefill your own values in this script. 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. 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.

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. Because changing the organization name in the Chef console UI is currently not possible, leave `CHEF_ORGANIZATION` set to `default`.

```
#!/bin/bash  
  
# Required settings  
NODE_NAME=$(curl --silent --show-error --retry 3 http://169.254.169.254/latest/meta-data/instance-id)" # This uses the EC2 instance ID as the node name  
REGION="region" # Valid values are us-east-1, us-west-2, or eu-west-1  
CHEF_SERVER_NAME="serverName" # The name of your Chef server
```

```

CHEF_SERVER_ENDPOINT="endpoint" # Provide the FQDN or endpoint; it's the string after
# 'https://'

# Optional settings
CHEF_ORGANIZATION="default"      # Leave as "default"; do not change. AWS OpsWorks for
# Chef Automate always creates the organization "default"
NODE_ENVIRONMENT=""              # e.g. development, staging, onebox ...
CHEF_CLIENT_VERSION="12.16.42"    # latest if empty

# Recommended: upload the chef-client cookbook from the chef supermarket https://
supermarket.chef.io/cookbooks/chef-client
# Use this to apply sensible default settings for your chef-client configuration like
logrotate, and running as a service.
# You can add more cookbooks in the run list, based on your needs
RUN_LIST="" # e.g. "recipe[chef-client],recipe[apache2]"

# -----
set -e -o pipefail

AWS_CLI_TMP_FOLDER=$(mktemp --directory "/tmp/awscli_XXXX")
CHEF_CA_PATH="/etc/chef/opsworks-cm-ca-2016-root.pem"

install_aws_cli() {
    # see: http://docs.aws.amazon.com/cli/latest/userguide/installing.html#install-
bundle-other-os
    cd "$AWS_CLI_TMP_FOLDER"
    curl --retry 3 -L -o "awscli-bundle.zip" "https://s3.amazonaws.com/aws-cli/awscli-
bundle.zip"
    unzip "awscli-bundle.zip"
    ./awscli-bundle/install -i "$PWD"
}

aws_cli() {
    "${AWS_CLI_TMP_FOLDER}/bin/aws" opsworks-cm --region "${REGION}" --output text "$@"
    --server-name "${CHEF_SERVER_NAME}"
}

associate_node() {
    client_key="/etc/chef/client.pem"
    mkdir /etc/chef
    ( umask 077; openssl genrsa -out "${client_key}" 2048 )

    aws_cli associate-node \
        --node-name "${NODE_NAME}" \
        --engine-attributes \
        "Name=CHEF_ORGANIZATION,Value=${CHEF_ORGANIZATION}" \
        "Name=CHEF_NODE_PUBLIC_KEY,Value='$(openssl rsa -in "${client_key}" -pubout)'"
}

write_chef_config() {
(
    echo "chef_server_url      'https://${CHEF_SERVER_ENDPOINT}/organizations/
${CHEF_ORGANIZATION}'"
    echo "node_name            '${NODE_NAME}'"
    echo "ssl_ca_file          '${CHEF_CA_PATH}'"
) >> /etc/chef/client.rb
}

install_chef_client() {
    # see: https://docs.chef.io/install_omnibus.html
    curl --silent --show-error --retry 3 --location https://omnitruck.chef.io/install.sh
    | bash -s -- -v "${CHEF_CLIENT_VERSION}"
}

install_trusted_certs() {
    curl --silent --show-error --retry 3 --location --output "${CHEF_CA_PATH}" \

```

```
"https://opsworks-cm-${REGION}-prod-default-assets.s3.amazonaws.com/misc/opsworks-  
cm-ca-2016-root.pem"  
}  
  
wait_node_associated() {  
    aws_cli wait node-associated --node-association-status-token "$1"  
}  
  
install_aws_cli  
node_association_status_token=$(associate_node)  
install_chef_client  
write_chef_config  
install_trusted_certs  
wait_node_associated "${node_association_status_token}"  
  
if [ -z "${NODE_ENVIRONMENT}" ]; then  
    chef-client -r "${RUN_LIST}"  
else  
    chef-client -r "${RUN_LIST}" -E "${NODE_ENVIRONMENT}"  
fi
```

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 of the following disadvantages of other automation techniques:

- The `chef-client` agent runs in the foreground of instance processes, and logs to `stdout`. Because it is part of instance user data, the output is stored in `/var/log/cloud-init-output.log`. This directory is not rotated—or purged—and eventually runs out of disk space.
- Automation of this script doesn't survive or resume automatically if the node is restarted, or the process is terminated in some other way (for example, if the Chef server is restarted).
- Because `chef-client` runs processes in the foreground, its tasks in an instance's user data would never finish, and the instance could not signal other processes that it is finished booting. On Ubuntu nodes, for example, this can prevent other services from starting, and automatic updates from running.

Optionally, you can add one or more of the following parameters for `chef-client` in the last lines of the preceding script:

- `--runlist RunlistItem1,RunlistItem2,...`
Adds run list items to the initial Chef run list.
- `--environment ENVIRONMENT`
Sets the Chef environment on the node.
- `--json-attributes JSON_ATTRIBS`
Loads attributes from a JSON file or URL.

For a complete list of parameters you can provide to `chef-client`, see the [Chef documentation](#).

The user data shown in this topic describes how to install `chef-client`, and perform authentication similarly to `knife bootstrap`. To run specific recipes on a new node, associate a role or environment with the node, and then pass the parameter to the `chef-client` command.

Disassociate a Node from an AWS OpsWorks for Chef Automate Server

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. Both 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
```

The following command is an example.

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

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

For more information about how to delete an AWS OpsWorks for Chef Automate server, see [Delete an AWS OpsWorks for Chef Automate Server \(p. 20\)](#).

Delete an AWS OpsWorks for Chef Automate Server

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.

Important

Do not delete a server while a cookbook run, system maintenance, or backups are in progress.

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

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 command that's documented in this section to change the Chef Automate dashboard password if you have lost it.

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*](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 East (N. Virginia) Region, the console URL would be the following.

[https://console.aws.amazon.com/ec2/v2/home?
region=us-east-1#Instances:search=aws-opsworks-cm-*MyChefServer*](https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#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 the following AWS CLI command. Replace `enterprise_name` with your enterprise or organization name, `user_name` with the IAM 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.

```
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
```

3. Wait for output text (in this case, the command ID) to show that the password change is finished.

For more information about working with the Chef Automate dashboard, see [An Overview of Visibility in Chef Automate](#).

Logging AWS OpsWorks for Chef Automate API Calls with AWS CloudTrail

AWS OpsWorks for Chef Automate is integrated with CloudTrail, a service that captures all of the AWS OpsWorks for Chef Automate API calls and delivers the log files to an Amazon S3 bucket that you specify. CloudTrail captures API calls from the AWS OpsWorks for Chef Automate console, or from your code to the AWS OpsWorks for Chef Automate APIs. Using the information collected by CloudTrail, you can determine the request that was made to AWS OpsWorks for Chef Automate, the source IP address from which the request was made, who made the request, when it was made, and so on.

To learn more about CloudTrail, including how to configure and enable it, see the [AWS CloudTrail User Guide](#).

AWS OpsWorks for Chef Automate Information in CloudTrail

When CloudTrail logging is enabled in your AWS account, API calls made to AWS OpsWorks for Chef Automate actions are tracked in CloudTrail log files, where they are written with other AWS service records. CloudTrail determines when to create and write to a new file based on a time period and file size.

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](#) APIs generate entries in the CloudTrail log files.

Every log entry contains information about who generated the request. The user identity information in the log entry 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](#).

You can store your log files in your Amazon S3 bucket for as long as you want, but you can also define Amazon S3 lifecycle rules to archive or delete log files automatically. By default, your log files are encrypted with Amazon S3 server-side encryption (SSE).

If you want to be notified about log file delivery, you can configure CloudTrail to publish Amazon SNS notifications when new log files are delivered. For more information, see [Configuring Amazon SNS Notifications for CloudTrail](#).

You can also aggregate AWS OpsWorks for Chef Automate log files from multiple AWS regions and multiple AWS accounts into a single Amazon S3 bucket.

For more information, see [Receiving CloudTrail Log Files from Multiple Regions](#) and [Receiving CloudTrail Log Files from Multiple Accounts](#).

Understanding AWS OpsWorks for Chef Automate Log File Entries

CloudTrail log files can contain one or more log entries. Each entry lists multiple JSON-formatted events. A log entry 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. Log entries 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 an 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-e8346428cfcc3",
"eventType": "AwsApiCall",
"recipientAccountId": "831000000000"
}
```

Troubleshooting AWS OpsWorks for Chef Automate

This topic contains some common AWS OpsWorks for Chef Automate issues, and suggested solutions for those issues.

Topics

- [General Troubleshooting Tips \(p. 24\)](#)
- [Troubleshooting Specific Errors \(p. 24\)](#)
- [Additional help and support \(p. 27\)](#)

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 \(p. 24\)](#) 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

- [Cannot create a Chef vault; knife vault command fails with errors \(p. 25\)](#)

- Server creation fails with "requested configuration is currently not supported" message (p. 25)
- Chef server doesn't recognize organization names added in the Chef Automate dashboard (p. 26)
- Unable to create the server's Amazon EC2 instance (p. 26)
- Service role error prevents server creation (p. 26)
- Elastic IP address limit exceeded (p. 26)
- Cannot sign into the Chef Automate dashboard (p. 27)
- Unattended node association fails (p. 27)

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_ORGANIZATION value other than "default" in the [unattended node association script \(p. 16\)](#), 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 **AWSOpsWorksCMServerRole** 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 explicitly 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 \(p. 21\)](#).

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 [Adding Nodes Automatically in AWS OpsWorks for Chef Automate \(p. 16\)](#).

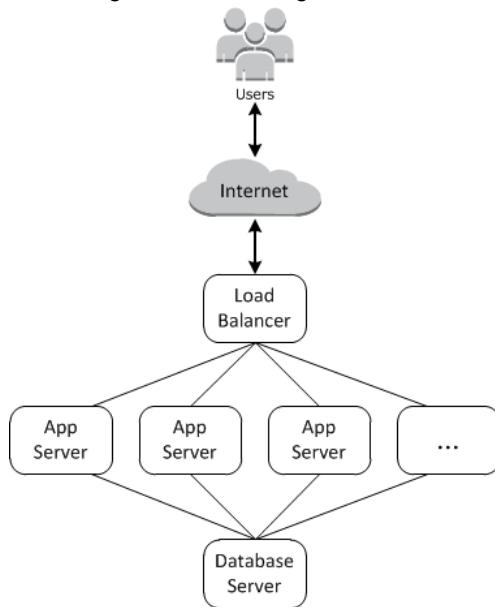
Additional help and support

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 [AWS 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 Stacks

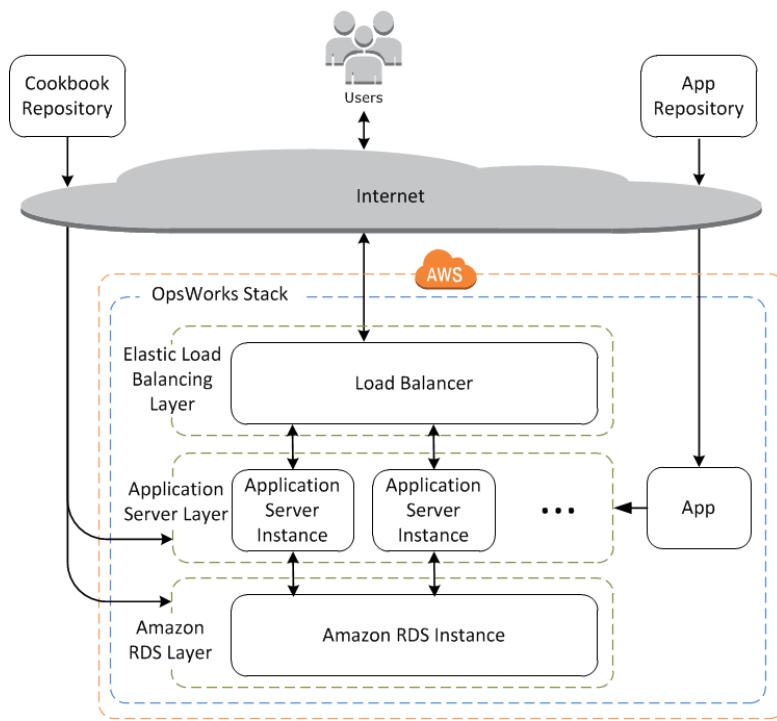
Cloud-based computing usually involves groups of AWS resources, such as Amazon EC2 instances and Amazon Relational Database Service (RDS) instances, which must be created and managed collectively. For example, a web application typically requires application servers, database servers, load balancers, 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.

AWS 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 AWS 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 AWS OpsWorks Stacks features. Here's how it's put together.

Topics

- [Stacks \(p. 30\)](#)
- [Layers \(p. 30\)](#)
- [Recipes and LifeCycle Events \(p. 30\)](#)
- [Instances \(p. 31\)](#)
- [Apps \(p. 32\)](#)
- [Customizing your Stack \(p. 32\)](#)
- [Resource Management \(p. 33\)](#)
- [Security and Permissions \(p. 33\)](#)
- [Monitoring and Logging \(p. 33\)](#)
- [CLI, SDK, and AWS CloudFormation Templates \(p. 33\)](#)
- [Getting Started with AWS OpsWorks Stacks \(p. 34\)](#)
- [AWS OpsWorks Stacks Best Practices \(p. 105\)](#)
- [Stacks \(p. 120\)](#)
- [Layers \(p. 138\)](#)
- [Instances \(p. 158\)](#)
- [Apps \(p. 216\)](#)
- [Cookbooks and Recipes \(p. 235\)](#)
- [Resource Management \(p. 256\)](#)
- [Monitoring \(p. 268\)](#)
- [Security and Permissions \(p. 281\)](#)
- [AWS OpsWorks Stacks Support for Chef 12 Linux \(p. 309\)](#)

- [Support for Previous Chef Versions in AWS OpsWorks Stacks \(p. 312\)](#)
- [Using AWS OpsWorks Stacks with Other AWS Services \(p. 561\)](#)
- [Using the AWS OpsWorks Stacks CLI \(p. 621\)](#)
- [Debugging and Troubleshooting Guide \(p. 633\)](#)
- [AWS OpsWorks Stacks Agent CLI \(p. 651\)](#)
- [AWS OpsWorks Stacks Data Bag Reference \(p. 659\)](#)
- [OpsWorks Agent Changes \(p. 673\)](#)

Stacks

The *stack* is the core AWS 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, AWS OpsWorks Stacks includes *service layers*, which represent the following AWS services.

- Amazon Relational Database Service
- Elastic Load Balancing
- Amazon EC2 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 AWS 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, AWS 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, AWS 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 AWS 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 AWS OpsWorks Stacks to create instances and add them to a layer. When you start the instance, AWS 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, AWS OpsWorks Stacks installs an agent that handles communication between the instance and the service and runs the appropriate recipes in response to lifecycle events.

AWS 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 AWS 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 AWS 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.

AWS OpsWorks Stacks supports instance autohealing. If an agent stops communicating with the service, AWS OpsWorks Stacks automatically stops and restarts the instance.

You can also incorporate Linux-based computing resources into a stack that was created outside of AWS 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 AWS OpsWorks Stacks instance and you can manage it in much the same way as instances that you create with AWS 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, AWS 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, AWS 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 AWS 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

AWS OpsWorks Stacks provides a variety of ways to customize layers to meet your specific requirements:

- You can modify how AWS 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 as Amazon S3 or Git.

You can run recipes manually, but AWS 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, AWS OpsWorks Stacks runs the associated recipes. For example, when a Deploy event occurs on an app server instance, AWS 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 AWS 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 AWS 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 AWS OpsWorks Stacks can act on your behalf to interact with AWS resources such as Amazon EC2 instances.
- How apps that run on AWS 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

AWS OpsWorks Stacks provides several features to help you monitor your stack and troubleshoot issues with your stack and any recipes. For all stacks:

- AWS OpsWorks Stacks provides a set of custom CloudWatch metrics for Linux stacks, which are summarized for your convenience on the **Monitoring** page.

AWS 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 AWS 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, AWS 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 AWS 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.

- AWS 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 AWS 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](#).

Getting Started with AWS OpsWorks Stacks

AWS 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 (p. 35)
Experiment with a Linux-based stack	Getting Started: Linux (p. 48)
Experiment with a Windows-based stack	Getting Started: Windows (p. 67)
Learn how to create your own Chef cookbooks	Getting Started: Cookbooks (p. 86)

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 \(p. 188\)](#), along with instances that you created with AWS OpsWorks Stacks. You can then use AWS OpsWorks Stacks to manage all related instance as a group, regardless of how they were created.

Region Support

You can access AWS OpsWorks Stacks globally; you can also create and manage instances globally. Users can configure AWS OpsWorks Stacks instances to be launched in any AWS region except AWS GovCloud (US) and the China (Beijing) Region. To work with AWS OpsWorks Stacks, instances must be able to connect to one of the following AWS 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 (N. Virginia) Region
- US East (Ohio) Region
- US West (Oregon) Region
- US West (N. California) Region
- Asia Pacific (Mumbai) Region
- Asia Pacific (Singapore) Region
- Asia Pacific (Sydney) Region
- Asia Pacific (Tokyo) Region
- Asia Pacific (Seoul) Region

- EU (Frankfurt) Region
- EU (Ireland) Region
- EU (London) Region
- South America (São Paulo) Region

Getting Started with a Sample Stack

In this walkthrough, you will learn how to use AWS 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 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.

Topics

- [Step 1: Complete the Prerequisites \(p. 35\)](#)
- [Step 2: Create a Stack \(p. 36\)](#)
- [Step 3: Start the Instance and Deploy the App \(p. 38\)](#)
- [Step 4: Test the Deployed App on the Instance \(p. 40\)](#)
- [Step 5: Explore the Stack's Settings \(p. 41\)](#)
- [Step 6: Explore the Layer's Settings \(p. 42\)](#)
- [Step 7: Explore the Instance's Settings and Logs \(p. 43\)](#)
- [Step 8: Explore the App's Settings \(p. 45\)](#)
- [Step 9: Explore Layer Monitoring Reports \(p. 46\)](#)
- [Step 10: Explore Additional Stack Settings \(p. 46\)](#)
- [Step 11 \(Optional\): Clean Up \(p. 46\)](#)
- [Next Steps \(p. 48\)](#)

Step 1: Complete the Prerequisites

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 IAM user in your AWS account, and assigning the IAM user access permissions to AWS OpsWorks Stacks.

Topics

- [Step 1.1: Sign up for an AWS Account \(p. 35\)](#)
- [Step 1.2: Create an IAM User in Your AWS Account \(p. 36\)](#)
- [Step 1.3: Assign Service Access Permissions to Your IAM User \(p. 36\)](#)

Step 1.1: Sign up for an AWS Account

In this step, you will sign up for an AWS account. If you already have an AWS account that you want to use for this walkthrough, you can skip ahead to [Step 1.2: Create an IAM User in Your AWS Account \(p. 36\)](#).

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

To sign up for AWS

1. Open <https://aws.amazon.com/> and choose **Create an AWS Account**.
2. Follow the online instructions.

Step 1.2: Create an IAM User in Your AWS Account

Use your AWS account to create an AWS Identity and Access Management (IAM) user. You use an IAM user to access the AWS OpsWorks Stacks service. (We don't recommend that you use your AWS account to access AWS OpsWorks Stacks directly. Doing so is generally less secure and can make it more difficult for you to troubleshoot service access issues later.) If you already have an IAM user that you want to use for this walkthrough, you can skip ahead to [Step 1.3: Assign Service Access Permissions to Your IAM User \(p. 36\)](#).

To create an IAM user, see [Creating IAM Users \(AWS Management Console\)](#). For this walkthrough, we will refer to the user as `OpsWorksDemoUser`. However, you can use a different name.

Step 1.3: Assign Service Access Permissions to Your IAM User

Set up your IAM user to enable access to the AWS OpsWorks Stacks service (and related services that AWS OpsWorks Stacks relies on).

To assign service access permissions to your IAM user, see [Attaching Managed Policies](#). Attach the `AWSOpsWorksFullAccess` and `AmazonS3FullAccess` policies to the user you created in the previous step or to the existing IAM user that you want to use.

You have now completed all of the setup steps and can [start this walkthrough \(p. 36\)](#).

Step 2: Create a Stack

In this step, you use the AWS 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 \(p. 120\)](#).) There will be only one instance for this walkthrough.

Before you begin this step, complete the [prerequisites \(p. 35\)](#).

To create the stack

1. Using your IAM user, sign in to the AWS OpsWorks Stacks console at <https://console.aws.amazon.com/opsworks>.
2. Do any of the following, if they apply:
 - If the **Welcome to AWS OpsWorks Stacks** page is displayed, choose **Add your first stack** or **Add your first AWS 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**.

Add stack

Which type of stack do you want to create?

Sample stack
Explore AWS OpsWorks with a sample Node.js app

Chef 12 stack
Bring your own cookbooks and use community cookbooks

Chef 11 stack
Use built-in cookbooks for application and deployments

Create a Chef 12 sample stack with a Node.js app
A Node.js app will be set up to help you explore the features and configuration options of AWS OpsWorks, for example: layers and lifecycle events.

[Learn more.](#)

Operating system type Linux Windows

[Cancel](#) [Create stack](#)

5. AWS OpsWorks Stacks creates a stack named `My Sample Stack (Linux)`. AWS 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 \(p. 138\)](#).) The layer is named `Node.js App Server`.
 - An *instance*, which in this case is an Ubuntu Server 14.04 LTS EC2 instance. (For more information about instances, see [Instances \(p. 158\)](#).) The instance's hostname is `nodejs-app-1`.
 - An *app*, which is code to run on the instance. (For more information about apps, see [Apps \(p. 216\)](#).) The app is named `Node.js Sample App`.
6. After AWS 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**):

Setting up a sample stack

- ✓ 1. Creating a stack named "My Sample Stack (Linux)"
- ✓ 2. Setting the Chef cookbook repository of the stack
- ✓ 3. Creating a layer named "Node.js App Server" in the stack
- ✓ 4. Assigning a recipe to the deploy lifecycle event in the layer
- ✓ 5. Adding an instance to the layer

[Cancel](#) | [Explore the sample stack](#)

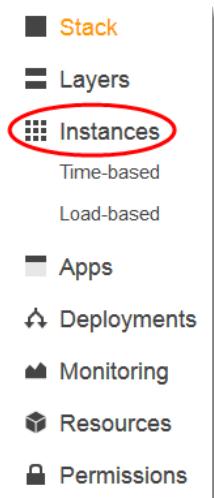
In the [next step \(p. 38\)](#), you will start the instance and deploy the app to the instance.

Step 3: Start the Instance and Deploy the App

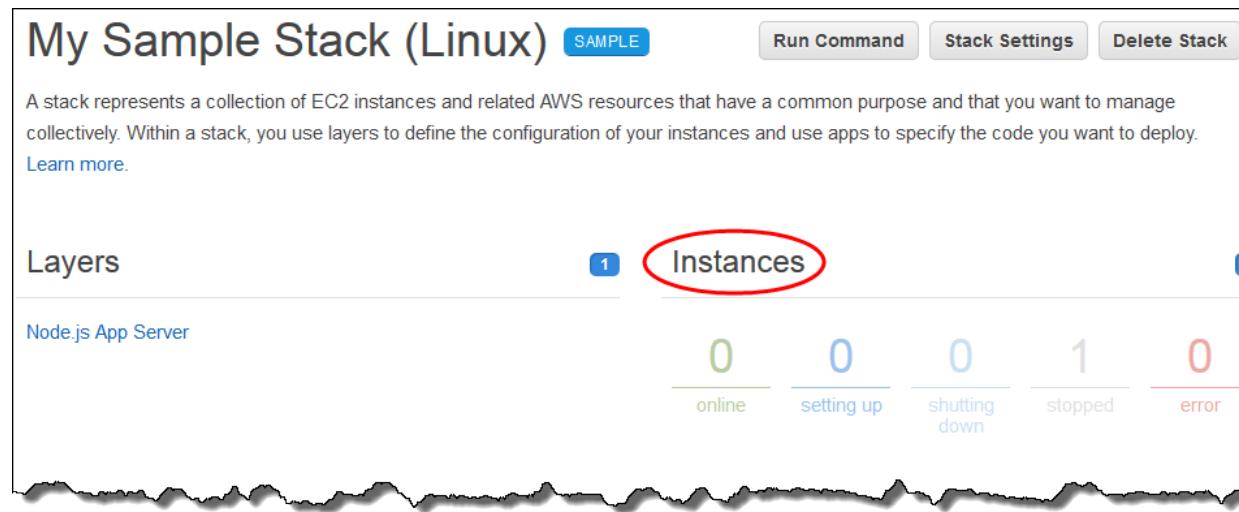
Now that you have an instance and an app, start the instance and deploy the app to the instance.

To start the instance and deploy the app

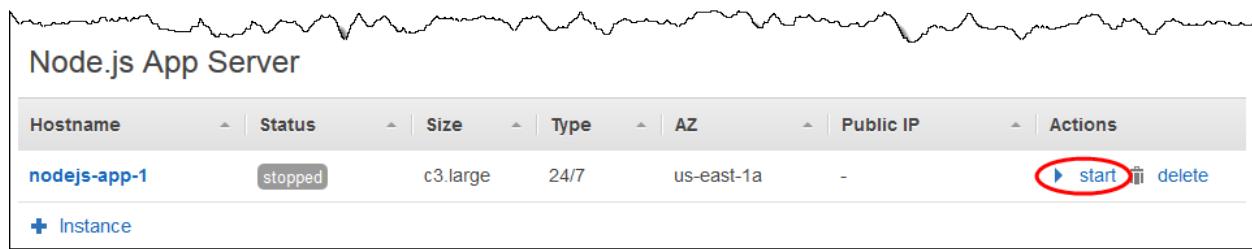
1. Do one of the following:
 - In the service navigation pane, choose **Instances**:



- On the **My Sample Stack (Linux)** page, choose **Instances**:



2. On the Instances page, for Node.js App Server, for nodejs-app-1, choose start:



3. Over the course of several minutes, the following occurs:
 - The **setting up** circle is highlighted, and its number changes from **0** to **1**.
 - **Status** changes 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 changes from **1** to **0**, **online** changes from **0** to **1**, and the **online** circle indicator is bright green. Do not proceed until the **online** circle is bright green. (If you receive a failure message, consult the [Debugging and Troubleshooting Guide \(p. 633\)](#).)
4. As the instance is setting up, AWS OpsWorks Stacks deploys the app to the instance.
5. 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 \(p. 633\)](#)):

Instances  1 | 1 | 0 | 0 | 0 | 0 | 0 | Stop All Instances

An instance represents a server. It can belong to one or more layers, that define the instance's settings, resources, installed packages, profiles and security groups. When you start the instance, OpsWorks uses the associated layer's blueprint to create and configure a corresponding EC2 instance. [Learn more](#).

Node.js App Server

Search for instances in this layer by name, status, size, type, AZ or IP						
Hostname	Status	Size	Type	AZ	Public IP	Actions
nodejs-server1	online	t2.medium	24/7	us-west-2a		■ stop ▶ ssh
+ Instance						

You now have an instance with an app that has been deployed to the instance.

In the [next step \(p. 40\)](#), you will test the app on the instance.

Step 4: Test the Deployed App on the Instance

Test the results of the app deployment on the instance.

To test the deployment on the instance

- With the **Instances** page displayed from the previous step, for **Node.js App Server**, for **Public IP**, choose the IP address:

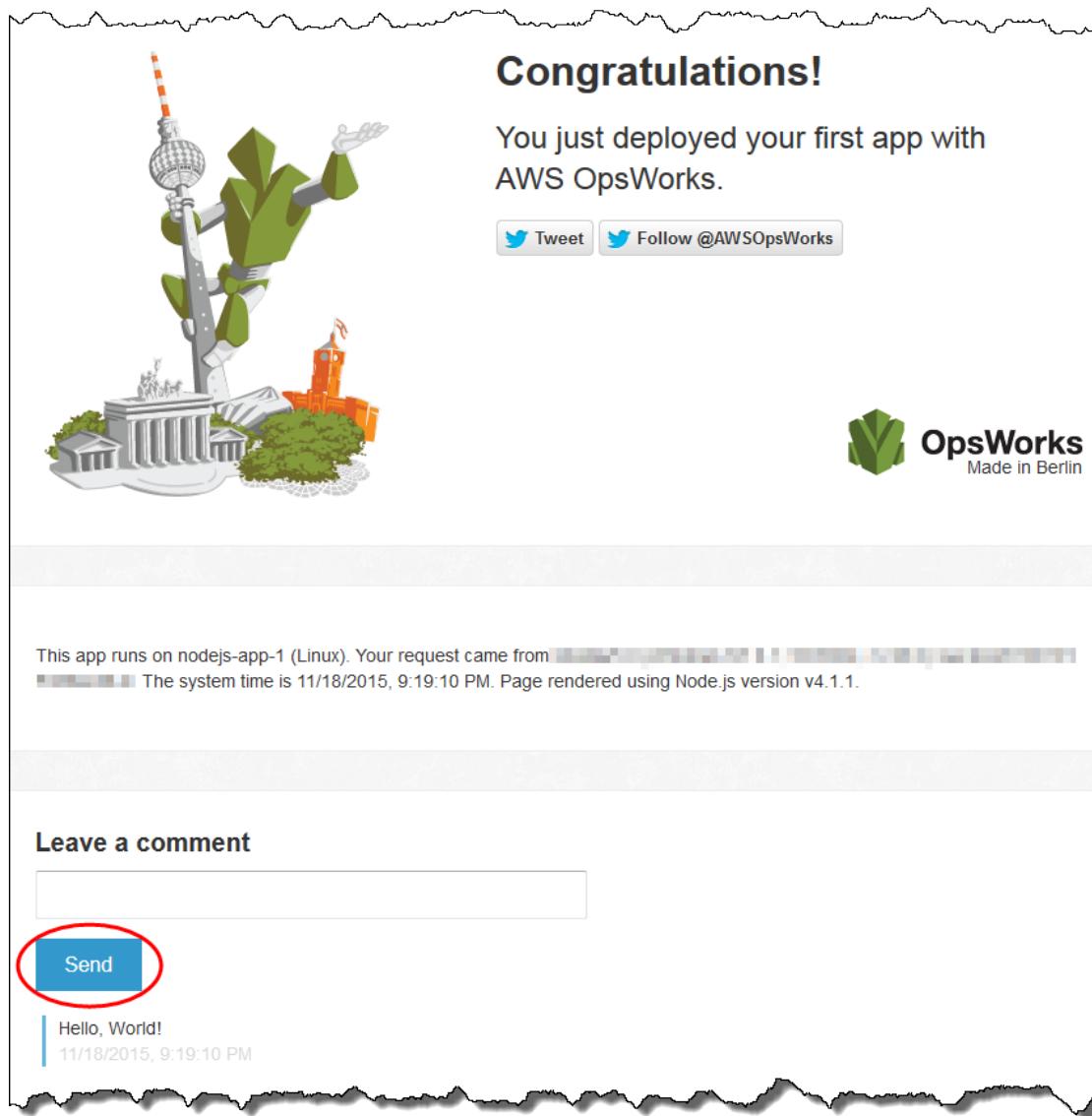
Instances  1 | 1 | 0 | 0 | 0 | 0 | 0 | Stop All Instances

An instance represents a server. It can belong to one or more layers, that define the instance's settings, resources, installed packages, profiles and security groups. When you start the instance, OpsWorks uses the associated layer's blueprint to create and configure a corresponding EC2 instance. [Learn more](#).

Node.js App Server

Search for instances in this layer by name, status, size, type, AZ or IP						
Hostname	Status	Size	Type	AZ	Public IP	Actions
nodejs-server1	online	t2.medium	24/7	us-west-2a		■ stop ▶ ssh
+ Instance						

- 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 AWS OpsWorks Stacks console to explore settings of the stack and its components. In the [next step \(p. 41\)](#), you can start your exploration by examining the stack's settings.

Step 5: Explore the Stack's Settings

Examine how AWS 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:

The screenshot shows the 'Settings' page for a stack named 'My Sample Stack (Linux)'. The page includes a table of settings with the following data:

Setting	Value
Stack name	My Sample Stack (Linux)
Region	US East (N. Virginia)
VPC	No VPC
Default Availability Zone	us-east-1a
Default operating system	Amazon Linux 2015.09
Default SSH key	No default key
Chef version	12
Use custom Chef cookbooks	yes
Repository type	Http Archive
Repository URL	https://s3.amazonaws.com/opsworks-demo-assets/opsworks-linux-demo-cookbooks-nodejs.tar.gz
User name	-

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 \(p. 120\)](#).

To explore the Chef cookbook used in this walkthrough, go to the [opsworks-linux-demo-cookbooks-nodejs](#) repository on GitHub.

In the [next step \(p. 42\)](#), you can explore the layer's settings.

Step 6: Explore the Layer's Settings

Examine how AWS 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**:

Layer Node.js App Server

Edit Delete Instances Monitoring

General Settings Recipes Network EBS Volumes Security

The Custom layer type allows you to create a fully customized layer. Standard recipes handle basic setup and configuration for the layer's instances. You can implement custom Chef recipes to install and configure any required software. You can create as many custom layers as you require. [Learn more.](#)

Settings

Name	Node.js App Server
Short name	nodejs-app-server
OpsWorks ID	[REDACTED]
Instance shutdown timeout	120 seconds
Auto healing enabled	yes

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 \(p. 140\)](#).

In the [next step \(p. 43\)](#), you can explore the instance's settings and logs.

Step 7: Explore the Instance's Settings and Logs

Examine the settings that AWS OpsWorks Stacks used to launch the instance. You can also examine the instance logs that AWS 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-app-1**. The **nodejs-app-1** page displays:

The screenshot shows the instance details for 'nodejs-app-1'. Key information includes:

- Hostname:** nodejs-app-1
- Status:** online
- Layers:** Node.js App Server
- EC2 instance ID:** i-0000000000000000
- OpsWorks ID:** [REDACTED]
- Instance type:** 24/7
- Size:** c3.large
- Availability Zone:** us-east-1a
- Operating system:** Amazon Linux 2015.09
- Reported OS:** Amazon Linux 2015.09

On the right side, there are sections for Monitoring, Volumes, and Elastic Load Balancing. The monitoring section indicates CloudWatch metrics are used for detailed monitoring.

- To explore the instance logs, in the **Logs** section, for **Log**, choose **show**:

	Created at	Command	Comment	Duration	Log
✓	2015-11-18 21:14:11 UTC	configure		00:01:09	show
✓	2015-11-18 21:10:09 UTC	setup		00:04:02	show

- AWS OpsWorks Stacks displays the corresponding log in a separate web browser tab:

The browser tab title is: ✓ Instance: nodejs-app-1 | Stack: My Sample Stack (Linux) | Layer: Node.js App Server | Type: configure

```

1 [2015-11-18T21:15:11+00:00] INFO: AWS OpsWorks instance , Agent version 4002-20151110164726
2 [2015-11-18T21:15:12+00:00] INFO: Started chef-zero at chefzero://localhost:8889 with repository at /opt/aws/opsworks/current
3 One version per cookbook
4 data_bags at /var/lib/aws/opsworks/data.internal/data_bags
5 nodes at /var/lib/aws/opsworks/data.internal/nodes
6
7 [2015-11-18T21:15:12+00:00] INFO: Forking chef instance to converge...
8 [2015-11-18T21:15:12+00:00] INFO: *** Chef 12.4.1 ***
9 [2015-11-18T21:15:12+00:00] INFO: Chef-client pid: 586
10 [2015-11-18T21:15:14+00:00] WARN: Run List override has been provided.
11 [2015-11-18T21:15:14+00:00] WARN: Original Run List: []
12 [2015-11-18T21:15:14+00:00] WARN: Overridden Run List: [recipe[aws_opsworks_agent]]
13 [2015-11-18T21:15:14+00:00] INFO: Run List is [recipe[aws_opsworks_agent]]
14 [2015-11-18T21:15:14+00:00] INFO: Run List expands to [aws_opsworks_agent]
15 [2015-11-18T21:15:14+00:00] INFO: Starting Chef Run for nodejs-app-1.us-east-1.amazonaws.com
16 [2015-11-18T21:15:14+00:00] INFO:Chef Client: 0

```

To learn more about what some of the instance settings represent, return to the **nodejs-app-1** 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 \(p. 168\)](#).

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 AWS 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 IAM user to log in to the instance. For instructions, see [Registering an IAM User's Public SSH Key \(p. 305\)](#) and [Logging In with SSH \(p. 212\)](#).

In the [next step \(p. 45\)](#), explore the app's settings.

Step 8: Explore the App's Settings

Examine the settings that AWS 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:

The screenshot shows the AWS OpsWorks 'App Node.js Sample App' configuration page. At the top right are three buttons: 'Deploy App', 'Edit', and 'Delete'. The main area is divided into sections: 'Settings', 'Application Source', and 'Data Sources'. The 'Settings' section contains fields for Name (Node.js Sample App), Short name (nodejs_sample_app), OpsWorks ID (redacted), and Type (Other). The 'Application Source' section shows App source type as Git and Repository URL as <https://github.com/awslabs/opsworks-windows-demo-nodejs.git>. The 'Data Sources' section shows Data source type as OpsWorks, with Database instance set to '(automatic selection)' and Database name set to nodejs_sample_app. Below these sections is a 'Environment Variables' section, which is currently empty.

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 \(p. 217\)](#).

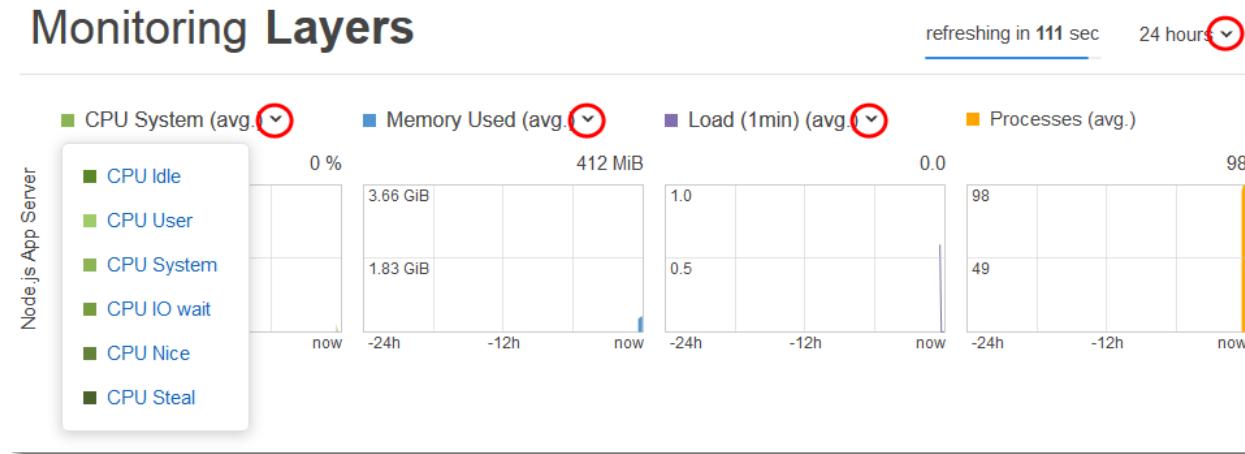
In the [next step \(p. 46\)](#), you can explore layer monitoring reports.

Step 9: Explore Layer Monitoring Reports

Examine reports that AWS 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 \(p. 269\)](#) and [Monitoring \(p. 268\)](#).

In the [next step \(p. 46\)](#), you can explore additional stack settings.

Step 10: Explore Additional Stack Settings

In this step, you can examine additional stack settings.

AWS 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**, **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 \(p. 221\)](#), [Resource Management \(p. 256\)](#), and [Managing User Permissions \(p. 282\)](#).

In the [next step \(p. 46\)](#), 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 AWS 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 \(p. 48\)](#).

Step 11 (Optional): Clean Up

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 AWS OpsWorks Stacks stack. (For more information, see [AWS OpsWorks Pricing](#).) However, you might want to keep using these AWS resources as you continue to learn more about AWS OpsWorks Stacks. If you want to keep these AWS resources available, you have now completed this walkthrough, and you can skip to [Next Steps \(p. 48\)](#).

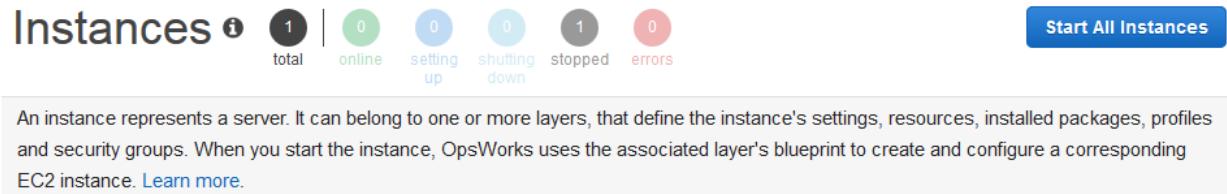
To delete the app from the stack

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-app-1**, 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 AWS OpsWorks Stacks is finished, the following results will be displayed:



Node.js App Server

Search for instances in this layer by name, status, size, type, AZ or IP						
Hostname	Status	Size	Type	AZ	Public IP	Actions
nodejs-server1	stopped	t2.medium	24/7	us-west-2a	-	start info delete
+ Instance						

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 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 [Deleting Your Key Pair](#).

You have now completed this walkthrough. For more information, see [Next Steps \(p. 48\)](#).

Next Steps

Now that you have completed this walkthrough, you can learn more about using AWS OpsWorks Stacks:

- Practice manually recreating this stack by yourself using AWS OpsWorks Stacks. See [Getting Started: Linux \(p. 48\)](#).
- Explore the cookbook and the app that AWS OpsWorks Stacks used for this walkthrough. See [Learning More: Explore the Cookbook Used in This Walkthrough \(p. 65\)](#) and [Learning More: Explore the App Used in This Walkthrough \(p. 66\)](#) in the companion [Getting Started: Linux \(p. 48\)](#) walkthrough.
- Practice using AWS OpsWorks Stacks with Windows instances. See [Getting Started: Windows \(p. 67\)](#).
- Learn more about stacks by learning how to [Create a New Stack \(p. 120\)](#).
- Learn more about layers by [Editing an OpsWorks Layer's Configuration \(p. 140\)](#).
- Learn more about instances by [Adding an Instance to a Layer \(p. 168\)](#).
- Learn more about apps by [Deploying Apps \(p. 221\)](#).
- Learn more about [Cookbooks and Recipes \(p. 235\)](#).
- Create your own cookbooks. See [Getting Started: Cookbooks \(p. 86\)](#).
- Learn about controlling access to stacks with [Security and Permissions \(p. 281\)](#).

Getting Started with Linux Stacks

In this walkthrough, you will learn how to use AWS 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 AWS OpsWorks Stacks has to offer. (For more information, see the [Learn Chef](#) website.)

AWS OpsWorks Stacks supports four Linux distributions: Amazon Linux, Ubuntu Server, CentOS, and Red Hat Enterprise Linux. For this walkthrough, we use Ubuntu Server. AWS 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 AWS OpsWorks Stacks and Chef that are not repeated there. After you complete this walkthrough, see the [Getting Started: Windows \(p. 67\)](#) walkthrough.

Topics

- [Step 1: Complete the Prerequisites \(p. 49\)](#)
- [Step 2: Create a Stack \(p. 49\)](#)
- [Step 3: Add a Layer to the Stack \(p. 52\)](#)
- [Step 4: Specify the App to Deploy to the Instance \(p. 54\)](#)
- [Step 5: Launch an Instance \(p. 56\)](#)
- [Step 6: Deploy the App to the Instance \(p. 58\)](#)
- [Step 7: Test the Deployed App on the Instance \(p. 62\)](#)
- [Step 8 \(Optional\): Clean Up \(p. 63\)](#)
- [Next Steps \(p. 65\)](#)
- [Learning More: Explore the Cookbook Used in This Walkthrough \(p. 65\)](#)
- [Learning More: Explore the App Used in This Walkthrough \(p. 66\)](#)

Step 1: Complete the Prerequisites

Complete the following setup steps before you can start the walkthrough. These setup steps include signing up for an AWS account, creating an IAM user in your AWS account, and assigning the IAM user access permissions to AWS 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 \(p. 49\)](#).

Topics

- [Step 1.1: Sign up for an AWS Account \(p. 49\)](#)
- [Step 1.2: Create an IAM User in Your AWS Account \(p. 49\)](#)
- [Step 1.3: Assign Service Access Permissions to Your IAM User \(p. 49\)](#)

Step 1.1: Sign up for an AWS Account

In this step, you will sign up for an AWS account. If you already have an AWS account that you want to use for this walkthrough, you can skip ahead to [Step 1.2: Create an IAM User in Your AWS Account \(p. 49\)](#).

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

To sign up for AWS

1. Open <https://aws.amazon.com/> and choose **Create an AWS Account**.
2. Follow the online instructions.

Step 1.2: Create an IAM User in Your AWS Account

Use your AWS account to create an AWS Identity and Access Management (IAM) user. You use an IAM user to access the AWS OpsWorks Stacks service. (We don't recommend that you use your AWS account to access AWS OpsWorks Stacks directly. Doing so is generally less secure and can make it more difficult for you to troubleshoot service access issues later.) If you already have an IAM user that you want to use for this walkthrough, you can skip ahead to [Step 1.3: Assign Service Access Permissions to Your IAM User \(p. 49\)](#).

To create an IAM user, see [Creating IAM Users \(AWS Management Console\)](#). For this walkthrough, we will refer to the user as `OpsWorksDemoUser`. However, you can use a different name.

Step 1.3: Assign Service Access Permissions to Your IAM User

Set up your IAM user to enable access to the AWS OpsWorks Stacks service (and related services that AWS OpsWorks Stacks relies on).

To assign service access permissions to your IAM user, see [Attaching Managed Policies](#). Attach the `AWSOpsWorksFullAccess` and `AmazonS3FullAccess` policies to the user you created in the previous step or to the existing IAM user that you want to use.

You have now completed all of the setup steps and can [start this walkthrough \(p. 49\)](#).

Step 2: Create a Stack

You will use the AWS 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 \(p. 120\)](#).) For this walkthrough, there is only one instance.

Before you begin, complete the [prerequisites \(p. 49\)](#) if you haven't already.

To create the stack

1. Using your IAM user, sign in to the AWS OpsWorks Stacks console at <https://console.aws.amazon.com/opsworks>.
 2. Do any of the following, if they apply:
 - If the **Welcome to AWS OpsWorks Stacks** page is displayed, choose **Add your first stack** or **Add your first AWS 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 \(p. 126\)](#).)
 - Otherwise, choose **No VPC**.
 7. For **Default operating system**, choose **Linux** and **Ubuntu 16.04 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 AWS OpsWorks Stacks to Act on Your Behalf \(p. 298\)](#)):
 - 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 \(p. 300\)](#)):
 - 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 \(p. 34\)](#).
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**:

Add stack

Which type of stack do you want to create?

  **Sample stack**

Explore AWS OpsWorks Stacks with a sample Node.js app

  **Chef 12 stack**

Bring your own cookbooks and use community cookbooks

  **Chef 11 stack**

Use built-in cookbooks for applications and deployments

Create a stack with Linux or Windows instances that run Chef 12

The more advanced experience. Bring your own cookbooks and use community cookbooks. AWS OpsWorks Stacks does separate Chef runs to isolate its internal cookbooks from yours. [Learn more](#).

Stack name: MyLinuxDemoStack

Region: US West (Oregon)

VPC: No VPC

Default Availability Zone: us-west-2a

Default operating system:  Linux  Windows
Ubuntu 16.04 LTS Need a different OS? [Let us know.](#)

Default SSH key: Do not use a default SSH key

Chef version: 12

Use custom Chef cookbooks: Yes Define the source of your Chef cookbooks

Repository type: Git

Repository URI: `https://github.com/opsworks-cookbooks/nginx`

Repository type: Git

Repository URL: https://github.com/user/cookbooks.git

Repository SSH key: Optional

Branch/Revision: Optional

Stack color: #333, #555, #777, #999, #B3B3B3, #D3D3D3, #F3F3F3

Advanced options

Default root device type: EBS backed Instance store

IAM role: aws-opsworks-service-role

Default IAM instance profile: aws-opsworks-ec2-role

API endpoint region: **us-west-2 REGIONAL** **us-east-1 CLASSIC**

Hostname theme: Layer Dependent

OpsWorks Agent version: 4021 (Dec 16th 2016)

Custom JSON: Optional

Enter custom JSON that is passed to your Chef recipes for all instances in your stack. You can use this to override and customize built-in recipes or pass variables to your own recipes. [Learn more](#).

Security

Use OpsWorks security groups:

Add stack

18. Choose **Add Stack**. AWS OpsWorks Stacks creates the stack and displays the **MyLinuxDemoStack** page.

You now have a stack with the correct settings for this walkthrough.

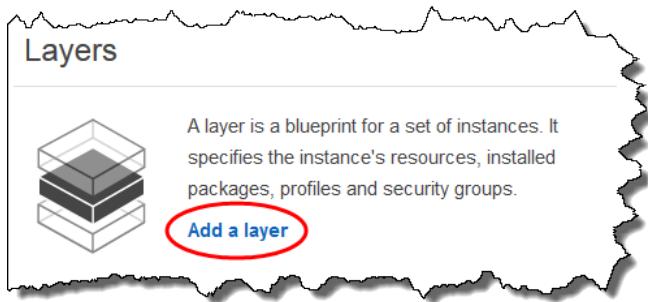
In the [next step \(p. 52\)](#), you will add a layer to the stack.

Step 3: Add a Layer to the Stack

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 \(p. 138\)](#).)

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):

Add layer

OpsWorks ECS RDS

A layer is a blueprint and container for your instances. You can add Chef recipes to lifecycle events of your instances, for example to install and configure any required software. [Learn more](#).

Name	<input type="text" value="MyLinuxDemoLayer"/>
Short name	<input type="text" value="demo"/>

Need further support? [Let us know](#).

[Cancel](#) Add layer

4. Choose **Add layer**. AWS 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**.

Automatically Assign IP Addresses i

Public IP addresses	<input checked="" type="radio"/> yes
Elastic IP addresses	<input type="radio"/> No

7. On the **Layers** page, choose **Security**:

Layers

A layer is a blueprint for a set of Amazon EC2 instances. It specifies the instance's settings, associated resources, installed packages, profiles, and security groups. You can also add recipes to lifecycle events of your instances, for example: to set up, deploy, configure your instances, or discover your resources. [Learn more](#).



8. The **Layer MyLinuxDemoLayer** page is displayed with the **Security** tab open. For **Security groups**, choose **AWS-OpsWorks-WebApp**, and then choose **Save**:

The screenshot shows the 'Layer MyLinuxDemoLayer' configuration page. The 'Security' tab is selected. In the 'Security Groups' section, a dropdown menu labeled 'Select a security group' is shown, with two options: 'AWS-OpsWorks-Default-Server' and 'AWS-OpsWorks-WebApp'. The 'AWS-OpsWorks-WebApp' option is circled in red. In the 'EC2 Instance Profile' section, a dropdown menu labeled 'Use default stack profile (aws-opswo...' is shown. At the bottom right, there are 'Cancel' and 'Save' buttons, with 'Save' being highlighted.

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 \(p. 54\)](#), you will specify the app to deploy to the instance.

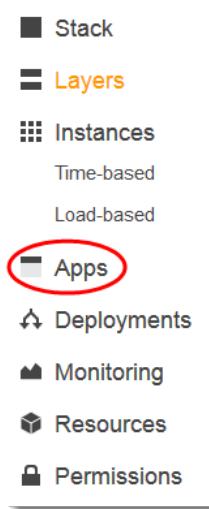
Step 4: Specify the App to Deploy to the Instance

Tell AWS OpsWorks Stacks about the app that you will deploy to the instance later in this walkthrough. In this context, AWS OpsWorks Stacks defines an **app** as code you want to run on an instance. (For more information, see [Apps \(p. 216\)](#).)

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 \(p. 319\)](#).

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/aws-labs/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)**

Add App

Settings

Name: MyLinuxDemoApp
Document root: Optional

Data Sources

Data source type: RDS None

Application Source

Repository type: Git
Repository URL: s/opsworks-windows-demo-nodejs.git
Repository SSH key: Optional

Branch/Revision: Optional

Environment Variables

KEY VALUE Protected value

Add Domains

Domain name: Optional

SSL Settings

Enable SSL: No

6. Choose **Add App**. AWS 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 \(p. 56\)](#), you will launch the instance.

Step 5: Launch an Instance

Use AWS 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 \(p. 158\)](#).)

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 16.04 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:

The screenshot shows the 'New' tab of the AWS OpsWorks instance configuration dialog. The form fields are as follows:

Hostname	demo1
Size	c3.large
Subnet	- us-west-2a
Scaling type	<input checked="" type="radio"/> 24/7 <input type="radio"/> Time-based <input type="radio"/> Load-based
SSH key	Do not set an SSH key
Operating system	Ubuntu 14.04 LTS
OpsWorks Agent version	Inherit from stack
Tenancy	Default - Rely on VPC settings
Root device type	<input checked="" type="radio"/> EBS backed <input type="radio"/> Instance store
Volume type	General Purpose (SSD)
Volume size	8 <small>Min: 8 GiB, Max: 16384 GiB</small>

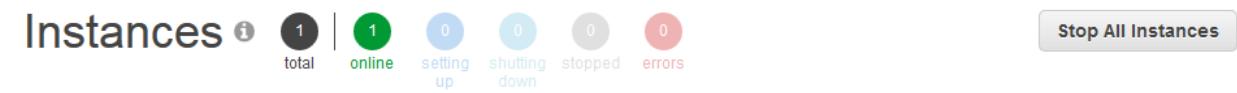
At the bottom right of the dialog are two buttons: 'Cancel' and 'Add Instance'.

7. Choose **Add Instance**. AWS OpsWorks Stacks adds the instance to the layer and displays the **Instances** page.

- For **MyLinuxDemoLayer**, for **demo1**, for **Actions**, choose **start**:

Hostname	Status	Size	Type	AZ	Public IP	Actions
demo1	stopped	c3.large	24/7	us-west-2a	-	start delete
+ Instance						

- 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.
- 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 \(p. 633\)](#)):



Hostname	Status	Size	Type	AZ	Public IP	Actions
demo1	online	c3.large	24/7	us-west-2a		stop ssh
+ Instance						

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 AWS 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 IAM user to log in to the instance. For instructions, see [Registering an IAM User's Public SSH Key \(p. 305\)](#) and [Logging In with SSH \(p. 212\)](#). 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 \(p. 58\)](#), you will deploy the app to the instance.

Step 6: Deploy the App to the Instance

In this step, you will deploy the app from GitHub to the running instance. (For more information, see [Deploying Apps \(p. 221\)](#).) Before you deploy the app, you must specify the *recipe* to use to coordinate the

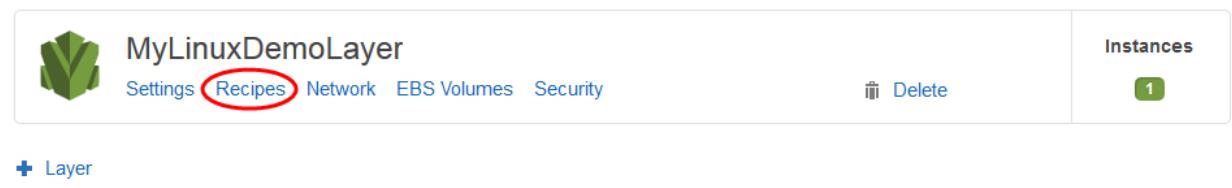
deployment. A recipe is a Chef concept. Recipes are instructions, written with Ruby language syntax, that specify the resources to use and the order in which those resources are applied. (For more information, go to [About Recipes](#) on the [Learn Chef](#) website.)

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**:

Layers

A layer is a blueprint for a set of Amazon EC2 instances. It specifies the instance's settings, associated resources, installed packages, profiles, and security groups. You can also add recipes to lifecycle events of your instances, for example: to set up, deploy, configure your instances, or discover your resources. [Learn more](#).



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 \(p. 65\)](#).) Your results must match the following screenshot:

The screenshot shows the 'Layer MyLinuxDemoLayer' configuration page. Under the 'Custom Chef Recipes' section, the 'Deploy' tab is selected. It lists the following recipes:

Event	Target	Action
Setup	mycookbook::myrecipe, mycookt	+
Configure	mycookbook::myrecipe, mycookt	+
Deploy	mycookbook::myrecipe, mycookt	+
	nodejs_demo::default	✖
Undeploy	mycookbook::myrecipe, mycookt	+
Shutdown	mycookbook::myrecipe, mycookt	+

At the bottom right, there are 'Cancel' and 'Save' buttons.

4. Choose **Save**. AWS 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:

Name	Type	Data Source	Last Deployment	Actions
MyLinuxDemoApp	Other			deploy
+ App				

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:

Deploy App

Settings

App	MyLinuxDemoApp
Command	Deploy
Deploy an app.	
Comment	Optional
Custom Chef JSON	Optional

Enter custom JSON that is passed to your Chef recipes for all instances in your stack. You can use this to override and customize built-in recipes or pass variables to your own. [Learn more](#).

Instances i

OpsWorks will run this command on **1 of 1** instances. The assigned recipes are run on all selected instances.

- Select all**
- MyLinuxDemoLayer** demo1 ●
Click to select instances in this layer

[Cancel](#) [Deploy](#)

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:

Deployment MyLinuxDemoApp - deploy

[Repeat](#)

Status	successful	User	OpsWorksDemoUser	
Created at	2015-11-12 17:12:49 UTC			
Completed at	2015-11-12 17:14:02 UTC			
Duration	00:01:13			
<hr/>				
▲ Hostname	▲ SSH	▲ Layers	▲ Duration	▲ Log
✓ demo1	ssh	MyLinuxDemoLayer	00:01:13	show

You have now successfully deployed the app to the instance.

In the [next step \(p. 62\)](#), you will test the deployed app on the instance.

Step 7: Test the Deployed App on the Instance

Now, test the app deployment on the instance.

To test the deployment on the instance

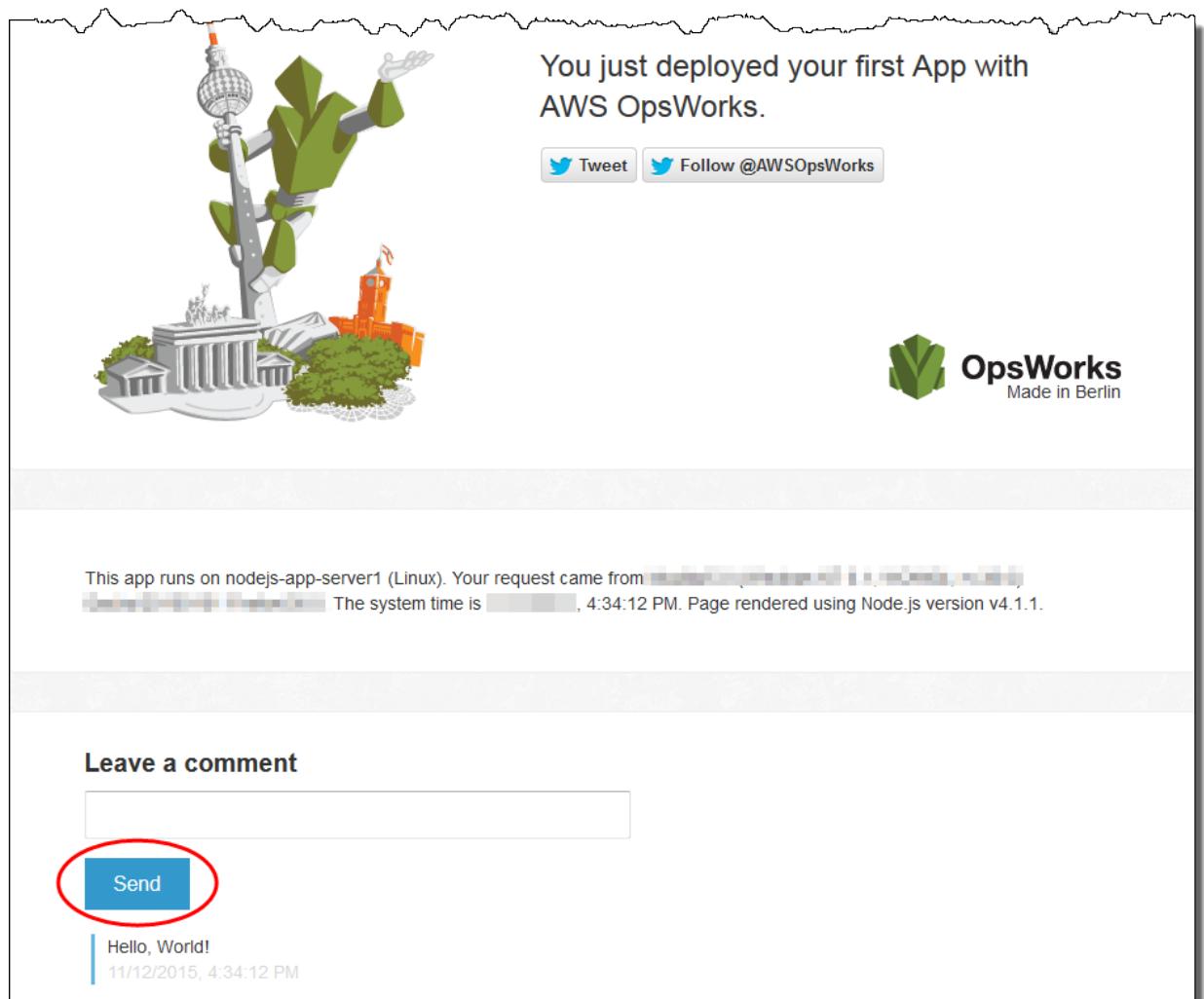
1. In the service navigation pane, choose **Instances**. The **Instances** page is displayed.
2. For **MyLinuxDemoLayer**, for **demo1**, for **Public IP**, choose the IP address:

MyLinuxDemoLayer						
Search for instances in this layer by name, status, size, type, AZ or IP						
Hostname	Status	Size	Type	AZ	Public IP	Actions
demo1	online	c3.large	24/7	us-west-2a	[Public IP]	stop ssh

[+ Instance](#)

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 \(p. 63\)](#), 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 AWS 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 \(p. 65\)](#).

Step 8 (Optional): Clean Up

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 AWS 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 AWS OpsWorks Stacks. If you want to keep these AWS resources available, you have now completed this walkthrough, and you can skip to [Next Steps \(p. 65\)](#).

To delete the app from the stack

1. In the AWS 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**. AWS 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 AWS OpsWorks Stacks is finished, the following results will be displayed:

The screenshot shows the AWS OpsWorks Stacks Instances page. At the top, there's a summary bar with the following counts: total 1, online 0, setting up 0, shutting down 0, stopped 1, and errors 0. To the right is a blue button labeled "Start All Instances". Below this is a table titled "MyLinuxDemoLayer" showing one instance named "demo1" which is currently "stopped". The table has columns for Hostname, Status, Size, Type, AZ, Public IP, and Actions. The Actions column includes links for "start" and "delete". At the bottom left of the table is a "+ Instance" button.

MyLinuxDemoLayer						
Search for instances in this layer by name, status, size, type, AZ or IP						
Hostname	Status	Size	Type	AZ	Public IP	Actions
demo1	stopped	c3.large	24/7	us-west-2a		start delete
+ Instance						

3. For **Actions**, choose **delete**. When you see the confirmation message, choose **Delete**. AWS 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**. AWS OpsWorks Stacks deletes the stack and displays the **OpsWorks 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 [Deleting Your Key Pair](#).

You have now completed this walkthrough. For more information, see [Next Steps \(p. 65\)](#).

Next Steps

Now that you have completed this walkthrough, you can learn more about using AWS OpsWorks Stacks:

- Explore the cookbook and the app that you used for this walkthrough. See [Learning More: Explore the Cookbook Used in This Walkthrough \(p. 65\)](#) and [Learning More: Explore the App Used in This Walkthrough \(p. 66\)](#).
- Practice using AWS OpsWorks Stacks with Windows instances. See [Getting Started: Windows \(p. 67\)](#).
- Learn more about stacks by learning how to [Create a New Stack \(p. 120\)](#).
- Learn more about layers by [Editing an OpsWorks Layer's Configuration \(p. 140\)](#).
- Learn more about instances by [Adding an Instance to a Layer \(p. 168\)](#).
- Learn more about apps by [Deploying Apps \(p. 221\)](#).
- Learn more about [Cookbooks and Recipes \(p. 235\)](#).
- Create your own cookbooks. See [Getting Started: Cookbooks \(p. 86\)](#).
- Learn about controlling access to stacks with [Security and Permissions \(p. 281\)](#).

Learning More: Explore the Cookbook Used in This Walkthrough

This topic describes the cookbook that AWS 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 starts running 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"] ==
  "16.04"
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
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, AWS 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 [AWS OpsWorks Stacks Data Bag Reference \(p. 659\)](#). 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, AWS 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 AWS OpsWorks Stacks created the cookbook used for this walkthrough, you can create your own cookbooks. To learn how, see [Getting Started: Cookbooks \(p. 86\)](#). 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

This topic describes the app that AWS 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);
});

```

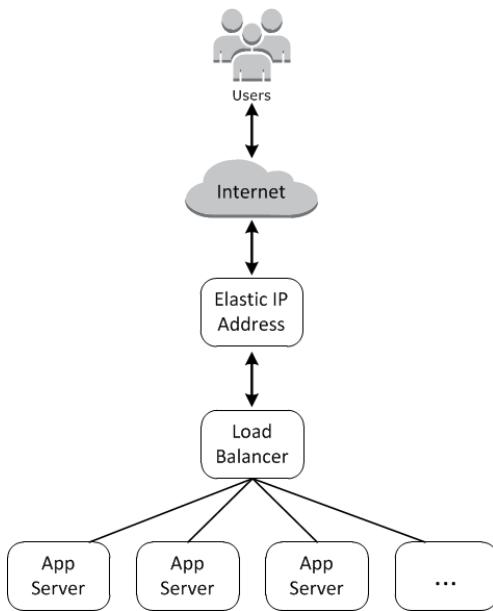
Here's what the file does:

- `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 \(p. 218\)](#).

Getting Started with Windows Stacks

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.

AWS 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 AWS 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 AWS 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 \(p. 69\)](#) shows how to get set up to start the walkthrough.
- [Step 2: Create a Basic Application Server Stack \(p. 69\)](#) shows how to create a basic stack to support Internet Information Server (IIS) and deploy an app to the server.
- [Step 3: Scale Out IISExample \(p. 83\)](#) 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 \(p. 69\)](#)
- [Step 2: Create a Basic Application Server Stack \(p. 69\)](#)
- [Step 3: Scale Out IISExample \(p. 83\)](#)
- [Next Steps \(p. 86\)](#)

Step 1: Complete the Prerequisites

Complete the following setup steps before you can start the walkthrough. These setup steps include signing up for an AWS account, creating an IAM user in your AWS account, and assigning the IAM user access permissions to AWS OpsWorks Stacks.

If you have already completed the [Getting Started: Sample \(p. 35\)](#) or [Getting Started: Linux \(p. 48\)](#) walkthroughs, then you have met the prerequisites for this walkthrough, and you can skip ahead to [Step 2: Create a Basic Application Server Stack \(p. 69\)](#).

Topics

- [Step 1.1: Sign up for an AWS Account \(p. 69\)](#)
- [Step 1.2: Create an IAM User in Your AWS Account \(p. 69\)](#)
- [Step 1.3: Assign Service Access Permissions to Your IAM User \(p. 69\)](#)

Step 1.1: Sign up for an AWS Account

In this step, you will sign up for an AWS account. If you already have an AWS account that you want to use for this walkthrough, you can skip ahead to [Step 1.2: Create an IAM User in Your AWS Account \(p. 69\)](#).

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

To sign up for AWS

1. Open <https://aws.amazon.com/> and choose **Create an AWS Account**.
2. Follow the online instructions.

Step 1.2: Create an IAM User in Your AWS Account

Use your AWS account to create an AWS Identity and Access Management (IAM) user. You use an IAM user to access the AWS OpsWorks Stacks service. (We don't recommend that you use your AWS account to access AWS OpsWorks Stacks directly. Doing so is generally less secure and can make it more difficult for you to troubleshoot service access issues later.) If you already have an IAM user that you want to use for this walkthrough, you can skip ahead to [Step 1.3: Assign Service Access Permissions to Your IAM User \(p. 69\)](#).

To create an IAM user, see [Creating IAM Users \(AWS Management Console\)](#). For this walkthrough, we will refer to the user as `OpsWorksDemoUser`. However, you can use a different name.

Step 1.3: Assign Service Access Permissions to Your IAM User

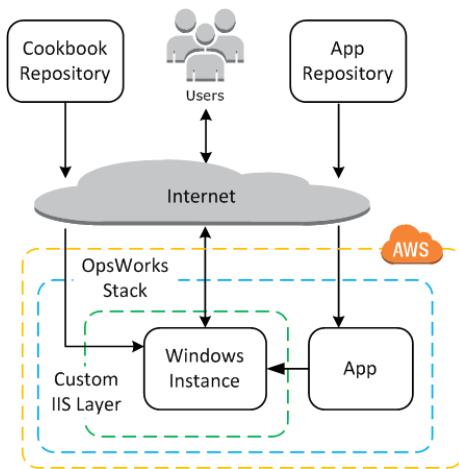
Set up your IAM user to enable access to the AWS OpsWorks Stacks service (and related services that AWS OpsWorks Stacks relies on).

To assign service access permissions to your IAM user, see [Attaching Managed Policies](#). Attach the `AWSOpsWorksFullAccess` and `AmazonS3FullAccess` policies to the user you created in the previous step or to the existing IAM user that you want to use.

You have now completed all of the setup steps and can [start this walkthrough \(p. 69\)](#).

Step 2: Create a Basic Application Server Stack

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 AWS OpsWorks Stacks console to create the stack and deploy the application.

Topics

- [Step 2.1: Create the Stack \(p. 70\)](#)
- [Step 2.2: Authorize RDP Access \(p. 71\)](#)
- [Step 2.3: Implement a Custom Cookbook \(p. 72\)](#)
- [Step 2.4: Add an IIS Layer \(p. 76\)](#)
- [Step 2.5: Deploy an App \(p. 78\)](#)

Step 2.1: Create the Stack

You start an AWS 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 [AWS 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 AWS 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 2012 R2 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. AWS 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 AWS OpsWorks Stacks assumes to work with other AWS services on your behalf. For more information, see [Allowing AWS OpsWorks Stacks to Act on Your Behalf \(p. 298\)](#).

- If your account has an existing AWS OpsWorks Stacks IAM role, you can select it from the list.

If the role was created by AWS OpsWorks Stacks, it will be named `aws-opsworks-service-role`.

- Otherwise, select **New IAM Role** to direct AWS OpsWorks Stacks to create a new role for you with the correct permissions.

Note: If you have AWS OpsWorks Stacks Full Access permissions, creating a new role requires several additional IAM permissions. For more information, see [Example Policies \(p. 292\)](#).

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 \(p. 120\)](#).

Step 2.2: Authorize RDP Access

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 AWS OpsWorks Stacks permissions for this stack to allow RDP access.

When you create the first stack in a region, AWS OpsWorks Stacks creates a set of security groups. They include one named something like `AWS-OpsWorks-RDP-Server`, which AWS OpsWorks Stacks attaches to

API Version 2013-02-18

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 AWS OpsWorks Stacks. To have AWS OpsWorks Stacks generate that password, you must explicitly authorize RDP access for the user.

To authorize RDP for a user

1. In the AWS 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 IAM user to whom you want to grant necessary permissions. If you want the user to also have administrator permissions, select **sudo/admin** as well.

Stack	Permission level					Instance access	
	Deny	IAM Policies Only	Show	Deploy	Manage	SSH / RDP	sudo / admin
CLITest	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>
Chef9Test	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>
EC2Register	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
JavaStack	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>

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 \(p. 216\)](#).

Step 2.3: Implement a Custom Cookbook

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 AWS 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 AWS OpsWorks Stacks, each recipe is usually assigned to one of the layer's [lifecycle events \(p. 253\)](#): Setup, Configuration, Deploy, Undeploy, and Shutdown. When a lifecycle event occurs on an instance, AWS OpsWorks Stacks runs the event's recipes to perform the appropriate tasks. For example, the Setup event occurs after an instance finishes booting. AWS OpsWorks Stacks then runs the Setup recipes, which typically perform tasks such as installing and configuring server software and starting related services.

AWS 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 \(p. 157\)](#), 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 \(p. 73\)](#)
- [Implement a Recipe to Install and Start IIS \(p. 74\)](#)
- [Enable the Custom Cookbook \(p. 75\)](#)

[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 \(p. 235\)](#). For a detailed tutorial introduction to implementing cookbooks, including some Windows-specific topics, see [Cookbooks 101 \(p. 382\)](#).

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 \(p. 235\)](#).

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 `Add-WindowsFeature` cmdlet.
- By using the Chef `windows cookbook's windows_feature resource`.

The `windows` cookbook contains a set of resources whose underlying providers use [Deployment Image Servicing and Management](#) 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 wide variety of tasks on an instance by running the appropriate Windows PowerShell script. It's especially useful for tasks that aren't supported by a Chef resource.

This example uses a Windows PowerShell script to install and start 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 \(p. 452\)](#).

Add a recipe named `install.rb` with the following contents to the cookbook's `recipes` directory.

```
powershell_script 'Install IIS' do
  code 'Add-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 Windows PowerShell script. The example has the following attribute settings:

- `code` – The Windows PowerShell cmdlets to be run.

This example runs a single `Add-WindowsFeature` cmdlet, which installs 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 Windows PowerShell script. For more information, see [Running a Windows PowerShell Script \(p. 420\)](#).

service

A `service` manages a service, the 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 \(p. 454\)](#).

[Enable the Custom Cookbook](#)

AWS OpsWorks Stacks runs recipes from a local cache on each instance. To run your custom recipes, you must do the following:

- Put the cookbook in a remote repository.

AWS 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.

AWS OpsWorks Stacks supports S3 archives and Git repositories for custom cookbooks; this example uses an S3 archive. For more information, see [Cookbook Repositories \(p. 235\)](#).

To use an S3 archive

1. Create a `.zip` archive of the `iis-cookbook` directory.

AWS OpsWorks Stacks also supports .tgz (gzip compressed tar) archives for Windows stacks.

2. Upload the archive to an S3 bucket in the US Standard (us-east-1) 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.

To upload the archive to an S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. If you do not already have a bucket in us-east-1, click **Create Bucket** and create a bucket in the US Standard region.
3. In the buckets list, click the name of bucket to which you want to upload the file, and then click **Upload**.
4. Choose **Add Files**.
5. Select the archive file that you will upload, and then click **Open**.
6. At the bottom of the **Upload - Select Files and Folders** dialog, choose **Set Details**.
7. At the bottom of the **Set Details** dialog, choose **Set Permissions**.
8. In the **Set Permissions** dialog, choose **Make everything public**.
9. At the bottom of the **Set Permissions** dialog, choose **Start Upload**. When the upload finishes, the `iis-cookbook.zip` file will appear in your bucket.
10. Select the bucket in the bucket list, and then choose the **Properties** tab for the bucket. Next to **Link**, record the archive file's URL for later use. It should look something like `https://s3-us-west-2.amazonaws.com/windows-cookbooks/iis-cookbook.zip`.

For more information about uploading files to an Amazon S3 bucket, see [Uploading Objects into Amazon S3](#).

Important

Up to this point, the walkthrough has cost you only a little time; the AWS 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 AWS 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.

AWS OpsWorks Stacks will now install your custom cookbook on all new instances. Note that AWS 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

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`

AWS 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 \(p. 157\)](#).

3. Choose **Add Layer**.

If you were to add an instance to IISWalkthrough at this point and start it, AWS 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 \(p. 133\)](#). However, a better approach is to assign the recipe to one of the layer's [lifecycle events \(p. 253\)](#). AWS OpsWorks Stacks then automatically runs the recipe at the appropriate point in the instance's lifecycle.

You will probably want to install and start IIS as soon as the instance finishes booting. You can accomplish this by assigning `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

You 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. AWS OpsWorks Stacks will automatically install the cookbooks and run `install.rb` during setup, as soon as the instance finishes booting.

To add an instance to a layer and start it

1. In the AWS 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, AWS 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. AWS 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, AWS 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. AWS 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. AWS 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 AWS OpsWorks Stacks console and choose the **iisexample1** instance's public IP address. 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

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 \(p. 397\)](#). 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 \(p. 79\)](#)
- [Implement a Recipe to Deploy the Application \(p. 79\)](#)
- [Update the Instance's Cookbooks \(p. 81\)](#)
- [Add the Recipe to the Custom IIS Layer \(p. 82\)](#)
- [Add an App \(p. 82\)](#)
- [Deploy the App and Open the Application \(p. 83\)](#)

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 that file and extract the contents to the appropriate directory. There's no need to download multiple files, create a directory structure, and 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 AWS OpsWorks Stacks Windows Instance \(p. 448\)](#).
- 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 recipies` directory, with the following contents.

```

chef_gem "aws-sdk" do
  compile_time false
  action :install
end

ruby_block "download-object" do
  block do
    require 'aws-sdk'

    #1
    Aws.config[:ssl_ca_bundle] = 'C:\ProgramData\Git\bin\curl-ca-bundle.crt'

    #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, AWS 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. AWS 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, AWS 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, you use a Chef search query to retrieve the app's hash table, which are all 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 \(p. 429\)](#)

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

AWS 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 will overwrite the old cookbook, but the URL will stay 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 \(p. 133\)](#). This command will install the updated cookbook on the instance.
5. Choose **Update Custom Cookbooks**. The command may take a few minutes to complete.

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, AWS 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 \(p. 221\)](#) 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 piece of the puzzle is to add an app to the stack to represent your application in the AWS 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 as follows:
 - **Name** – `IIS-Example-App`
 - **Repository Type** – `Other`
 - **Environment Variables** – Add the following three environment variables:
 - `S3REGION` – The bucket's region (in this case, `us-east-1`).
 - `BUCKET` – The bucket name, such as `windows-example-app`.
 - `FILENAME` – The file name: `default.htm`.
3. Accept the 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. AWS OpsWorks Stacks will add the information, along with optional data, such as your AWS credentials, to the app's `app_source` attribute. Your deploy recipe would then need to retrieve 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

AWS OpsWorks Stacks automatically deploys apps to new instances, but not to online instances. Because your instance is already running, you will have to 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**, so click the **Deploy** button at the lower right of the **Deploy App** page. The command may take a few minutes to complete.

After the deployment is complete, you will automatically return to the **Apps** page. The **Status** indicator will show **successful** in green, and the app name will have 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 \(p. 376\)](#), as described earlier.
- Triggers a Deploy event on the stack's instances, which runs your custom Deploy recipes.

Note

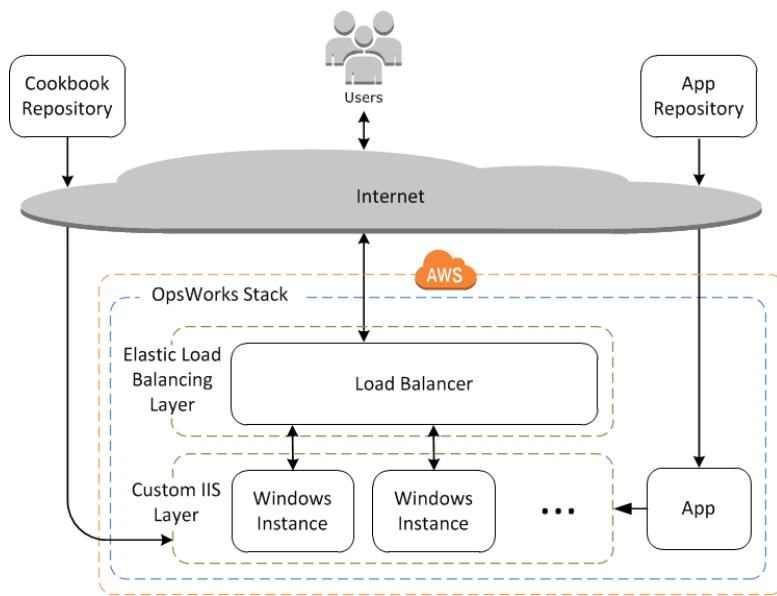
For more information about how to troubleshoot failed deployments or applications, see [Debugging Recipes \(p. 634\)](#).

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.

Hello World!

Step 3: Scale Out IIS Example

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.

AWS 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 AWS OpsWorks Stacks scale your stack automatically. For more information, see [Managing Load with Time-based and Load-based Instances \(p. 181\)](#).

Add a Load Balancer

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

AWS OpsWorks Stacks does not support Application Load Balancer. You can only use Classic Load Balancer with AWS 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. AWS 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 AWS 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

AWS 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 AWS 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, AWS 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, AWS OpsWorks Stacks automatically installs the current custom cookbooks and deploys the current app version during setup. When the instance comes online, AWS 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

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 \(p. 86\)](#) provides a tutorial introduction to implementing cookbooks, and includes a number of AWS OpsWorks Stacks-specific examples.
- You can add an [Amazon Relational Database Service \(Amazon RDS\) layer \(p. 150\)](#) 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 \(p. 225\)](#).

Getting Started with Cookbooks in AWS OpsWorks Stacks

A production-level AWS 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 AWS OpsWorks Stacks. To learn more about Chef, cookbooks, recipes, and resources, see the links in [Next Steps \(p. 105\)](#).

This walkthrough mostly describes how to create your own cookbooks. However, you can also use community-provided cookbooks through locations such as 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, you need to complete a few setup steps. If you have already completed any of the other walkthroughs in this chapter, such as [Getting Started: Sample \(p. 35\)](#), then you have

met the prerequisites for this walkthrough and can skip to [start this walkthrough \(p. 87\)](#). Otherwise, be sure to complete the [prerequisites \(p. 35\)](#), and then return to this walkthrough.

Topics

- [Step 1: Create the Cookbook \(p. 87\)](#)
- [Step 2: Create the Stack and its Components \(p. 88\)](#)
- [Step 3: Run and Test the Recipe \(p. 90\)](#)
- [Step 4: Update the Cookbook to Install a Package \(p. 91\)](#)
- [Step 5: Update the Cookbook on the Instance and Run the Recipe \(p. 91\)](#)
- [Step 6: Update the Cookbook to Add a User \(p. 92\)](#)
- [Step 7: Update the Cookbook to Create a Directory \(p. 93\)](#)
- [Step 8: Update the Cookbook to Create and Copy Files \(p. 94\)](#)
- [Step 9: Update the Cookbook to Run a Command \(p. 95\)](#)
- [Step 10: Update the Cookbook to Run a Script \(p. 96\)](#)
- [Step 11: Update the Cookbook to Manage a Service \(p. 97\)](#)
- [Step 12: Update the Cookbook to Use Custom JSON \(p. 98\)](#)
- [Step 13: Update the Cookbook to Use Data Bags \(p. 99\)](#)
- [Step 14: Update the Cookbook to Use Iteration \(p. 100\)](#)
- [Step 15: Update the Cookbook to Use Conditional Logic \(p. 101\)](#)
- [Step 16: Update the Cookbook to Use Community Cookbooks \(p. 102\)](#)
- [Step 17: \(Optional\) Clean Up \(p. 104\)](#)
- [Next Steps \(p. 105\)](#)

Step 1: Create the Cookbook

Start by creating a cookbook. This cookbook won't do much to start. However, 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, use a text editor to create a file named `metadata.rb` with the following code to specify the cookbook's name (for more information, go to [metadata.rb](#)):

```
name "opsworks_cookbook_demo"
```

3. In the `opsworks_cookbook_demo` directory, create a subdirectory named `recipes`. This subdirectory will contain all of the recipes that you will 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 \(p. 88\)](#), you will create an AWS OpsWorks Stacks stack and its components—including an instance—that you will use later to upload your cookbook and to run the cookbook's recipes.

Step 2: Create the Stack and its Components

Create an AWS OpsWorks Stacks stack and its components, which include a layer and an instance. In later steps, you will upload your cookbook to the instance and then run the cookbook's recipes on that instance.

To create the stack

1. Using your AWS Identity and Access Management (IAM) user, sign in to the AWS OpsWorks Stacks console at <https://console.aws.amazon.com/opsworks>.
2. Do one of the following, if they apply:
 - If the **Welcome to AWS OpsWorks Stacks** page is displayed, choose **Add your first stack** or **Add your first AWS 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 \(p. 126\)](#).
 - 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 \(p. 48\)](#) 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/opsworks-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**. AWS 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**. AWS 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 \(p. 90\)](#), you will run and test the default recipe from within the cookbook on the instance.

Step 3: Run and Test the Recipe

Run and test the `default` recipe from within the cookbook that AWS 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 \(p. 91\)](#), 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

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 AWS 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 AWS 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 \(p. 91\)](#), 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 be able to log in to the instance, you must first provide AWS 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 IAM user to log in to the instance. For instructions, see [Registering an IAM User's Public SSH Key \(p. 305\)](#) and [Logging In with SSH \(p. 212\)](#).

Step 5: Update the Cookbook on the Instance and Run the Recipe

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 AWS OpsWorks Stacks will run those recipes automatically when the event occurs. For more information, see [AWS OpsWorks Stacks Lifecycle Events \(p. 253\)](#).

In the [next step \(p. 92\)](#), you will update the cookbook to add a user to the instance.

Step 6: Update the Cookbook to Add a User

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 \(p. 284\)](#).

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 \(p. 91\)](#) 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 \(p. 93\)](#), you will update the cookbook to create a directory on the instance.

Step 7: Update the Cookbook to Create a Directory

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 \(p. 91\)](#) 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.

- 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 \(p. 94\)](#), you will update the cookbook to create a file on the instance.

Step 8: Update the Cookbook to Create and Copy Files

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 \(p. 93\)](#) to make sure that the files' parent directory already exists.

To update the cookbook on the instance and run the new recipe

- On your local workstation, in the `opsworks_cookbook_demo` directory, create a subdirectory named `files`.
- In the `files` subdirectory, create a file named `hello.txt` with the following text: `Hello, World!`
- 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.

- 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.
- 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 \(p. 91\)](#) 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 \(p. 95\)](#), you will update the cookbook to run a command on the instance.

Step 9: Update the Cookbook to Run a Command

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 \(p. 91\)](#) 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 \(p. 96\)](#), you will update the cookbook to run a script on the instance.

Step 10: Update the Cookbook to Run a Script

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 \(p. 91\)](#) 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 \(p. 97\)](#), you will update the cookbook to manage a service on the instance.

Step 11: Update the Cookbook to Manage a Service

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 \(p. 91\)](#) 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 \(p. 98\)](#), you will update the cookbook to reference information stored as custom JSON on the instance.

Step 12: Update the Cookbook to Use Custom JSON

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 \(p. 135\)](#).

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

- 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.

- 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.
- Upload the updated `opsworks_cookbook_demo.tar.gz` file to your S3 bucket.
- Follow the procedures in [Step 5: Update the Cookbook on the Instance and Run the Recipe \(p. 91\)](#) 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

- 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.
- 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 \(p. 99\)](#), you will update the cookbook to get information from data bags, which are collections of stack settings that AWS OpsWorks Stacks stores on each instance.

Step 13: Update the Cookbook to Use Data Bags

Update your cookbook by adding a recipe that references stack settings that AWS 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 [AWS OpsWorks Stacks Data Bag Reference \(p. 659\)](#).

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 \(p. 91\)](#) 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/opsworks-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 \(p. 100\)](#), you will update the cookbook to run recipe code multiple times.

Step 14: Update the Cookbook to Use Iteration

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

- 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']}\"]' *****")
```

- 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.
- Upload the updated `opsworks_cookbook_demo.tar.gz` file to your S3 bucket.
- Follow the procedures in [Step 5: Update the Cookbook on the Instance and Run the Recipe \(p. 91\)](#) 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

- 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.
- 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/
opsworks-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 \(p. 101\)](#), you will update the cookbook to run recipe code only if certain conditions are met.

Step 15: Update the Cookbook to Use Conditional Logic

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

- 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 12.04 LTS" || os == "Ubuntu 14.04 LTS" || os == "Ubuntu 16.04 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"
  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 12.04 LTS", "Ubuntu 14.04 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", "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 \(p. 91\)](#) 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 \(p. 102\)](#), 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

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 \(p. 91\)](#). 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. To complete this step, install Berkshelf on your local workstation by following the instructions at [Berkshelf](#).

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:

```
berks init .
```

(Don't forget the period at the end of the command.) 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 can access through 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 then creates a list of the dependencies for both your cookbook and the application cookbook. Berkshelf will use 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, substitute 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 \(p. 91\)](#) 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 \(p. 104\)](#), 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 AWS 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 \(p. 105\)](#).

Step 17: (Optional) Clean Up

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 AWS 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 AWS OpsWorks Stacks. If you want to keep these AWS resources available, you have now completed this walkthrough, and you can skip to [Next Steps \(p. 105\)](#).

To delete the S3 bucket

- See [Delete the Amazon S3 Bucket](#).

To delete the instance for the stack

1. In the AWS 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, so be patient, and 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**. AWS 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**. AWS 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 [Deleting Your Key Pair](#).

You have now completed this walkthrough. For more information, see [see Next Steps \(p. 105\)](#).

Next Steps

Now that you have completed this walkthrough, you can learn more about AWS OpsWorks Stacks support for Chef cookbooks by reviewing the following resources:

- [Cookbooks and Recipes \(p. 235\)](#) – Describes the versions of Chef and Ruby that AWS 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.
 - [Chef Tutorials](#) – Describes how to use Chef to manage an instance, manage a basic web app, develop and test infrastructure code, use Chef analytics, and more.
 - [Chef Fundamentals Series](#) – Provides six videos describing how to work with cookbooks, data bags, community cookbooks, 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.

AWS OpsWorks Stacks Best Practices

The strategies, techniques, and suggestions presented here will help you get the maximum benefit and optimal outcomes from AWS OpsWorks Stacks.

Topics

- [Best Practices: Root Device Storage for Instances \(p. 105\)](#)
- [Best Practices: Optimizing the Number of Application Servers \(p. 107\)](#)
- [Best Practices: Managing Permissions \(p. 108\)](#)
- [Best Practices: Managing and Deploying Apps and Cookbooks \(p. 110\)](#)
- [Packaging Cookbook Dependencies Locally \(p. 116\)](#)
- [Installing Linux Security Updates \(p. 117\)](#)
- [Using Security Groups \(p. 118\)](#)

Best Practices: Root Device Storage for Instances

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 \(p. 278\)](#) 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

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.

AWS OpsWorks Stacks provides three ways to manage the number of server instances.

- [24/7 instances \(p. 178\)](#) are started manually and run until they are manually stopped.
- [Time-based instances \(p. 181\)](#) are automatically started and stopped by AWS OpsWorks Stacks on a user-specified schedule.
- [Load-based instances \(p. 181\)](#) are automatically started and stopped by AWS OpsWorks Stacks when they cross a threshold for a user-specified load metric such as CPU or memory utilization.

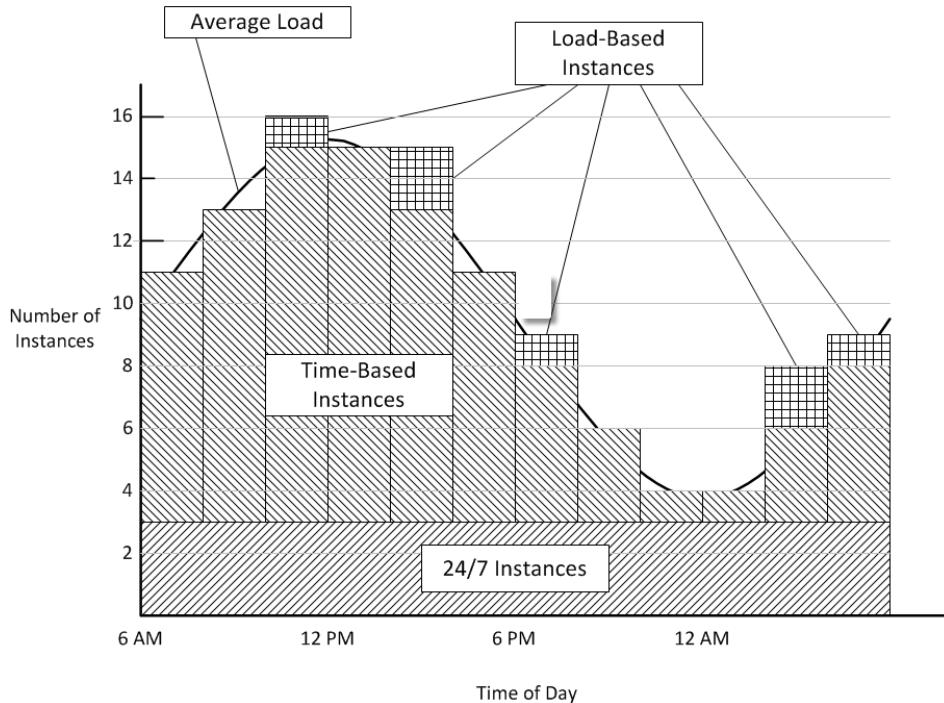
Note

After you have created and configured your stack's time and load-based instances, AWS 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.

AWS 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.

Tip

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

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 users](#) for your employees and attach policies that provide appropriate access. Each employee can then use their IAM user 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 AWS OpsWorks Stacks or IAM to manage user permissions. Note that the two options are not mutually exclusive; it is sometimes desirable to use both.

AWS 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 AWS 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 AWS OpsWorks Stacks permissions management, see [Granting Per-Stack Permissions \(p. 289\)](#). For more information about managing SSH access, see [Managing SSH Access \(p. 302\)](#).

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 AWS 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. AWS OpsWorks Stacks then evaluates both policies to determine the user's permissions.

Important

If an IAM 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.

```
{  
    "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 [Attaching an IAM Policy \(p. 291\)](#).

Tip

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

AWS 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 \(p. 110\)](#)
- [Deploying Code to Online Instances \(p. 111\)](#)

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, AWS 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 \(p. 221\)](#) (for apps) or an [Update Custom Cookbooks command \(p. 133\)](#) (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, AWS 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 \(p. 218\)](#) or [cookbook \(p. 249\)](#) source.

Deploying Code to Online Instances

AWS 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 \(p. 221\)](#) (for apps) or [Update Custom Cookbooks command \(p. 133\)](#) (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 \(p. 112\)](#)
- [Using Separate Stacks \(p. 112\)](#)
- [Managing a Backend Database \(p. 114\)](#)

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 \(p. 221\)](#), 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.

Tip

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 \(p. 113\)](#)
- [Using a Blue-Green Deployment Strategy \(p. 113\)](#)

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 AWS OpsWorks Stacks stack management functionality by using the stack's [Permissions page \(p. 282\)](#).

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 \(p. 132\)](#). 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 AWS 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 \(p. 282\)](#).

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 \(p. 113\)](#).

Tip

Instead of using the AWS 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 AWS OpsWorks Stacks stacks, in conjunction with [Amazon 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 \(p. 181\)](#) 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 Amazon 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 \(p. 147\)](#) 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 Amazon 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 \(p. 147\)](#) 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 \(p. 137\)](#).

Managing a Backend Database

If your application depends on a backend database, you will need to transition from the old application to the new. AWS OpsWorks Stacks supports the following database options.

Amazon RDS Layer

With an [Amazon Relational Database Service \(Amazon RDS\) layer \(p. 150\)](#), 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.

AWS OpsWorks Stacks installs a file with the connection data on your application servers in a format that easily can be used by your application. AWS 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 \(p. 224\)](#).

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 \(p. 225\)](#).

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

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.

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 AWS 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 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 AWS 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` 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 AWS OpsWorks Stacks stack to use the new cookbook source. To do this, follow steps 2 through 4 in [Packaging Dependencies Locally for One Cookbook \(p. 116\)](#).

Installing Linux 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, AWS OpsWorks Stacks automatically installs the latest updates during setup, after an instance finishes booting. AWS 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 \(p. 133\)](#), which installs the current set of security patches and other updates on the specified instances.

For both of these approaches, AWS 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.

For more information, see [CentOS Product Specifications](#).

- **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 \(p. 401\)](#) to install your preferred updates.

Because system updates don't map naturally to a [lifecycle event \(p. 253\)](#), include the recipe in your custom cookbooks but [execute it manually \(p. 255\)](#). 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 \(p. 212\)](#) and run the appropriate commands manually.

Using Security Groups

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, AWS 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 AWS OpsWorks Stacks layers
- Inbound TCP traffic on port 22 (SSH login)

Caution

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 \(p. 214\)](#).

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.

Tip

If you accidentally delete an AWS OpsWorks Stacks security group, the preferred way to recreate it is to have AWS OpsWorks Stacks perform the task for you. Just create a new stack in the same AWS region—and VPC, if present—and AWS 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, AWS 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 AWS 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) – AWS 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** – AWS 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, AWS 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.

Stacks

The stack is the top-level AWS 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

- [Create a New Stack \(p. 120\)](#)
- [Running a Stack in a VPC \(p. 126\)](#)
- [Update a Stack \(p. 132\)](#)
- [Clone a Stack \(p. 132\)](#)
- [Run AWS OpsWorks Stacks Stack Commands \(p. 133\)](#)
- [Using Custom JSON \(p. 135\)](#)
- [Shut Down a Stack \(p. 137\)](#)

Create a New Stack

To create a new stack, on the AWS 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 \(p. 121\)](#)
- [Basic Options \(p. 121\)](#)
- [Advanced Options \(p. 123\)](#)

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 (p. 35)
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 (p. 309) .	Getting Started: Linux (p. 48)
A Windows-based Chef 12.2 stack	Create a Windows-based stack.	Getting Started: Windows (p. 67)
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 (p. 313)

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 AWS OpsWorks Stacks console. The name does not need to be unique. AWS 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 its 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 AWS OpsWorks Stacks stack, see [Running a Stack in a VPC \(p. 126\)](#). 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 AWS OpsWorks Stacks launch an instance in a different Availability Zone or subnet by overriding this setting when you [create the instance \(p. 168\)](#).

For more information about how to run a stack in a VPC, see [Running a Stack in a VPC \(p. 126\)](#).

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. (Not available outside the US East (N. Virginia) Region regional endpoint.)
- 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 \(p. 172\)](#).

For more information on the available operating systems, see [AWS OpsWorks Stacks Operating Systems \(p. 159\)](#).

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, AWS 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 \(p. 212\)](#).

- 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 \(p. 214\)](#).

For more information on how to manage SSH keys, see [Managing SSH Access \(p. 302\)](#).

Note

You can override this setting by specifying a different key pair, or no key pair, when you [create an instance \(p. 168\)](#).

Chef version

This shows the Chef version that you have chosen.

For more information on Chef versions, see [Chef Versions \(p. 237\)](#).

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 AWS 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 \(p. 249\)](#).

Stack color

(Optional) The hue used to represent the stack on the AWS OpsWorks Stacks console. You can use different colors for different stacks to help distinguish, for example, among development, staging, and production stacks.

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 AWS OpsWorks Stacks uses to interact with AWS on your behalf. You should use a role that was created by AWS OpsWorks Stacks to ensure that it has all the required permissions. If you don't have an existing AWS OpsWorks Stacks role, select **New IAM Role** to have AWS OpsWorks Stacks create a new IAM role for you. For more information, see [Allowing AWS OpsWorks Stacks to Act on Your Behalf \(p. 298\)](#).

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.
- Otherwise, select **New IAM Instance Profile** to have AWS OpsWorks Stacks create a new instance profile for you.
- 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 \(p. 300\)](#).

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
- Asia Pacific (Mumbai) Region
- Asia Pacific (Singapore) Region
- Asia Pacific (Sydney) Region
- Asia Pacific (Tokyo) Region
- Asia Pacific (Seoul) Region
- EU (Frankfurt) Region
- EU (Ireland) Region
- South America (São Paulo) Region

Stacks that are created in one API endpoint are not available in another API endpoint. Because AWS OpsWorks Stacks users are also region-specific, if you want AWS 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 AWS 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 AWS 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.

AWS 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 \(p. 253\)](#). This agent is regularly updated. You have two options for specifying the agent version for your stack.

- **Auto-update** – AWS OpsWorks Stacks automatically installs each new agent version on the stack's instances as soon as the update is available.
- **Manual update** – AWS OpsWorks Stacks installs the specified agent version on the stack's instances.

AWS 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 \(p. 132\)](#) to specify a new agent version and AWS 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 \(p. 210\)](#). 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 AWS 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 \(p. 210\)](#) and specify a new version.

Tip

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 \(p. 376\)](#) 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 \(p. 348\)](#).

Security has one option, **Use OpsWorks security groups**, which allows you to specify whether to associate the AWS OpsWorks Stacks built-in security groups with the stack's layers.

AWS 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 \(p. 307\)](#).

Use OpsWorks security groups has the following settings:

- **Yes** - AWS 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** - AWS 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, AWS 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 AWS 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 AWS OpsWorks Stacks perform the task for you. Just create a new stack in the same AWS region—and VPC, if present—and AWS 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

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 AWS OpsWorks Stacks.

Important

If you use the VPC Endpoint feature, be aware 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
<ul style="list-style-type: none"> • opswsks-instance-agent-sa-east-1 • opswsks-instance-agent-ap-south-1 • opswsks-instance-agent-ap-northeast-1 • opswsks-instance-agent-ap-northeast-2 • opswsks-instance-agent-ap-southeast-1 • opswsks-instance-agent-ap-southeast-2 • opswsks-instance-agent-eu-central-1 • opswsks-instance-agent-eu-west-1 	<ul style="list-style-type: none"> • opswsks-instance-assets-us-east-1 • opswsks-instance-assets-us-east-2 • opswsks-instance-assets-ap-south-1 • opswsks-instance-assets-ap-northeast-1 • opswsks-instance-assets-ap-northeast-2 • opswsks-instance-assets-ap-southeast-1 • opswsks-instance-assets-ap-southeast-2 • opswsks-instance-assets-eu-central-1 • opswsks-instance-assets-eu-west-1 	<ul style="list-style-type: none"> • opswsks-us-east-1-log • opswsks-us-east-2-log • opswsks-ap-south-1-log • opswsks-ap-northeast-1-log • opswsks-ap-northeast-2-log • opswsks-ap-southeast-1-log • opswsks-ap-southeast-2-log • opswsks-eu-central-1-log • opswsks-eu-west-1-log 	<ul style="list-style-type: none"> • opswsks-us-east-1-dna • opswsks-us-east-2-dna • opswsks-ap-south-1-dna • opswsks-ap-northeast-1-dna • opswsks-ap-northeast-2-dna • opswsks-ap-southeast-1-dna • opswsks-ap-southeast-2-dna • opswsks-eu-central-1-dna • opswsks-eu-west-1-dna

Agent Buckets	Asset Buckets	Log Buckets	DNA Buckets
<ul style="list-style-type: none"> opsworks-instance-agent-eu-west-2 opsworks-instance-agent-us-east-1 opsworks-instance-agent-us-east-2 opsworks-instance-agent-us-west-1 opsworks-instance-agent-us-west-2 	<ul style="list-style-type: none"> opsworks-instance-assets-eu-west-1 opsworks-instance-assets-eu-west-2 opsworks-instance-assets-sa-east-1 opsworks-instance-assets-us-west-1 opsworks-instance-assets-us-west-2 	<ul style="list-style-type: none"> opsworks-eu-west-2-log opsworks-sa-east-1-log opsworks-us-west-1-log opsworks-us-west-2-log 	<ul style="list-style-type: none"> opsworks-eu-west-2-dna opsworks-sa-east-1-dna opsworks-us-west-1-dna opsworks-us-west-2-dna

For Chef 11.10 and earlier versions for Linux, the buckets are as follows. Note that 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
<ul style="list-style-type: none"> opsworks-instance-agent-us-east-1 opsworks-instance-agent-us-east-2 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-eu-central-1 opsworks-instance-agent-eu-west-1 opsworks-instance-agent-eu-west-2 opsworks-instance-agent-us-east-1 opsworks-instance-agent-us-west-1 opsworks-instance-agent-us-west-2 	<ul style="list-style-type: none"> opsworks-instance-assets-us-east-1 opsworks-instance-assets-us-east-2 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-eu-central-1 opsworks-instance-assets-eu-west-1 opsworks-instance-assets-eu-west-2 opsworks-instance-assets-sa-east-1 opsworks-instance-assets-us-west-1 opsworks-instance-assets-us-west-2 	<ul style="list-style-type: none"> prod_stage-log 	<ul style="list-style-type: none"> prod_stage-dna

For more information, see [VPC Endpoints](#).

Note

For AWS OpsWorks Stacks to connect to the VPC endpoints that you enable, you must also configure routing for your NAT or public IP, as the AWS OpsWorks Stacks agent still requires access to the public endpoint.

Topics

- [VPC Basics \(p. 128\)](#)
- [Create a VPC for an AWS OpsWorks Stacks Stack \(p. 130\)](#)

VPC Basics

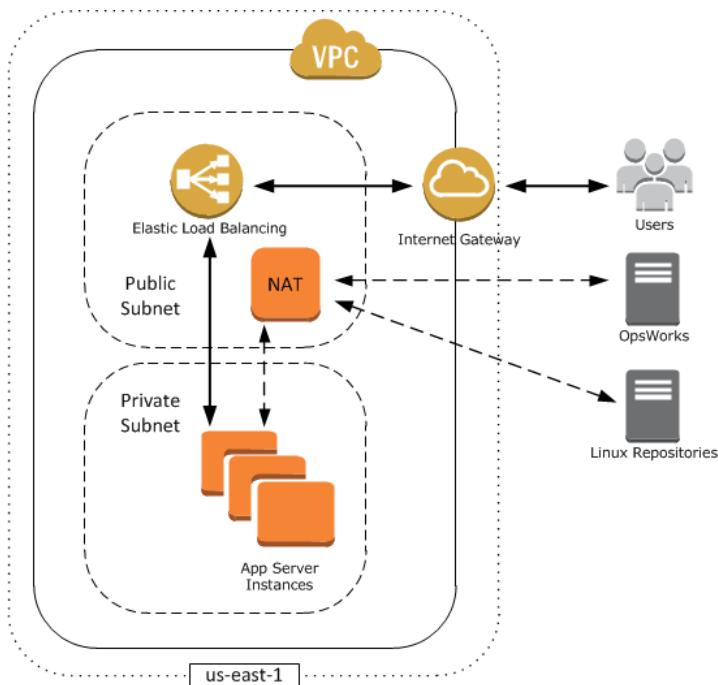
For a detailed discussion of VPCs, see [Amazon Virtual Private Cloud](#). Briefly, a VPC consists of one or more *subnets*, each of which contains one or more instances. Each subnet has an associated routing table that directs outbound traffic based on its destination IP address.

- Instances within a VPC can generally communicate with each other, regardless of their subnet.
- 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*.

AWS 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 AWS OpsWorks Stacks service endpoints listed in the "Region Support" section of [Getting Started with AWS OpsWorks Stacks \(p. 34\)](#).
- 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 AWS 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 AWS OpsWorks Stacks instances must be able to communicate with AWS 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.

Tip

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 AWS 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 \(p. 139\)](#).

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 AWS OpsWorks Stacks Stack

This section shows how to create a VPC for an AWS OpsWorks Stacks stack by using an example [AWS CloudFormation](#) template. You can download the template from <http://cloudformation-templates-us-east-1.s3.amazonaws.com/OpsWorksInVPC.template>. 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.

Tip

By default, AWS 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. AWS 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 \(p. 131\)](#) 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 **Specify an Amazon S3 template URL** and paste the template URL: `http://cloudformation-templates-us-east-1.s3.amazonaws.com/OpsWorksinVPC.template`. Choose **Continue**.

Select Template

Select the template that describes the stack that you want to create. A stack is a group of related resources that you manage as a single unit.

The screenshot shows the 'Select Template' step of the AWS CloudFormation 'Create Stack' wizard. It has two main sections: 'Design a template' (using CloudFormation Designer) and 'Choose a template' (using a JSON-formatted template file). In the 'Choose a template' section, the 'Specify an Amazon S3 template URL' option is selected, and the URL `http://cloudformation-templates-us-east-1.s3.amazonaws.com/OpsWorksinVPC.template` is entered into the input field. There is also a 'View/Edit template' link next to the input field.

You can also launch this stack by opening [AWS CloudFormation Sample Templates](#), locating the AWS 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 AWS 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 AWS 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 AWS OpsWorks Stacks stack. You can see them by choosing the **Outputs** tab at the bottom of the AWS CloudFormation console window.

Key	Value	Description
VPC	vpc-d1ed9ebf	VPC
PublicSubnets	subnet-66ec9f08	Public Subnet
PrivateSubnets	subnet-6dec9f03	Private Subnet
LoadBalancer	OpsWorks-ElasticL-K0FQ6TD5S14A	Load Balancer

Update a Stack

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 \(p. 120\)](#). 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 AWS 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 \(p. 140\)](#).

AWS OpsWorks Stacks then deletes the built-in security groups from every layer.

- If you change **Use OpsWorks security groups** from **No** to **Yes**, AWS OpsWorks Stacks adds the appropriate built-in security group to each layer but does not delete the existing security groups.

Clone a Stack

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 AWS 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.

OpsWorks Dashboard

Add stack

Register instances

Stack name	Region	Layers	Instances	Apps	Actions
[redacted]	us-east-1	1	1	0	
[blue]	us-west-2	2	1	0	
MyLinuxDemoStack	us-west-2	1	1	1	

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 \(p. 120\)](#). 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 East (N. Virginia) Region (us-east-1) to the Asia Pacific (Mumbai) Region (ap-south-1).

When you have finalized the settings, choose **Clone stack**. AWS 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. AWS 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 AWS OpsWorks Stacks Stack Commands

AWS 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

AWS OpsWorks Stacks also supports a set of *deployment commands*, which you use to manage app deployment. For more information, see [Deploying Apps \(p. 221\)](#).

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 \(p. 221\)](#) to run the Deploy recipes. For more information on custom cookbooks, see [Cookbooks and Recipes \(p. 235\)](#).

Execute Recipes

Executes a specified set of recipes on the instances. For more information, see [Manually Running Recipes \(p. 255\)](#).

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 \(p. 252\)](#).

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 \(p. 306\)](#).

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 [AWS OpsWorks Stacks Operating Systems \(p. 159\)](#).

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, AWS 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 AWS 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 \(p. 376\)](#).

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, AWS OpsWorks Stacks uses `execute_recipes` to update permissions on the instances.

Using Custom JSON

Several AWS OpsWorks Stacks actions allow you to specify custom JSON, which AWS 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 \(p. 253\)](#).

- 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"  
    ...  
}
```

AWS 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 AWS 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 AWS 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 AWS 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 \(p. 140\)](#) and [Deploying Apps \(p. 221\)](#).

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. AWS 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 80 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.

Shut Down a Stack

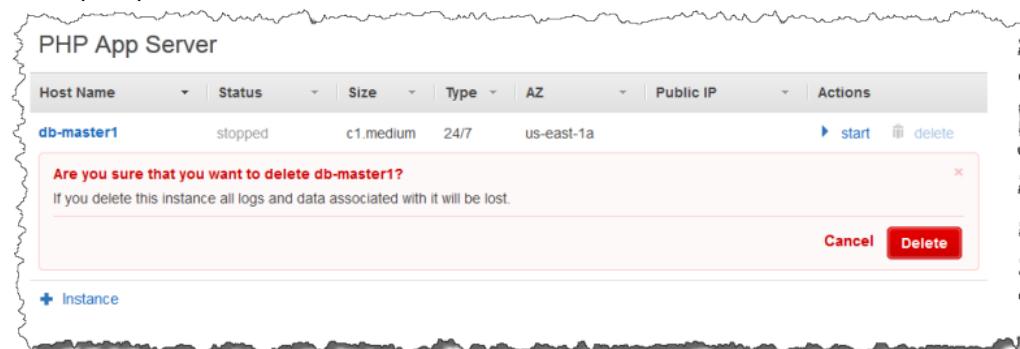
If you no longer need a stack, you can shut it down.

To shut down a stack

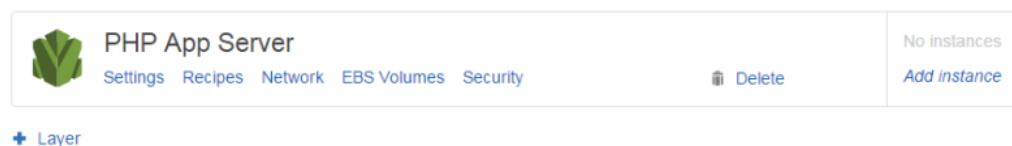
1. On the AWS OpsWorks Stacks dashboard, click the stack that you want to shut down.
2. In the navigation pane, click **Instances**.
3. On the **Instances** page, click **Stop all Instances**.



4. After the instances have stopped, for each instance in the layer, click **delete** in the **Actions** column. When prompted to confirm, click **Yes, Delete**.



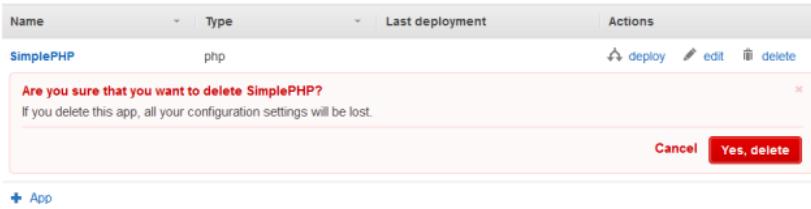
5. When all the instances are deleted, in the navigation pane, click **Layers**.
6. On the **Layers** page, for each layer in the stack, click **delete**. When a confirmation prompt appears, click **Yes, Delete**.



7. When all the layers are deleted, in the navigation pane, click **Apps**.
8. On the **Apps** page, for each app in the stack, click **delete** in the **Actions** column. When prompted to confirm, click **Yes, Delete**.

Apps

An app represents code stored in a repository that you want to install on application server instances. When you deploy the app, OpsWorks downloads the code from the repository to the specified server instances. [Learn more.](#)



9. When all the apps are deleted, in the navigation pane, click **Stack**.
10. On the stack page, click **Delete stack**. When prompted to confirm, click **Yes, Delete**.



Layers

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 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 \(p. 188\)](#).

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 OpsWorks layers. In that case, AWS 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 \(p. 139\)](#)

- [Elastic Load Balancing Layer \(p. 147\)](#)
- [Amazon RDS Service Layer \(p. 150\)](#)
- [ECS Cluster Layers \(p. 153\)](#)
- [Custom AWS OpsWorks Stacks Layers \(p. 157\)](#)
- [Per-layer Operating System Package Installations \(p. 158\)](#)

OpsWorks Layer Basics

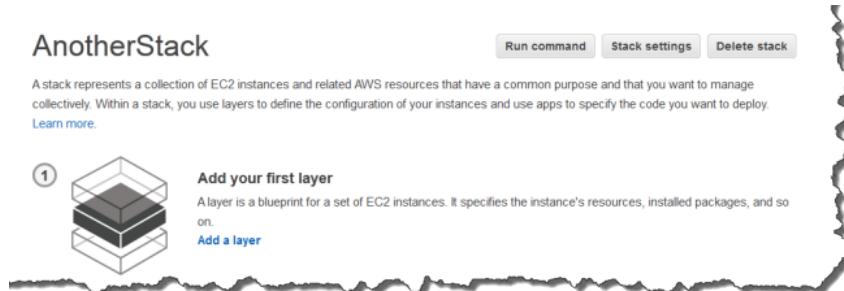
This section describes how to perform operations that are common to all OpsWorks layers.

Topics

- [Creating an OpsWorks Layer \(p. 139\)](#)
- [Editing an OpsWorks Layer's Configuration \(p. 140\)](#)
- [Using Auto Healing to Replace Failed Instances \(p. 145\)](#)
- [Deleting an OpsWorks Layer \(p. 146\)](#)

Creating an OpsWorks Layer

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.

Tip

The **Add Layer** page displays only the more commonly used configuration settings for each layer. You can specify additional settings by [editing the layer \(p. 140\)](#).

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, AWS 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 \(p. 345\)](#) in a variety of ways,

such as by assigning custom recipes to the appropriate lifecycle events. AWS OpsWorks Stacks runs custom recipes after the standard recipes for each event. For more information, see [Cookbooks and Recipes \(p. 235\)](#).

The following layer-specific sections describe how handle Steps 2 and 3 for the various AWS OpsWorks Stacks layers. For more information how to add instances, see [Adding an Instance to a Layer \(p. 168\)](#).

Editing an OpsWorks Layer's Configuration

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, click a layer name to open the details page, which shows the current configuration.

Tip
Clicking 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 \(p. 140\)](#)
- [Recipes \(p. 141\)](#)
- [Network \(p. 141\)](#)
- [EBS Volumes \(p. 142\)](#)
- [Security \(p. 144\)](#)

General Settings

All layers have the following settings:

Auto healing enabled

Whether [auto healing \(p. 145\)](#) 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 \(p. 135\)](#).

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 AWS 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 AWS 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 \(p. 135\)](#) and [Deploying Apps \(p. 221\)](#).

Instance shutdown timeout

Specifies how long (in seconds) AWS OpsWorks Stacks waits after triggering a [Shutdown lifecycle event \(p. 253\)](#) 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 \(p. 180\)](#).

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 \(p. 252\)](#).

Network

The **Network** tab includes the following settings.

Elastic Load Balancing

You can attach an Elastic Load Balancing load balancer to any layer. AWS 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 AWS OpsWorks Stacks supports it. For more information, see [Elastic Load Balancing Layer \(p. 147\)](#).

Automatically Assign IP Addresses

You can control whether AWS 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, AWS OpsWorks Stacks automatically assigns an address each time the instance is started.
- For Amazon EBS-backed instances, AWS OpsWorks Stacks automatically assigns an address when the instance is started for the first time.
- If an instance belongs to more than one layer, AWS 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. AWS 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:

Public IP addresses		
Elastic 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 AWS 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 \(p. 126\)](#).

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, AWS 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 \(p. 256\)](#).

- **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 GB.

For a RAID array, this setting specifies the total array size, not the size of each disk.

- **Volume Type** – (Optional) Specify whether to create a magnetic or PIOPS volume.

The default value is **Magnetic**.

- **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 AWS 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 AWS OpsWorks Stacks or Amazon EC2 and should not be used for Amazon EBS volumes.

The following mount points are reserved for use by AWS 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.

Tip

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 AWS OpsWorks Stacks, see [Using Custom AMIs \(p. 172\)](#).

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 \(p. 235\)](#) and [Customizing AWS OpsWorks Stacks \(p. 345\)](#).

```
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. \(p. 252\)](#)

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 \(p. 120\)](#) or [update \(p. 132\)](#) a stack. AWS OpsWorks Stacks provides a standard set of built-in security groups.

- The default option is to have AWS 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 \(p. 307\)](#).

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, AWS 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.) AWS 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 AWS 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, AWS OpsWorks Stacks does not add it to any new or restarted instances. AWS OpsWorks Stacks does not remove security groups from online instances.

Tip

If your stack is running in a VPC, including a [default 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 AWS 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, AWS 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 \(p. 132\)](#) and switch the **Use OpsWorks security groups** setting to **No**.

AWS 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 \(p. 300\)](#).

Using Auto Healing to Replace Failed Instances

Every instance has an AWS OpsWorks Stacks agent that communicates regularly with the service. AWS OpsWorks Stacks uses that communication to monitor instance health. If an agent does not communicate with the service for more than approximately five minutes, AWS 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.

Layer windowscompute

The screenshot shows the AWS OpsWorks Stacks interface for managing a layer named 'windowscompute'. At the top, there are tabs for General Settings, Recipes, Network, EBS Volumes, and Security. The General Settings tab is active. Below it, the 'Settings' section contains the following fields:

- Name: windowscompute
- Short name: compute
- Instance shutdown timeout: 120
- Auto healing enabled: Yes (with a blue button)

A red oval highlights the 'Auto healing enabled' field and its button.

Note

An instance can be a member of multiple layers. If any of those layers has auto healing disabled, AWS OpsWorks Stacks does not heal the instance if it fails.

If a layer has auto healing enabled—the default setting—AWS 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, AWS OpsWorks Stacks triggers a Configure [lifecycle event \(p. 253\)](#) on all of the stack's instances. The associated [stack configuration and deployment attributes \(p. 376\)](#) 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 \(p. 142\)](#) for a layer's instances, AWS 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 \(p. 256\)](#) page.

When AWS 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 AWS OpsWorks Stacks attaches it to the new instance.
- If the volume was not attached to the instance when the instance failed, AWS 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 \(p. 140\)](#) to disable it.

Important

If you have auto healing enabled, be sure to do the following:

- Use only the AWS 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, AWS 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

If you no longer need an AWS 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.

PHP App Server

Host Name	Status	Size	Type	AZ	Public IP	Actions
php-app1	online	c1.medium	24/7	us-east-1a	54.242.127.207	stop

Are you sure you want to stop php-app1?
All data not stored on EBS volumes will be lost.

Cancel **Stop**

+ Instance

- After each instance has stopped, click **delete** to remove it from the layer.

Host Name	Status	Size	Type	AZ	Public IP	Actions
db-master1	stopped	c1.medium	24/7	us-east-1a		start delete

Are you sure that you want to delete db-master1?
If you delete this instance all logs and data associated with it will be lost.

Cancel **Delete**

+ Instance

- In the navigation pane, click **Layers**.
- On the **Layers** page, choose **Delete**.

PHP App Server

Settings **Recipes** **Network** **EBS Volumes** **Security**

Delete

No instances

Add instance

+ Layer

Elastic Load Balancing Layer

Elastic Load Balancing works somewhat differently than an AWS 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, AWS OpsWorks Stacks does the following:

- Deregisters any currently registered instances.
- Automatically registers the layer's instance's when they come online and deregisters instances when they leave the online state, including load-based and time-based instances.
- Automatically activates and deactivates the instances' Availability Zones.

If you have enabled the load balancer's [connection draining](#) feature, you can specify whether AWS OpsWorks Stacks supports it. If you enable connection draining support (the default setting), after an instance is shut down, AWS 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 \(p. 253\)](#) until the load balancer has completed connection draining.

If you don't enable connection draining support, AWS 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.
- AWS OpsWorks Stacks does not support Application Load Balancer. You can only use Classic Load Balancer with AWS 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 AWS 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 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 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 \(p. 139\)](#) that you want to have balanced or [edit an existing layer's Network settings \(p. 140\)](#).

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 AWS OpsWorks Stacks to support connection draining.

After you attach a load balancer to a layer, AWS OpsWorks Stacks triggers a [Configure lifecycle event \(p. 253\)](#) on the stack's instances to notify them of the change. AWS OpsWorks Stacks also triggers a Configure event when you detach a load balancer.

Note

After an instance has booted, AWS OpsWorks Stacks runs the [Setup and Deploy recipes \(p. 252\)](#), which install packages and deploy applications. After those recipes have finished, the instance is in the online state and AWS OpsWorks Stacks registers the instance with Elastic Load Balancing. AWS 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 \(p. 252\)](#).

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. AWS 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.

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 \(p. 157\)](#) 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 \(p. 168\)](#) 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 Availability Zones. A green check indicates a healthy instance. You can click on the name to connect to a server, through the load balancer.

ELB My-Stack-PHP

[Disconnect ELB](#)

Elastic Load Balancing associates your load balancer with your EC2 instances using IP addresses. [Learn more](#).

Settings

DNS Name	My-Stack-PHP-1556928710.us-west-2.elb.amazonaws.com
Layer	PHP App Server
Region	us-west-2



Amazon RDS Service Layer

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](#) (p. 380).
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” \(p. 224\)](#)

Caution

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 \(p. 151\)](#)
- [Registering an Amazon RDS Instance with a Stack \(p. 151\)](#)
- [Associating Amazon RDS Service Layers with Apps \(p. 153\)](#)
- [Removing an Amazon RDS Service Layer from a Stack \(p. 153\)](#)

Specifying Security Groups

To use an Amazon RDS instance with AWS 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 AWS OpsWorks Stacks instances that need to access the database. AWS 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 AWS OpsWorks Stacks instances is to assign the appropriate AWS 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 AWS 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 AWS 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 \(p. 152\)](#).
3. Click the RDS tab to list the available Amazon RDS instances.

Tip

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 \(p. 140\)](#) to provide valid user and password values and then redeploy the app.

Add Layer

The screenshot shows the 'Add Layer' interface. At the top, there are two tabs: 'OpsWorks' (disabled) and 'RDS' (selected). Below the tabs is a table with columns: Instance Identifier, Engine, Storage (GB), Type, Status, Multi-AZ, and Availability Zone. One row is visible for 'opsinstance2', which is a mysql instance with 5 GB storage, type t1.micro, available status, no Multi-AZ, and located in us-east-1a. A large box below the table is titled 'Connection Details for opsinstance2'. It contains fields for 'User' (opsuser) and 'Password' (redacted). A 'SHOW' button is next to the password field. A note at the bottom of this box says: 'Please verify that OpsWorks can connect to your RDS Instance by setting [Security Groups](#) on that instance. [Learn more](#)'. At the bottom right of the main area are 'Cancel' and 'Register with Stack' buttons.

When you add an Amazon RDS service layer to a stack, AWS OpsWorks Stacks assigns it an ID and adds the associated Amazon RDS configuration to the [stack configuration and deployment attribute's \(p. 376\)](#) `[:opsworks][:stack] (p. 517)` attribute.

Note

If you change a registered Amazon RDS instance's password, you must manually update the password in AWS 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 \(p. 152\)](#)

Updating the Stack's Service Role

Every stack has an [IAM service role \(p. 298\)](#) that specifies what actions AWS 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 AWS 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.

Add Layer

The screenshot shows the 'Add Layer' interface with the RDS tab selected. A message box in the center states: 'To use RDS instances, your OpsWorks IAM role needs to have an RDS instances access policy.' At the bottom right of this message box is an 'Update' button.

Click **Update** to have AWS 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 \(p. 217\)](#), or later by [editing the app's configuration \(p. 224\)](#).
- 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, AWS 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” \(p. 224\)](#).

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

The [Amazon EC2 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:

- ECS Cluster layers are not supported in the following regions, because the Amazon EC2 Container Service (Amazon ECS) feature is not available in those regions.
 - Asia Pacific (Seoul) Region (ap-northeast-2)
 - Asia Pacific (Mumbai) Region (ap-south-1)
 - South America (São Paulo) Region (sa-east-1)
- 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 Amazon Linux 2015.03 or a newer release of Amazon Linux, or Ubuntu 14.04 LTS or a newer release of Ubuntu. For more information, see [Linux Operating Systems \(p. 160\)](#).
- The instances' [AWS OpsWorks Stacks agent version \(p. 123\)](#) must be 3425-20150727112318 or later.

Topics

- [Adding an ECS Cluster Layer to a Stack \(p. 154\)](#)
- [Managing the ECS Cluster \(p. 156\)](#)
- [Deleting an ECS Cluster Layer from a Stack \(p. 156\)](#)

Adding an ECS Cluster Layer to a Stack

AWS 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 EC2 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 AWS 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 AWS OpsWorks Stacks AWS Identity and Access Management (IAM) service role, which is usually named `aws-opsworks-service-role`, to allow AWS OpsWorks Stacks to interact with Amazon ECS on your behalf. For more information on the service role, see [Allowing AWS OpsWorks Stacks to Act on Your Behalf \(p. 298\)](#).

The first time you create an ECS Cluster layer, the console provides an **Update** button that you can choose to direct AWS OpsWorks Stacks to update the role for you. AWS 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": [
```

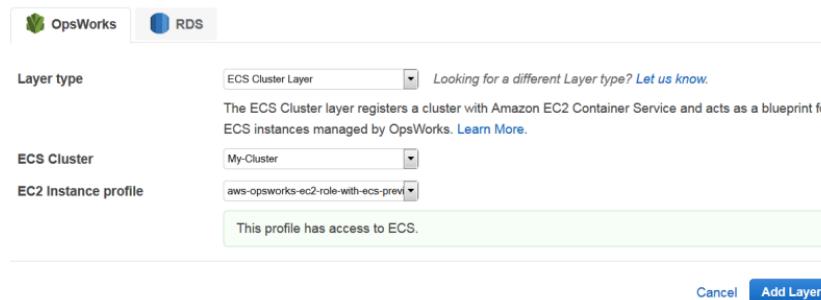
```

    "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 AWS OpsWorks Stacks console combines these steps; layer creation automatically registers the specified cluster. If you use the AWS 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.

Add Layer



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 AWS 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 \(p. 300\)](#).

You can specify other settings by [editing the layer's configuration \(p. 140\)](#), including:

- [Attaching an Elastic Load Balancing load balancer \(p. 141\)](#) 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 \(p. 141\)](#) 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 \(p. 126\)](#).

Managing the ECS Cluster

After you create an ECS Cluster layer, you can use AWS 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 \(p. 168\)](#) to the layer and [delete them \(p. 211\)](#) when they are no longer needed.
- Add or delete instances on a schedule by adding [time-based instances \(p. 182\)](#) to the layer.
- Add or delete instances based on AWS OpsWorks Stacks host metrics or CloudWatch alarms by adding [load-based instances \(p. 184\)](#) 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 2015.03 or later or Ubuntu 14.04 LTS—when you create the container instances.

For more information, see [Optimizing the Number of Servers \(p. 107\)](#). AWS OpsWorks Stacks assigns the AWS-OpsWorks-ECS-Cluster security group to each instance. After each new instance finishes booting, AWS 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 \(p. 189\)](#) and [assign them to the ECS Cluster layer \(p. 204\)](#). Note that the instances must be running a supported operating system, Amazon Linux 2015.03 or later or Ubuntu 14.04 LTS.

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 \(p. 157\)](#).

Execute operating system and package updates

After a new instance finishes booting, AWS OpsWorks Stacks installs the latest updates. You can then use AWS OpsWorks Stacks to keep the container instances up to date. For more information, see [Managing Security Updates \(p. 306\)](#).

Manage user permissions

AWS 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 \(p. 282\)](#) and [Managing SSH Access \(p. 302\)](#).

Monitor performance metrics

AWS OpsWorks Stacks provides a variety of ways to monitor performance metrics for the stack, layer, or individual instances. For more information, see [Monitoring \(p. 268\)](#).

You handle other management tasks, such as creating tasks or services, through Amazon ECS. For more information, see the [Amazon EC2 Container Service Developer Guide](#).

Tip

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 AWS OpsWorks Stacks console combines these steps; layer deletion automatically deregisters the specified cluster. If you use the AWS OpsWorks Stacks API, CLI, or SDK, you must use separate operations to deregister the cluster and delete the associated layer.

Note

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 \(p. 180\)](#), and then [delete them \(p. 211\)](#). When you stop a container instance, AWS 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 \(p. 204\)](#) and then [deregister them \(p. 204\)](#), which returns the instances to ECS control.

3. [Delete the layer \(p. 146\)](#). AWS OpsWorks Stacks deregisters the associated cluster, but does not delete it. The cluster remains in Amazon ECS.

Custom AWS OpsWorks Stacks Layers

A custom layer has only a minimal set of recipes. You then add appropriate functionality to the layer by implementing [custom recipes \(p. 235\)](#) and assigning them to the layer's [lifecycle events \(p. 253\)](#).

The custom layer has the following configuration settings.

Name

(Required) The layer's name, which is used for display purposes.

Short name

(Required) The layer's short name, which is used internally and by recipes. It can have a maximum of 200 characters, which are limited to the lowercase alphanumeric characters and the following punctuation marks: -, _, and ..

Security groups

This setting appears if you chose to not automatically associate a built-in AWS 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 \(p. 120\)](#).

Tip

AWS 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 \(p. 248\)](#).

The basic procedure for creating a custom layer has the following steps:

1. Implement a [cookbook \(p. 235\)](#) 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 \(p. 235\)](#).

2. Create a custom layer.

3. Assign your recipes to the appropriate [lifecycle events \(p. 253\)](#).

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

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

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 AWS 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 AWS OpsWorks Stacks to manage these instances in much the same way as instances created with AWS OpsWorks Stacks.

This section describes how to use AWS OpsWorks Stacks to create and manage instances.

Topics

- [Using AWS OpsWorks Stacks Instances \(p. 159\)](#)
- [Using Computing Resources Created Outside of AWS OpsWorks Stacks \(p. 188\)](#)
- [Editing the Instance Configuration \(p. 210\)](#)
- [Deleting AWS OpsWorks Stacks Instances \(p. 211\)](#)
- [Using SSH to Log In to a Linux Instance \(p. 212\)](#)
- [Using RDP to Log In to a Windows Instance \(p. 214\)](#)

Using AWS OpsWorks Stacks Instances

You can use AWS OpsWorks Stacks to create instances and add them to the stack.

Topics

- [AWS OpsWorks Stacks Operating Systems \(p. 159\)](#)
- [Adding an Instance to a Layer \(p. 168\)](#)
- [Using Custom AMIs \(p. 172\)](#)
- [Manually Starting, Stopping, and Rebooting 24/7 Instances \(p. 178\)](#)
- [Managing Load with Time-based and Load-based Instances \(p. 181\)](#)

AWS OpsWorks Stacks Operating Systems

AWS 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 \(p. 172\)](#) (Amazon Machine Images), but they must be based on one of the AWS OpsWorks Stacks-supported AMIs that are described in topics in this section.
 - [Linux Operating Systems \(p. 160\)](#)
 - [Microsoft Windows Server \(p. 166\)](#)
- You can [start and stop instances manually \(p. 178\)](#) or have AWS OpsWorks Stacks [automatically scale \(p. 181\)](#) 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 AWS OpsWorks Stacks to create Amazon EC2 instances, you can also [register instances with a Linux stack \(p. 181\)](#) that were created outside of AWS 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.

Topics

- [Linux Operating Systems \(p. 160\)](#)
- [Microsoft Windows Server \(p. 166\)](#)

Linux Operating Systems

AWS OpsWorks Stacks supports the 64-bit versions of the following Linux operating systems.

- [Amazon Linux](#) (see the [AWS OpsWorks Stacks console](#) for the currently supported versions)
- [Ubuntu 12.04 LTS](#)
- [Ubuntu 14.04 LTS](#)
- [Ubuntu 16.04 LTS](#)
- [CentOS 7](#)
- [Red Hat Enterprise Linux 7](#)

You can also use [custom AMIs \(p. 172\)](#) 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, AWS 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, AWS 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 \(p. 133\)](#). For more information on how to manage security updates, see [Managing Security Updates \(p. 306\)](#).

For more control over how AWS 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 \(p. 172\)](#).

Hosts File

Each online instance has a `/etc/hosts` file that maps IP addresses to host names. AWS 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
```

AWS OpsWorks Stacks Agent Proxy Support

The AWS 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="https://myproxy.example.com:8080/"
no_proxy="169.254.169.254"
```

To enable proxy support, we recommend [creating a custom AMI \(p. 172\)](#) 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. AWS OpsWorks Stacks needs the proxy server data early in the setup process, before any custom recipes have executed.

Topics

- [Amazon Linux \(p. 161\)](#)
- [Ubuntu LTS \(p. 163\)](#)
- [CentOS \(p. 164\)](#)
- [Red Hat Enterprise Linux \(p. 165\)](#)

Amazon Linux

AWS OpsWorks Stacks supports the 64-bit version of Amazon Linux. In addition to regular updates and patches, Amazon Linux releases a new version approximately every six months, which can involve significant changes. For example, the 2012.09 to 2013.03 update upgraded the kernel from 3.2 to 3.4. 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 \(p. 133\)](#).

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 bullet item.
- For offline instance store-backed instances, including time-based and load-based instances, [edit the instance's Operating system setting \(p. 210\)](#) to specify the new version.

AWS 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
2017.03	0.12.15
2016.09	0.12.15
2016.03	0.12.15 0.12.13 0.12.12 0.12.10
2015.09	0.12.15 0.12.13 0.12.12 0.12.10 0.12.7 0.10.40
2015.03	0.12.15 0.12.13 0.12.12 0.12.10 0.12.7 0.10.40

Amazon Linux: Supported Chef Versions

Chef Version	Supported Amazon Linux Versions
12	Amazon Linux 2017.03 Amazon Linux 2016.09 Amazon Linux 2016.03 Amazon Linux 2015.09 Amazon Linux 2015.03

Chef Version	Supported Amazon Linux Versions
11.10	Amazon Linux 2017.03 Amazon Linux 2016.09 Amazon Linux 2016.03 Amazon Linux 2015.09 Amazon Linux 2015.03 Amazon Linux 2014.09 (deprecated) Amazon Linux 2014.03 (deprecated)
11.4 (deprecated)	Amazon Linux 2016.09 Amazon Linux 2016.03 Amazon Linux 2015.09 Amazon Linux 2015.03 Amazon Linux 2014.09 (deprecated) Amazon Linux 2014.03 (deprecated)

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](#).

AWS OpsWorks Stacks supports the 64-bit versions of Ubuntu 12.04 LTS and Ubuntu 14.04 LTS.

- Ubuntu 12.04 is supported for all stacks.
- Ubuntu 14.04 is supported only for Chef 11.10 and higher stacks.
- You cannot update an existing Ubuntu 12.04 instance to Ubuntu 14.04, or from either Ubuntu 12.04 or 14.04 to Ubuntu 16.04.

You must [create a new Ubuntu 14.04 or Ubuntu 16.04 instance \(p. 168\)](#) and [delete the older instance \(p. 211\)](#).

- Ubuntu 16.04 LTS is supported only for Chef 12 and higher stacks.

Ubuntu: Supported Node.js Versions

Ubuntu Version	Node.js Versions
16.04 LTS	(Not applicable to operating systems that are available for Chef 12 and higher stacks only)
14.04 LTS	0.10.27 0.10.29 0.10.40 0.12.10 0.12.12 0.12.13 0.12.15
12.04 LTS	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

Ubuntu: Supported Chef Versions

Chef Version	Supported Ubuntu Versions
12	Ubuntu 16.04 LTS Ubuntu 14.04 LTS Ubuntu 12.04 LTS
11.10	Ubuntu 14.04 LTS Ubuntu 12.04 LTS
11.4 (deprecated)	Ubuntu 12.04 LTS

All versions of Node.js that are older than 0.10.40 are deprecated. 0.12.7 and 0.12.9 are also deprecated.

CentOS

AWS 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. For more information, see [Questions about CentOS 7](#).

When you start a new instance in a CentOS stack, AWS OpsWorks Stacks automatically installs the most current CentOS version. Because AWS 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 \(p. 133\)](#), 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, AWS 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

AWS OpsWorks Stacks supports Apache 2.4 for CentOS instances.

Red Hat Enterprise Linux

AWS 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, AWS OpsWorks Stacks automatically installs the current RHEL 7 version. Because AWS 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 \(p. 133\)](#), 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, AWS 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

AWS OpsWorks Stacks supports Apache 2.4 for RHEL 7 instances.

Microsoft Windows Server

The following notes describe AWS OpsWorks Stacks support for Windows instances. Windows instances are available only for Chef 12.2 stacks.

Currently, the AWS OpsWorks Stacks agent cannot be installed on—and AWS OpsWorks Stacks cannot manage—Windows-based instances that use a system UI language other than **English - United States** (en-US).

Versions

AWS OpsWorks Stacks supports the following 64-bit versions of [Microsoft Windows Server 2012 R2](#):

- Microsoft Windows Server 2012 R2 Standard
- Microsoft Windows Server 2012 R2 with SQL Server Express
- Microsoft Windows Server 2012 R2 with SQL Server Standard
- Microsoft Windows Server 2012 R2 with SQL Server Web

Amazon Elastic Compute Cloud (Amazon EC2) refers to Microsoft Windows Server 2012 R2 Standard as Microsoft Windows Server 2012 R2 Base, which is what you will see reported as the operating system on Windows instances. You can also use a custom AMI based on these operating systems. For

pricing information, go to the [Amazon EC2 Pricing](#) page and visit the Windows tabs under **On-Demand Instances**.

Creating Instances

You create Windows instances with the AWS 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 \(p. 178\)](#) instances, which you start and stop manually. They can also use [time-based automatic scaling \(p. 182\)](#), which automatically starts and stops instances based on a user-specified schedule, or [load-based automatic scaling \(p. 184\)](#), which allows a stack to handle variable loads by starting additional instances when traffic is high and stopping instances when traffic is low.

You cannot [register Windows instances \(p. 188\)](#) that were created outside of AWS 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, AWS 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 \(p. 157\)](#) with custom recipes.

Chef

Windows instances use Chef 12.2 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

AWS OpsWorks Stacks provides authorized AWS Identity and Access Management (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 \(p. 214\)](#). For more information, see [Logging In with RDP \(p. 214\)](#).

AWS SDK

AWS 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](#). If you want 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 12.2 client that AWS OpsWorks Stacks installs on Windows instances comes with Ruby 2.0.0. However, AWS OpsWorks Stacks does not add the executable's directory to the PATH environment variable. If you want to have your applications use this Ruby version, you can typically find it at `C:\opscode\chef\embedded\bin\`.

AWS OpsWorks Stacks Agent CLI

The AWS OpsWorks Stacks agent on Windows instances does not expose a [command-line interface \(p. 651\)](#).

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 (AWS OpsWorks Stacks agent) applications:

```
<system.net>
  <defaultProxy>
    <proxy autoDetect="false" bypassOnLocal="true"
    proxyAddress="http://10.100.1.91:3128" usesSystemDefault="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 "https://10.100.1.91:3128"
```

Tip

For more control over how AWS OpsWorks Stacks updates your instances, create a custom AMI based on Microsoft Windows Server 2012 R2 Base. For example, with custom AMIs you can specify which software is installed on an instance, such as Internet Information Server. For more information, see [Using Custom AMIs \(p. 172\)](#).

Adding an Instance to a Layer

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 \(p. 181\)](#) 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, AWS OpsWorks Stacks runs the recipes for each of the instance's layers when a lifecycle event occurs, or when you run a [stack \(p. 133\)](#) or [deployment \(p. 221\)](#) command.

You can also make an instance a member of multiple layers by editing its configuration. For more information, see [Editing the Instance Configuration \(p. 210\)](#).

Note

[Spot instances](#) are not supported.

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:

- 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 \(p. 120\)](#).

Hostname

Identifies the instance on the network. By default, AWS 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. AWS OpsWorks Stacks specifies a default size for each instance, which you can override with your preferred instance type. AWS OpsWorks Stacks supports all instance types except Cluster Compute, Cluster GPU, and High Memory Cluster. Be aware that micro instances such as t1.micro might not have sufficient resources to support some layers. For more information, see [Instance Families and Types](#).

Note

If you are using [load-balanced instances \(p. 184\)](#), note that [Configure lifecycle events \(p. 253\)](#) 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 \(p. 184\)](#).

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.

Tip

By default, AWS 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. AWS 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.
- AWS OpsWorks Stacks starts and stops **time-based** instances based on a specified schedule.
- (Linux only) AWS 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 AWS OpsWorks Stacks starts and stops them based on the configuration. For more information, see [Managing Load with Time-based and Load-based Instances \(p. 181\)](#).

SSH key

An Amazon EC2 key pair. AWS 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 \(p. 212\)](#).
- For Windows instances, you can use the corresponding private key to [retrieve the instance's Administrator password \(p. 216\)](#). 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.

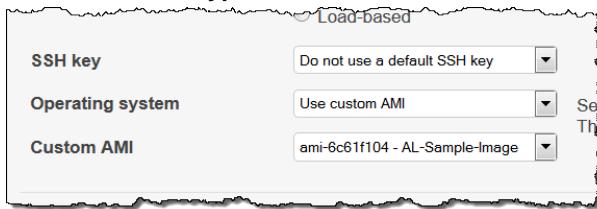
- If the default value is set to **Do not use a default SSH key**, you can specify one of your account's Amazon EC2 keys.
- If the default value is set to an Amazon EC2 key, you can specify a different key or no key.

Operating system

Operating system specifies which operating system the instance is running. AWS 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 \(p. 172\)](#).

OpsWorks Agent version

OpsWorks Agent version specifies the version of the AWS OpsWorks Stacks agent that you want to run on the instance. If you want AWS 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 AWS 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](#).

Tip

After the initial boot, Amazon EBS-backed instances boot faster than instance store-backed instances because AWS OpsWorks Stacks does not have to reinstall the instance's software from scratch. For more information, see [Root Device Storage \(p. 105\)](#).

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.

Tip

You cannot override the [stack's default agent version \(p. 124\)](#) setting when you create an instance. To specify a custom agent version setting, you must create the instance and then [edit its configuration \(p. 210\)](#).

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 AWS 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 AWS OpsWorks Stacks service, Amazon S3, and package repositories. If, for example, you remove a default gateway, the instances will lose their connection to the AWS OpsWorks Stacks service, which will then treat the instances as failed and [auto heal \(p. 145\)](#) them. However, AWS 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 AWS 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 AWS OpsWorks Stacks \(p. 188\)](#).

Using Custom AMIs

AWS 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 \(p. 181\)](#) 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 on recipes, see [Cookbooks and Recipes \(p. 235\)](#).

Topics

- [How Custom AMIs work with AWS OpsWorks Stacks \(p. 173\)](#)
- [Creating a Custom AMI for AWS OpsWorks Stacks \(p. 174\)](#)

How Custom AMIs work with AWS 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. AWS 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 \(p. 168\)](#).

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 \(p. 168\)](#) and [Create a New Stack \(p. 120\)](#).

This topic discusses some general issues that you should consider before creating or using a custom AMI.

Topics

- [Startup Behavior \(p. 173\)](#)
- [Choosing a Layer \(p. 173\)](#)
- [Handling Applications \(p. 174\)](#)

Startup Behavior

When you start the instance, AWS OpsWorks Stacks uses the specified custom AMI to launch a new Amazon EC2 instance. AWS OpsWorks Stacks then uses [cloud-init](#) to install the AWS 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 AWS OpsWorks Stacks agent usually does not conflict with installed packages. However, the instance must be a member of at least one layer. AWS 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.

Tip

For Linux AMIs, one way to reduce the possibility of conflicts is to use AWS 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 AWS OpsWorks Stacks Instance \(p. 175\)](#).

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 AWS OpsWorks Stacks deploy the application.

One option is to include the application in your AMI and also [create an app \(p. 217\)](#) 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 AWS OpsWorks Stacks

To use a custom AMI with AWS OpsWorks Stacks, you must first create an AMI from a customized instance. You can choose from two basic approaches:

- Use the Amazon EC2 console or API to create and customize an instance, based on a 64-bit version of one of the [AWS OpsWorks Stacks-supported AMIs \(p. 159\)](#).
- For Linux AMIs, use OpsWorks to create an Amazon EC2 instance, based on the configuration of its associated layers.

Tip

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 \(p. 168\)](#).

Note

By default, AWS 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 \(p. 175\)](#)
- [Create a Custom Linux AMI from an AWS OpsWorks Stacks Instance \(p. 175\)](#)
- [Create a Custom Windows AMI \(p. 176\)](#)

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 [AWS OpsWorks Stacks-supported AMIs \(p. 159\)](#).
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 AWS OpsWorks Stacks Instance

If you want to use a customized AWS OpsWorks Stacks Linux instance to create an AMI, you should 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 will include that identity and all instances based on the AMI will 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 AWS OpsWorks Stacks instance

1. [Create a Linux stack \(p. 120\)](#) and [add one or more layers \(p. 139\)](#) 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 AWS OpsWorks Stacks \(p. 345\)](#).
2. [Edit the layers \(p. 140\)](#) and disable AutoHealing.
3. [Add an instance with your preferred Linux distribution \(p. 168\)](#) to the layer or layers and [start it \(p. 178\)](#). 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 \(p. 212\)](#), and run the following commands, in order, depending upon your instance operating system.

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

1. `sudo /etc/init.d/monit stop`
2. `sudo /etc/init.d/opsworks-agent stop`
3. `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`
4. `sudo rpm -e opsworks-agent-ruby`
 5. `sudo rpm -e chef`

For an Ubuntu 16.04 LTS instance in a Chef 12 stack

1. `sudo systemctl stop opsworks-agent`

2. `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`
3. `sudo apt-get -y remove chef`
4. `sudo dpkg -r opsworks-agent-ruby`

For other supported Ubuntu versions in a Chef 12 stack

1. `sudo /etc/init.d/monit stop`
2. `sudo /etc/init.d/opsworks-agent stop`
3. `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`
4. `sudo apt-get -y remove chef`
5. `sudo dpkg -r opsworks-agent-ruby`

For a Red Hat Enterprise Linux 7 instance in a Chef 12 stack

1. `sudo systemctl stop opsworks-agent`
2. `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`
3. `sudo rpm -e opsworks-agent-ruby`
4. `sudo rpm -e chef`
5. This step depends on the instance type:
 - For an Amazon EBS-backed instance, use the AWS OpsWorks Stacks console to [stop the instance \(p. 178\)](#) 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 AWS 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.

6. Clean up your stack by returning to the AWS OpsWorks Stacks console and [deleting the instance \(p. 211\)](#) from the stack.

Create a Custom Windows AMI

The following procedures create custom AMIs for Windows Server 2012 R2. You can choose other Windows Server operating systems in the Amazon EC2 management console; the oldest available release of Windows Server is Windows Server 2003 R2.

Important

Currently, the AWS OpsWorks Stacks agent cannot be installed on—and AWS 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 \(p. 177\)](#)
- [Creating a Custom Windows AMI Without Sysprep \(p. 177\)](#)

- [Adding a New Instance by Using a Custom Windows AMI \(p. 178\)](#)

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, AWS OpsWorks Stacks provisioning, and the first AWS 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 2012 R2 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 AWS 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 AWS 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 2012 R2 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, AWS 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 AWS 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

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. AWS 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 \(p. 181\)](#).

Topics

- [Starting or Restarting an Instance \(p. 178\)](#)
- [Stopping an Instance \(p. 180\)](#)
- [Rebooting an Instance \(p. 181\)](#)

Starting or Restarting an Instance

To start a new instance, on the **Instances** page, click **start** in the instance's **Actions** column.



Hostname	Status	Size	Type	AZ	Public IP	Actions
lb1	online	c1.medium	24/7	us-east-1a	123.45.67.89 (EIP)	<input type="button" value="stop"/> <input type="button" value="ssh"/>
lb2	stopped	c1.medium	24/7	us-east-1a		<input type="button" value="start"/> <input type="button" value="delete"/>
+ Instance						

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, AWS 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** - AWS OpsWorks Stacks has called the Amazon EC2 service to create the Amazon EC2 instance.
2. **pending** - AWS OpsWorks Stacks is waiting for the Amazon EC2 instance to start.
3. **booting** - The Amazon EC2 instance is booting.
4. **running_setup** - AWS 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 \(p. 252\)](#). If you have added [added custom cookbooks \(p. 249\)](#), to the stack, AWS OpsWorks Stacks installs the current version from your repository prior to 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, AWS OpsWorks Stacks adds the instance to it.
- AWS 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.
- AWS 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 \(p. 633\)](#).

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 – AWS OpsWorks Stacks launches a new Amazon EC2 instance with the same configuration.
- Amazon EBS-backed instances – AWS OpsWorks Stacks restarts the Amazon EC2 instance, which reattaches the root volume.

After the instance finishes booting, AWS OpsWorks Stacks installs operating system updates and runs the Setup and Deploy recipes, just as with the initial start. AWS 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.

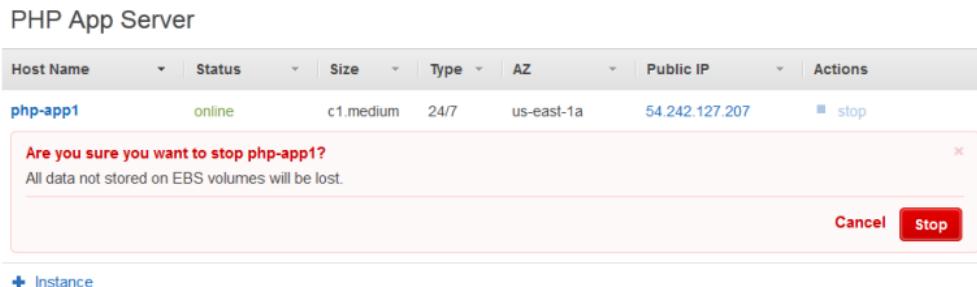
Amazon EBS-backed instances continue to use the custom cookbooks that were stored on the root volume. If your custom cookbooks have changed since you stopped the instance, you must manually update them after the instance is online. For more information, see [Updating Custom Cookbooks \(p. 251\)](#).

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 AWS 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, AWS OpsWorks Stacks performs several tasks:

1. If the instance's layer has an attached Elastic Load Balancing load balancer, AWS OpsWorks Stacks deregisters the instance.
If the layer supports the load balancer's connection draining feature, AWS OpsWorks Stacks delays triggering the Shutdown event until connection draining is complete. For more information, see [Elastic Load Balancing Layer \(p. 147\)](#).
2. AWS OpsWorks Stacks triggers a Shutdown event, which runs the instance's Shutdown recipes.
3. After triggering the Shutdown event, AWS 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 \(p. 140\)](#) 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** - AWS OpsWorks Stacks is terminating the Amazon EC2 instance.
2. **shutting_down** - AWS 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 \(p. 253\)](#).

Tip

To have AWS OpsWorks Stacks automatically replace failed instances, enable auto healing. For more information, see [Using Auto Healing \(p. 145\)](#).

Managing Load with Time-based and Load-based Instances

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 AWS 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 Windows and Linux 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 AWS 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. AWS 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 AWS 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 AWS OpsWorks Stacks starts and stops to handle unpredictable traffic variations. AWS 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 \(p. 107\)](#).

Note

If you have created apps for the instances' layer or created custom cookbooks, AWS OpsWorks Stacks automatically deploys the latest version to time-based and load-based instances when they are first started. However, AWS OpsWorks Stacks does not necessarily deploy the latest cookbooks to restarted offline instances. For more information, see [Editing Apps \(p. 224\)](#) and [Updating Custom Cookbooks \(p. 251\)](#).

Topics

- [Using Automatic Time-based Scaling \(p. 182\)](#)
- [Using Automatic Load-based Scaling \(p. 184\)](#)
- [How Load-based Scaling Differs from Auto Healing \(p. 188\)](#)

Using Automatic Time-based Scaling

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. AWS 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

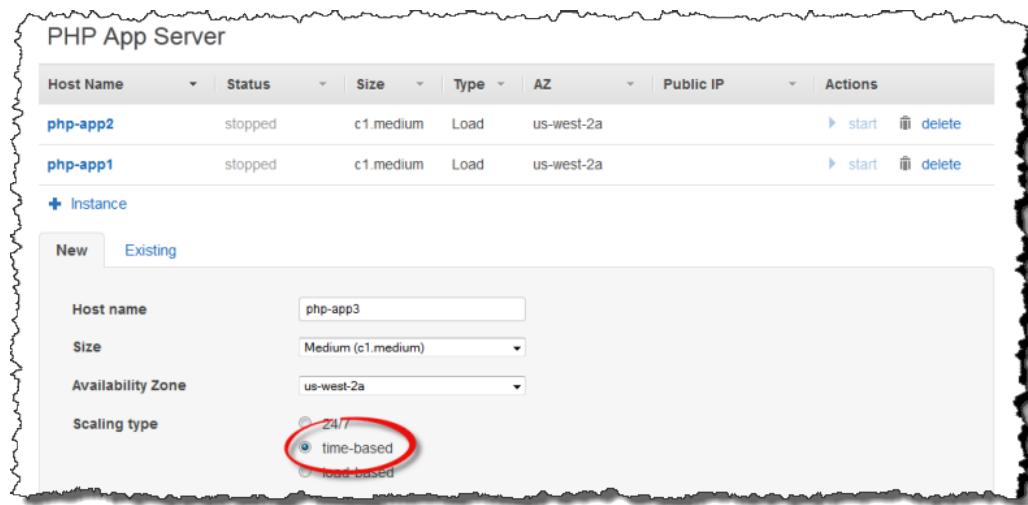
- [Adding a Time-Based Instance to a Layer \(p. 182\)](#)
- [Configuring a Time-Based Instance \(p. 183\)](#)

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

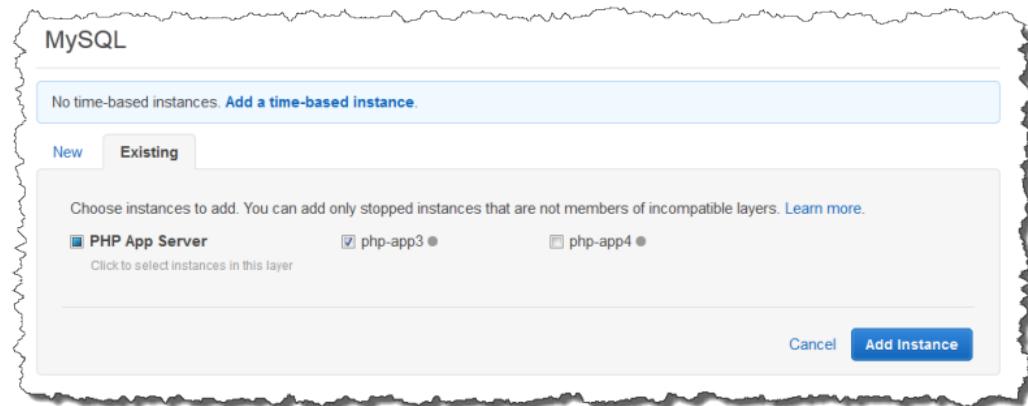
1. On the **Instances** page, click **+ Instance** to add an instance. On the **New** tab, click **Advanced >>** and then click **time-based**.



- Configure the instance as desired. Then click **Add Instance** to add the instance to the layer.

To add an existing time-based instance to a layer

- On the **Time-based Instances** page, click **+ Instance** if a layer already has a time-based instance. Otherwise, click **Add a time-based instance**. Then click the **Existing** tab.



- On the **Existing** tab, select an instance from the list. The list shows only time-based instances.

Note

If you change your mind about using an existing instance, click the **New** tab to create a new instance, as described in the preceding procedure.

- Click **Add instance** to add the instance to the layer.

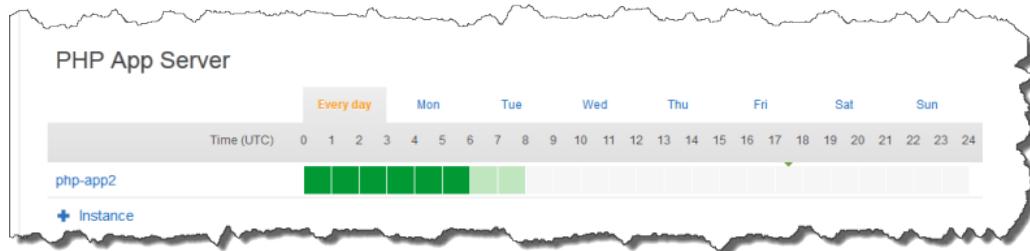
Configuring a Time-Based Instance

After you add a time-based instance to a layer, you configure its schedule as follows.

To configure a time-based instance

- In the navigation pane, click **Time-based** under **Instances**.
- Specify the online periods for each time-based instance by clicking the appropriate boxes below the desired hour.

- To use the same schedule every day, click the **Every day** tab and then specify the online time periods.
- To use different schedules on different days, click each day and select the appropriate time periods.



Note

Make sure to allow for the amount of time it takes to start an instance and the fact that AWS 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, AWS OpsWorks Stacks might not start the instance until several minutes past 1:00 UTC, and it will take several more minutes for it to come online.

You can modify an instance's online time periods at any time using the previous configuration steps. The next time AWS OpsWorks Stacks checks, it uses the new schedule to determine whether to start or stop instances.

Note

You can also add a new time-based instance to a layer by going to the **Time-based** page and clicking **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

Load-based instances let you rapidly start or stop instances in response to changes in incoming traffic. AWS 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 *upscale* and *downscale* 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 AWS OpsWorks Stacks responds to scaling events by specifying the following:

- How many instances to start or stop.
- How long AWS 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 AWS 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.

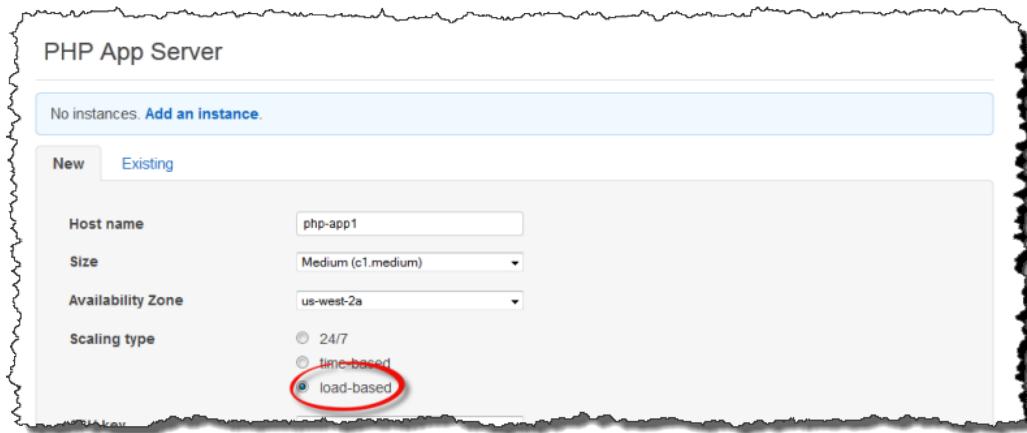
When a scaling event occurs, AWS OpsWorks Stacks starts or stops only load-based instances. It does not start or stop 24/7 instances or time-based instances.

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, click **+Instance** to add an instance. Click **Advanced >>** and then click **load-based**.



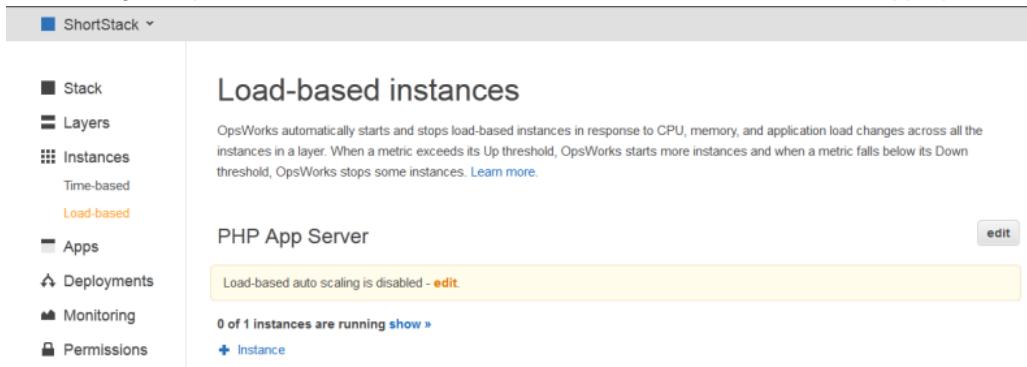
2. Configure the instance as desired. Then click **Add Instance** to add the instance to the layer.

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, click **Load-based** under **Instances** and click **edit** for the appropriate layer.



2. Set **Load-based auto scaling enabled** to **Yes**. Then set threshold and scaling parameters to define how and when to add or delete instances.

Load-based Rails App Server Configuration

The screenshot shows the 'Scaling configuration' section of the AWS OpsWorks Stacks interface. At the top, there's a toggle switch labeled 'On'. Below it, two main sections are shown: 'Based on Layer averages' and 'Based on Amazon CloudWatch alarms'.

Based on Layer averages:

Metric	UP	DOWN
Average CPU	80 %	30 %
Average memory	[] %	[] %
Average load	[]	[]

Scaling parameters:

UP	DOWN
Start servers in batches of <input type="text" value="1"/>	Stop servers in batches of <input type="text" value="1"/>
If thresholds are exceeded <input type="text" value="5"/> min	If thresholds are undershot <input type="text" value="10"/> min
After scaling, ignore metrics <input type="text" value="5"/> min	After scaling, ignore metrics <input type="text" value="10"/> min

Based on Amazon CloudWatch alarms:

UP	DOWN
Select an alarm ...	Select an alarm ...

At the bottom right are 'Cancel' and 'Save' buttons.

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 \(computing\)](#).

Crossing a threshold causes a scaling event, upscaling if more instances are needed and downscaling if fewer instances are needed. AWS 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 AWS OpsWorks Stacks update the role for you when you first use this feature or you can edit the role manually. For more information, see [Allowing AWS OpsWorks Stacks to Act on Your Behalf \(p. 298\)](#).

Scaling parameters

These parameters control how AWS 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 AWS OpsWorks Stacks triggers a scaling event.
- **After scaling, ignore metrics** – The amount of time (in minutes) after a scaling event occurs that AWS OpsWorks Stacks should ignore metrics and suppress additional scaling events.

For example, AWS OpsWorks Stacks adds new instances following an upscaling event but the instances won't start running 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 AWS OpsWorks Stacks to suppress scaling events long enough to get the new instances online.

You can also 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 being downscaled, an instance upscale event could be immediately followed by a memory downscaling event. To prevent this from happening, you can increase the number of minutes in the **After scaling, ignore metrics** setting. In the example mentioned, the CPU scaling would occur, but the memory downscaling event would not.

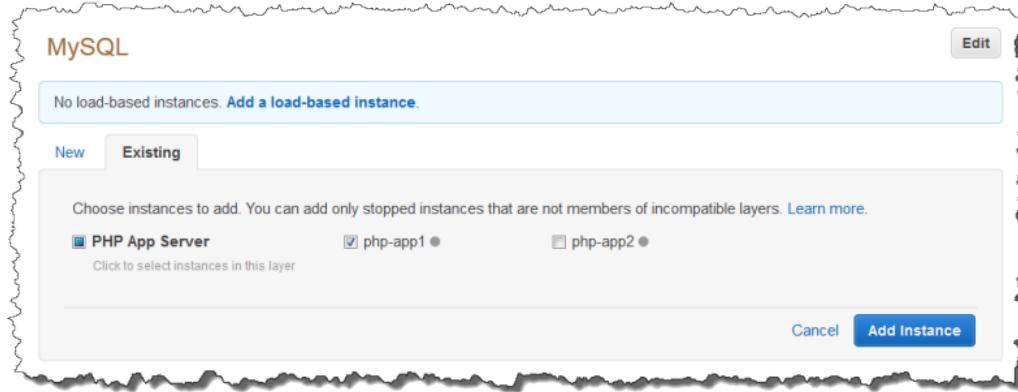
3. To add additional load-based instances, click **+ Instance**, configure the settings, and click **Add Instance**. Repeat until you have enough load-based instances to handle your maximum anticipated load. Then click **Save**.

Note

You can also add a new load-based instance to a layer by going to the **Load-based** page and clicking **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, click **Load-based** under **Instances**.
2. If you have already enabled load-based automatic scaling for a layer, click **+Instance**. Otherwise, click **Add a load-based instance**. Then click the **Existing** tab.



3. On the **Existing** tab, select an instance from the list. The list shows only load-based instances.

Note

If you change your mind about using an existing instance, click the **New** tab to create a new instance as described in the preceding procedure.

4. Click **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, click **Load-based** under **Instances** and click **edit** for the appropriate layer.
2. Toggle **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, AWS OpsWorks Stacks does not start or stop any instances. With auto healing, on the other hand, AWS 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 that your CPU upscaling threshold is 80%, and then one instance stops responding.

- If auto healing is disabled, and the remaining running instances are able to keep average CPU utilization below 80%, AWS 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, AWS OpsWorks Stacks starts a replacement instance irrespective of the load thresholds.

Using Computing Resources Created Outside of AWS OpsWorks Stacks

Note

This feature is supported only for Linux stacks.

[Instances \(p. 158\)](#) describes how to use AWS 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 AWS 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 AWS OpsWorks Stacks-managed instances, and you can manage them much like regular AWS OpsWorks Stacks instances:

- **Manage user permissions** – You can use [AWS OpsWorks Stacks user management \(p. 282\)](#) 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 AWS 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 \(p. 138\)](#), AWS OpsWorks Stacks automatically runs a specified set of Chef recipes on the instance at key points in its [lifecycle \(p. 253\)](#), including your custom recipes. Note that you can assign registered Amazon EC2 instances to [custom layers \(p. 157\)](#) 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** – AWS OpsWorks Stacks provides a simple way to install operating system security patches and updates on a stack's instances.
- **Deploy applications** – AWS OpsWorks Stacks deploys applications consistently to all of the stack's application server instances.
- **Monitoring** – AWS 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 AWS OpsWorks Stacks.

2. Optionally, assign the instance to a layer.

This step lets you take full advantage of AWS 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 AWS OpsWorks Stacks to manage the instance.

4. When you no longer need the instance in the stack, deregister it, which removes the instance from AWS OpsWorks Stacks.

The following sections describe this process in detail.

Topics

- [Registering an Instance with an AWS OpsWorks Stacks Stack \(p. 189\)](#)
- [Managing Registered Instances \(p. 202\)](#)
- [Assigning a Registered Instance to a Layer \(p. 204\)](#)
- [Unassigning a Registered Instance \(p. 204\)](#)
- [Deregistering a Registered Instance \(p. 204\)](#)
- [Registered Instance Life Cycle \(p. 205\)](#)

Registering an Instance with an AWS OpsWorks Stacks Stack

Note

This feature is supported only for Linux stacks.

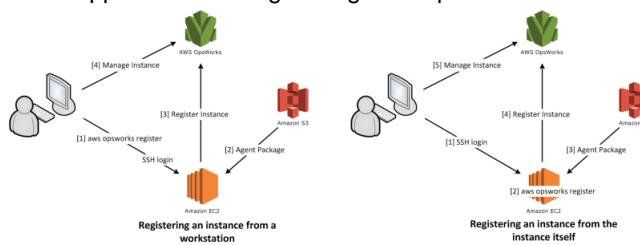
The registration process installs an agent on the instance, which AWS OpsWorks Stacks then uses to manage the instance, and registers the instance with the specified stack. At that point, the instance is part of the stack and is managed by AWS OpsWorks Stacks. For more information, see [Managing Registered Instances \(p. 202\)](#).

You register an instance by running the `register` AWS CLI command. You can run this command from the instance itself, or from a separate workstation.

Note

The `register` command is not included in the [AWS Tools for Windows PowerShell](#).

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 [AWS OpsWorks Stacks console](#) to manage a registered instance, but you must use an AWS CLI command to register the instance. The reason for this requirement 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 \(p. 190\)](#)
- [Registering Amazon EC2 and On-premises Instances \(p. 193\)](#)

Walkthrough: Register an Instance from Your Workstation

Note

This feature is supported only for Linux stacks.

The registration process is designed to support a variety of scenarios. This section gets you started by walking you through an end-to-end example of one of those scenarios, how use your workstation to register an Amazon EC2 instance. The other registration scenarios use a similar procedure. For more information, see [Registering Amazon EC2 and On-premises Instances \(p. 193\)](#).

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 \(p. 190\)](#)
- [Step 2: Install and Configure the AWS CLI \(p. 190\)](#)
- [Step 3: Register the Instance with the EC2Register Stack \(p. 191\)](#)

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 [AWS OpsWorks Stacks console](#) to [create a new stack \(p. 120\)](#) 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 will need those values later.
 - The instance must be based on one of the [supported Linux operating systems \(p. 160\)](#). 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

The key part of the registration process must be handled by running the `aws opsworks register` AWS CLI command (which, for convenience, is abbreviated to `register` for the rest of this topic). Before you register

your first instance, you must install a current version of the AWS CLI or update your existing version. The installation details depend on your workstation's operating system, so see [Installing the AWS Command Line Interface](#) for directions.

You must provide `register` with a set of AWS credentials that have appropriate permissions. This walkthrough does so by creating a new IAM user with appropriate permissions, installing the user's credentials on the workstation, and then passing those credentials to `register`.

To create the IAM User

1. On the [IAM console](#), choose **Users** in the navigation pane, and then choose **Add user**.
2. Add a user named `EC2Register`. In the **Select AWS access type** area, select **Programmatic access**, and then choose **Next: Permissions**.
3. On the **Set permissions** page, choose **Attach existing policies directly**.
4. Enter `opsworks` in the **Policy type** filter box to display the AWS OpsWorks policies, select **AWSOpsWorksRegisterCLI**, and then choose **Next: review**. This policy grants your user the permissions that are required to run `register`.

Attach one or more existing policies directly to the user or create a new policy. [Learn more](#)

	Policy name	Type	Attachments	Description
<input type="checkbox"/>	<code>AWSOpsWorksFullAccess</code>	AWS managed	3	Provides full access to A
<input type="checkbox"/>	<code>AWSOpsWorksRole</code>	AWS managed	0	Default policy for AWS O
<input checked="" type="checkbox"/>	<code>AWSOpsWorksRegisterCLI</code>	AWS managed	0	Provides access to regist
<input type="checkbox"/>	<code>AWSOpsWorksInstanceRegistration</code>	AWS managed	0	Provides access for an A

5. On the **Review** page, choose **Create user**.
6. Choose **Download .csv**, save the credentials file to a convenient location on your system, and then choose **Close**.

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 AWS OpsWorks Stacks, return to the EC2Register stack, click **Instances** in the navigation pane, and then click **register an instance**.
2. Select **EC2 Instances**, click **Next: Select Instances**, and select your instance from the list.
3. Click **Next: Install AWS CLI**, and **Next: Register Instances**. AWS 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 will have `register` 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 will look something like the following:

```
aws opsworks register --infrastructure-class ec2 --region us-east-1  
--stack-id 247be7ea-3551-4177-9524-1ff804f453e3 --ssh-username [username]  
--ssh-private-key [key-file] i-f1245d10
```

Note

You must set the region to `us-east-1`—the AWS OpsWorks Stacks service's endpoint region—not the stack's region. AWS OpsWorks Stacks will determine 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.
 - Replace `[private 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 will be either `ubuntu`, for an Ubuntu instance, or `ec2-user`, for a Red Hat Enterprise Linux (RHEL) or Amazon Linux instance.

- Insert `--profile ec2register`, which runs `register` with the EC2Register user's credentials.

If your default credentials have appropriate permissions, you can omit this argument.

The edited command string should look something like:

```
aws opsworks register --profile ec2register --infrastructure-class ec2 \  
--region us-east-1 --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 AWS OpsWorks Stacks console and click **Done**. Then click **Instances** in the navigation pane. Your instance should be listed under **Unassigned Instances**. You can then [assign the instance to a layer \(p. 204\)](#) or leave it where it is, depending on how you intend to manage the instance.

6. When you are finished, [stop the instance \(p. 180\)](#) and then [delete it \(p. 180\)](#). This terminates the Amazon EC2 instance, so you don't incur any further charges.

Registering Amazon EC2 and On-premises Instances

Note

This feature is supported only for Linux stacks.

This section describes how to register an Amazon EC2 or on-premises instance with an AWS OpsWorks Stacks stack.

Topics

- [Preparing the Instance \(p. 193\)](#)
- [Installing and Configuring the AWS CLI \(p. 194\)](#)
- [Registering the Instance \(p. 196\)](#)
- [Using the register Command \(p. 198\)](#)
- [Example register Commands \(p. 200\)](#)
- [The AWSOpsWorksRegisterCLI Policy \(p. 201\)](#)

Preparing the Instance

Note

This feature is supported only for Linux stacks.

Before registering an instance, you must ensure that it is compatible with AWS 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 \(p. 160\)](#).

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 AWS 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 have Internet access that allows it to communicate with the AWS 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.

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 [AWS OpsWorks Stacks Operating Systems \(p. 159\)](#).

For more information, see [Using Custom AMIs \(p. 172\)](#).

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 AWS 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 AWS OpsWorks Stacks.
- [Spot instances](#) are not supported, because they do not work with [auto healing](#).

When you register an Amazon EC2 instance, AWS OpsWorks Stacks does not modify the instance's [security groups](#) or rules. You must ensure that the instance's security group rules are compatible with AWS OpsWorks Stacks, as follows:

Ingress Rules

The 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

The egress rules should allow the following:

- Traffic to the AWS 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](#)

Before you register your first instance, you must install and configure a current version of the AWS CLI on the computer that you will run `register` from. For more information about installing and configuring the AWS CLI, see [Installing the AWS Command Line Interface](#) and [Configuring the AWS Command Line Interface](#).

Note

Although [AWS Tools for Windows 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 typically handle this task by installing IAM user credentials with appropriate permissions on the workstation or instance to be registered. You can then run `register` with those credentials, as described later.

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 AWS IAM Role to an Existing Amazon EC2 Instance by Using the AWS CLI](#) (blog post).

You specify permissions by attaching an IAM policy to the IAM user or role. For `register`, you usually attach the `AWSOpsWorksRegisterCLI` policy, which grants permissions to register both on-premises and Amazon EC2 instances.

To examine the `AWSOpsWorksRegisterCLI` policy, see [The AWSOpsWorksRegisterCLI Policy \(p. 201\)](#). For more information about creating and managing AWS credentials, see [AWS Security Credentials](#).

Topics

- [Using Installed Credentials \(p. 195\)](#)
- [Using an IAM Role \(p. 195\)](#)

Using Installed Credentials

There are several ways to install IAM user credentials on a system and provide them to an AWS CLI command. The following describes the preferred approach and assumes that you will create a new IAM user. You can also use an existing IAM 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.
2. Attach the `AWSOpsWorksRegisterCLI` policy to the user. If you prefer, you can attach a policy that grants broader permissions, as long as it includes the `AWSOpsWorksRegisterCLI` 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 an IAM 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` argument to specify the profile name. `register` will then run 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 on how the AWS CLI determines default credentials, see [Configuring the AWS Command Line Interface](#).

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. This approach allows you to avoid

installing your credentials on the instance. For more information about how to attach an IAM role to an existing instance, see [Attach an AWS IAM Role to an Existing Amazon EC2 Instance by Using the AWS CLI](#) (blog post).

If the instance is running and has a role, you can grant the required permissions by attaching the `AWSOpsWorksRegisterCLI` 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` could therefore not have the required permissions. For more information, see [Configuration Settings and Precedence](#).

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 \(p. 195\)](#).

Registering the Instance

Note

This feature is supported only for Linux stacks.

You register an instance by running the `register` command from your workstation or from the instance. The simplest way to handle the operation is to use the [AWS 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 AWS 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 [AWS 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. AWS 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 have a current version of the AWS CLI. To install or update the AWS CLI, the installation wizard page provides links to installation and configuration instructions. After you have verified the CLI installation, specify whether you will run the command from the instance to be registered or 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-east-1 --stack-id  
247be7ea-3551-4177-9524-1ff804f453e3 --ssh-username [username] i-f1245d10
```

If you set **I use SSH keys** to **Yes**, AWS OpsWorks Stacks adds a `--ssh-private-key` argument to the string, which you can use to specify a private SSH key file.

Tip

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 will run `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 \(p. 198\)](#). 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 \(p. 200\)](#).

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 AWS 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 AWS OpsWorks Stacks service to provide secure communication.

The user's permissions allow only the `opsworks:RegisterInstance` action, and they expire after 15 minutes.

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 AWS OpsWorks Stacks agent to be unable to communicate with the service. To help prevent problems managing registered instances in the event that the IAM user is accidentally deleted, add the `--use-instance-profile` parameter to your `register` command to use the instance's built-in instance profile instead.

5. An IAM group for the stack, which contains the registered instances' IAM users.
5. Creates an RSA key pair and sends the public key to AWS OpsWorks Stacks.

This key pair is used to encrypt communications between the agent and AWS OpsWorks Stacks.

6. Registers the instance with AWS 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 AWS 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 on managing stack resources, see [Resource Management \(p. 256\)](#). For more information on layer mount points, see [AWS OpsWorks Stacks Layer Reference \(p. 467\)](#).

For a complete description of the initial setup configuration changes, see [Initial Setup Configuration Changes \(p. 209\)](#).

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 \(p. 306\)](#).

Using the register Command

Note

This feature is supported only for Linux stacks.

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 on how the AWS CLI determines default credentials, see [Configuring the AWS Command Line Interface](#).

--region

(Optional) The AWS OpsWorks Stacks service endpoint's region. Do not set `--region` to the stack's region. AWS OpsWorks Stacks automatically determines the stack's region from the stack ID.

Tip

If your default region is already set, you can omit this argument. For more information on 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.

Tip

To find a stack ID, on the **Stack** page, click **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` will attempt 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 on 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 will use this command string to log in to the registered instance.

For more information on `ssh`, see [SSH](#).

--override-hostname

(Optional) Specifies a host name for the instance, which is used only by AWS 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 \(p. 650\)](#).

--use-instance-profile

(Optional) 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.

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.

AWS OpsWorks Stacks uses the instance ID to obtain the instance configuration, including the instance's public IP address. By default, AWS 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 \(p. 200\)](#).

Tip

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) AWS 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 clicking the address on the **Instances** page, AWS OpsWorks Stacks uses the specified address. AWS OpsWorks Stacks automatically determines the argument's default value.

--override-private-ip

(Optional) AWS OpsWorks Stacks displays the specified address as the instance's private IP address. It does not change the instance's private IP address. AWS OpsWorks Stacks automatically determines the argument's default value.

Example register Commands

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-east-1 \
--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-east-1 \
--infrastructure-class on-premises \
--stack-id ad21bce6-7623-47f1-bf9d-af2affad8907 \
--override-ssh "ssh your-user@192.0.2.0"
```

Tip

You can use `--override-ssh` to specify any custom SSH command string. AWS 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 \(p. 201\)](#).

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` will use 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-east-1 \
--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-east-1 \
--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. If you want 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-east-1 \
--infrastructure-class ec2 \
--stack-id 2f92ff9d-04f2-4728-879b-f4283b40783c \
--override-ssh "ssh -i mykey.pem ec2-user@10.183.201.93" \
i-2422b9c5
```

The AWSOpsWorksRegisterCLI Policy

The following example shows the `AWSOpsWorksRegisterCLI` policy, which grants permissions to use `register` to for on-premises and Amazon EC2 instances. Note that the Amazon EC2 permissions are required only for registering Amazon EC2 instances.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "opsworks:AssignInstance",  
                "opsworks>CreateStack",  
                "opsworks>CreateLayer",  
                "opsworks:DeregisterInstance",  
                "opsworks:DescribeInstances",  
                "opsworks:DescribeStackProvisioningParameters",  
                "opsworks:DescribeStacks",  
                "opsworks:UnassignInstance"  
            ],  
            "Resource": [  
                "*"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ec2:DescribeInstances"  
            ],  
            "Resource": [  
                "*"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iam:AddUserToGroup",  
                "iam>CreateAccessKey",  
                "iam>CreateGroup",  
                "iam>CreateUser",  
                "iam>ListInstanceProfiles",  
                "iam:PassRole",  
                "iam:PutUserPolicy"  
            ],  
            "Resource": [  
                "*"  
            ]  
        }  
    ]  
}
```

Managing Registered Instances

Note

This feature is supported only for Linux stacks.

When you register an instance, it becomes an AWS OpsWorks Stacks instance, and you can manage it in much the same way as instances created with AWS 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. AWS OpsWorks Stacks provides the following management functionality to all registered instances:

- **Health checks** – AWS OpsWorks Stacks monitors the agent to evaluate whether the instance continues to function.

If an instance fails a health check, AWS OpsWorks Stacks [autoheals \(p. 145\)](#) registered Amazon EC2 instances and changes the status of registered on-premises instances to `connection lost`.

- [CloudWatch monitoring \(p. 269\)](#) – 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** – AWS 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 \(p. 282\)](#).
- **Recipe execution** – You can use the [Execute Recipes stack command \(p. 133\)](#) to execute Chef recipes on the instance.
- **Operating system updates** – You can use the [Update Dependencies stack command \(p. 133\)](#) to update the instance's operating system.

To take full advantage of AWS OpsWorks Stacks management functionality, you can assign the instance to a layer. For more information, see [Assigning a Registered Instance to a Layer \(p. 204\)](#).

Some differences exist in the way AWS OpsWorks Stacks manages Amazon EC2 and on-premises instances.

Amazon EC2 Instances

- If you stop a registered Amazon EC2 instance, AWS 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 \(p. 204\)](#) or by [deleting the instance \(p. 211\)](#), 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 – AWS OpsWorks Stacks starts a new instance with the same AMI.

Note that AWS 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 AWS 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 – AWS 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

- AWS 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.

- AWS OpsWorks Stacks cannot autoheal a registered on-premises instance if it fails, but the instance will be 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

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 \(p. 253\)](#). AWS 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 \(p. 157\)](#). 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 AWS OpsWorks Stacks [built-in layers \(p. 138\)](#).

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 \(p. 138\)](#) and [AWS OpsWorks Stacks Layer Reference \(p. 467\)](#).

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. Click **Instances** in the navigation pane and then click **assign** in the instance's **Actions** column.
3. Select the appropriate layers and click **Save**.

When you assign an instance to a layer AWS 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 AWS OpsWorks Stacks to manage these resources. For more information, see [Resource Management \(p. 256\)](#).

After they are finished, the instance is in the online status and fully incorporated into the stack. AWS OpsWorks Stacks then runs the layer's assigned recipes each time a lifecycle event occurs.

Unassigning a Registered Instance

Note

This feature is supported only for Linux stacks.

To remove a registered instance from its layers, on the **Instances** page, click the instance name and then click **Unassign**. AWS OpsWorks Stacks then 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.

Deregistering a Registered Instance

When you no longer want a registered instance in your stack, you can deregister it. On the **Instances** page, just click the instance name and then click **Deregister**. AWS OpsWorks Stacks then does the following:

- 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 AWS 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.

- Removes the instance from the stack.
- For on-premises instances, stops the charges.

The instance remains in the running state, but it is under your direct control and is no longer managed by AWS OpsWorks Stacks.

Note

Both registering and deregistering computers or instances are supported only within Linux stacks. 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.

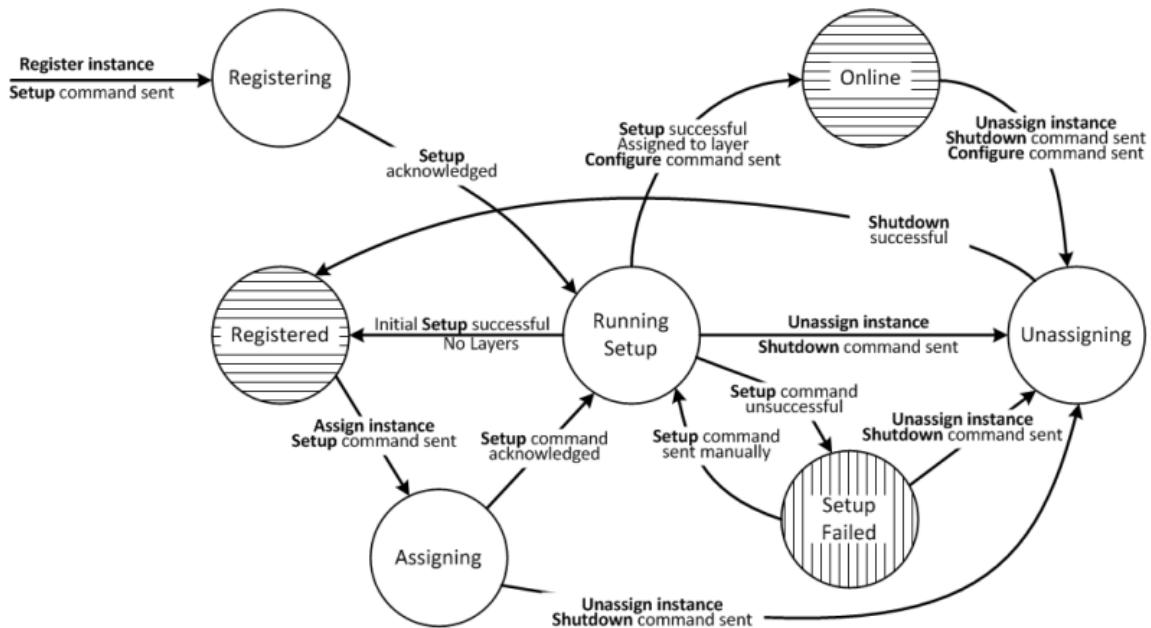
- `/etc/hosts` is backed up to `/var/lib/aws/opsworks/local-mode-cache/backup/etc/`, but is not restored.
- Entries remain for `aws` and `opsworks` in `passwd`, `group`, and `shadow` files, etc.
- `/etc/sudoers` contains a reference to an AWS OpsWorks Stacks directory.
- The following files are safe 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

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 AWS 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 AWS OpsWorks Stacks commands. The details are discussed in the following sections.

- **Setup** – This command corresponds to the Setup lifecycle event (p. 253) and runs the instance's Setup recipes.
- **Configure** – This command corresponds to the Configure lifecycle event.

AWS 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, AWS 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 \(p. 207\)](#)
- [Running Setup \(p. 207\)](#)
- [Registered \(p. 208\)](#)
- [Assigning \(p. 208\)](#)

- [Online \(p. 208\)](#)
- [Setup Failed \(p. 208\)](#)
- [Unassigning \(p. 208\)](#)
- [Initial Setup Configuration Changes \(p. 209\)](#)

Registering

After the agent sends a registration request, AWS 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 \(p. 207\)](#) state.

Running Setup

The Running Setup state runs the instance's Setup recipes. The details how setup works depend on the preceding state.

Note

If you unassign the instance while it is in the Running Setup state, AWS 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 \(p. 208\)](#) state.

Topics

- [Registering \(p. 207\)](#)
- [Assigning \(p. 207\)](#)
- [Setup Failed \(p. 208\)](#)

Registering

During the Registering process, setup creates an AWS OpsWorks Stacks instance to represent the registered instance in the stack and performs an initial setup by running 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 AWS 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 \(p. 209\)](#).

Note

For consistency, AWS 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 \(p. 208\)](#) state.
- If setup is unsuccessful, the instance moves to the [Setup Failed \(p. 208\)](#) state.

Assigning

The instance has at least one assigned layer. AWS OpsWorks Stacks runs each layer's Setup recipes, including any custom recipes that you have [assigned to the layers' Setup event \(p. 252\)](#).

- If setup is successful, the instance moves to the Online state and AWS 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 \(p. 208\)](#) state fails, you can try again by using the [Setup stack command \(p. 133\)](#) to manually rerun the instance's Setup recipes.

- If setup is successful, the assigned instance moves to the [Online \(p. 208\)](#) state and AWS 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 AWS 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, AWS OpsWorks Stacks sends a Setup command to the instance and it moves to the [Assigning \(p. 208\)](#) state.

Assigning

After the instance acknowledges the Setup command, it moves to the [Running Setup \(p. 207\)](#) state.

If you unassign the instance while it is in the Assigning state, AWS OpsWorks Stacks terminates the setup process and sends a Shutdown command. The instance moves to the [Unassigning \(p. 208\)](#) state.

Online

The instance is now a member of at least one layer and is treated like a regular AWS OpsWorks Stacks instance. It can remain in this state indefinitely.

If you unassign the instance while it is in the Online state, AWS 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 \(p. 208\)](#) state.

Setup Failed

The Setup command has failed.

- You can try again by running the [Setup stack command \(p. 133\)](#).
The instance returns to the [Running Setup \(p. 207\)](#) state.
- If you unassign the instance, AWS OpsWorks Stacks sends a Shutdown command to the instance.
The instance moves to the [Unassigning \(p. 208\)](#) state.

Unassigning

After the Shutdown command finishes, the instance is no longer assigned to any layers and returns to the [Registered \(p. 208\)](#) 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.

<code>autofs</code>	<code>libicu-dev</code>	<code>libopenssl-ruby</code>
<code>libssl-dev</code>	<code>libxml2-dev</code>	<code>libxslt-dev</code>
<code>libyaml-dev</code>	<code>monit</code>	<code>ntpd</code>
<code>procps</code>	<code>ruby</code>	<code>ruby-dev</code>
<code>rubygems</code>	<code>screen</code>	<code>sqlite</code>
<code>vim</code>	<code>xfs</code>	

Editing the Instance Configuration

You can edit instance configurations, including [registered Amazon Elastic Compute Cloud \(Amazon EC2\) instances \(p. 188\)](#), 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 \(p. 140\)](#).

- 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 \(p. 124\)](#) 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, AWS 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. AWS 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 \(p. 168\)](#). 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, AWS OpsWorks Stacks starts a new Amazon EC2 instance with the updated configuration.

Deleting AWS OpsWorks Stacks Instances

Important

This topic applies only to Amazon EC2 instances that are managed by AWS OpsWorks Stacks. For more information about how to delete instances that are managed by the Amazon EC2 console or API, see [Terminate Your Instance](#).

You can use AWS OpsWorks Stacks to stop an instance, including [registered Amazon EC2 instances \(p. 188\)](#). 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.

Note

You cannot use AWS 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 \(p. 210\)](#).

Important

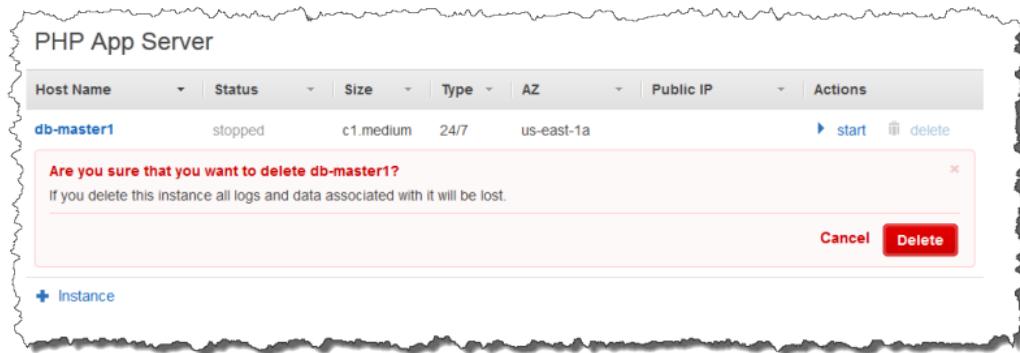
You should delete AWS OpsWorks Stacks instances only by using the AWS OpsWorks Stacks console or API. In particular, you should not delete AWS OpsWorks Stacks instances by using the Amazon EC2 console or API because Amazon EC2 actions are not automatically synchronized with AWS OpsWorks Stacks. For example, if auto healing is enabled and you terminate an instance by using the Amazon EC2 console, AWS 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 \(p. 145\)](#).

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, AWS 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, AWS OpsWorks Stacks displays the following section.



Click **Yes, delete** to confirm. In addition to deleting the instance from the stack, this action deletes any associated logs or data; it also deletes any Amazon EBS volumes on the instance.

Using SSH to Log In to a Linux Instance

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. AWS 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 \(p. 120\)](#).

You can optionally override the default key pair and specify a different pair when you create an instance. In either case, AWS 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 \(p. 305\)](#) with AWS OpsWorks Stacks.

The user or an administrator registers the public key with AWS 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. AWS 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 an IAM user

1. In the AWS OpsWorks Stacks navigation pane, click **Permissions**.
2. Select **SSH/RDP** for the desired AWS Identity and Access Management (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 \(p. 651\)](#) commands—select **sudo/admin** also.

Stack	Permission level						Instance access	
	Deny	IAM Policies Only	Show	Deploy	Manage	SSH / RDP	sudo / admin	
CLITest	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Chef9Test	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>	
EC2Register	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
JavaStack	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>	

For more information on how to use AWS OpsWorks Stacks to manage SSH access, see [Managing SSH Access \(p. 302\)](#).

Topics

- [Using the Built-in MindTerm SSH Client \(p. 213\)](#)
- [Using a Third-Party SSH Client \(p. 214\)](#)

Using the Built-in MindTerm SSH Client

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 IAM user.
3. On the **Instances** page, choose **ssh** in the **Actions** column for the appropriate instance.

PHP App Server								
Hostname	Status	Size	Type	AZ	Public IP	Actions		
php-app1	online	c1.medium	24/7	us-east-1a	123.45.67.89	<input type="button" value="stop"/>	<input checked="" type="button" value="ssh"/>	
+ Instance								

4. For **Private key**, provide a path to the IAM 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

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 AWS 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

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 \(p. 307\)](#).
- Ordinary users – AWS 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 \(p. 289\)](#) 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 \(p. 215\)](#)
- [Logging in As an Ordinary User \(p. 215\)](#)
- [Logging in As Administrator \(p. 216\)](#)

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, AWS OpsWorks Stacks creates a set of security groups. They include one named something like `AWS-OpsWorks-RDP-Server`, which AWS 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 AWS OpsWorks Stacks.

To authorize RDP for an IAM user

1. In the AWS OpsWorks Stacks navigation pane, click **Permissions**.
2. Select the **SSH/RDP** checkbox for the desired IAM user to grant the necessary permissions. If you want the user to have administrator permissions, you should also select **sudo/admin**.

Permissions							
Stack	Permission level				Instance access		
	Deny	IAM Policies Only	Show	Deploy	Manage	SSH / RDP	sudo / admin
CLITest	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>
Chef9Test	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>
EC2Register	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
JavaStack	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>

Authorized users can log in to any of the stack's online instances, as follows.

To log in as an ordinary 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.

Tip

AWS 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 AWS OpsWorks Stacks is concerned, the EC2 instance is running and it's often useful to log in to troubleshoot the issue. AWS 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 \(p. 302\)](#) 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 \(p. 120\)](#) when you create the stack, or you can [specify a key pair for a particular instance \(p. 168\)](#) 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

An AWS 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, AWS OpsWorks Stacks triggers a Deploy event, which runs each layer's Deploy recipes. AWS OpsWorks Stacks also installs [stack configuration and deployment attributes \(p. 376\)](#) 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 \(p. 217\)](#)
- [Deploying Apps \(p. 221\)](#)
- [Editing Apps \(p. 224\)](#)
- [Connecting an Application to a Database Server \(p. 224\)](#)
- [Using Environment Variables \(p. 226\)](#)
- [Passing Data to Applications \(p. 226\)](#)
- [Using Git Repository SSH Keys \(p. 228\)](#)
- [Using Custom Domains \(p. 229\)](#)
- [Using SSL \(p. 230\)](#)

Adding Apps

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 \(p. 282\)](#).

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 \(p. 319\)](#).

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 \(p. 218\)](#).
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 \(p. 218\)](#)
- [Application Source \(p. 218\)](#)
- [Data Sources \(p. 220\)](#)
- [Environment Variables \(p. 220\)](#)
- [Domain and SSL Settings \(p. 221\)](#)

Settings

Name

The app name, which is used to represent the app in the UI. AWS 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 \(p. 376\)](#). 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

AWS OpsWorks Stacks assigns the **Document root** setting to the [\[:document_root\] \(p. 527\)](#) 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 \(p. 369\)](#).

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

AWS 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, AWS OpsWorks Stacks puts the repository information in the app's [deploy attributes \(p. 380\)](#), but you must implement custom recipes to handle the deployment tasks.

Topics

- [HTTP Archive \(p. 218\)](#)
- [Amazon S3 Archive \(p. 218\)](#)
- [Git Repository \(p. 219\)](#)
- [Other Repositories \(p. 219\)](#)

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

AWS 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 user name and password.

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 AWS OpsWorks Stacks to access private buckets, you must be an IAM 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 \(IAM\) Documentation](#).
3. Put the code and any associated files in a folder and store the folder in a compressed archive—zip, gzip, bz2, Java WAR, or tarball.

Note

AWS 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 AWS 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. AWS 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 \(p. 313\)](#) 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. For Git submodules, the specified key must have access to those submodules. For more information, see [Using Git Repository SSH Keys \(p. 228\)](#).

Important

The deploy SSH key cannot require a password; AWS OpsWorks Stacks has no way to pass it through.

Branch/Revision

If the repository has multiple branches, AWS 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, AWS 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 \(p. 369\)](#).

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 \(p. 150\)](#).
- **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, AWS OpsWorks Stacks adds the database instance's connection to the app's [deploy attributes \(p. 380\)](#).

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 \(p. 224\)](#).

To detach a database server from an app, [edit the app's configuration \(p. 224\)](#) 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 10 KB (10240 Bytes). 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 10KB)."

AWS OpsWorks Stacks stores the variables as attributes in the app's [deploy attributes \(p. 380\)](#). 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 \(p. 226\)](#).

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

By default, the Chef deployment logs do not include environment variables unless deployment fails during a step that used them. In that case, the environment variables are logged as part of the failure's backtrace. To routinely include environment variables in deployment logs, specify an environment variable named SHOW_ENVIRONMENT_VARIABLES. You don't need to specify a value; AWS OpsWorks Stacks checks only for the variable's presence and ignores the value.

For alternative ways to pass data to your application, see [Passing Data to Applications \(p. 226\)](#) and [Using an Amazon S3 Bucket \(p. 574\)](#).

Domain and SSL Settings

For the Other app type, AWS 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 \(p. 229\)](#).

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 \(p. 230\)](#).

Deploying Apps

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, AWS 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 \(p. 282\)](#).

To deploy an app

1. On the **Apps** page, click the app's **deploy** action.

Apps

An app represents code stored in a repository that you want to install on application server instances. When you deploy the app, OpsWorks downloads the code from the repository to the specified server instances. [Learn more](#).

Name	Type	Last deployment	Actions
SimplePHP	PHP		deploy edit delete
+ App			

Tip

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. AWS OpsWorks Stacks adds a set of [stack configuration and deployment attributes \(p. 376\)](#) 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 AWS OpsWorks Stacks settings \(p. 348\)](#) or pass custom settings to your custom recipes. For more information about how to use custom JSON, see [Using Custom JSON \(p. 135\)](#).

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 \(p. 135\)](#). Custom JSON is limited to 80 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 \(p. 442\)](#).

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 \(p. 235\)](#).

5. Click **Deploy** to run the deploy recipes on the specified instances, which displays the Deployment page. When the process is complete, AWS OpsWorks Stacks marks the app with a green check to indicate successful deployment. If deployment fails, AWS 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.

Deployment PHPTestApp - deploy

[Repeat](#)

Status	successful	User	
Created at	2017-04-11 18:59:10 UTC		
Completed at	2017-04-11 18:59:59 UTC		
Duration	00:00:49		

Tip

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 \(p. 253\)](#), 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, AWS 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 \(p. 133\)](#). 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 \(p. 482\)](#).

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 \(p. 133\)](#) to run them. For more information on how to implement and install custom recipes, see [Cookbooks and Recipes \(p. 235\)](#).

Editing Apps

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 AWS 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 \(p. 282\)](#).

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, AWS 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 AWS OpsWorks Stacks, but does not affect the stack's instances. When you first [deploy an app \(p. 221\)](#), 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. AWS OpsWorks Stacks automatically deploys the current app version to new instances when they are started. For existing instances, however, the situation is different:

- AWS OpsWorks Stacks automatically deploys the current app version to new instances when they are started.
- AWS OpsWorks Stacks automatically deploys the latest app version to offline instances, including [load-based and time-based instances \(p. 181\)](#), 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 \(p. 221\)](#)

Connecting an Application to a Database Server

You can associate an Amazon RDS database server with an app when you [create the app \(p. 217\)](#) or later by [editing the app \(p. 224\)](#). Your application can then use the database connection information—user name, password, ...—to connect to the database server. When you [deploy an app \(p. 221\)](#), AWS OpsWorks Stacks provides this information to applications in two ways:

- For Linux stacks, AWS 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.
- AWS OpsWorks Stacks includes the connection information in the [stack configuration and deployment attributes \(p. 376\)](#) 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 \(p. 226\)](#).

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 \(p. 380\)](#) 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 \(p. 217\)](#) or later by [editing the app \(p. 224\)](#). When you deploy the app, AWS OpsWorks Stacks installs a [stack configuration and deployment attributes \(p. 376\)](#) 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' \(p. 376\)](#) `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, AWS 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 AWS OpsWorks Stacks.

Note

AWS 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 \(p. 226\)](#).

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.

Connecting to a Database Server for a Windows Stack

With Windows stacks, you can use the `aws_opsworks_rds_db_instance` search index to retrieve database connection information. The following example recipe shows how to retrieve the RDS instance's ID, which is assigned to the `[:db_instance_identifier]` attribute.

```
dbserver = search(:aws_opsworks_rds_db_instance, "*:*").first
Chef::Log.info("*****The DB instance ID is:
'#{dbserver[:db_instance_identifier]}*****")
```

The search code returns a list of RDS instances that are [registered with the stack \(p. 260\)](#). This example simply retrieves the first RDS instance's instance ID and puts it in the Chef log. You can easily modify this code to write the connection information to a file in an appropriate format.

For a complete list of database connection attributes, see [Amazon RDS Data Bag \(aws_opsworks_rds_db_instance\) \(p. 670\)](#).

Using Environment Variables

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 \(p. 220\)](#), AWS OpsWorks Stacks adds the variable definitions to the app's `deploy` attributes (p. 380).

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 \(p. 226\)](#).

Passing Data to Applications

It is often useful to pass data such as key-value pairs to an application on the server. To do so, use [custom JSON \(p. 135\)](#) to add the data to the stack. AWS 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 \(p. 135\)](#).

```
{  
    "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 \(p. 218\)](#).

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 \(p. 525\)](#).

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 AWS OpsWorks Stacks installs apps, see [Deploy Recipes \(p. 369\)](#).
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 \(p. 217\)](#).
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 \(p. 135\)](#).
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 \(p. 235\)](#). 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 \(p. 249\)](#).

5. Assign the recipe to the app server layer's Deploy lifecycle event. AWS OpsWorks Stacks will then run the recipe on each new instance, after it has booted. For more information, see [Executing Recipes \(p. 252\)](#).
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, AWS 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 AWS 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

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 AWS 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 AWS 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

AWS 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 \(p. 217\)](#).

AWS 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 \(p. 376\)](#) as `node[:deploy]['appshortname'][:scm][:ssh_key]` (p. 529), and is accessible only to the root user.

Using Custom Domains

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.

AWS OpsWorks Stacks simply adds the domain settings to the app's [deploy attributes \(p. 380\)](#). 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 \(p. 235\)](#).

Running Multiple Applications on the Same Application Server

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 \(p. 229\)](#).

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

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 if possible that you disable SSLv3 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 AWS OpsWorks Stacks Layer \(p. 484\)](#)
- [Node.js App Server AWS OpsWorks Stacks Layer \(p. 491\)](#)
- [PHP App Server AWS OpsWorks Stacks Layer \(p. 492\)](#)
- [Rails App Server AWS OpsWorks Stacks Layer \(p. 493\)](#)
- [Static Web Server AWS OpsWorks Stacks Layer \(p. 497\)](#)
- [Ganglia Layer \(p. 503\)](#)

Topics

- [Step 1: Install and Configure OpenSSL \(p. 231\)](#)
- [Step 2: Create a Private Key \(p. 232\)](#)
- [Step 3: Create a Certificate Signing Request \(p. 232\)](#)
- [Step 4: Submit the CSR to Certificate Authority \(p. 233\)](#)
- [Step 5: Edit the App \(p. 233\)](#)

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 [x64](#) or [x86](#) version of the redistributable, whichever is appropriate for your environment.
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
Organizational Unit	(Optional) For additional organization information.	Marketing

Name	Description	Example
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 [Third-Party Certificate Authorities](#).

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-----
MIICuTCCAiICCQCrqFKItVQJpzANBgkqhkiG9w0BAQUFADCBoDELMAkGA1UEBhMC
dXMxEzARBgNVBAgMCnhdhc2hpbd0b24xEDEAOBgNVBACMB3N1YXR0bGUxDzANBgNV
BAoMBmFtYXpvbjEWMBQGA1UECwwNRGV2IGFuZCBUb29sczEdMBsGA1UEAwUc3Rl
cGhhbmllYXBpZXJjZS5jb20xIjAgBgkqhkiG9w0BCQEWE3Nhcg11cmNlQGFtYXpv
...
-----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:

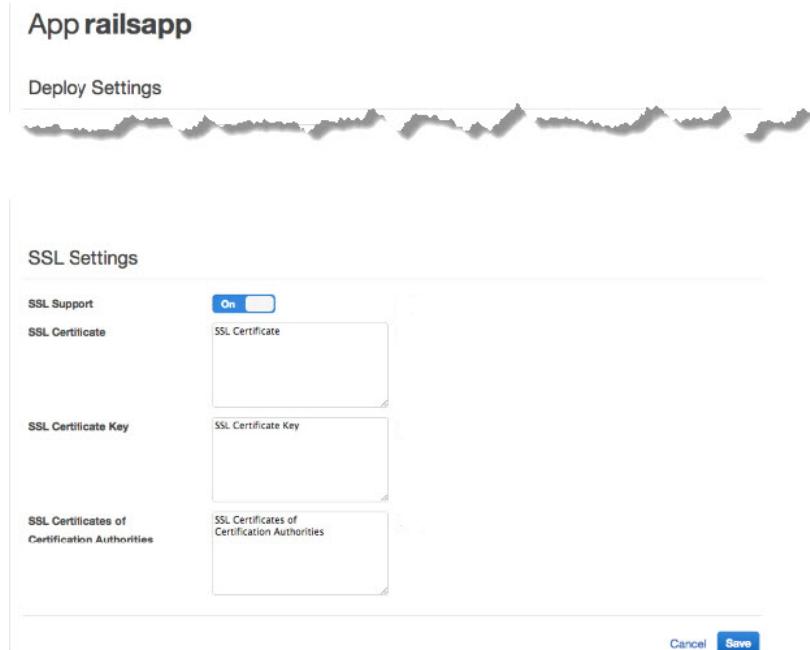
```
----BEGIN RSA PRIVATE KEY----  
MIICXQIBAAKBgQC0CYk1JY5r4vV2NHQYEpwtsLuMMBhylMrgBSHkq+HHVLYQQCL6  
+wGIIiRq5qXqZlRXje3GM5Jvcm6q0R71MFRI11FuzKyqDtneZaAIEYniZibHiUnm0  
/UNqpFDosw/6hY3ONk0fSBLU4ivD0Gjpf6J80jL3DJ4R23Ed0sdL4pRT3QIDAQAB  
AoGBAKmMfWrNRqYVtGKgnWB6Tji9QrKQLMXjmHeGg95mppdJELiXHhpMvrHtpIyK  
...  
----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 Key**.



After you click **Save**, [redeploy the application](#) (p. 221) to update your online instances.

For the [built-in application server layers](#) (p. 380), AWS 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.

If your app does not run as expected, or the webpage does not work as expected, see the "USER Questions on using the OpenSSL applications" section of the [OpenSSL FAQ](#) for troubleshooting information.

For all other layers, including custom layers, AWS OpsWorks Stacks simply adds the SSL settings to the app's [deploy attributes \(p. 380\)](#). You must implement a custom recipe to retrieve the information from the node object and configure the server appropriately.

Cookbooks and Recipes

AWS 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 AWS OpsWorks Stacks. For more information, see [Chef](#).

Note

AWS 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 \(p. 120\)](#). Windows stacks must use Chef 12.2. For more information, including guidelines for migrating stacks to more recent Chef versions, see [Chef Versions \(p. 237\)](#).

Topics

- [Cookbook Repositories \(p. 235\)](#)
- [Chef Versions \(p. 237\)](#)
- [Ruby Versions \(p. 248\)](#)
- [Installing Custom Cookbooks \(p. 249\)](#)
- [Updating Custom Cookbooks \(p. 251\)](#)
- [Executing Recipes \(p. 252\)](#)

Cookbook Repositories

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, AWS 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.

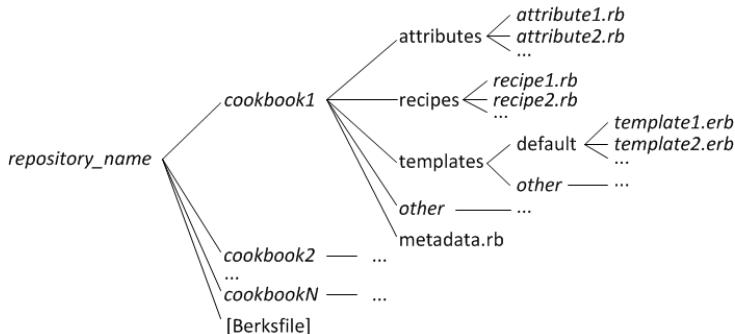
The following describes how to structure your cookbook repository, which depends on the type. The italicized text in the illustrations represents user-defined directory and file names, including the repository or archive name.

Source Control Manager

AWS 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

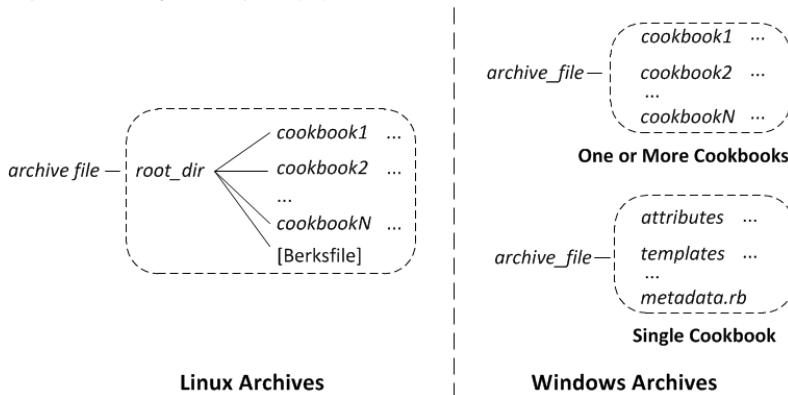
AWS OpsWorks Stacks supports the following archives:

- Linux stacks – zip, gzip, bzip2, or tarball files, stored on Amazon S3 or a web site (HTTP archive).

AWS 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, AWS 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 earlier.
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.

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 \(p. 249\)](#).

[Installing Custom Cookbooks \(p. 249\)](#) describes how to have AWS 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 AWS OpsWorks Stacks to update each online instance's local cache. For more information, see [Run Stack Commands \(p. 133\)](#).

Chef Versions

AWS OpsWorks Stacks supports multiple versions of Chef. You select the version when you [create the stack \(p. 120\)](#). AWS 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.

AWS OpsWorks Stacks currently supports Chef versions 12, 11.10, 11.4 and 0.9 for Linux stacks, and Chef 12.2 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 \(p. 120\)](#). Windows stacks must use Chef 12.2. For more information, including guidelines for migrating stacks to more recent Chef versions, see [Chef Versions \(p. 237\)](#). For complete version information, see [AWS OpsWorks Stacks Operating Systems \(p. 159\)](#).

Chef 12.2

Chef 12.2 support was introduced in May 2015, and is used only by Windows stacks. It runs with Ruby 2.0.0 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 \(p. 239\)](#), 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 [AWS OpsWorks Stacks Operating Systems \(p. 159\)](#).

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 \(p. 239\)](#), 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 [AWS OpsWorks Stacks Operating Systems \(p. 159\)](#). 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 AWS OpsWorks Stacks, but you must modify them as described in [Migrating to a new Chef Version \(p. 247\)](#). 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 AWS 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 EU (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 AWS OpsWorks Stacks, we recommend [that you specify Chef 12 \(p. 120\)](#) for new Linux stacks and migrate your existing Linux stacks to Chef 12. You can use the

AWS 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 \(p. 247\)](#).

Topics

- [Implementing Recipes for Chef 12.2 Stacks \(p. 239\)](#)
- [Implementing Recipes for Chef 12 Stacks \(p. 239\)](#)
- [Implementing Recipes for Chef 11.10 Stacks \(p. 239\)](#)
- [Implementing Recipes for Chef 11.4 Stacks \(p. 245\)](#)
- [Migrating an Existing Linux Stack to a new Chef Version \(p. 247\)](#)

Implementing Recipes for Chef 12.2 Stacks

Chef 12.2 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 2.0.0, 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 \(p. 241\)](#) and [Using Data Bags \(p. 242\)](#).

- Most of the stack configuration and deployment attributes described in [AWS OpsWorks Stacks Data Bag Reference \(p. 659\)](#) and [Built-in Cookbook Attributes \(p. 532\)](#) 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 \(p. 429\)](#). For a list of attributes, see [AWS OpsWorks Stacks Data Bag Reference \(p. 659\)](#).

Implementing Recipes for Chef 12 Stacks

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 AWS OpsWorks Stacks has provided pre-built packages for. Berkshelf is no longer installed on AWS 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 AWS 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

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"`.

- AWS OpsWorks Stacks installs a MySQL client on a stack's instances only if the stack includes a MySQL layer.
- AWS 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 \(p. 240\)](#)
- [Using Chef Search \(p. 241\)](#)
- [Using Data Bags \(p. 242\)](#)
- [Using Berkshelf \(p. 243\)](#)

Cookbook Installation and Precedence

The procedure for installing AWS OpsWorks Stacks cookbooks works somewhat differently for Chef 11.10 stacks than for earlier Chef versions. For Chef 11.10 stacks, after AWS 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 AWS 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 AWS 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 AWS OpsWorks Stacks \(p. 345\)](#).

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 AWS 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 \(p. 135\)](#).
- 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. AWS OpsWorks Stacks updates these attributes for each [lifecycle event \(p. 253\)](#), 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 AWS 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 AWS 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 \(p. 221\)](#) 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 (p. 135), 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 AWS 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 AWS 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 AWS 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 AWS OpsWorks Stacks instance. If you are writing code specifically for AWS 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 AWS OpsWorks Stacks obtains the data from the instance's stack configuration and deployment attributes. However, AWS 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 \(p. 135\)](#), 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 \(p. 221\)](#).

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 \(p. 116\)](#).) In particular, with Berkshelf, you can install AWS 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 [AWS OpsWorks Stacks Operating Systems \(p. 159\)](#).

To use Berkshelf, you must explicitly enable it, as described in [Installing Custom Cookbooks \(p. 249\)](#). 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 AWS 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, AWS 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 \(p. 133\)](#). AWS OpsWorks Stacks runs `berks install` for every Chef run, so your Berkshelf cookbooks are updated for each [lifecycle event \(p. 253\)](#), 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

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, AWS OpsWorks Stacks installs [stack configuration and deployment attributes \(p. 376\)](#) 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 AWS 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.

AWS 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 AWS OpsWorks Stacks \(p. 345\)](#). For more information on deploy directories, see [Deploy Recipes \(p. 369\)](#).

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 \(p. 135\)](#). Your recipes can then access the data by using standard Chef node syntax. For more information, see [Using Custom JSON \(p. 348\)](#).

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 AWS OpsWorks Stacks instances—to download the data from the bucket.

Note

Each AWS 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:::yourbucketname/*",
```

For more information on instance profiles, see [Specifying Permissions for Apps Running on EC2 instances \(p. 300\)](#).

Migrating an Existing Linux Stack to a new Chef Version

You can use AWS 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 AWS 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 \(p. 239\)](#).
- 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 \(p. 247\)](#). For more information on the Chef 11.4 to Chef 11.10 transition, see [Implementing Recipes: Chef 11.10 \(p. 239\)](#). For more information on the Chef 11.10 to Chef 12 transition, see [Implementing Recipes: Chef 12 \(p. 239\)](#).

- 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 \(p. 248\)](#).

- 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 \(p. 240\)](#).

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 \(p. 132\)](#). 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.

Tip

When you are testing recipes, you can use [SSH to connect to \(p. 212\)](#) the instance and then use the [Instance Agent CLI \(p. 651\)](#) [run_command \(p. 656\)](#) 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 \(p. 133\)](#) 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 \(p. 132\)](#) to change the **Chef version** stack setting.
2. Save the new settings and wait for AWS OpsWorks Stacks to update the instances, which typically takes 15 - 20 minutes.

Important

AWS 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 \(p. 253\)](#) 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

All instances in a Linux stack have Ruby installed. AWS OpsWorks Stacks installs a Ruby package on each instance, which it uses to run Chef recipes and the instance agent. AWS 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.

AWS 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 AWS 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 [AWS OpsWorks Stacks Operating Systems \(p. 159\)](#).

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
12 (b)	2.1.6, 2.2.3	None
12.2 (d)	2.0.0	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.

(e) Deprecated; nearing end of support.

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

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, AWS OpsWorks Stacks automatically installs your cookbooks on all new instances at startup. However, you must explicitly direct AWS OpsWorks Stacks to install new or updated cookbooks on any existing instances by running the [Update Custom Cookbooks stack command \(p. 133\)](#). For more information, see [Updating Custom Cookbooks \(p. 251\)](#). 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**.

The screenshot shows a configuration interface for enabling custom Chef cookbooks. At the top, there is a toggle switch labeled "Use custom Chef cookbooks" which is set to "Yes". Below this, there are several input fields:

- "Repository type": A dropdown menu currently set to "Git".
- "Repository URL": An input field containing the URL "https://github.com/awslabs/op".
- "Repository SSH key": A text input field with the placeholder "Optional".
- "Branch/Revision": A text input field with the placeholder "Optional".
- "Stack color": A color palette with several colored squares, with the first one being dark red.

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

AWS 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. AWS OpsWorks Stacks can access your repository by using Amazon EC2 roles (host operating system manager authentication), whether you use the AWS 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 \(p. 228\)](#).

Important

The deploy SSH key cannot require a password; AWS OpsWorks Stacks has no way to pass it through.

Branch/Revision

If the repository has multiple branches, AWS 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

When you provide AWS 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. AWS 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. AWS 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.

AWS 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, AWS 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. AWS 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 \(p. 132\)](#) 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.

Run Command

Settings

Command Deployes an updated set of custom Chef cookbooks from the repository to each instance's cookbooks cache.

Comment

[Advanced »](#)

Instances i

OpsWorks will run this command on **1 of 2** instances. The assigned recipes are run on all selected instances.

Select all

Rails App Server rails-app1 Click to select instances in this layer

MySQL db-master1 Click to select instances in this layer

Cancel

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 AWS OpsWorks Stacks installs on the instances. For more information, see [Using Custom JSON \(p. 135\)](#) and [Overriding Attributes \(p. 346\)](#).
5. By default, AWS 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. AWS 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, AWS OpsWorks Stacks deploys the cookbooks that are currently in the repository so they automatically get the latest version.

Executing Recipes

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 \(p. 133\)](#) or by using the agent CLI.

Topics

- [AWS OpsWorks Stacks Lifecycle Events \(p. 253\)](#)
- [Automatically Running Recipes \(p. 255\)](#)
- [Manually Running Recipes \(p. 255\)](#)

AWS OpsWorks Stacks Lifecycle Events

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, AWS 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 \(p. 255\)](#) for each layer. AWS 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 \(p. 133\)](#). AWS 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 \(p. 263\)](#) with an instance or [disassociate one from an instance \(p. 266\)](#).
- You [attach an Elastic Load Balancing load balancer \(p. 147\)](#) 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, AWS OpsWorks Stacks triggers the **Configure** event on A, B, C, and D. If you subsequently stop A, AWS OpsWorks Stacks triggers the **Configure** event on B, C, and D. AWS 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 \(p. 133\)](#).

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 AWS OpsWorks Stacks to shut an instance down but before the associated Amazon EC2 instance is actually terminated. AWS 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 \(p. 147\)](#), AWS OpsWorks Stacks waits until connection draining is complete before triggering the Shutdown event.

After triggering a Shutdown event, AWS 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 \(p. 140\)](#) to change the timeout value. For more information on instance shutdown, see [Stopping an Instance \(p. 180\)](#).

Note

[Rebooting an instance \(p. 181\)](#) does not trigger any lifecycle events.

For more discussion about the **Deploy** and **Undeploy** app commands, see [Deploying Apps \(p. 221\)](#).

After a started instance has finished booting, the remaining startup sequence is as follows:

1. AWS OpsWorks Stacks runs the instance's built-in Setup recipes, followed by any custom Setup recipes.
2. AWS OpsWorks Stacks runs the instance's built-in Deploy recipes, followed by any custom Deploy recipes.

The instance is now online.

3. AWS OpsWorks Stacks triggers a Configure event on all instances in the stack, including the newly started instance.

AWS OpsWorks Stacks runs the instances' built-in Configure recipes, followed by any custom Configure recipes.

Tip

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 \(p. 635\)](#).

Created at	Command	Duration	Log
✓ 2014-02-21 17:25:13 UTC	shutdown	00:00:28	show
✓ 2014-02-21 17:09:21 UTC	configure	00:01:04	show
✓ 2014-02-21 17:08:35 UTC	setup	00:00:45	show
✓ 2014-02-21 17:03:34 UTC	shutdown	00:01:06	show
✓ 2014-02-17 21:17:49 UTC	configure	00:01:05	show
✓ 2014-02-17 21:11:15 UTC	setup	00:06:33	show

For each lifecycle event, AWS OpsWorks Stacks installs a set of [stack configuration and deployment attributes \(p. 376\)](#) 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 \(p. 376\)](#).

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, AWS 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. AWS OpsWorks Stacks labels the unprocessed Configure events as **superseded**.

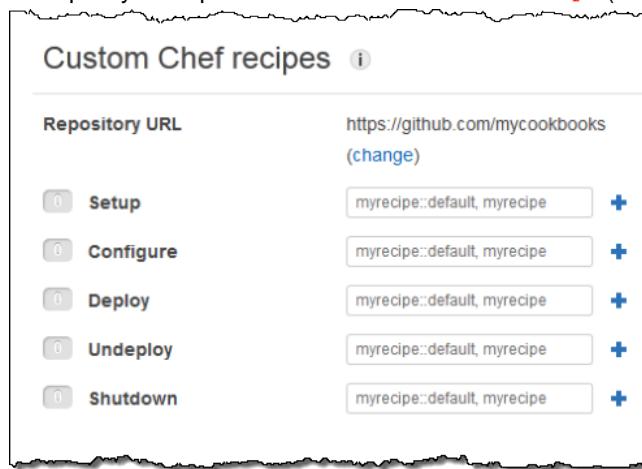
Automatically Running Recipes

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, AWS OpsWorks Stacks runs the appropriate set of recipes for the associated layer.

If you have installed custom cookbooks, you can have AWS OpsWorks Stacks run some or all of the recipes automatically by assigning each recipe to a layer's lifecycle event. After an event occurs, AWS 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 \(p. 249\)](#).
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, AWS 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

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, AWS 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**.

Run Command

Settings

Command	Execute Recipes
Recipes to execute	
Comment	Optional
Custom Chef JSON	Optional

Enter custom JSON that is passed to your Chef recipes for all instances in your stack. You can use this to override and customize built-in recipes or pass variables to your own. [Learn more](#).

Instances ⓘ

No running instances with the OpsWorks status online or setup_failed. Start [instances](#) now.

[Cancel](#) [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 \(p. 135\)](#) and [Overriding Attributes \(p. 346\)](#).
4. Under **Instances**, select the instances on which AWS OpsWorks Stacks should run the recipes.

When a lifecycle event occurs, the AWS 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 \(p. 133\)](#) or by using the agent CLI's [run_command \(p. 656\)](#) command. For more information on how to use the agent CLI, see [AWS OpsWorks Stacks Agent CLI \(p. 651\)](#).

Resource Management

The **Resources** page enables you to use your account's [Elastic IP address](#), [Amazon EBS volume](#), or [Amazon RDS](#) instance resources in an AWS OpsWorks Stacks stack. You can use **Resources** to do the following:

- [Register a resource \(p. 257\)](#) with a stack, which allows you to attach the resource to one of the stack's instances.
- [Attach a resource \(p. 261\)](#) to one of the stack's instances.

- [Move a resource \(p. 261\)](#) from one instance to another.
- [Detach a resource \(p. 265\)](#) from an instance. The resource remains registered and can be attached to another instance.
- [Deregister a resource \(p. 267\)](#). An unregistered resource cannot be used by AWS 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, or General Purpose (SSD) Amazon EBS volumes, but not RAID arrays.
- Amazon EBS volumes must be xfs formatted.

AWS 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.

- 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 AWS 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 \(p. 289\)](#) 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 \(p. 281\)](#).

Topics

- [Registering Resources with a Stack \(p. 257\)](#)
- [Attaching and Moving Resources \(p. 261\)](#)
- [Detaching Resources \(p. 265\)](#)
- [Deregistering Resources \(p. 267\)](#)

Registering Resources with a Stack

Amazon EBS volumes or Elastic IP addresses must be registered with a stack before you can attach them to instances. When AWS 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, AWS OpsWorks Stacks deregisters all resources.

Topics

- [Registering Amazon EBS Volumes with a Stack \(p. 258\)](#)
- [Registering Elastic IP Addresses with a Stack \(p. 259\)](#)
- [Registering Amazon RDS Instances with a Stack \(p. 260\)](#)

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, PIOPS, or General Purpose (SSD), but not a RAID array.
- Volumes must be XFS formatted.
- AWS 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`) or PIOPS (`io1`, followed by the IOPS per disk value in parentheses).

Resources Unregistered Volumes

Name	EC2 ID	EC2 Instance ID	Size (GiB)	Device	Volume Type	AZ	Status
Disk 1 of 2	vol-3753f475		50		standard	us-east-1a	available
Disk 2 of 2	vol-eb54f3a9		50		standard	us-east-1a	available
PHP-LB-Standard	vol-6a4bec28		100		standard	us-east-1a	available
no name	vol-68702625	i-9a5328ba	8	/dev/sda1	standard	us-east-1c	in-use

[Cancel](#) [Register with Stack](#)

- Select the appropriate volumes and click **Register to Stack**. The **Resources** page now lists the newly registered volumes.

Name	EC2 ID	Instance	Size (GiB)	Volume Type	AZ	Actions
PHP-LB-Standard	vol-6a4bec28	assign to instance	100	standard	us-east-1a	edit

[Show Unregistered Volumes](#)

[+ Unregistered 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

- 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.

No Elastic IPs have been registered yet. [Show unregistered Elastic IPs](#).

[Show Unregistered Elastic IPs](#)

- Click **Show Unregistered Elastic IPs** to display the available Elastic IP addresses in your account that are in the stack's region.

Resources Unregistered Elastic IPs

Address	Instance	Domain
192.0.2.0		standard
<input checked="" type="checkbox"/> 192.0.2.10		standard
192.0.2.20		standard

Cancel **Register with Stack**

- Select the appropriate addresses and click **Register to Stack**. This returns you to the **Resources** page, which now lists the newly registered addresses.

Address	Name	Instance	Public DNS	Actions
192.0.2.0	-	associate with instance	-	edit

Show Unregistered Elastic IPs

+ Unregistered Elastic IPs

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

- 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.

No RDS DB instances have been registered yet. [Show unregistered RDS DB instances](#).

Show Unregistered RDS DB instances

- Click **Show Unregistered RDS DB instances** to display the available Amazon RDS instances in your account that are in the stack's region.

Resources Unregistered RDS DB instances

Instance Identifier	Engine	Storage (GB)	Type	Status	Multi-AZ	Availability Zone
opsinstance1	mysql	5	t1.micro	available	No	us-east-1d
opsinstance2	mysql	5	t1.micro	available	No	us-east-1d

Connection Details for opsinstance1

User: opsuser

Password: *****

Your RDS DB instance must accept connections from your OpsWorks instances. [Learn more.](#)

[Cancel](#) [Register with Stack](#)

- 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.

Instance Identifier	Engine	Apps	Type	Multi-AZ	AZ	Actions
opsinstance1	mysql	Add app	t1.micro	No	us-east-1d	edit

[Show Unregistered RDS DB instances](#)

[+ Unregistered RDS DB instances](#)

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 AWS OpsWorks Stacks, see [Amazon RDS Service Layer \(p. 150\)](#).

Note

You can also register Amazon RDS instances through the **Layers** page. For more information, see [Amazon RDS Service Layer \(p. 150\)](#).

Attaching and Moving Resources

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 (p. 178). 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 \(p. 262\)](#)
- [Associating Elastic IP Addresses with an Instance \(p. 263\)](#)
- [Attaching Amazon RDS Instances to an App \(p. 265\)](#)

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.

Name	EC2 ID	Instance	Size (GiB)	Volume Type	AZ	Actions
Created for db-master1	vol-24ac9267	db-master1 ●	10	standard	us-east-1a	
PHP-LB-PIOPS	vol-0faf914c	assign to instance	100	io1 (2000)	us-east-1a	
PHP-LB-Standard	vol-53af9110	assign to instance	100	standard	us-east-1a	

Unregistered Volumes

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.

Volume PHP-LB-PIOPs

Name	PHP-LB-PIOPs
EC2 Volume ID	vol-0faf914c
Mount point	/vol/mountpoint
Availability Zone	us-east-1a
Instance	<input style="width: 150px; height: 20px; border: none; background-color: #f0f0f0; border-bottom: 2px solid #0072bc; color: #0072bc; font-weight: bold; font-size: 10px; padding: 2px 5px; margin-bottom: 5px;" type="button" value="PHP App Server"/>
Status	<input style="width: 150px; height: 20px; border: none; background-color: #f0f0f0; border-bottom: 2px solid #0072bc; color: #0072bc; font-weight: bold; font-size: 10px; padding: 2px 5px; margin-bottom: 5px;" type="button" value="php-app1"/>
Size	100 GiB
Device	—
Volume Type	io1
IOPS	2000
Snapshot ID	—
OpsWorks ID	a402f9f9-6814-403d-8b2d-dfee98950e9c

[Cancel](#) [Save](#)

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 \(p. 267\)](#), [register the volume \(p. 257\)](#) with the new stack, and [attach it \(p. 261\)](#) 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.

Resources

[Show Unregistered Elastic IPs](#)

The screenshot shows the AWS OpsWorks Resources page with the 'Elastic IPs' tab selected. A table lists an elastic IP address: 23.21.119.187. The 'Instance' column contains a link labeled 'associate with instance', which is circled in red. Below the table is a link to 'Unregistered Elastic IPs'. The main content area shows the details for the IP 23.21.119.187, including fields for Name (PHP-EIP), Region (us-east-1), Domain (standard), Stack (MyStack), and Instance (a dropdown menu). The dropdown menu lists 'PHP App Server' and three instances: 'php-app1', 'php-app2', and 'php-app3'. The 'php-app1' option is selected. A note below the dropdown says 'Select the instance the Elastic IP should be associated with.' At the bottom right are 'Cancel' and 'Save' buttons.

Note

If the Elastic IP address is currently associated with another online instance, AWS 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**.

Elastic IP PHP-EIP1

IP	54.221.232.99
Name	PHP-EIP1
Region	us-east-1
Domain	standard
Stack	MyStack change
Instance	php-app1 [current]

After you attach or move an address, AWS OpsWorks Stacks triggers a [Configure lifecycle event \(p. 253\)](#) 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.

The screenshot shows the AWS OpsWorks Resources page. At the top, there are tabs for Volumes, Elastic IPs, and RDS, with RDS selected. Below the tabs is a search bar. The main area displays a table with columns: Instance Identifier, Engine, Apps, Type, Multi-AZ, AZ, and Actions. One row is visible, showing 'opsinstance1' as the instance identifier, 'mysql' as the engine, 'Add app' under Apps, 't1.micro' as the type, 'No' for Multi-AZ, 'us-east-1d' for AZ, and an 'edit' link under Actions. Below the table is a link '+ Unregistered RDS DB instances'.

Instance Identifier	Engine	Apps	Type	Multi-AZ	AZ	Actions
opsinstance1	mysql	Add app	t1.micro	No	us-east-1d	edit

2. Use the **Add App** page to attach the Amazon RDS instance. For more information, see [Adding Apps \(p. 217\)](#).

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 \(p. 224\)](#).

Detaching Resources

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 \(p. 265\)](#)
- [Disassociating Elastic IP Addresses \(p. 266\)](#)
- [Detaching Amazon RDS Instances \(p. 266\)](#)

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**.

Volume PHP-LB-PIOPS

Volumes are the block level storage associated with your instance. [Learn more](#).

[Edit](#) [Unassign](#)

Settings

Name	PHP-LB-PIOPS
EC2 Volume ID	vol-0faf914c
Mount point	/vol/mountpoint
Availability Zone	us-east-1a
Instance	php-app1 ●
Status	available
Size	100 GiB
Device	/dev/sdi
Volume Type	io1
IOPS	2000
Snapshot ID	—
OpsWorks ID	a402f9f9-6814-403d-8b2d-dfee98950e9c

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**.

Elastic IP PHP-Vol2

[Edit](#) [Disassociate](#)

Elastic IPs are static IP addresses for your instance. [Learn more](#).

Settings

IP	23.21.119.187
Name	PHP-Vol2
Region	us-east-1
Domain	standard
Instance	php-app1 ●

After you disassociate an address, AWS OpsWorks Stacks triggers a [Configure lifecycle event \(p. 253\)](#) 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 \(p. 224\)](#).

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

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 AWS 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

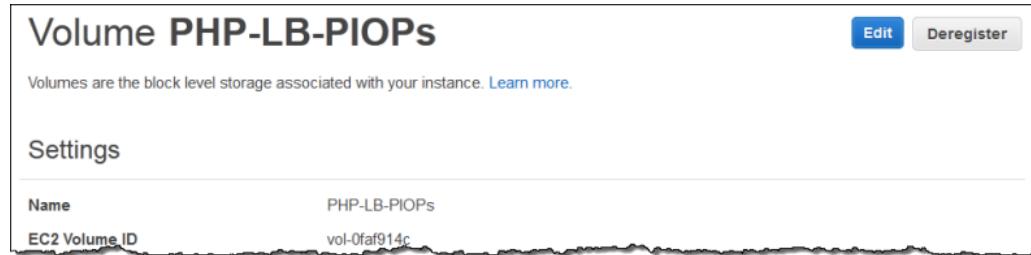
- [Deregistering Amazon EBS Volumes \(p. 267\)](#)
- [Deregistering Elastic IP Addresses \(p. 267\)](#)
- [Deregistering Amazon RDS Instances \(p. 268\)](#)

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 \(p. 265\)](#).
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 \(p. 266\)](#).
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**.

Elastic IP PHP-Vol2

[Edit](#) [Deregister](#)

Elastic IPs are static IP addresses for your instance. [Learn more](#).

Settings

IP	23.21.119.187
Name	PHP-Vol2
Region	us-east-1
Domain	standard
Instance	<i>associate with instance</i>

Tip

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 \(p. 261\)](#).

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 \(p. 265\)](#).
2. On the **Resources** page, click **RDS** and then instance's name.
3. On the instance's details page, click **Deregister**.

RDS opsinstance1

[Edit](#) [Deregister](#)

Amazon Relational Database Service (RDS) makes it easy to set up, operate, and scale a relational database in the cloud. [Learn more](#).

Settings	Configuration details
Instance Identifier opsinstance1	DB Engine mysql (5.6.13)

Monitoring

You can monitor your stacks in the following ways.

- AWS OpsWorks Stacks uses Amazon CloudWatch to provide thirteen custom metrics with detailed monitoring for each instance in the stack.
- AWS OpsWorks Stacks integrates with AWS CloudTrail to log every AWS 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 \(p. 269\)](#)
- [Logging AWS OpsWorks Stacks API Calls By Using AWS CloudTrail \(p. 276\)](#)
- [Using Amazon CloudWatch Logs with AWS OpsWorks Stacks \(p. 278\)](#)

Monitoring Stacks using Amazon CloudWatch

AWS OpsWorks Stacks uses Amazon CloudWatch (CloudWatch) to provide monitoring for stacks.

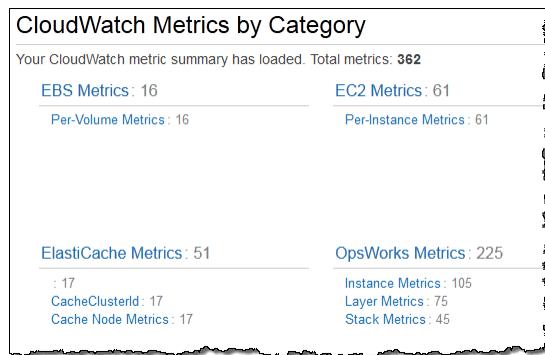
- For Linux stacks, AWS OpsWorks Stacks supports thirteen custom metrics to provide detailed monitoring for each instance in the stack and summarizes the data for your convenience on the **Monitoring** page.
- For Windows stacks, you can monitor standard Amazon EC2 metrics for your instances with the [CloudWatch console](#).

The **Monitoring** page does not display Windows metrics.

The **Monitoring** page displays metrics for an entire stack, a layer, or an instance. AWS 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

AWS 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 AWS 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 \(p. 270\)](#)
- [Dimensions for AWS OpsWorks Stacks Metrics \(p. 273\)](#)
- [Stack Metrics \(p. 273\)](#)
- [Layer Metrics \(p. 274\)](#)
- [Instance Metrics \(p. 275\)](#)

AWS OpsWorks Stacks Metrics

AWS OpsWorks Stacks sends the following metrics to CloudWatch every five minutes.

Metric	Description
cpu_idle	<p>The percentage of time that the CPU is idle.</p> <p>Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId.</p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples.</p> <p>Unit: Percent</p>
cpu_nice	<p>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).</p> <p>Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId.</p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples.</p> <p>Unit: Percent</p>
cpu_steal	<p>As AWS allocates hypervisor CPU resources among increasing numbers of instances, virtualization load rises, and can affect how often the hypervisor can perform requested work on an instance. <code>cpu_steal</code> measures the percentage of time that an instance is waiting for the hypervisor to allocate physical CPU resources.</p> <p>Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId.</p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples.</p> <p>Unit: Percent</p>
cpu_system	<p>The percentage of time that the CPU is handling system operations.</p>

Metric	Description
	<p>Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId.</p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples.</p> <p>Unit: Percent</p>
cpu_user	<p>The percentage of time that the CPU is handling user operations.</p> <p>Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId.</p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples.</p> <p>Unit: Percent</p>
cpu_waitio	<p>The percentage of time that the CPU is waiting for input/output operations.</p> <p>Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId.</p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples.</p> <p>Unit: Percent</p>

Metric	Description
memory_buffers	<p>The amount of buffered memory.</p> <p>Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId.</p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples.</p> <p>Unit: Kilobytes</p>
memory_cached	<p>The amount of cached memory.</p> <p>Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId.</p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples.</p> <p>Unit: Kilobytes</p>

Metric	Description
memory_free	<p>The amount of free memory.</p> <p>Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or Instanceld.</p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples.</p> <p>Unit: Kilobytes</p>
memory_swap	<p>The amount of swap space.</p> <p>Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or Instanceld.</p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples.</p> <p>Unit: Kilobytes</p>
memory_total	<p>The total amount of memory.</p> <p>Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or Instanceld.</p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples.</p> <p>Unit: Kilobytes</p>
memory_used	<p>The amount of memory in use.</p> <p>Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or Instanceld.</p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples.</p> <p>Unit: Kilobytes</p>

Metric	Description
load_1	<p>The load averaged over a one-minute window.</p> <p>Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or Instanceld.</p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples.</p> <p>Unit: Unix load units></p>

Metric	Description
load_5	<p>The load averaged over a five-minute window.</p> <p>Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId.</p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples.</p> <p>Unit: Unix load units></p>
load_15	<p>The load averaged over a 15-minute window.</p> <p>Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId.</p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples.</p> <p>Unit: Unix load units></p>

Metric	Description
procs	<p>The number of active processes.</p> <p>Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId.</p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples.</p> <p>Unit: Count</p>

Dimensions for AWS OpsWorks Stacks Metrics

AWS OpsWorks Stacks metrics use the AWS 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 AWS 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.

Monitoring Layers

refreshing in 69 sec 1 hour ▾



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:

- AWS 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.

Layer PHP App Server

refreshing in 111 sec 1 hour ▾



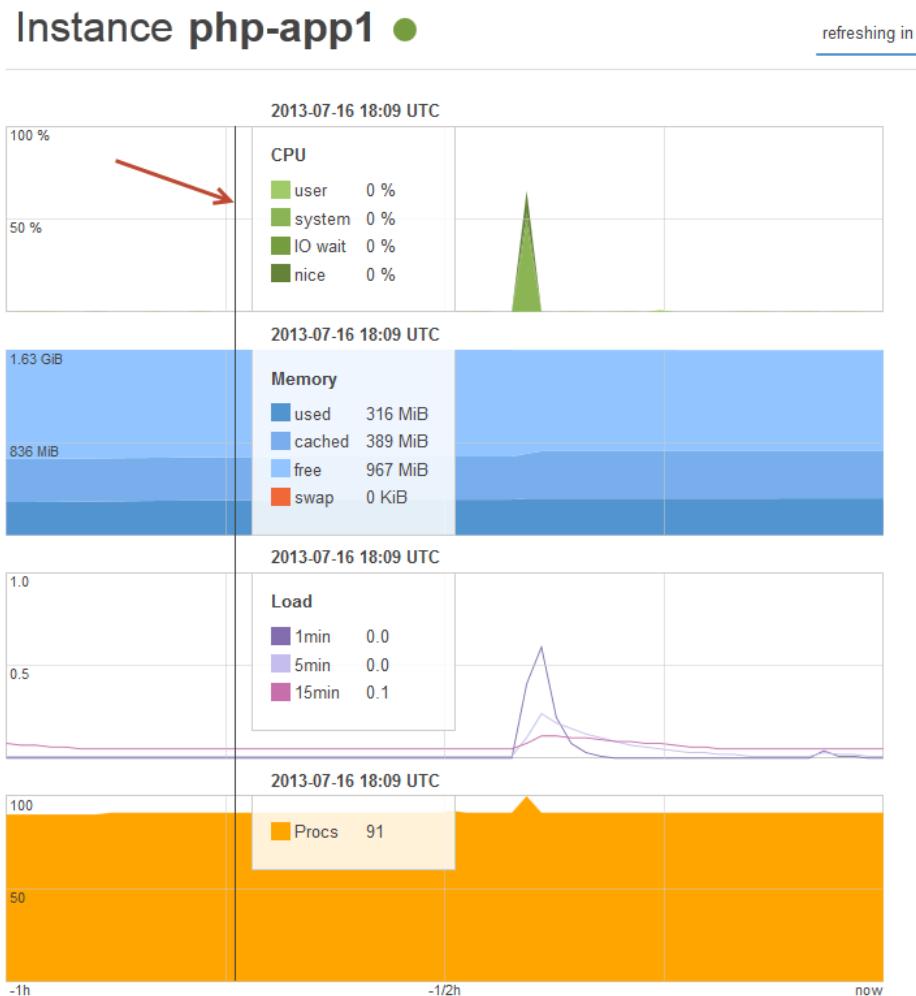
The metric types are the same as for the stack metrics, and for each type, you can use the list at the top of the graph to select the particular metric that you want to view.

Tip

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.



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.

Tip

You can also display instance metrics by going to the instance's details page and choosing **Monitoring** at the upper right.

Logging AWS OpsWorks Stacks API Calls By Using AWS CloudTrail

AWS OpsWorks Stacks is integrated with CloudTrail, a service that captures API calls made by or on behalf of AWS OpsWorks Stacks in your AWS account and delivers the log files to an Amazon S3 bucket that you specify. CloudTrail captures API calls from the AWS OpsWorks Stacks console or from the AWS OpsWorks Stacks API. Using the information collected by CloudTrail, you can determine what request was made to AWS OpsWorks Stacks, the source IP address from which the request was made, who made the request, when it was made, and so on. To learn more about CloudTrail, including how to configure and enable it, see the [AWS CloudTrail User Guide](#).

AWS OpsWorks Stacks Information in CloudTrail

When CloudTrail logging is enabled in your AWS account, API calls made to AWS OpsWorks Stacks actions are tracked in log files. AWS OpsWorks Stacks records are written together with other AWS service records in a log file. CloudTrail determines when to create and write to a new file based on a time period and file size.

Important

Although your stacks can be in any AWS region, all calls made by AWS OpsWorks Stacks on your behalf originate in the US East (N. Virginia) Region. To log API calls made on your behalf to other regions, you must enable CloudTrail in those regions.

All of the AWS OpsWorks Stacks actions are logged and are documented in the [AWS OpsWorks Stacks API Reference](#). For example, each call to [CreateLayer](#), [DescribeInstances](#) or [StartInstance](#) generates a corresponding entry in the CloudTrail log files.

Every log entry contains information about who generated the request. The user identity information in the log helps you determine whether the request was made with root or IAM user credentials, with temporary security credentials for a role or federated user, or by another AWS service. For more information, see the [userIdentity](#) field in the [CloudTrail Event Reference](#).

You can store your log files in your bucket for as long as you want, but you can also define Amazon S3 lifecycle rules to archive or delete log files automatically. By default, your log files are encrypted by using Amazon S3 server-side encryption (SSE).

You can choose to have CloudTrail publish Amazon SNS notifications when new log files are delivered if you want to take quick action upon log file delivery. For more information, see [Configuring Amazon SNS Notifications](#).

You can also aggregate AWS OpsWorks Stacks log files from multiple AWS regions and multiple AWS accounts into a single Amazon S3 bucket. For more information, see [Aggregating CloudTrail Log Files to a Single Amazon S3 Bucket](#).

Understanding AWS OpsWorks Stacks Log File Entries

CloudTrail log files can contain one or more log entries where each entry is made up of multiple JSON-formatted events. A log entry represents a single request from any source and includes information about the requested action, any parameters, the date and time of the action, and so on. The log entries are not guaranteed to be in any particular order. That is, they are not an ordered stack trace of the public API calls.

The following example shows a CloudTrail log entry that demonstrates the [CreateLayer](#) and [DescribeInstances](#) actions.

Note

Potentially sensitive information, including database passwords and custom JSON attributes, is redacted and does not appear in the CloudTrail logs. The information is instead represented by `**redacted**`.

```
{  
  "Records": [  
    {  
      "awsRegion": "us-east-1",  
      "eventID": "342cdlec-8214-4a0f-a68f-8e6352feb5af",  
      "eventName": "CreateLayer",  
      "eventSource": "opsworks.amazonaws.com",  
      "eventTime": "2014-05-28T16:05:29Z",  
      "eventVersion": "1.01",  
      "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-east-1",  
      "eventID": "a860d8f8-cleb-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-a990-3fbelefa25f6"  
        ]  
      },  
      "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"  
      }  
    }  
  ]  
}
```

Using Amazon CloudWatch Logs with AWS OpsWorks Stacks

To simplify the process of monitoring logs on multiple instances, AWS OpsWorks Stacks supports Amazon CloudWatch Logs. You enable CloudWatch Logs at the layer level in AWS 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 AWS OpsWorks Stacks, the AWS OpsWorks Stacks agent sends the logs to CloudWatch Logs. For more information about CloudWatch Logs, see [Getting Started with CloudWatch Logs](#).

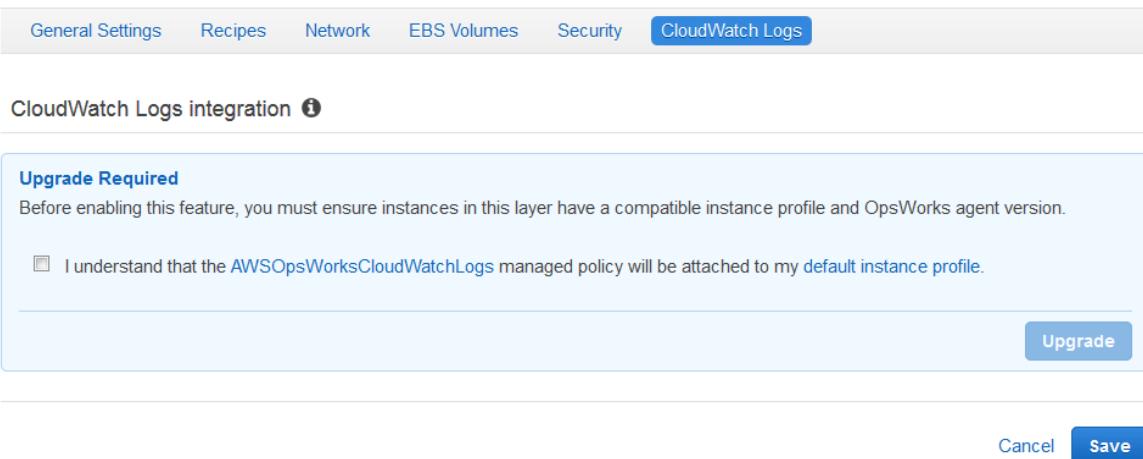
Prerequisites

Before you can enable CloudWatch Logs, your instances must be running version 3444 or later of the AWS 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. AWS OpsWorks Stacks prompts you to let it upgrade the agent version and instance profile when you open the CloudWatch Logs tab on the **Layer** page.

If you are using a custom instance profile (one that AWS OpsWorks Stacks did not provide when you created the stack), AWS OpsWorks Stacks cannot automatically upgrade the instance profile. You must manually attach the **AWSOpsWorksCloudWatchLogs** policy to your profile by using IAM. For information, see [Attaching Managed Policies](#) in the *IAM User Guide*.

The following screenshot shows the upgrade prompt.

Layer PHP App Server



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.

OpsWorks Agent Upgrade in Progress
1 instances in this layer are upgrading their OpsWorks agent to a version compatible with CloudWatch Logs. If this upgrade has not completed within 15 minutes, visit [this page](#) for details on how to resolve the issue.

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**.

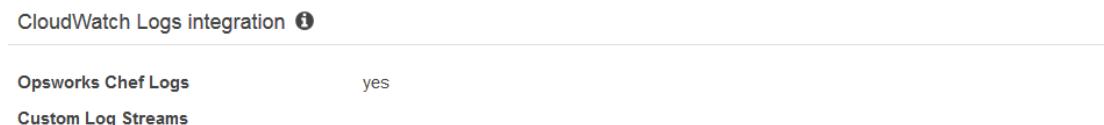
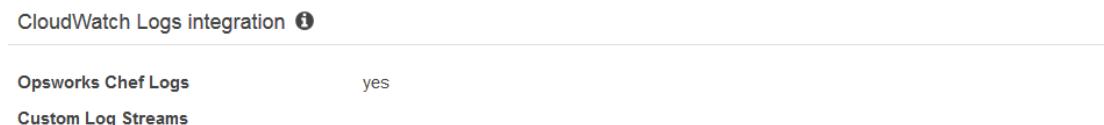
Layer PHP App Server



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, AWS 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**.

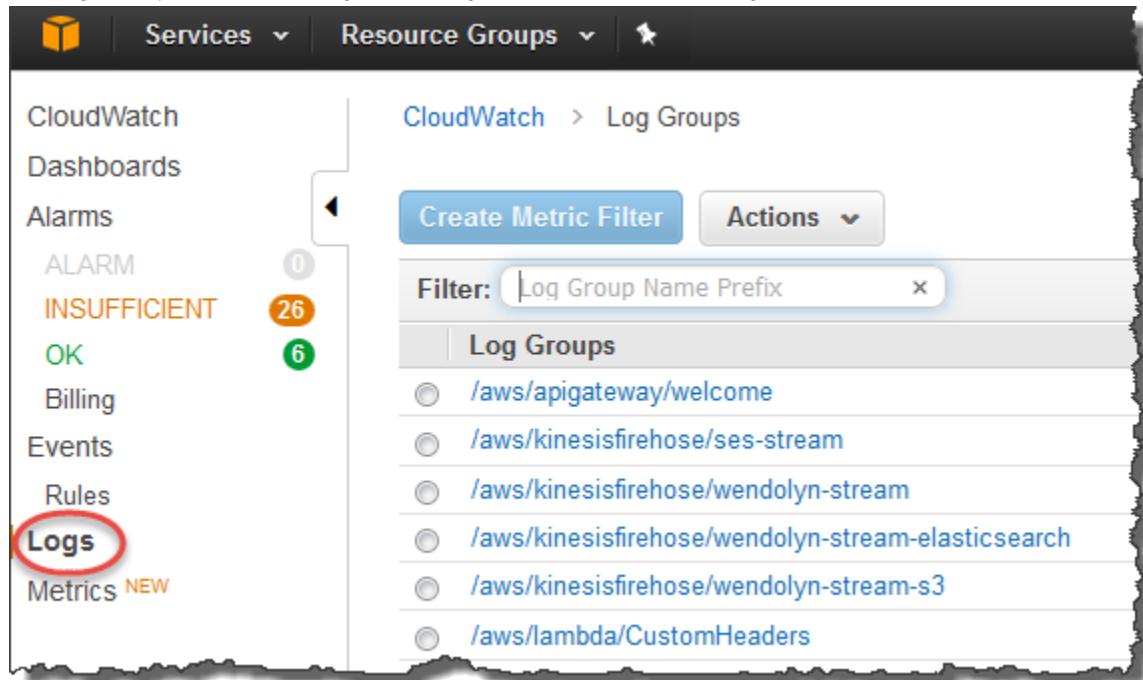
Deleting Streamed Logs from CloudWatch Logs

After you turn off CloudWatch Logs streaming from AWS 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.



AWS OpsWorks creates default log groups and log streams automatically. Log groups for AWS 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:

`full_path_name_of_file`. 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, AWS OpsWorks Stacks automatically runs your recipes after the instance finishes booting, enabling CloudWatch Logs. For more information about manually configuring CloudWatch Logs streams for Windows-based instances, see the following.

- [Configuring a Windows Instance Using the EC2Config Service](#)
- [Sending Logs, Events, and Performance Counters to Amazon CloudWatch](#)
- [CloudWatch Update – Enhanced Support for Windows Log Files](#) (blog post)

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.

Security and Permissions

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 \(IAM\)](#). AWS OpsWorks Stacks integrates with IAM to let you control the following:

- How individual users can interact with AWS 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 AWS OpsWorks Stacks can act on your behalf to access stack resources such as Amazon EC2 instances and Amazon S3 buckets.

AWS OpsWorks Stacks provides a service role that grants permissions for these tasks.

- How apps that run on Amazon EC2 instances controlled by AWS 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 IAM 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 \(p. 306\)](#).
- 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 AWS OpsWorks Stacks default security groups. For more information, see [Using Security Groups \(p. 307\)](#).

Topics

- [Managing AWS OpsWorks Stacks User Permissions \(p. 282\)](#)
- [Signing in as an IAM User \(p. 297\)](#)
- [Allowing AWS OpsWorks Stacks to Act on Your Behalf \(p. 298\)](#)
- [Specifying Permissions for Apps Running on EC2 instances \(p. 300\)](#)
- [Managing SSH Access \(p. 302\)](#)
- [Managing Linux Security Updates \(p. 306\)](#)
- [Using Security Groups \(p. 307\)](#)

Managing AWS OpsWorks Stacks User Permissions

One way to handle AWS OpsWorks Stacks permissions is to attach an IAM `AWSOpsWorksFullAccess` policy to every IAM user. However, this policy allows a user to perform every AWS OpsWorks Stacks action on every stack. It is often desirable instead to restrict AWS OpsWorks Stacks users to a specified set of actions or set of stack resources. You can control AWS OpsWorks Stacks user permissions in two ways: By using the AWS OpsWorks Stacks **Permissions** page and by attaching 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 \(p. 302\)](#).

You can use the [IAM console](#), CLI, or API to attach policies to your users that grant explicit permissions for the various AWS 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 groups](#), which grant permissions to groups of users, or define [roles that can be associated with federated users](#).
- An IAM policy is the only way to grant permissions for certain key AWS 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 AWS 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 IAM users.

The two approaches are not mutually exclusive and it is sometimes useful to combine them; AWS 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 AWS 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 attach an IAM policy that denies permissions to add or remove layers. For more information, see [Overview of AWS IAM Permissions](#).

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 attach `AWSOpsWorksFullAccess` policies to one or more administrative users.
2. Create an IAM user for each nonadministrative user with a policy that grants no AWS OpsWorks Stacks permissions.

If a user requires access only to AWS OpsWorks Stacks, you might not need to attach a policy at all. You can instead manage their permissions with the AWS OpsWorks Stacks **Permissions** page.

3. Use the AWS OpsWorks Stacks **Users** page to import the nonadministrative users into AWS 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 attaching an appropriately configured IAM policy.

For more recommendations on managing users, see [Root Device Storage \(p. 105\)](#).

Important

As a best practice, don't use root (account owner) credentials to perform everyday work in AWS. Instead, create an IAM administrators group with appropriate permissions. Then create IAM users for the people in your organization who need to perform administrative tasks (including for yourself), and add those users to the administrative group. For more information, see [IAM Best Practices](#) in the *Using IAM* guide.

Topics

- [Managing AWS OpsWorks Stacks Users \(p. 284\)](#)
- [Granting AWS OpsWorks Stacks Users Per-Stack Permissions \(p. 289\)](#)
- [Managing AWS OpsWorks Stacks Permissions by Attaching an IAM Policy \(p. 291\)](#)
- [Example Policies \(p. 292\)](#)
- [AWS OpsWorks Stacks Permissions Levels \(p. 296\)](#)

Managing AWS OpsWorks Stacks Users

Before you can import users into AWS OpsWorks Stacks and grant them permissions, you must first have created an IAM user for each individual. To create IAM users, start by signing in to AWS as an IAM user that has been granted the permissions defined in the `IAMFullAccess` policy. You then use the IAM console to [create IAM users \(p. 286\)](#) for everyone who needs to access AWS OpsWorks Stacks. You can then import those users into AWS OpsWorks Stacks and grant user permissions as follows:

Regular AWS OpsWorks Stacks Users

Regular users don't require an attached policy. If they do have one, it typically does not include any AWS OpsWorks Stacks permissions. Instead, use the AWS 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 attaching an IAM policy. For example, you might want to use the AWS 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 attach 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 [Attaching an IAM Policy \(p. 291\)](#).

AWS OpsWorks Stacks Administrative Users

Administrative users are the account owner or an IAM user with the permissions that are defined by the [AWSOpsWorksFullAccess policy \(p. 293\)](#). 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 AWS OpsWorks Stacks
- Creating and cloning stacks

For the complete policy, see [Example Policies \(p. 292\)](#). For a detailed list of permissions that can be granted to users only by attaching an IAM policy, see [Permissions Levels \(p. 296\)](#).

Topics

- [Users and Regions \(p. 285\)](#)
- [Creating an AWS OpsWorks Stacks Administrative User \(p. 285\)](#)
- [Creating IAM Users for AWS OpsWorks Stacks \(p. 286\)](#)
- [Importing Users into AWS OpsWorks Stacks \(p. 287\)](#)
- [Editing AWS OpsWorks Stacks User Settings \(p. 288\)](#)

Users and Regions

AWS OpsWorks Stacks user accounts are available within the regional endpoint in which they were created. You can create users in any of the following regions.

- US East (N. Virginia) Region
- US East (Ohio) Region
- US West (Oregon) Region
- US West (N. California) Region
- Asia Pacific (Mumbai) Region
- Asia Pacific (Singapore) Region
- Asia Pacific (Sydney) Region
- Asia Pacific (Tokyo) Region
- Asia Pacific (Seoul) Region
- EU (Frankfurt) Region
- EU (Ireland) Region
- EU (London) Region
- South America (São Paulo) Region

When you import IAM users to AWS OpsWorks Stacks, you import them to one of the regional endpoints; if you want an IAM user to be available in more than one region, you must import the user to that region. You can also import AWS 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 \(p. 287\)](#).

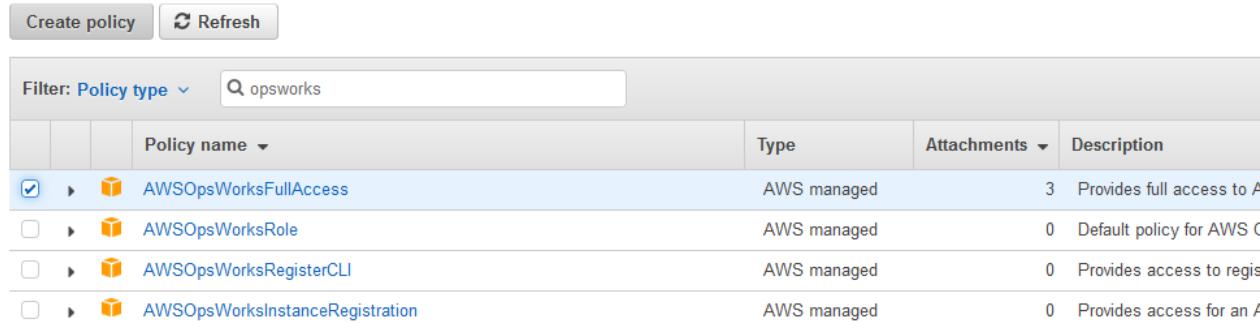
Creating an AWS OpsWorks Stacks Administrative User

You can create an AWS OpsWorks Stacks administrative user by attaching an IAM policy to a user that grants AWS OpsWorks Stacks Full Access permissions. This procedure assumes that you will create a new IAM user for this purpose. For existing users, simply add the policy to the user as described in Step 4.

To create the IAM User

1. Sign in to the IAM console at <https://console.aws.amazon.com/iam/>.
2. On the [IAM console](#), choose **Users** in the navigation pane, and then choose **Add user**.
3. Type a user name. In the **Select AWS access type** area, select **Programmatic access**, and then choose **Next: Permissions**.
4. On the **Set permissions** page, choose **Attach existing policies directly**.
5. Enter **opsWorks** in the **Policy type** filter box to display the AWS OpsWorks policies, select **AWSOpsWorksFullAccess**, and then choose **Next: review**. This policy grants your user rights to perform all tasks in AWS OpsWorks.

Attach one or more existing policies directly to the user or create a new policy. [Learn more](#)



The screenshot shows the AWS IAM Policies page. At the top, there are buttons for 'Create policy' and 'Refresh'. Below that is a search bar with the text 'opsworks'. A filter dropdown says 'Filter: Policy type'. The main area is a table with columns: Policy name, Type, Attachments, and Description. There are four rows:

Policy name	Type	Attachments	Description
<input checked="" type="checkbox"/> AWSOpsWorksFullAccess	AWS managed	3	Provides full access to A
<input type="checkbox"/> AWSOpsWorksRole	AWS managed	0	Default policy for AWS C
<input type="checkbox"/> AWSOpsWorksRegisterCLI	AWS managed	0	Provides access to regis
<input type="checkbox"/> AWSOpsWorksInstanceRegistration	AWS managed	0	Provides access for an A

6. On the **Review** page, choose **Create user**.
7. Choose **Download .csv**, save the credentials file to a convenient location on your system, and then choose **Close**.

Note

The AWSOpsWorksFullAccess policy allows users to create and manage AWS 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 \(p. 293\)](#). When this user creates the first stack, AWS 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 AWS OpsWorks Stacks to Act on Your Behalf \(p. 298\)](#).

8. Attach 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 [Attaching an IAM Policy \(p. 291\)](#).

If you have multiple administrative users, instead of setting permissions separately for each user, you can attach an AWSOpsWorksFullAccess policy to an [IAM group](#) and add the users to that group.

- To create a new group, choose **Groups** in the navigation pane, and then choose **Create New Group**. In the **Create New Group Wizard**, name your group and specify the **AWSOpsWorksFullAccess** policy. You can also specify the **AdministratorAccess** policy, which includes the **AWSOpsWorksFullAccess** permissions.
- To add full AWS OpsWorks Stacks permissions to an existing group, choose **Groups** in the navigation pane, select the group, and then choose the **Permissions** tab. Choose **Attach Policy** or **Attach Another Policy**. In the **Manage Group Permissions** wizard, select the **AWSOpsWorksFullAccess** policy.

Creating IAM Users for AWS OpsWorks Stacks

Before you can import IAM users into AWS OpsWorks Stacks, you need to create them. You can do this using the [IAM console](#), command line, or API. For full instructions, see [Adding an IAM User in Your AWS Account](#).

Note that unlike [administrative users \(p. 285\)](#), you don't need to attach a policy to define permissions. You can set permissions after [importing the users into AWS OpsWorks Stacks \(p. 287\)](#), as explained in [Managing User Permissions \(p. 282\)](#).

For more information on creating IAM users and groups, see [IAM Getting Started](#).

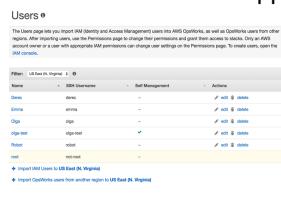
Importing Users into AWS OpsWorks Stacks

Administrative users can import IAM users into AWS OpsWorks Stacks; they can also import AWS OpsWorks Stacks users from one regional endpoint to another. When you import IAM users to AWS OpsWorks Stacks, you import them to one of the AWS OpsWorks Stacks regional endpoints. If you want an IAM 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 AWS 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 IAM users.

To import IAM users into AWS OpsWorks Stacks

1. Sign in to AWS 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 user accounts 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

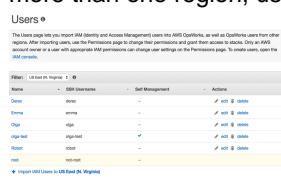
After you have imported an IAM user into AWS OpsWorks Stacks, if you use the IAM console or API to delete the user from your account, the user does not automatically lose SSH access that you have granted through AWS OpsWorks Stacks. You must also delete the user from AWS OpsWorks Stacks by opening the **Users** page, and choosing **delete** in the user's **Actions** column.

To import AWS OpsWorks Stacks users from one region to another

AWS OpsWorks Stacks user accounts are available within the regional endpoint in which they were created. You can create users in the regions shown in [Users and Regions \(p. 285\)](#).

You can import AWS 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 AWS 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 AWS 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 AWS OpsWorks Stacks users from another region to <current region>**.

Import OpsWorks users from another region to **US East (N. Virginia)**

OpsWorks users are created and stored regionally. You can import users from another region to this region. Duplicate users are replaced by users that you import.

Step 1.

Select the region from which you want to import users.

Asia Pacific (Mumbai) 

Step 2.

Select the user(s) that you want to import to this region, and then choose **Import to this region**.

Select all users

Kroos

Emma

4. Select the region from which you want to import AWS OpsWorks Stacks users.
5. Select one or more users to import, or select all users, and then choose **Import to this region**. Wait for AWS OpsWorks Stacks to display the imported users in the **Users** list.

Editing AWS OpsWorks Stacks User Settings

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 attaching an IAM policy to the user 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 an IAM User's Public SSH Key \(p. 305\)](#).

AWS 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 \(p. 212\)](#). 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 \(p. 289\)](#).

User windows-test-user

Name	windows-test-user
ARN	arn:aws:iam:645732743964:user/windows-test-user
Self Management	<input type="checkbox"/> No
SSH Username	windows-test-user
Public SSH key	<input type="text"/>

The user will be created on **linux-based instances** if they have a **Public SSH Key**.
 Clearing the public key will cause all SSH logins of the user to be deleted on **linux-based** instances.
 Running processes will be terminated.

Permissions

Stack	Permission level						Instance access	
	Deny	IAM Policies Only	Show	Deploy	Manage	SSH / RDP	sudo / admin	
CLITest	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Chef9Test	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>	
EC2Register	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
JavaStack	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Granting AWS OpsWorks Stacks Users Per-Stack Permissions

The simplest way to manage AWS 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 AWS OpsWorks Stacks. For information on how to create and import users, see [Managing Users \(p. 284\)](#).

The default permission level is IAM Policies Only, which grants users only those permissions that are in their attached 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 \(p. 290\)](#)
- [Viewing your Permissions \(p. 291\)](#)

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 attached IAM policy, AWS OpsWorks Stacks evaluates both sets of permissions. For an example see [Example Policies \(p. 292\)](#).
 - 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.

User Name	Permission level					Instance access	
	Deny	IAM Policies Only	Show	Deploy	Manage	SSH / RDP	sudo / admin
admin_user	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
cli-user-test	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>
development	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>

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 [Permissions Levels \(p. 296\)](#).

Deny

The user cannot perform any AWS OpsWorks Stacks actions on the stack, even if they have an attached IAM policy that grants AWS 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 attached IAM policy. If a user has no IAM policy, or their policy has no explicit AWS OpsWorks Stacks permissions, they cannot access the stack. Administrative users are typically assigned this level because their attached 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 attached IAM policy. For an example, see [Manage Permissions \(p. 294\)](#).

If the user also has an attached policy, AWS OpsWorks Stacks evaluates both sets of permissions. This allows you to assign a permission level to a user and then attach a policy to the user to restrict or augment the level's allowed actions. For example, you could attach 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 \(p. 292\)](#).

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 an attached 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 `AWSOpsWorksFullAccess` policies. For more information, see [Overview of AWS IAM Permissions](#).

Viewing your Permissions

If [self-management \(p. 288\)](#) 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 attached policy grants permissions for the `DescribeMyUserProfile` and `UpdateMyUserProfile` actions.

Managing AWS OpsWorks Stacks Permissions by Attaching an IAM Policy

You can specify a user's AWS 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 [Permissions Levels \(p. 296\)](#).

You can also use an attached policy to customize permission levels that were granted through the **Permissions** page. This section provides a brief summary of how to attach an IAM policy to a user to specify AWS OpsWorks Stacks permissions. For more information, see [Permissions and Policies](#).

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 AWS 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 AWS OpsWorks Stacks actions, see the [AWS OpsWorks Stacks API Reference](#).

Effect

Whether the specified actions are allowed or denied.

Resource

The AWS resources that the permission affects. AWS 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`.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": "opsworks:StopInstance",  
            "Effect": "Deny",  
            "Resource": "arn:aws:opsworks:*:*:stack/2f18b4cb-4de5-4429-a149-ff7da9f0d8ee/"  
        }  
    ]  
}
```

To attach a policy to an IAM user

1. Open the [IAM console](#).
2. To create a new managed policy for this user, choose **Policies** in the navigation pane, and then in the list of policies, choose **Create Policy** to create a new customer-managed policy with the appropriate permissions. For more information about how to create a customer-managed policy, see [Customer Managed Policies](#).
3. When you have created the customer-managed policy, choose **Users** in the IAM console navigation pane, select the user name, and then choose **Add permissions**.
4. On the **Add permissions** page, choose **Attach existing policies directly**.
5. In the **Policy Type** drop-down list, select **Customer managed**, select the policy that you want to attach, and then choose **Next: Review**.
6. On the **Review** page, choose **Add permissions**.

For more information about how to create or modify IAM policies, see [Permissions and Policies](#). For some examples of AWS OpsWorks Stacks policies, see [Example Policies \(p. 292\)](#).

Example Policies

This section shows some examples of IAM policies that can be attached to AWS OpsWorks Stacks users.

- [Administrative Permissions \(p. 293\)](#) shows two policies that can be used to grant permissions to administrative users.
- [Manage Permissions \(p. 294\)](#) and [Deploy Permissions \(p. 295\)](#) show examples of policies that can be attached to a user to augment or restrict the Manage and Deploy permissions levels.

AWS OpsWorks Stacks determines the user's permissions by evaluating the permissions granted by attached IAM policies as well as the permissions granted by the **Permissions** page. For more

information, see [Overview of AWS IAM Permissions](#). For more information on the **Permissions** page permissions, see [Permissions Levels \(p. 296\)](#).

Administrative Permissions

The following is the AWSOpsWorksFullAccess policy, which can be attached to a user to grant them permissions to perform all AWS OpsWorks Stacks actions. The IAM permissions are required, among other things, to allow an administrative user to import users.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "opsworks:*",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeKeyPairs",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "elasticloadbalancing:DescribeInstanceHealth",
                "elasticloadbalancing:DescribeLoadBalancers",
                "iam:GetRolePolicy",
                "iam>ListRoles",
                "iam>ListInstanceProfiles",
                "iam:PassRole",
                "iam>ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

You must create an [IAM role](#) that allows AWS 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 AWS OpsWorks Stacks create the role for you. You can then use that role for all subsequent stacks. For more information, see [Allowing AWS OpsWorks Stacks to Act on Your Behalf \(p. 298\)](#).

The administrative user who creates the first stack must have permissions for some IAM actions that are not included in the AWSOpsWorksFullAccess policy, as follows:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "opsworks:*",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeKeyPairs",
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
```

```

        "ec2:DescribeVpcs",
        "elasticloadbalancing:DescribeInstanceHealth",
        "elasticloadbalancing:DescribeLoadBalancers",
        "iam:GetRolePolicy",
        "iam>ListRoles",
        "iam>ListInstanceProfiles",
        "iam:PassRole",
        "iam>ListUsers",
        "iam:PutRolePolicy",
        "iam>AddRoleToInstanceProfile",
        "iam>CreateInstanceProfile",
        "iam>CreateRole"
    ],
    "Resource": "*"
}
]
}

```

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 attach 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 attaching the following IAM policy.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "opsworks:CreateLayer",
                "opsworks:DeleteLayer"
            ],
            "Resource": "*"
        }
    ]
}

```

Allow a **Manage** user to create or clone stacks

The Manage permissions level doesn't allow users to create or clone stacks. You can augment the **Manage** permissions to allow a user to create or clone stacks by attaching the following IAM policy.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "opsworks:CreateStack",
                "opsworks:CloneStack"
            ],
            "Resource": "*"
        }
    ]
}

```

```
}
```

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 \(p. 257\)](#) with the stack. You can restrict the **Manage** permissions to allow the user to perform all **Manage** actions except registering resources by attaching the following policy.

```
{
    "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 AWS OpsWorks Stacks. You can augment the **Manage** permissions to allow a user to import and delete users by attaching the following IAM policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRolePolicy",
                "iam>ListRoles",
                "iam>ListInstanceProfiles",
                "iam>ListUsers",
                "iam:PassRole",
                "iam>ListInstanceProfiles",
                "opsworks:DescribeUserProfiles",
                "opsworks>CreateUserProfile",
                "opsworks>DeleteUserProfile"
            ],
            "Resource": "*"
        }
    ]
}
```

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 attaching the following IAM policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
{  
    "Effect": "Allow",  
    "Action": [  
        "opsworks:CreateApp",  
        "opsworks:DeleteApp"  
    ],  
    "Resource": "*"  
}  
]  
}
```

AWS OpsWorks Stacks Permissions Levels

This section lists the actions that are allowed by the **Show**, **Deploy**, and **Manage** permissions levels on the AWS OpsWorks Stacks **Permissions** page. It also includes a list of actions that you can grant permissions only by attaching an IAM policy to the user.

Show

The **Show** level allows `DescribeXYZ` 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 \(p. 288\)](#).

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 attaching an appropriate IAM policy to the user. For some examples, see [Example Policies \(p. 292\)](#).

```
CloneStack
CreateStack
CreateUserProfile
DeleteUserProfile
DescribeUserProfiles
UpdateUserProfile
```

Signing in as an IAM User

Each account has a URL called an account alias, which IAM users use to sign in to the account. The URL is on your IAM console's **Dashboard (Getting Started)** page under **Account Alias**.

When creating a new IAM user, an administrative user specifies or has the IAM service generate the following:

- A user name
- An optional password

You can specify a password or have IAM generate one for you.

- An optional Access Key ID and Secret Access Key

IAM generates a new key pair for each user. The keys are required for users who need to access AWS OpsWorks Stacks using the API or CLI. They are not required for console-only users.

The administrative user then provides this information and the account alias to each new user, for example, in an e-mail.

To sign in to AWS OpsWorks Stacks as an IAM user

1. In your browser, navigate to the account alias URL.
2. Enter your account name, user name, and password and click **Sign in**.

3. In the navigation bar, click **Services** and select **OpsWorks**.

Allowing AWS OpsWorks Stacks to Act on Your Behalf

AWS OpsWorks Stacks needs to interact with a variety of AWS services on your behalf. For example, AWS OpsWorks Stacks interacts with Amazon EC2 to create instances and with Amazon CloudWatch to obtain monitoring statistics. When you create a stack, you specify an IAM role, usually called a service role, that grants AWS OpsWorks Stacks the appropriate permissions.

The screenshot shows the configuration options for creating a new AWS OpsWorks stack. The fields include:

- Name: ShortStack
- Region: US East (N. Virginia)
- VPC: No VPC
- Default Availability Zone: us-east-1a
- Default operating system:
 - Amazon Linux
 - Ubuntu 12.04 LTS
 - Use custom AMI
- Default root device type:
 - Instance store
 - EBS backed
- IAM role: aws-opsworks-service-role (highlighted with a red circle)

When you specify a new stack's service role, you can do one of the following:

- Have AWS OpsWorks Stacks create a new service role with a standard set of permissions.

The role will be named something like `aws-opsworks-service-role`.

- 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 AWS 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 AWS OpsWorks Stacks to create a new IAM service role and allow you to import users, [as described earlier \(p. 287\)](#). 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 AWS OpsWorks Stacks and your Amazon EC2 instances.

If you create a custom service role, you must ensure that it grants all the permissions that AWS OpsWorks Stacks needs to manage your stack.

A service role also has a trust relationship. Service roles created by AWS OpsWorks Stacks have the following trust relationship.

```
{  
    "Version": "2008-10-17",  
    "Statement": [  
        {  
            "Sid": "",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "opsworks.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

The service role must have this trust relationship for AWS 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 as follows:

- If you are using the **Create Role** wizard in the [IAM console](#), specify the **AWS Opsworks** 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": {  
    "OpsWorksServiceRole": {  
        "Type": "AWS::IAM::Role",  
        "Properties": {  
            "AssumeRolePolicyDocument": {  
                "Statement": [ {  
                    "Effect": "Allow",  
                    "Principal": {  
                        "Service": [ "opsworks.amazonaws.com" ]  
                    },  
                    "Action": [ "sts:AssumeRole" ]  
                } ]  
            },  
            "Path": "/",  
            "Policies": [ {  
                "PolicyName": "opsworks-service",  
                "PolicyDocument": {  
                    ...  
                }  
            } ]  
        }  
    }  
}
```

Specifying Permissions for Apps Running on EC2 instances

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 AWS OpsWorks Stacks stack \(p. 120\)](#).



You can also specify a profile for a layer's instances by [editing the layer configuration \(p. 140\)](#).

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:

- Have AWS OpsWorks Stacks create a new profile when you create a stack.

The profile will be named something like `aws-opsworks-ec2-role` and will have a trust relationship but no policy.

- 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. Instance profiles created by AWS OpsWorks Stacks have the following trust relationship.

```
{  
    "Version": "2008-10-17",  
    "Statement": [  
        {  
            "Sid": "",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "ec2.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

The instance profile must have this trust relationship for AWS 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 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"
                } ]
            }
        }
    }
}
```

If you create your own instance profile, you can attach an appropriate policy to the profile's role at that time. If you have AWS OpsWorks Stacks create an instance profile for you when you create the stack, it does not have an attached policy and grants no permissions. 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 any Amazon S3 bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [ {
        "Effect": "Allow",
        "Action": "s3:*",
        "Resource": "*"
    } ]
}
```

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 AWS OpsWorks Stacks API from an EC2 instance, the policy must allow the `iam:PassRole` action in addition to the appropriate actions for AWS OpsWorks Stacks and other AWS services. The `iam:PassRole` permission allows AWS OpsWorks Stacks to assume

the service role on your behalf. For more information about the AWS OpsWorks Stacks API, see [AWS OpsWorks API Reference](#).

The following is an example of an IAM policy that allows you to call any AWS OpsWorks Stacks action from an EC2 instance, as well as any Amazon EC2 or Amazon S3 action.

```
{  
    "Version": "2012-10-17",  
    "Statement": [ {  
        "Effect": "Allow",  
        "Action": [ "ec2:*", "s3:*", "opsworks:*", "iam:PassRole"],  
        "Resource": "*"  
    }  
]
```

Note

If you do not allow `iam:PassRole`, any attempt to call an AWS 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 Using IAM* guide.

Managing SSH Access

AWS 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 \(p. 651\)](#) commands.
For more information, see [Logging In with SSH \(p. 212\)](#).
- 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 \(p. 214\)](#).

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 AWS 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 AWS 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 AWS 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 \(p. 304\)](#).

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, AWS 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 AWS OpsWorks Stacks, as described in [Registering an IAM User's Public SSH Key \(p. 305\)](#).

- When you set user permissions for a stack, you specify which users should have SSH access to the stack's instances.

AWS 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 \(p. 212\)](#).

Using personal SSH keys has the following advantages.

- There's no need to manually configure keys on the instances; AWS OpsWorks Stacks automatically installs the appropriate public keys on every instance.
- AWS 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 \(p. 288\)](#) to revoke the user's SSH/RDP permissions.

AWS 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 AWS OpsWorks Stacks automatically updates the instances.

Installing an Amazon EC2 Key

When you create a stack, you can specify an Amazon EC2 SSH key that is installed by default on every instance in the stack.

Add Stack

Name

Region

VPC **NEW**

Default Availability Zone

Default operating system

Default root device type Instance store
 EBS backed

IAM role

Default SSH key
Existing keys
somekey
None
Do not use a default SSH key
Layer Dependent

Host name theme

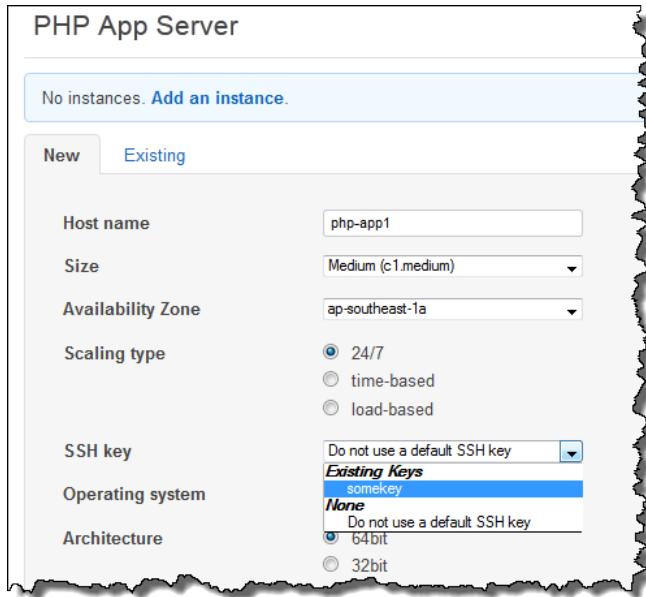
Stack color 

[Advanced **NEW** »](#)

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 AWS OpsWorks Stacks installs the public key in the `authorized_keys` file.

Registering an IAM User's Public SSH Key

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 \(p. 288\)](#).

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 an IAM 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](#).

Tip

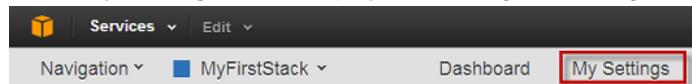
If you use [PuTTYgen](#) to generate your key pair, copy the public key from the **Public key for pasting into OpenSSH authorized_keys file** box. Clicking **Save Public Key** saves the public key in a format that is not supported by MindTerm.

2. Sign into the AWS 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, AWS 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 \(p. 288\)](#).

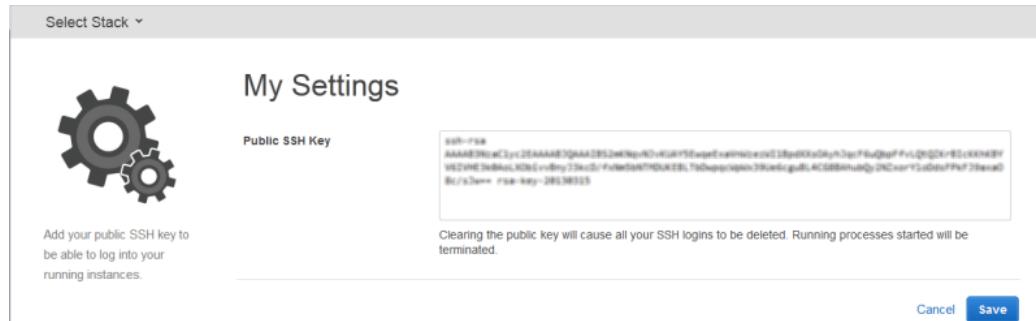
3. Select **My Settings**, which displays the settings for the signed-in IAM user.



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 AWS OpsWorks Stacks. For more information, see [Using the Built-in MindTerm SSH Client \(p. 213\)](#).

Managing Linux 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, AWS OpsWorks Stacks automatically installs the latest updates during setup, after an instance finishes booting. AWS 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 \(p. 133\)](#), which installs the current set of security patches and other updates on the specified instances.

For both of these approaches, AWS 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.

For more information, see [CentOS Product Specifications](#).

- **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 \(p. 401\)](#) to install your preferred updates.

Because system updates don't map naturally to a [lifecycle event \(p. 253\)](#), include the recipe in your custom cookbooks but [execute it manually \(p. 255\)](#). 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 \(p. 212\)](#) and run the appropriate commands manually.

Using Security Groups

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, AWS 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 AWS OpsWorks Stacks layers
- Inbound TCP traffic on port 22 (SSH login)

Caution

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 \(p. 214\)](#).

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.

Tip

If you accidentally delete an AWS OpsWorks Stacks security group, the preferred way to recreate it is to have AWS OpsWorks Stacks perform the task for you. Just create a new stack in the same AWS region—and VPC, if present—and AWS 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, AWS 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 AWS 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) – AWS 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** – AWS 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, AWS 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.

AWS OpsWorks Stacks Support for Chef 12 Linux

This section provides a brief overview of AWS OpsWorks Stacks for Chef 12 Linux. For information about Chef 12 on Windows, see [Getting Started: Windows \(p. 67\)](#). For information about previous Chef versions on Linux, see [Chef 11.10 and Earlier Versions for Linux \(p. 313\)](#).

Overview

AWS OpsWorks Stacks supports Chef 12, the latest version of Chef, for Linux stacks. For more information, see [Learn Chef](#).

AWS 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 AWS 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 AWS 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 [AWS OpsWorks Stacks Lifecycle Events \(p. 253\)](#). This second run lets you use your own Chef cookbooks or community cookbooks.
- **Resolution of namespace conflicts** - Prior to Chef 12, AWS 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 AWS OpsWorks Stacks had run. Unwanted default configurations had to be manually overwritten, a time consuming and error-prone task. In Chef 12 for Linux, AWS OpsWorks Stacks no longer supports built-in Chef cookbooks for application server environments like PHP, Node.js, or Rails. By eliminating built-in recipes, AWS OpsWorks Stacks eliminates the issue of naming collisions between built-in and custom recipes.

- **Strong support for Chef community cookbooks** – AWS 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 AWS 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** - AWS 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 AWS 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 \(p. 313\)](#).

Moving to Chef 12

Key AWS 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 AWS OpsWorks Stacks recipes. Removal of built-in layers also enables AWS 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 AWS OpsWorks Stacks is running recipes that you provide, there is no longer a need to override built-in AWS 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.
- AWS OpsWorks Stacks continues to provide support for the following layers for Chef 12 Linux stacks:
 - [Custom Layers \(p. 157\)](#)
 - [Amazon RDS Service Layer \(p. 150\)](#)
 - [ECS Cluster Layers \(p. 153\)](#)
- 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. For more information, see [AWS OpsWorks Stacks Data Bag Reference \(p. 659\)](#).
- 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 \(p. 116\)](#).
- 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 \(p. 158\)](#).

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 \(p. 160\)](#).

Supported Instance Types

AWS 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 \(p. 35\)](#)

Introduces you to AWS OpsWorks Stacks by guiding you through a brief hands-on exercise with the AWS OpsWorks Stacks console to create a Node.js application environment.

- [Getting Started: Linux \(p. 48\)](#)

Introduces you to AWS OpsWorks Stacks and Chef 12 Linux by guiding you through a hands-on exercise with the AWS 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 \(p. 157\)](#)

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 \(p. 311\)](#)

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

AWS 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 AWS 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 \(p. 510\)](#).

In Chef 12 Linux, stack settings are available as [Chef data bags](#) and are accessed only through Chef search. Data bags are stored on AWS 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 AWS 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 [AWS OpsWorks Stacks Data Bag Reference \(p. 659\)](#).

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 AWS OpsWorks Stacks data bag names such as `aws_opsworks_app`, instead of using AWS OpsWorks Stacks attribute names such as `opsworks` and `deploy`.

For more information, see the [AWS OpsWorks Stacks Data Bag Reference \(p. 659\)](#).

Support for Previous Chef Versions in AWS OpsWorks Stacks

This section provides a brief overview of the AWS OpsWorks Stacks documentation for previous Chef versions.

[Chef 11.10 and Earlier Versions for Linux \(p. 313\)](#)

Provides documentation about AWS OpsWorks Stacks support for Chef 11.10, 11.4, and 0.9 for Linux stacks.

Chef 11.10 and Earlier Versions for Linux

This section provides a brief overview of the AWS OpsWorks Stacks documentation for Chef 11.10, 11.4, and 0.9 for Linux.

[Getting Started with Chef 11 Linux Stacks \(p. 313\)](#)

Provides a walkthrough that shows you how to create a simple but functional PHP application server stack.

[Creating Your First Node.js Stack \(p. 335\)](#)

Describes how to create a Linux stack that supports a Node.js application server and how to deploy a simple application.

[Customizing AWS OpsWorks Stacks \(p. 345\)](#)

Describes how to customize AWS OpsWorks Stacks to meet your specific requirements.

[Cookbooks 101 \(p. 382\)](#)

Describes how to implement recipes for AWS OpsWorks Stacks instances.

[Load Balancing a Layer \(p. 462\)](#)

Describes how to use available AWS OpsWorks Stacks load balancing options.

[Running a Stack in a VPC \(p. 126\)](#)

Describes how to create and run a stack in a virtual private cloud.

[Migrating from Chef Server \(p. 463\)](#)

Provides guidelines for migrating from Chef Server to AWS OpsWorks Stacks.

[AWS OpsWorks Stacks Layer Reference \(p. 467\)](#)

Describes the available AWS OpsWorks Stacks built-in layers.

[Cookbook Components \(p. 504\)](#)

Describes the three standard cookbook components: attributes, templates, and recipes.

[Stack Configuration and Deployment Attributes: Linux \(p. 510\)](#)

Describes stack configuration and deployment attributes for Linux.

[Built-in Cookbook Attributes \(p. 532\)](#)

Describes how to use built-in recipe attributes to control the configuration of installed software.

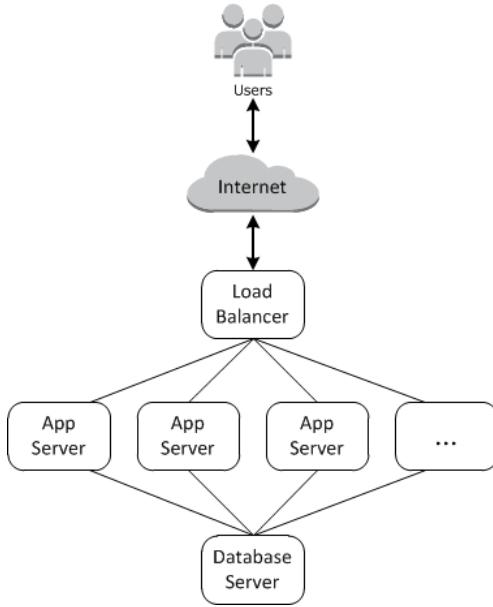
[Troubleshooting Chef 11.10 and Earlier Versions for Linux \(p. 561\)](#)

Describes approaches to troubleshooting various issues in AWS OpsWorks Stacks.

Getting Started with Chef 11 Linux Stacks

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 \(p. 48\)](#). For information about getting started with Chef 12 Windows stacks, see [Getting Started: Windows \(p. 67\)](#).

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.

AWS 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 AWS 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 AWS 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 \(p. 315\)](#) shows how to get set up to start the walkthrough.
- [Step 2: Create a Simple Application Server Stack \(p. 315\)](#) shows how to create a minimal stack that consists of a single application server.
- [Step 3: Add a Back-end Data Store \(p. 320\)](#) shows how to add a database server and connect it to the application server.
- [Step 4: Scale Out MyStack \(p. 329\)](#) 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 \(p. 315\)](#)
- [Step 2: Create a Simple Application Server Stack \(p. 315\)](#)
- [Step 3: Add a Back-end Data Store \(p. 320\)](#)
- [Step 4: Scale Out MyStack \(p. 329\)](#)

- [Step 5: Delete MyStack \(p. 334\)](#)

Step 1: Complete the Prerequisites

Complete the following setup steps before you can start the walkthrough. These setup steps include signing up for an AWS account, creating an IAM user in your AWS account, and assigning the IAM user access permissions to AWS OpsWorks Stacks.

If you have already completed any of the [Getting Started with AWS OpsWorks Stacks \(p. 34\)](#) walkthroughs, then you have met the prerequisites for this walkthrough, and you can skip ahead to [Step 2: Create a Simple Application Server Stack \(p. 315\)](#).

Topics

- [Step 1.1: Sign up for an AWS Account \(p. 315\)](#)
- [Step 1.2: Create an IAM User in Your AWS Account \(p. 315\)](#)
- [Step 1.3: Assign Service Access Permissions to Your IAM User \(p. 315\)](#)

Step 1.1: Sign up for an AWS Account

In this step, you will sign up for an AWS account. If you already have an AWS account that you want to use for this walkthrough, you can skip ahead to [Step 1.2: Create an IAM User in Your AWS Account \(p. 315\)](#).

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

To sign up for AWS

1. Open <https://aws.amazon.com/> and choose **Create an AWS Account**.
2. Follow the online instructions.

Step 1.2: Create an IAM User in Your AWS Account

Use your AWS account to create an AWS Identity and Access Management (IAM) user. You use an IAM user to access the AWS OpsWorks Stacks service. (We don't recommend that you use your AWS account to access AWS OpsWorks Stacks directly. Doing so is generally less secure and can make it more difficult for you to troubleshoot service access issues later.) If you already have an IAM user that you want to use for this walkthrough, you can skip ahead to [Step 1.3: Assign Service Access Permissions to Your IAM User \(p. 315\)](#).

To create an IAM user, see [Creating IAM Users \(AWS Management Console\)](#). For this walkthrough, we will refer to the user as `OpsWorksDemoUser`. However, you can use a different name.

Step 1.3: Assign Service Access Permissions to Your IAM User

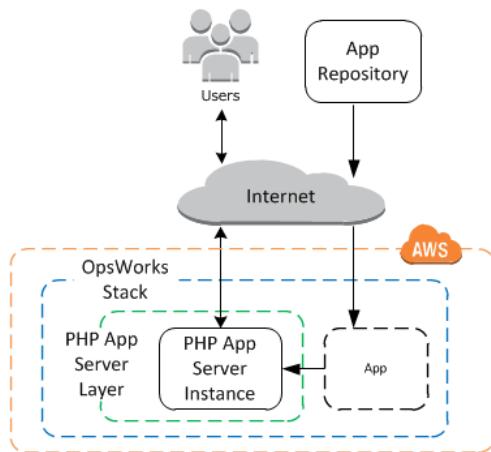
Set up your IAM user to enable access to the AWS OpsWorks Stacks service (and related services that AWS OpsWorks Stacks relies on).

To assign service access permissions to your IAM user, see [Attaching Managed Policies](#). Attach the `AWSOpsWorksFullAccess` and `AmazonS3FullAccess` policies to the user you created in the previous step or to the existing IAM user that you want to use.

You have now completed all of the setup steps and can [start this walkthrough \(p. 315\)](#).

Step 2: Create a Simple Application Server Stack

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. AWS OpsWorks Stacks also supports several other app servers. For more information, see [Application Server Layers \(p. 482\)](#).

- 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 AWS 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 \(p. 316\)](#)
- [Step 2.2: Add a PHP App Server Layer \(p. 317\)](#)
- [Step 2.3: Add an Instance to the PHP App Server Layer \(p. 317\)](#)
- [Step 2.4: Create and Deploy an App \(p. 319\)](#)

[Step 2.1: Create a Stack](#)

You start an AWS 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 [AWS 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 AWS 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 \(p. 120\)](#).

Step 2.2: Add a PHP App Server Layer

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 AWS 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. AWS 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 AWS OpsWorks Stacks \(p. 345\)](#).

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 \(p. 138\)](#).

To add a PHP App Server layer to MyStack

1. Open the Add Layer Page

After you finish creating the stack, AWS 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 \(p. 140\)](#).

Step 2.3: Add an Instance to the PHP App Server Layer

An AWS OpsWorks Stacks instance represents a particular Amazon EC2 instance:

- The instance's configuration specifies some basics like the Amazon EC2 operating 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.

AWS OpsWorks Stacks installs an agent on each instance that interacts with the service. To add a layer's functionality to an instance, AWS OpsWorks Stacks directs the agent to run small applications called [Chef recipes](#), which can install applications and packages, create configuration files, and so on. AWS OpsWorks Stacks runs recipes at key points in the instance's [lifecycle \(p. 253\)](#). 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.

Tip

If you are curious about how the recipes work, all of the AWS OpsWorks Stacks built-in recipes are in a public GitHub repository: [OpsWorks Cookbooks](#). You can also create your own custom recipes and have AWS 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, AWS 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 AWS OpsWorks Stacks. In this example, AWS 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 \(p. 158\)](#).

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. AWS 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 AWS 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 \(p. 181\)](#).

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** - AWS OpsWorks Stacks has called the Amazon EC2 service to create the Amazon EC2 instance.
2. **pending** - AWS OpsWorks Stacks is waiting for the Amazon EC2 instance to start.
3. **booting** - The Amazon EC2 instance is booting.
4. **running_setup** - The AWS 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

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:

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 \(p. 217\)](#).

1. Create an app.

An app contains the information that AWS 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, AWS 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, AWS 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. AWS OpsWorks Stacks also supports several other repository types. For more information, see [Application Source \(p. 218\)](#).

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 AWS OpsWorks Stacks uses for display purposes. The example app is named `simplePHPApp`. AWS 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 \(p. 320\)](#).

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/aws-labs/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 \(p. 217\)](#).

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.

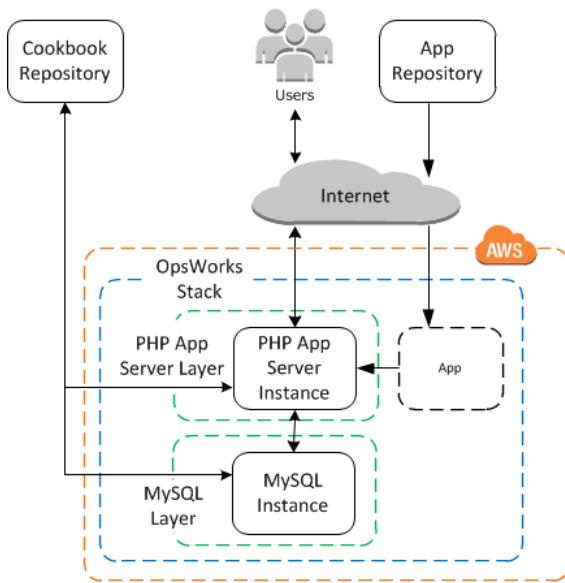
You should see a page such as the following in your browser.

Tip

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 AWS 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

[Step 2.1: Create a Stack \(p. 316\)](#) 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. AWS 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 \(p. 321\)](#)
- [Step 3.2: Update SimplePHPApp \(p. 322\)](#)
- [A Short Digression: Cookbooks, Recipes, and AWS OpsWorks Stacks Attributes \(p. 323\)](#)
- [Step 3.3: Add the Custom Cookbooks to MyStack \(p. 327\)](#)
- [Step 3.4: Run the Recipes \(p. 327\)](#)
- [Step 3.5: Deploy SimplePHPApp, Version 2 \(p. 328\)](#)
- [Step 3.6: Run SimplePHPApp \(p. 328\)](#)

Step 3.1: Add a Back-end Database

The new version of SimplePHPApp stores its data in a back-end database. AWS OpsWorks Stacks supports two types of database servers:

- The [MySQL AWS OpsWorks Stacks layer \(p. 474\)](#) 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” \(p. 562\)](#).

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

AWS 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

To start, you need a new version of SimplePHPApp that uses a back-end data store. With AWS 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 '`/`'—AWS 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; AWS 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, AWS OpsWorks Stacks automatically deploys the new app version to any new app server instances when you start them. However, AWS 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 \(p. 319\)](#). You could deploy the updated SimplePHPApp now, but for this example, it's better to wait a bit.

A Short Digression: Cookbooks, Recipes, and AWS OpsWorks Stacks Attributes

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. AWS 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 AWS 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 My Stack \(p. 327\)](#) 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 \(p. 323\)](#)
- [Set Up the Database \(p. 324\)](#)
- [Connect the Application to the Database \(p. 326\)](#)

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.

AWS 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 \(p. 235\)](#) or [Customizing AWS OpsWorks Stacks \(p. 345\)](#).

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.
- AWS 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 \(p. 135\)](#) 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 AWS OpsWorks Stacks attributes, see [Stack Configuration and Deployment Attributes: Linux \(p. 510\)](#) and [Built-in Cookbook Attributes \(p. 532\)](#). For more information, see [Overriding Attributes \(p. 346\)](#).

Attributes are organized in a hierarchical structure, which can be represented as a JSON object. The following example shows a JSON representation of the deployment attributes for SimplePHPApp, which includes the values that you need for the task at hand.

```

"deploy": {
  "simplephpapp": {
    "sleep_before_restart": 0,
    "rails_env": null,
    "document_root": "web",
    "deploy_to": "/srv/www/simplephpapp",
    "ssl_certificate_key": null,
    "ssl_certificate": null,
    "deploying_user": null,
    "ssl_certificate_ca": null,
    "ssl_support": false,
    "migrate": false,
    "mounted_at": null,
    "application": "simplephpapp",
    "auto_bundle_on_deploy": true,
    "database": {
      "reconnect": true,
      "database": "simplephpapp",
      "host": null,
      "adapter": "mysql",
      "data_source_provider": "stack",
      "password": "d1zethv0sm",
      "port": 3306,
      "username": "root"
    },
    "scm": {
      "revision": "version2",
      "scm_type": "git",
      "user": null,
      "ssh_key": null,
      "password": null,
      "repository": "git://github.com/amazonwebservices/opsworks-demo-php-simple-app.git"
    },
    "symlink_before_migrate": {
      "config/opsworks.php": "opsworks.php"
    },
    "application_type": "php",
    "domains": [
      "simplephpapp"
    ],
    "memcached": {
      "port": 11211,
      "host": null
    },
    "symlinks": {
    },
    "restart_command": "echo 'restarting app'"
  }
}

```

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 AWS OpsWorks Stacks run

it for you. This section describes how the recipe, `dbsetup.rb`, is implemented. The procedure for having AWS 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.

```
node[:deploy].each do |app_name, deploy|
  execute "mysql-create-table" do
    command "/usr/bin/mysql -u#{deploy[:database][:username]} -p#{deploy[:database][:password]} #{deploy[:database][:database]} -e'CREATE TABLE #{node[:phpapp][:dbtable]}(
      id INT UNSIGNED NOT NULL AUTO_INCREMENT,
      author VARCHAR(63) NOT NULL,
      message TEXT,
      PRIMARY KEY (id)
    )'"
    not_if "/usr/bin/mysql -u#{deploy[:database][:username]} -p#{deploy[:database][:password]} #{deploy[:database][:database]} -e'SHOW TABLES' | grep #{node[:phpapp][:dbtable]}"
    action :run
  end
end
```

This recipe iterates over the apps in the `deploy` node. This example has only one app, but it's possible for a `deploy` node to have multiple apps.

`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.

```

node[:deploy].each do |app_name, deploy|
  script "install_composer" do
    interpreter "bash"
    user "root"
    cwd "#{deploy[:deploy_to]}/current"
    code <<-EOH
      curl -s https://getcomposer.org/installer | php
      php composer.phar install
    EOH
  end

  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

  only_if do
    File.directory?("#{deploy[:deploy_to]}/current")
  end
end
end

```

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:

```
<?php
  define('DB_NAME', '<%= @db%>');
  define('DB_USER', '<%= @user%>');
  define('DB_PASSWORD', '<%= @password%>');
  define('DB_HOST', '<%= @host%>');
  define('DB_TABLE', '<%= @table%>');
?>
```

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:

```
<?php
  define('DB_NAME', 'simplephpapp');
  define('DB_USER', 'root');
  define('DB_PASSWORD', 'ujn12pw');
  define('DB_HOST', '10.214.175.110');
  define('DB_TABLE', 'urler');
?>
```

Step 3.3: Add the Custom Cookbooks to MyStack

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 AWS 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.

AWS OpsWorks Stacks then installs the contents of your cookbook repository on all of the stack's instances. If you create new instances, AWS 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. AWS OpsWorks Stacks will automatically install the updated cookbooks on all new instances. However, AWS OpsWorks Stacks does not automatically install updated cookbooks on the stack's online instances. You must explicitly direct AWS OpsWorks Stacks to update the cookbooks by running the `Update Cookbooks` stack command. For more information, see [Run Stack Commands \(p. 133\)](#).

Step 3.4: Run the Recipes

After you have your custom cookbook, you need to run the recipes on the appropriate instances. You could [run them manually \(p. 255\)](#). 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 AWS OpsWorks Stacks automatically run them for you at the appropriate time.

AWS OpsWorks Stacks supports a set of [lifecycle events \(p. 253\)](#) 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 AWS 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 MySQL instance 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` (p. 324) 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

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

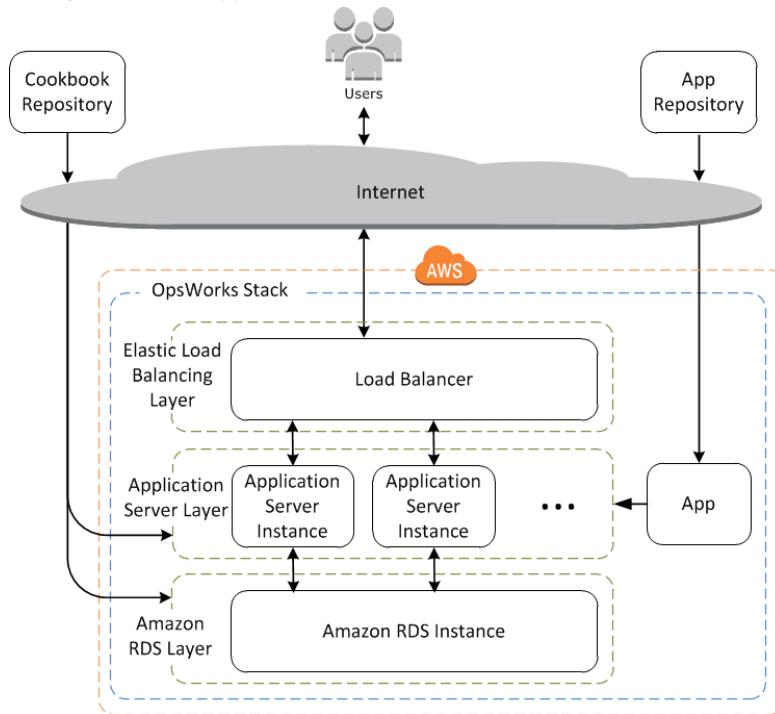
After the deployment status changes to **successful**, you can run the new SimplePHPApp version, as follows.

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

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:



AWS 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 AWS OpsWorks Stacks scale your stack automatically. For more information, see [Managing Load with Time-based and Load-based Instances \(p. 181\)](#).

Step 4.1: Add a Load Balancer

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 AWS 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 \(p. 126\)](#). For an AWS CloudFormation template that extends the example in this walkthrough to run in a VPC, see [OpsWorksVPCELB.template](#).

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. AWS 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

AWS OpsWorks Stacks does not support Application Load Balancer. You can only use Classic Load Balancer with AWS 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 [Get 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 AWS 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**; AWS OpsWorks Stacks automatically registers instances with the load balancer.

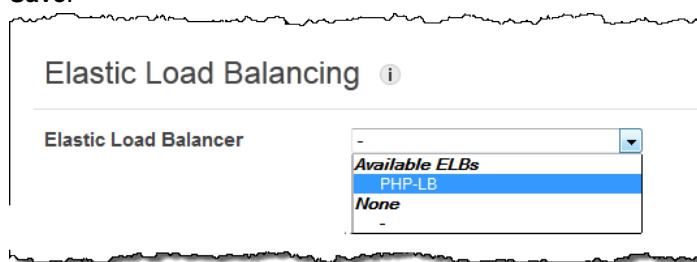
Add Tags

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

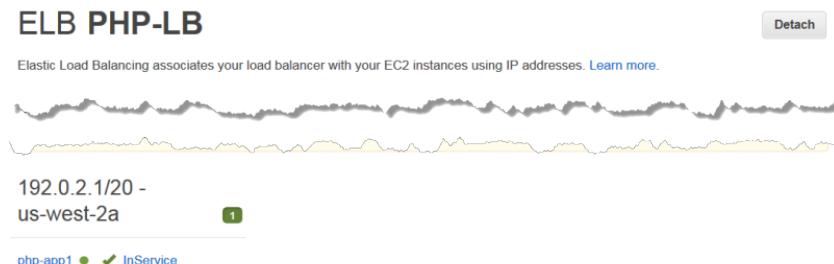
Review your choices, choose **Create**, and then choose **Close**, which starts the load balancer.

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 AWS 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, AWS 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, AWS 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.

The screenshot shows the 'ELB PHP-LB' configuration page. It includes a note about associating the load balancer with EC2 instances using IP addresses, a 'Learn more' link, and a 'Settings' section. The 'Settings' section contains the following information:

Layer	PHP App Server
DNS Name	PHP-LB-862966592.us-west-2.elb.amazonaws.com
Region	US West (Oregon)
Attached availability zones	us-west-2a

Note

AWS OpsWorks Stacks also supports the HAProxy load balancer, which might have advantages for some applications. For more information, see [HAProxy AWS OpsWorks Stacks Layer \(p. 469\)](#).

Step 4.2: Add PHP App Server Instances

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, AWS 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

AWS 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.

Monitoring Layers

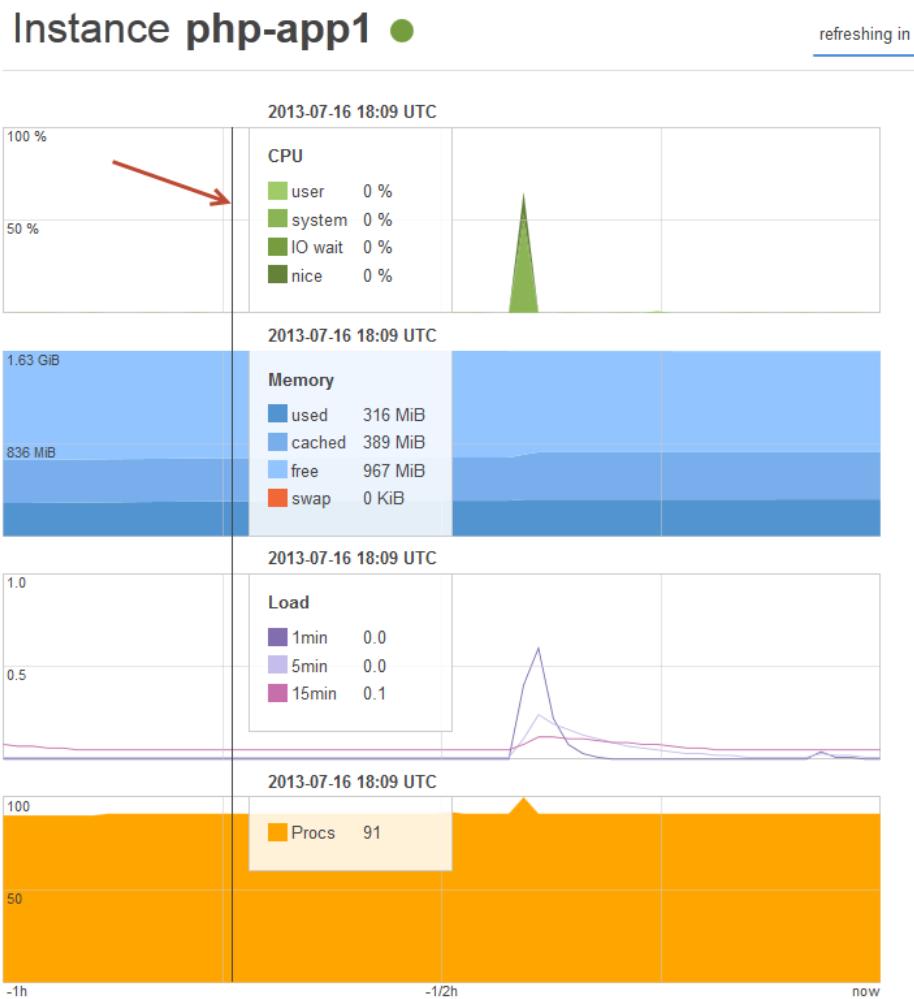


- Click **PHP App Server** to see metrics for each of the layer's instances.

Layer PHP App Server



- Click **php-app1** to see metrics for that instance. You can see metrics for any particular point in time by moving the slider.



Note

AWS OpsWorks Stacks also supports the Ganglia monitoring server, which might have advantages for some applications. For more information, see [Ganglia Layer \(p. 503\)](#).

Step 5: Delete MyStack

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**, AWS 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 state, you must delete each instance. In the **PHP App Server** layer click **Delete** in the php-app1 instance's **Actions** column.

AWS 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

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 \(p. 491\)](#) with two instances
- An [Elastic Load Balancing load balancer \(p. 147\)](#) to distribute traffic to the application server instances
- An [Amazon Relational Database Service \(Amazon RDS\) service layer \(p. 344\)](#) that provides a backend database

Topics

- [Prerequisites \(p. 335\)](#)
- [Implementing the Application \(p. 336\)](#)
- [Creating the Database Server and Load Balancer \(p. 339\)](#)
- [Creating the Stack \(p. 341\)](#)
- [Deploying the Application \(p. 342\)](#)
- [What Next? \(p. 344\)](#)

Prerequisites

This walkthrough assumes the following:

- You have an AWS account and a basic understanding of how to use AWS OpsWorks Stacks.

If you are new to AWS OpsWorks Stacks or to AWS, learn the basics by completing the introductory tutorial in [Getting Started with Chef 11 Linux Stacks \(p. 313\)](#).

- 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, AWS 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 AWS 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 \(p. 313\)](#). 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, AWS 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 AWS Identity and Access Management (IAM) user has full-access permissions for AWS OpsWorks Stacks.

For security reasons, we strongly recommend that you do not use your account's root credentials for this walkthrough. Instead, create an IAM user with AWS OpsWorks Stacks full-access permissions and use those credentials with AWS OpsWorks Stacks. For more information, see [Creating an Administrative User \(p. 285\)](#).

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 \(p. 336\)](#)
- [The Layout File \(p. 337\)](#)
- [The Code File \(p. 337\)](#)

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": "Node.js-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 AWS 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, AWS 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, AWS 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. AWS 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 \(p. 144\)](#). Do not modify inbound rules by editing the built-in security group. Each time you create a stack, AWS OpsWorks Stacks overwrites the built-in security groups with the standard settings, so any changes that you make will be lost.

Tip

You can associate custom environment variables with your application when you [create \(p. 220\)](#) or [update \(p. 224\)](#) 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 \(p. 226\)](#).

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

Tip

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 \(p. 126\)](#).

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**. AWS 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 AWS OpsWorks Stacks console.

Sign into the [AWS 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 \(p. 120\)](#).

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, AWS OpsWorks Stacks automatically registers or deregisters the instance with the load balancer.

Tip

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 \(p. 150\)](#).

Caution

AWS 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 \(p. 150\)](#), choose **Register with Stack**.

Deploying the Application

You must store the application in a remote repository. When you deploy it, AWS 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 \(p. 218\)](#).

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 AWS OpsWorks Stacks does not support uncompressed tarballs. For more information, see [Application Source \(p. 218\)](#).

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

AWS 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 \(p. 218\)](#).

3. Create an AWS OpsWorks Stacks app.

Return to the AWS 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 AWS 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. It should look something like `https://s3.amazonaws.com/node_example/nodedb.zip`.

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. AWS 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.

Tip

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

Tip

AWS OpsWorks Stacks automatically deploys apps to new instances during setup. Manual deployment is required only for online instances. For more information, see [Deploying Apps \(p. 221\)](#). For a general discussion of deployment, including some more sophisticated deployment strategies, see [Managing and Deploying Apps and Cookbooks \(p. 110\)](#).

What Next?

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 AWS OpsWorks Stacks uses to install and configure the software, and the built-in [deploy cookbook](#), which contains the recipes that AWS 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 AWS OpsWorks Stacks \(p. 345\)](#).

Add SSL support

You can enable SSL support for your app and provide AWS OpsWorks Stacks with the appropriate certificates when you create the app. AWS OpsWorks Stacks then installs the certificates in the appropriate directory. For more information, see [Using SSL \(p. 230\)](#).

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 AWS OpsWorks Stacks stack. For more information, see [ElastiCache Redis \(p. 566\)](#) and [Memcached \(p. 504\)](#).

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 \(p. 110\)](#).

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 \(p. 352\)](#).

Define environment variables

You can pass data to your application by defining environment variables for the associated app. When you deploy the app, AWS OpsWorks Stacks exports those variables so you can access them from your app. For more information, see [Using Environment Variables \(p. 226\)](#).

Customizing AWS OpsWorks Stacks

AWS 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, AWS 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.

AWS 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 AWS OpsWorks Stacks settings.
- Implement a custom Chef cookbook with an attributes file that overrides the default AWS OpsWorks Stacks settings.
- Implement a custom Chef cookbook with a template that overrides or extends a default AWS 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, AWS 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 \(p. 240\)](#).

- You cannot override built-in recipes on Windows stacks.

The way that AWS 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 \(p. 235\)](#) if you are not already familiar with cookbook implementation. [Cookbook Basics \(p. 383\)](#) provides a detailed tutorial introduction to implementing custom cookbooks, and [Implementing Cookbooks for AWS OpsWorks Stacks \(p. 411\)](#) covers some of the details about how to implement cookbooks for AWS OpsWorks Stacks instances.

Topics

- [Customizing AWS OpsWorks Stacks Configuration by Overriding Attributes \(p. 346\)](#)
- [Extending AWS OpsWorks Stacks Configuration Files Using Custom Templates \(p. 351\)](#)
- [Extending a Layer \(p. 352\)](#)
- [Creating a Custom Tomcat Server Layer \(p. 356\)](#)
- [Stack Configuration and Deployment Attributes \(p. 376\)](#)

Customizing AWS OpsWorks Stacks Configuration by Overriding Attributes

Note

For Windows stacks and Chef 12 Linux stacks, AWS 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**—AWS 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 \(p. 505\)](#), 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 \(p. 510\)](#) and [Built-in Cookbook Attributes \(p. 532\)](#). For more information about Ohai attributes, see [Ohai](#).

When a [lifecycle event \(p. 253\)](#) such as Deploy or Configure occurs, or you run a [stack command \(p. 133\)](#) such as `execute_recipes` or `update_packages`, AWS 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 \(p. 347\)](#)
- [Overriding Attributes With Custom JSON \(p. 348\)](#)
- [Overriding AWS OpsWorks Stacks Attributes Using Custom Cookbook Attributes \(p. 350\)](#)

Attribute Precedence

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. AWS 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 AWS 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 AWS 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](#).

Tip

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 AWS 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 cookbook 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 cookbook `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

Note

Because AWS 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 AWS 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 \(p. 347\)](#).

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 \(p. 376\)](#).

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 \(p. 510\)](#) and

[Built-in Cookbook Attributes \(p. 532\)](#) 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 \(p. 535\)](#) attribute, whose node is `node[:apache][:keepalivetimeout]` and has a default value of 3.
- The [logrotate schedule \(p. 536\)](#) 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 \(p. 120\)](#)
- [Update a stack \(p. 132\)](#)
- [Run a stack command \(p. 132\)](#)
- [Clone a stack \(p. 132\)](#)
- [Deploy an app \(p. 221\)](#)

For each task, AWS 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 AWS 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 AWS OpsWorks Stacks Attributes Using Custom Cookbook Attributes

Note

For Windows stacks, AWS 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 AWS 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 AWS OpsWorks Stacks definitions is straightforward.

To override AWS OpsWorks Stacks attribute definitions

1. Set up a cookbook repository, as described in [Cookbooks and Recipes \(p. 235\)](#).
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 AWS OpsWorks Stacks attribute. For a detailed list of AWS OpsWorks Stacks attributes, including node names, see [Stack Configuration and Deployment Attributes: Linux \(p. 510\)](#) and [Built-in Cookbook Attributes \(p. 532\)](#). For more information on attributes and attributes files, see [About Attribute Files](#).

Important

Your attributes must be `normal` type to override AWS 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 \(p. 346\)](#).

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 AWS OpsWorks Stacks to download your cookbooks to the stack's instances. For more information, see [Installing Custom Cookbooks \(p. 249\)](#).

Tip

For a complete walkthrough of this procedure, see [Overriding Built-In Attributes \(p. 456\)](#).

The node object used by subsequent lifecycle events, deploy commands, and stack commands will now contain your attribute definitions instead of the AWS OpsWorks Stacks values.

For example, to override the built-in Apache `keepalivetimeout` and `logrotate schedule` settings discussed in [How to Specify Custom JSON \(p. 348\)](#), 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 AWS 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 AWS 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 AWS OpsWorks Stacks Configuration Files Using Custom Templates

Note

Because AWS 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.

AWS 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 AWS OpsWorks Stacks definitions, your preferred settings are incorporated into the configuration files in place of the AWS OpsWorks Stacks settings. However, AWS 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 AWS 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 \(p. 507\)](#).

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 AWS 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 \(p. 249\)](#).

Tip

For a complete walkthrough of this procedure, see [Overriding Built-In Templates \(p. 459\)](#).

You don't have to implement any recipes or [add recipes to the layer configuration \(p. 255\)](#) to override a template. AWS 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 AWS 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. AWS 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 AWS OpsWorks Stacks repository for changes, and manually update your template as needed.

Extending a Layer

Sometimes, you need to customize a built-in layer beyond what can be handled by modifying AWS 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 \(p. 382\)](#), 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 \(p. 356\)](#).

Topics

- [Using Recipes to Run Scripts \(p. 352\)](#)
- [Using Chef Deployment Hooks \(p. 353\)](#)
- [Running Cron Jobs on Linux Instances \(p. 354\)](#)
- [Installing and Configuring Packages on Linux Instances \(p. 356\)](#)

Using Recipes to Run Scripts

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 \(p. 401\)](#). For an example of how to run a PowerShell script on a Windows instance, see [Running a Windows PowerShell Script \(p. 420\)](#).

Using Chef Deployment Hooks

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 \(p. 220\)](#) that you have specified. However, you must use `new_resource.environment["VARIABLE_NAME"]` to access the variable's value instead of `ENV["VARIABLE_NAME"]`. For more information, see [deploy](#).

For more information on how to use deployment hooks and what information a hook can access, see [Deploy Phases](#).

Running Cron Jobs on Linux Instances

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 AWS 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 \(p. 235\)](#).
2. Install the cookbook on your stack. For more information, see [Installing Custom Cookbooks \(p. 249\)](#).
3. Have AWS 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 \(p. 255\)](#).
 - **Setup** – Assigning `cronjob.rb` to this event directs AWS OpsWorks Stacks to run the recipe on all new instances.
 - **Deploy** – Assigning `cronjob.rb` to this event directs AWS 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 \(p. 133\)](#).

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 AWS 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 AWS 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, AWS 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

The built-in layers support only certain packages. For more information, see [Layers \(p. 138\)](#). 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 \(p. 356\)](#). 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

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 AWS OpsWorks Stacks instances is to [extend an existing layer \(p. 356\)](#). 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 \(p. 157\)](#). 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.

Tip

If you are new to Chef, you should first read [Cookbooks 101 \(p. 382\)](#), 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 \(p. 235\)](#).

Note

AWS OpsWorks Stacks includes a full-featured [Java App Server layer \(p. 484\)](#) 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 \(p. 357\)](#) contains configuration settings that are used by the various recipes.
- [Setup recipes \(p. 358\)](#) are assigned to the layer's Setup lifecycle event (p. 253). They run after an instance has booted and perform tasks such as installing packages and creating configuration files.
- [Configure recipes \(p. 365\)](#) 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 \(p. 369\)](#) 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 \(p. 371\)](#), 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 AWS 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 \(p. 357\)](#)
- [Setup Recipes \(p. 358\)](#)
- [Configure Recipes \(p. 365\)](#)
- [Deploy Recipes \(p. 369\)](#)
- [Create a Stack and Run an Application \(p. 371\)](#)

Attributes File

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 AWS 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 \(p. 346\)](#).

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 \(p. 348\)](#).
- 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

Setup recipes are assigned to the layer's Setup [lifecycle \(p. 253\)](#) 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, AWS OpsWorks Stacks runs the [Deploy recipes \(p. 369\)](#) to deploy any apps to the new instance.

Topics

- [tomcat::setup \(p. 359\)](#)
- [tomcat::install \(p. 360\)](#)
- [tomcat::service \(p. 361\)](#)

- [tomcat::container_config \(p. 361\)](#)
- [tomcat::apache_tomcat_bind \(p. 364\)](#)

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'
```

The `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 \(p. 360\)](#) recipe installs the Tomcat server package.
- The [tomcat::service \(p. 361\)](#) 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 \(p. 361\)](#) recipe creates configuration files.
- The `apache2` recipe (which is shorthand for `apache2::default`) is an AWS OpsWorks Stacks built-in recipe that installs and configures an Apache server.
- The [tomcat::apache_tomcat_bind \(p. 364\)](#) recipe configures the Apache server to function as a front-end for the Tomcat server.

Tip

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 \(p. 369\)](#).

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:

- Creates a list of packages to be installed, depending on the instance's operating system.
- 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.

- 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.

Tip

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 \(p. 132\)](#) 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 \(p. 346\)](#).

[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, AWS 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.

Tip

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 \(p. 373\)](#).

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 \(p. 346\)](#).

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'] !~ /\Proxy/
  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 AWS OpsWorks Stacks built-in recipe that defines the Apache service. For more information, see the [recipe](#) in the AWS 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

Configure recipes are assigned to the layer's [Configure lifecycle](#) (p. 253) 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 creates 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 \(p. 361\)](#) 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
```

```
end
```

In addition to Tomcat cookbook attributes, this recipe uses the [stack configuration and deployment attributes \(p. 376\)](#) that AWS OpsWorks Stacks installs with the Configure event. The AWS 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 \(p. 376\)](#). With Chef 11.10 stacks, you also can use Chef search to obtain stack configuration and deployment data. For more information, see [the section called "Using Chef Search" \(p. 241\)](#).

`deploy` attributes refers to the `[:deploy]` namespace, which contains deployment-related attributes that are defined through the console or API, or generated by the AWS 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]`). For example, here is a JSON representation of a highly abbreviated version of the stack configuration and deployment attributes for an app named `my_1st_jsp` that was deployed to a Tomcat-based custom layer from an Amazon S3 archive.

```
{
  ...
  "datasources": {
    "ROOT": "jdbc/mydb"
  }
  ...
  "deploy": {
    "my_1st_jsp": {
      "document_root": "ROOT",
      ...
      "scm": {
        "password": null,
        "repository": "https://s3.amazonaws.com/.../my_1st_jsp.zip",
        "ssh_key": null,
        "scm_type": "archive",
        "user": null,
        "revision": null
      },
      ...
      "deploy_to": "/srv/www/my_1st_jsp",
      ...
      "database": {
        "password": "86la64jagj",
        "username": "root",
        "reconnect": true,
        "database": "my_1st_jsp",
        "host": "10.254.3.69"
      },
      ...
    }
  },
  ...
}
```

The `context` recipe first ensures that the service is defined for this Chef run by calling [tomcat::service \(p. 361\)](#). 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 \(p. 371\)](#) 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 \(p. 371\)](#). 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.

```
<Context>
  <Resource name="<%=@resource_name%" auth="Container" type="javax.sql.DataSource"
            maxActive="20" maxIdle="5" maxWait="10000"
            username="<%=@node['deploy'][@webapp_name]['database']['username']%"%
            password="<%=@node['deploy'][@webapp_name]['database']['password']%"%
            driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://<%=@node['deploy'][@webapp_name]['database']['host']%>:3306/<%=@node['deploy'][@webapp_name]['database']['database']%>"%
            factory="org.apache.commons.dbcp.BasicDataSourceFactory" />
</Context>
```

These `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

Deploy recipes are assigned to the layer's Deploy [lifecycle \(p. 253\)](#) 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. AWS 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.

AWS 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 \(p. 346\)](#).

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 \(p. 235\)](#): 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, AWS 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. AWS 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 AWS 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

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 AWS OpsWorks Stacks, you should first read [Getting Started with Chef 11 Linux Stacks \(p. 313\)](#).

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 \(p. 366\)](#) 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 \(p. 235\)](#).

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 AWS OpsWorks Stacks dashboard, click **Add Stack** to create a new stack and click **Advanced >>** to display all options. Configure the stack as follows.

For the remaining options, you can accept the defaults.

- **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"  
    }  
}
```

}

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 \(p. 366\)](#).

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**. The example uses `TomCustom` and `tomcustom` as the layer's name and short name, respectively.

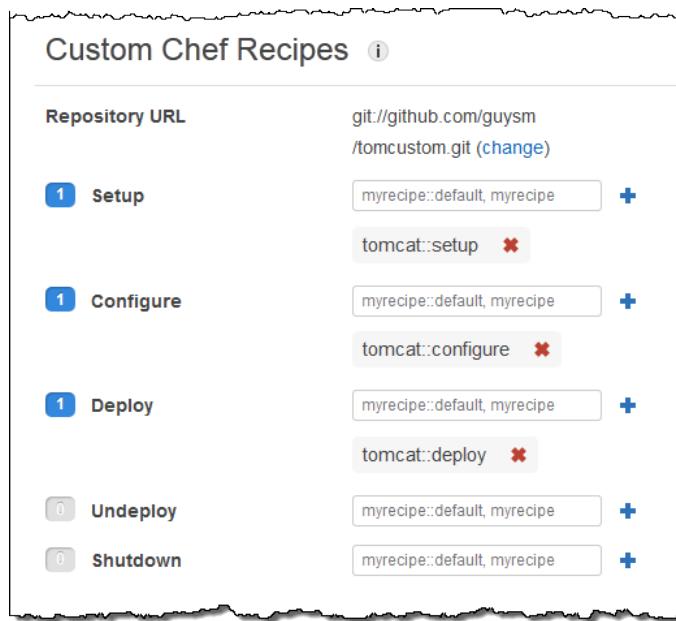
Add Layer

The 'Add Layer' dialog box shows the following fields:

Layer type	Custom
The Custom layer allows you to create a fully customized layer. Standard recipes handle basic setup and configuration for the layer instances, and you implement custom Chef recipes to install and configure any required software. You can create as many custom layers as you require. Learn more .	
Name	TomCustom
Short name	tomcustom

At the bottom right are 'Cancel' and 'Add layer' buttons.

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**:

Use default settings for the other options.

- **Name**—The app's name; the example uses SimpleJSP and the short name generated by AWS OpsWorks Stacks will be simplejsp.
- **App type**—Set this option to **Other**.

AWS OpsWorks Stacks automatically deploys standard app types to the associated server instances. If you set **App type** to other, AWS OpsWorks Stacks simply runs the Deploy recipes, and lets them handle deployment.

- **Document root**—Set this option to `/root`.

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.

App New

Settings

Name	SimpleJSP
App type	Other
Document root	ROOT

Application Source

Repository type	Http Archive
Repository URL	vs.com/opsworks-tomcat/simplejsp.zip
User name	Optional
Password	Optional

Add Domains

7. Use the **Instances** page to add an instance to the TomCustom layer and start it. AWS 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, AWS OpsWorks Stacks automatically creates a database for each app, named with the app's short name.

Stack Configuration and Deployment Attributes

When AWS 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 \(p. 133\)](#), AWS OpsWorks Stacks installs deploy attributes, which provide some additional deployment information.

For more information about the node object, see [Overriding Attributes \(p. 346\)](#). For a list of commonly used stack configuration and deployment attributes, including fully qualified node names, see [Stack Configuration and Deployment Attributes: Linux \(p. 510\)](#) and [Built-in Cookbook Attributes \(p. 532\)](#).

Tip

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 \(p. 652\)](#).

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 \(p. 377\)](#)
- [Deployment Attributes \(p. 380\)](#)

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 \(p. 135\)](#) 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 \(p. 510\)](#) and [Built-in Cookbook Attributes \(p. 532\)](#).

```
{
  "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-b67c48edaf4",
    "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 [AWS OpsWorks Stacks Layer Reference \(p. 467\)](#).

- `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 \(p. 133\)](#) 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 \(p. 510\)](#) and [Built-in Cookbook Attributes \(p. 532\)](#).

```
{
  ...
  "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 AWS 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 \(p. 220\)](#).
 - `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 \(p. 229\)](#).
 - `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

A production-level AWS OpsWorks Stacks stack typically requires some [customization \(p. 345\)](#), 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 AWS OpsWorks Stacks.

For more information on how AWS OpsWorks Stacks uses cookbooks, which includes a brief general introduction to cookbooks, see [Cookbooks and Recipes \(p. 235\)](#). 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 \(p. 383\)](#) 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 \(p. 383\)](#), you should first read [Vagrant and Test Kitchen \(p. 382\)](#) 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 AWS OpsWorks Stacks \(p. 411\)](#) describes how to implement recipes for AWS OpsWorks Stacks, including for Windows stacks.

It also includes some more advanced topics 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 \(p. 383\)](#). However AWS OpsWorks Stacks works a bit differently than Chef server, so we recommend that experienced Chef users at least read through this section.

Topics

- [Vagrant and Test Kitchen \(p. 382\)](#)
- [Cookbook Basics \(p. 383\)](#)
- [Implementing Cookbooks for AWS OpsWorks Stacks \(p. 411\)](#)

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 topic 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.

Topics

- [Vagrant \(p. 383\)](#)
- [Test Kitchen \(p. 383\)](#)

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 AWS 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`. It represents a Ubuntu 12.04 LTS system that is configured for Chef.

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 12.04 LTS 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` plugin 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.

Tip

The examples in this chapter use Test Kitchen as a convenient way to run recipes. If you prefer, you can skip 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

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.

Tip

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 AWS OpsWorks Stacks Windows instance, see [Running a Recipe on a Windows Instance \(p. 416\)](#).

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 \(p. 382\)](#).

Topics

- [Recipe Structure \(p. 384\)](#)
- [Example 1: Installing Packages \(p. 387\)](#)
- [Example 2: Managing Users \(p. 389\)](#)
- [Example 3: Creating Directories \(p. 389\)](#)
- [Example 4: Adding Flow Control \(p. 391\)](#)
- [Example 5: Using Attributes \(p. 395\)](#)
- [Example 6: Creating Files \(p. 397\)](#)
- [Example 7: Running Commands and Scripts \(p. 401\)](#)
- [Example 8: Managing Services \(p. 403\)](#)
- [Example 9: Using Amazon EC2 Instances \(p. 407\)](#)
- [Next Steps \(p. 411\)](#)

Recipe Structure

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 \(p. 469\)](#). 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
```

Tip

For this and other examples of working recipes and related files, see the [AWS OpsWorks Stacks built-in recipes](#).

The example highlights the key recipe elements, which are described in the following sections.

Topics

- [Resources \(p. 385\)](#)
- [Flow Control \(p. 386\)](#)
- [Included Recipes \(p. 387\)](#)

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.

Tip

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 \(p. 239\)](#).

Example 1: Installing Packages

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 \(p. 451\)](#).

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](#).

Tip

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.

Tip

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 12.04 LTS 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

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 \(p. 387\)](#), 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

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 12.04 LTS 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 \(p. 387\)](#), 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.

Tip

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 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.

Tip

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

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 \(p. 389\)](#) to address both scenarios.

Topics

- [Iteration \(p. 392\)](#)
- [Conditional Logic \(p. 393\)](#)

Iteration

[Example 3: Creating Directories \(p. 389\)](#) 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.

Tip

AWS OpsWorks Stacks recipes commonly use this approach to extract values from the [stack configuration and deployment JSON \(p. 376\)](#)—which is basically a large hash table—and insert them in a resource. For an example, see [Deploy Recipes \(p. 369\)](#).

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.

Tip

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`.

Tip

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

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 \(p. 392\)](#) 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 \(p. 347\)](#). 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 AWS 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 [\[:apache2\]\[:user\] \(p. 538\)](#), for example, unless you want to override that attribute's value. For more information, see [Using Custom Cookbook Attributes \(p. 350\)](#).

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
```

Tip

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.

Tip

With AWS OpsWorks Stacks, defining values as attributes provides an additional benefit; you can use [custom JSON \(p. 135\)](#) 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 \(p. 346\)](#).

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 \(p. 393\)](#), 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 \(p. 393\)](#).
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

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 \(p. 397\)](#)
- [Creating a File from a Template \(p. 399\)](#)

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 \(p. 389\)](#) 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 \(p. 387\)](#), 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.

Tip

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` contains `example_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

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 \(p. 401\)](#)
- [Running Scripts \(p. 402\)](#)

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 \(p. 420\)](#).

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 \(p. 387\)](#), 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

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 \(p. 356\)](#).

Topics

- [Defining and Starting a Service \(p. 403\)](#)
- [Using notifies to Start or Restart a Service \(p. 405\)](#)

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 \(p. 387\)](#), 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 12.04 LTS 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 \(p. 399\)](#). 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 \(p. 358\)](#).

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 \(p. 399\)](#) 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.

Tip

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

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.

Tip

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.

To sign up for AWS

1. Open <https://aws.amazon.com/>, and then choose **Create an AWS Account**.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

You should then [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.

- `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. For more information, see [Best Practices for Managing AWS Access Keys](#).

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 East (N. Virginia).

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 \(p. 387\)](#). 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 East (N. Virginia) region.

```
driver:
  name: ec2
  aws_ssh_key_id: US-East1
  region: us-east-1
  availability_zone: us-east-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 East (N. Virginia), which is represented by `us-east-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-east-1b` for US East (N. Virginia). 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-east-1b`, so the example explicitly specifies `us-east-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 an IAM 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 East (N. Virginia) 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 \(p. 392\)](#).

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 East (N. Virginia) 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

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 AWS OpsWorks Stacks \(p. 411\)](#) provides some more advanced examples, and shows how to implement cookbooks for AWS OpsWorks Stacks.

Implementing Cookbooks for AWS OpsWorks Stacks

[Cookbook Basics \(p. 383\)](#) 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 AWS OpsWorks Stacks instances. To implement more sophisticated cookbooks for AWS OpsWorks Stacks, you typically need to take full advantage of the AWS OpsWorks Stacks environment, which differs from standard Chef in a number of ways.

This topic describes the basics of implementing recipes for AWS OpsWorks Stacks instances.

Tip

If you are not familiar with how to implement cookbooks, you should start with [Cookbook Basics \(p. 383\)](#).

Topics

- [Running a Recipe on an AWS OpsWorks Stacks Linux Instance \(p. 412\)](#)
- [Running a Recipe on a Windows Instance \(p. 416\)](#)
- [Running a Windows PowerShell Script \(p. 420\)](#)
- [Mocking the Stack Configuration and Deployment Attributes on Vagrant \(p. 422\)](#)
- [Using Stack Configuration and Deployment Attribute Values \(p. 425\)](#)
- [Using an External Cookbook on a Linux Instance: Berkshelf \(p. 437\)](#)
- [Using the SDK for Ruby: Downloading Files from Amazon S3 \(p. 442\)](#)
- [Installing Windows Software \(p. 451\)](#)
- [Overriding Built-In Attributes \(p. 456\)](#)
- [Overriding Built-In Templates \(p. 459\)](#)

[Running a Recipe on an AWS OpsWorks Stacks Linux Instance](#)

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 AWS OpsWorks Stacks instance. This topic describes how to install a custom cookbook on an AWS 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 \(p. 416\)](#).

Topics

- [Creating and Running the Recipe \(p. 412\)](#)
- [Executing the Recipe Automatically \(p. 414\)](#)
- [Troubleshooting and Fixing Recipes \(p. 415\)](#)

[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 \(p. 120\)](#).

To create a stack

1. Open the [AWS 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 \(p. 157\)](#) 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 \(p. 168\)](#) with default settings to the layer and [start it \(p. 178\)](#).

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 \(p. 393\)](#), 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. AWS OpsWorks Stacks provides an easy way to view these logs, as described later.

Tip

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 AWS OpsWorks Stacks instance, you must store it in a repository and provide AWS 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 \(p. 235\)](#).

Tip

For simplicity, this example just archives the entire `opsworks_cookbooks` directory. However, it means that AWS 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`.

6. [Upload the archive to an Amazon S3 bucket, make the archive public, and record the archive's URL.](#) It should look something like `https://s3.amazonaws.com/cookbook_bucket/opsworks_cookbooks.zip`.

You can now install the cookbook and run the recipe.

To run the recipe

1. [Edit the stack to enable custom cookbooks \(p. 249\)](#), 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 \(p. 133\)](#), 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 \(p. 635\)](#). 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 \(p. 212\)](#) 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. AWS OpsWorks Stacks simplifies running recipes on your instance by supporting a set of [lifecycle events \(p. 253\)](#) for each layer: Setup, Configure, Deploy, Undeploy, and Shutdown. You can have AWS 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, AWS OpsWorks Stacks will run `opstest::default`.

4. After the `opstest2` instance is online, verify that `/srv/www/shared` is present.

Tip

If you have assigned recipes to the Setup, Configure, or Deploy events, you also run them manually by using a [stack command \(p. 133\)](#) (Setup and Configure) or a [deploy command \(p. 221\)](#) (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 \(p. 633\)](#).

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. AWS OpsWorks Stacks stores your cookbook in `/opt/aws/opsworks/current/site-cookbooks`, so navigate to `/opt/aws/opsworks/current/site-cookbooks/opstest/recipes`.

Tip

AWS 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, AWS 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.

Tip

If you have assigned your recipe to a lifecycle event so AWS 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 AWS 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 \(p. 133\)](#) to trigger Setup or Configure events.
- Use a [deploy command \(p. 221\)](#) to trigger Deploy or Undeploy events.

Running a Recipe on a Windows Instance

This topic is basically an abbreviated version of [Running a Recipe on a Linux Instance \(p. 412\)](#), 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 \(p. 412\)](#) 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 AWS OpsWorks Stacks Linux instances, see [Running a Recipe on a Linux Instance \(p. 412\)](#).

Topics

- [Enabling RDP Access \(p. 416\)](#)
- [Creating and Running the Recipe \(p. 417\)](#)
- [Executing the Recipe Automatically \(p. 419\)](#)

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, AWS OpsWorks Stacks creates a set of security groups. They include one named something like `AWS-OpsWorks-RDP-Server`, which AWS 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 \(p. 214\)](#).

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 \(p. 120\)](#).

Create a stack

1. Open the [AWS 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 \(p. 157\)](#) to the stack with the following settings.
 - **Name** – RecipeTest
 - **Short name** – recipetest
3. [Add a 24/7 instance \(p. 168\)](#) with default settings to the RecipeTest layer and [start it \(p. 178\)](#).

AWS 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.

Tip

You can also log in as Administrator, but it requires a different procedure. For more information, see [Logging In with RDP \(p. 214\)](#).

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 \(p. 389\)](#), modified for Windows.

Note

When implementing cookbooks for AWS OpsWorks Stacks Windows instances, you use a somewhat different directory structure than you do when implementing cookbooks for AWS OpsWorks Stacks Linux instances. For more information, see [Cookbook Repositories \(p. 235\)](#).

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 AWS OpsWorks Stacks instance, you must store it in a repository and provide AWS 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 \(p. 235\)](#).

6. [Upload the archive to an S3 bucket, make the archive public](#), and record the archive's URL. It should look something like `https://s3-us-west-2.amazonaws.com/opsworks-windows/opsworks_cookbooks.zip`. You can also use a private archive, but a public archive is sufficient for this example and somewhat easier to work with.

You can now install the cookbook and run the recipe.

To run the recipe

1. [Edit the stack to enable custom cookbooks \(p. 249\)](#) and specify the following settings.
 - **Repository type – S3 Archive**
 - **Repository URL** – The cookbook archive URL that you recorded earlierAccept the default values for the other settings and choose **Save** to update the stack configuration.
2. [Run the Update Custom Cookbooks stack command \(p. 133\)](#), 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 \(p. 133\)](#) 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 \(p. 635\)](#). 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 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 \(p. 214\)](#).

Tip

If your recipe isn't working properly, see [Troubleshooting and Fixing Recipes \(p. 415\)](#) 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 AWS 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. AWS OpsWorks Stacks simplifies running recipes on your instance by supporting a set of [lifecycle events \(p. 253\)](#) for each layer: Setup, Configure, Deploy, Undeploy, and Shutdown. You can have AWS 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, AWS OpsWorks Stacks will run `windowstest::default`.

4. After the `recipetest2` instance is online, verify that `c:\data` is present.

Tip

If you have assigned recipes to the Setup, Configure, or Deploy events, you can also run them manually by using a [stack command \(p. 133\)](#) (Setup and Configure) or a [deploy](#)

[command \(p. 221\)](#) (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

Note

These examples assume that you have already done the [Running a Recipe on a Windows Instance \(p. 416\)](#) example. If not, you should do that example first. In particular, it describes how to [enable RDP access \(p. 416\)](#) 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 \(p. 120\)](#).

Create a stack

1. Open the [AWS 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 \(p. 157\)](#) to the stack with the following settings.
 - **Name** – PowerShell
 - **Short name** – powershell
3. [Add a 24/7 instance \(p. 168\)](#) to with default settings to the PowerShell layer and start it (p. 178).
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 \(p. 389\)](#), 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. It should look something like `https://s3-us-west-2.amazonaws.com/opsworks-windows/powershell.zip`. You can also use a private archive, but a public archive is sufficient for this example, and somewhat easier to work with.

You can now install the cookbook and run the recipe.

To run the recipe

1. [Edit the stack to enable custom cookbooks \(p. 249\)](#) 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 \(p. 133\)](#) 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 \(p. 133\)](#) with **Recipes to execute** set to `powershell::default`.

Tip

This example uses **Execute Recipes** for convenience, but you typically have AWS OpsWorks Stacks [run your recipes automatically \(p. 255\)](#) 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 \(p. 221\)](#) 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 \(p. 635\)](#). 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 \(p. 214\)](#) and open the **Start** menu. XPS Viewer should be listed with **Windows Accessories**.

Mocking the Stack Configuration and Deployment Attributes on Vagrant

Note

This topic applies only to Linux instances. Test Kitchen does not yet support Windows, so you will run all Windows examples on AWS OpsWorks Stacks instances.

AWS OpsWorks Stacks adds [stack configuration and deployment attributes \(p. 376\)](#) 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 AWS OpsWorks Stacks instances use one or more of these attributes.

An instance running in a Vagrant box is not managed by AWS 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 AWS 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.

Tip

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 \(p. 387\)](#).
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 AWS 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 \(p. 313\)](#), 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 \(p. 313\)](#), including deploying SimplePHPApp. If you prefer, you can omit the second PHP App Server instance called for in [Step 4: Scale Out MyStack \(p. 329\)](#); 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 \(p. 212\)](#).
3. In the terminal window, run the following [agent cli \(p. 651\)](#) 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 AWS 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 AWS 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

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, AWS 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.

Tip

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 \(p. 226\)](#).

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.

Tip

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 \(p. 422\)](#).

Topics

- [Obtaining Attribute Values Directly \(p. 426\)](#)
- [Obtaining Attribute Values with Chef Search \(p. 429\)](#)

Obtaining Attribute Values Directly

Note

This approach works only for Linux stacks.

[Mocking the Stack Configuration and Deployment Attributes on Vagrant \(p. 422\)](#) 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 \(p. 387\)](#).
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": "192.0.2.6"  
          }  
        }  
      }  
    }  
  }  
}
```

```

        "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 : ef36017-ec42-4423-b655-53e4d3710652] action
    write[2014-07-17T22:56:19+00:00] INFO: Processing log[PHP App Server : ef36017-ec42-4423-
b655-53e4d3710652] action write (listip::default line 4)
  [2014-07-17T22:56:19+00:00] INFO: PHP App Server : ef36017-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 : ef36017-ec42-4423-b655-53e4d3710652] action
  write[2014-07-17T23:09:34+00:00] INFO: Processing log[PHP App Server : ef36017-ec42-4423-
b655-53e4d3710652] action write (listip::default line 2)
  [2014-07-17T23:09:34+00:00] INFO: PHP App Server : ef36017-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-blea-6d44c7841193] action
      write[2014-07-17T23:09:34+00:00] INFO: Processing log[HAProxy : d5c4dda9-2888-4b22-
      blea-6d44c7841193] action write (listip::default line 2)
[2014-07-17T23:09:34+00:00] INFO: HAProxy : d5c4dda9-2888-4b22-blea-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.

Tip

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 \(p. 369\)](#).

Obtaining Attribute Values with Chef Search

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 AWS OpsWorks Stacks. Because AWS 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 AWS OpsWorks Stacks instance includes the [stack configuration and deployment attributes \(p. 376\)](#). 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 AWS OpsWorks Stacks instances without modification. For more information, see [Using Chef Search \(p. 241\)](#).

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 \(p. 430\)](#)
- [Using Search on a Windows Stack \(p. 432\)](#)

Using Search on a Linux Stack

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 \(p. 426\)](#), 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 \(p. 120\)](#).

Tip

If you have not run a custom recipe on an AWS OpsWorks Stacks instance before, you should first go through the [Running a Recipe on a Linux Instance \(p. 412\)](#) example.

Create a stack

1. Open the [AWS 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 \(p. 157\)](#) to the stack with default settings.
4. [Add a 24/7 instance \(p. 168\)](#) with default settings to the layer and [start it \(p. 178\)](#).

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. It should look something like `https://s3.amazonaws.com/cookbook_bucket/opsworks_cookbooks.zip`. For more information on cookbook repositories, see [Cookbook Repositories \(p. 235\)](#).

You can now install the cookbook and run the recipe.

To run the recipe

1. [Edit the stack to enable custom cookbooks \(p. 249\)](#), 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` \(p. 255\)](#) to the layer's Setup event. AWS 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 \(p. 133\)](#), 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 \(p. 635\)](#) 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 \(p. 212\)](#) 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

AWS 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 AWS OpsWorks Stacks stacks without modification.

- An additional set of search indexes that can be used to query sets of AWS OpsWorks Stacks-specific attributes, and some standard attributes.

These indexes are discussed in [Using AWS OpsWorks Stacks-Specific Search Indexes on Windows Stacks \(p. 435\)](#).

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 AWS OpsWorks Stacks search indexes to retrieve information that is specific to AWS OpsWorks Stacks.

Topics

- [Using the node Search Index on Windows Stacks \(p. 432\)](#)
- [Using AWS OpsWorks Stacks-Specific Search Indexes on Windows Stacks \(p. 435\)](#)

Using the node Search Index on Windows Stacks

Note

This example assumes that you have already done the [Running a Recipe on a Windows Instance \(p. 416\)](#) 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 \(p. 120\)](#).

Create a stack

1. Open the [AWS 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 \(p. 157\)](#) to the stack with the following settings.
 - **Name** – IPTest
 - **Short name** – iptest
 4. [Add a 24/7 t2.micro instance \(p. 168\)](#) with default settings to the IPTest layer and [start it \(p. 178\)](#). It will be named iptest1.

AWS 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.

Tip

You also can log in as Administrator, but it requires a different procedure. For more information, see [Logging In with RDP \(p. 214\)](#).

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 \(p. 389\)](#).

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. It should look something like `https://s3.amazonaws.com/cookbook_bucket/nodesearch.zip`. For more information on cookbook repositories, see [Cookbook Repositories \(p. 235\)](#).

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 \(p. 249\)](#) 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 \(p. 133\)](#), 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 \(p. 133\)](#) 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 \(p. 635\)](#) 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 (p. 214) and examine the contents of `C:\tmp\addresses.txt`.

Using AWS OpsWorks Stacks-Specific Search Indexes on Windows Stacks

Note

This example assumes that you have already done the [Running a Recipe on a Windows Instance \(p. 416\)](#) example. If not, you should do that example first. In particular, it describes how to enable RDP access to your instances.

AWS 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 AWS OpsWorks Stacks-specific attributes. For example `aws_opsworks_instance` includes a `status` attribute that provides the instance's status, such as `online`.

Tip

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 \(p. 432\)](#).

This example shows how to use the AWS OpsWorks Stacks indexes to retrieve the value of an AWS 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 AWS 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 \(p. 120\)](#).

Create a stack

1. Open the [AWS 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 \(p. 157\)](#) to the stack with the following settings.
 - **Name** – IDCheck
 - **Short name** – idcheck
3. [Add a 24/7 t2.micro instance \(p. 168\)](#) with default settings to the IDCheck layer and [start it \(p. 178\)](#). It will be named `iptest1`.

AWS OpsWorks Stacks automatically assigns `AWS-OpsWorks-RDP-Server` to this instance. [Enabling RDP Access \(p. 416\)](#) 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.

Tip

You can also log in as Administrator, but it requires a different procedure. For more information, see [Logging In with RDP \(p. 214\)](#).

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 \(p. 666\)](#) 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 \(aws_opsworks_instance\) \(p. 666\)](#).

The instance attributes are basically a hash table, and the instance's AWS 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. It should look something like `https://s3-us-west-2.amazonaws.com/windows-cookbooks/ipaddress.zip`. For more information on cookbook repositories, see [Cookbook Repositories \(p. 235\)](#).

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 \(p. 249\)](#) 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 \(p. 133\)](#), 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 \(p. 133\)](#) 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 \(p. 635\)](#) for the most recent execute_recipes event. On the **Running command execute_recipes page**, choose **Show** in the ipstest1 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

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 AWS 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 AWS 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 AWS 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 \(p. 437\)](#)
- [Using Berkshelf with AWS OpsWorks Stacks \(p. 440\)](#)

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.

Tip

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 AWS 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 AWS 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 AWS OpsWorks Stacks instance.

Using Berkshelf with AWS OpsWorks Stacks

AWS 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.

AWS 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 AWS 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 \(p. 243\)](#).

This example shows how to use Berkshelf to install the getting-started community cookbook on an AWS 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 \(p. 397\)](#).

Tip

If this is the first time you have installed a custom cookbook on an AWS OpsWorks Stacks stack, you should first go through the [Running a Recipe on a Linux Instance \(p. 412\)](#) example.

Start by creating a stack, as summarized in the following. For more information, see [Create a New Stack \(p. 120\)](#).

Create a stack

1. Open the [AWS 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 \(p. 157\)](#) 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 \(p. 168\)](#) to the BerksTest layer with default settings, but don't start it yet.

With AWS OpsWorks Stacks, cookbooks must be in a remote repository with a standard directory structure. You then provide the download information to AWS 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 AWS OpsWorks Stacks also supports HTTP archives, Git repositories, and Subversion repositories. For more information, see [Cookbook Repositories \(p. 235\)](#).

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. It should look something like `https://s3.amazonaws.com/cookbook_bucket/berkstest_cookbooks.zip`.

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 ([p. 249](#)), 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 AWS 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 \(p. 255\)](#).
 - `getting-started::default`
 - `createfile::default`
3. [Start the instance \(p. 178\)](#). The Setup event occurs after the instance finishes booting. AWS 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 \(p. 212\)](#). You should see the following
 - `/srv/www/shared` should contain `example_data.json`.
 - `/root` should contain `chef-getting-started.txt`.

AWS 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

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.

Tip

For more information about how to use the [AWS SDK for Ruby](#) to handle encryption and decryption, see [AWS::S3::S3Object](#).

Topics

- [Using the SDK for Ruby on a Vagrant Instance \(p. 442\)](#)
- [Using the SDK for Ruby on an AWS OpsWorks Stacks Linux Instance \(p. 446\)](#)
- [Using the SDK for Ruby on an AWS OpsWorks Stacks Windows Instance \(p. 448\)](#)

Using the SDK for Ruby on a Vagrant Instance

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 an IAM user with an appropriate policy and provide those credentials to the recipe. For more information, see [Best Practices for Managing AWS Access Keys](#).

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 AWS OpsWorks Stacks Linux Instance \(p. 446\)](#).

If you don't already have an appropriate IAM user, you can create one as follows. For more information, see [What Is IAM](#).

To create an IAM user

1. Sign in to the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Users** and, if necessary, choose **Create New Users** to create a new IAM user for the administrative user.
3. Type a user name, select **Generate an access key for each user**, and choose **Create**.
4. Choose **Download Credentials**, save the credentials file to a convenient location on your system, and choose **Close**.
5. On the **Users** page, choose the new user's name and then choose **Attach Policy**.
6. Type `s3` in the **Policy Type** search box to display the Amazon S3 policies. Select **AmazonS3ReadOnlyAccess** and choose **Attach Policy**. 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.

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 \(p. 387\)](#).
3. Replace the text in `.kitchen.yml` with the following.

```
---  
driver:  
  name: vagrant
```

```

provisioner:
  name: chef_solo
  environments_path: ./environments

platforms:
  - name: ubuntu-12.04

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 IAM user. Save the file to the cookbook's `environments` folder.

```
{
  "default_attributes" : {
    "cookbooks_101" : {
      "access_key": "AKIAIOSFODNN7EXAMPLE",
      "secret_key" : "wJalrXUtnFEMI/K7MDENG/bPxRfCYEXAMPLEKEY"
    }
  },
  "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](#).

Tip

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.txt` from the `cookbook_bucket` bucket.

```

gem_package "aws-sdk" do
  action :install
end

ruby_block "download-object" do
  block do
    require 'aws-sdk'

    s3 = AWS::S3.new(
      :access_key_id => "#{node['cookbooks_101']['access_key']}",
      :secret_access_key => "#{node['cookbooks_101']['secret_key']}"
    )

    myfile = s3.buckets['cookbook_bucket'].objects['myfile.txt']
  end
end

```

```
Dir.chdir("/tmp")
File.open("myfile.txt", "w") do |f|
  f.write(myfile.read)
  f.close
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.

Tip

Your instance will usually have 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 the gem package is to be used by applications or recipes, 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.

- 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.

- Calls the `s3` object's `buckets.objects` method, which returns an [AWS::S3::S3Object](#) object named `myfile` that represents `myfile.txt`.
- Uses `Dir.chdir` to set the working directory to `/tmp`.
- Opens a file named `myfile.txt`, writes the contents of `myfile` to the file, and closes the file.

To run the recipe

- Create a file named `default.rb` with the example recipe and save it to the `recipes` directory.
- Run `kitchen converge`.
- 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 AWS OpsWorks Stacks Linux Instance

This topic describes how to use the SDK for Ruby on an AWS OpsWorks Stacks Linux instance to download a file from an Amazon S3 bucket. AWS 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.

[Using the SDK for Ruby on a Vagrant Instance \(p. 442\)](#) 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 \(p. 442\)](#) 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 \(p. 407\)](#). However, installing the SDK for Ruby on Amazon EC2 instances is somewhat complicated and not something you need to be concerned with for AWS OpsWorks Stacks. All AWS OpsWorks Stacks Linux instances have the SDK for Ruby installed by default. For simplicity, the example therefore uses an AWS 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 AWS OpsWorks Stacks creates when you create your first stack. It is named `aws-opsworks-ec2-role`. However, AWS OpsWorks Stacks does not attach a policy to that role, so by default it grants no permissions. You must attach a policy to the role that grants appropriate permissions, in this case, read-only permissions for Amazon S3. The procedure is much like the one you used to attach a policy to an IAM user in the preceding section.

To attach a policy to a role

1. Open the [IAM console](#) and click **Roles** in the navigation pane.
2. Click `aws-opsworks-ec2-role` and under **Permissions**, select **Attach Policy**.
3. Type `s3` in the **Policy Type** search box to display the Amazon S3 policies. Select `AmazonS3ReadOnlyAccess` and click **Attach Policy**.

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 \(p. 412\)](#), with one addition. On the **Add Stack** page, confirm that **Default IAM instance profile** is set to `aws-opsworks-ec2-role`. AWS 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 \(p. 412\)](#). 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 \(p. 235\)](#).

This recipe is similar the one used by the previous example, with the following exceptions.

- Because AWS 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. For more information, see [Using IAM Roles for Amazon EC2 Instances with the AWS SDK for Ruby](#).

- 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 [AWS 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 \(p. 157\)](#) 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 (p. 168) with default settings to the layer and start it (p. 178).

You can now install and run the recipe

To run the recipe

1. Edit the stack to enable custom cookbooks (p. 249), 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 (p. 133), 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`.

Tip

You typically have AWS OpsWorks Stacks run your recipes automatically (p. 255) 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 (p. 221) 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 (p. 212) and list the contents of `/tmp`.

Using the SDK for Ruby on an AWS OpsWorks Stacks Windows Instance

Note

This example assumes that you have already done the [Running a Recipe on a Windows Instance \(p. 416\)](#) example. If not, you should do that example first. In particular, it describes how to enable RDP access to your instances.

This topic describes how to use the [AWS SDK for Ruby](#) on an AWS 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 AWS OpsWorks Stacks creates when you create your first stack. It is named `aws-opsworks-ec2-role`. However, AWS OpsWorks Stacks does not attach a policy to that role, so by default it grants no permissions. You must attach a policy that grants appropriate permissions to the role, in this case, read-only permissions for Amazon S3.

To attach a policy to a role

1. Open the [IAM console](#) and choose **Roles** in the navigation pane.
2. Choose `aws-opsworks-ec2-role` and, under **Permissions**, choose **Attach Policy**.
3. Type `s3` in the **Policy Type** search box to display the Amazon S3 policies. Choose `AmazonS3ReadOnlyAccess` and choose **Attach Policy**.

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 \(p. 416\)](#), with one addition. On the **Add Stack** page, confirm that **Default IAM instance profile** is set to `aws-opsworks-ec2-role`. AWS 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 \(p. 412\)](#). 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" do
  compile_time false
  action :install
end

ruby_block "download-object" do
  block do
    require 'aws-sdk'

    Aws.config[:ssl_ca_bundle] = 'C:\ProgramData\Git\bin\curl-ca-bundle.crt'

    s3_client = Aws::S3::Client.new(region:'us-west-2')
  end
end
```

```

    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. It should look something like `https://s3-us-west-2.amazonaws.com/windows-cookbooks/s3download.zip`. For more information, see [Cookbook Repositories \(p. 235\)](#).
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, AWS 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: Specify a Certificate Bundle.

Amazon S3 uses SSL, so you need an appropriate certificate to download objects from an S3 bucket. The SDK for Ruby v2 does not include a certificate bundle, so you must provide one and configure the SDK for Ruby to use it. AWS OpsWorks Stacks does not install a certificate bundle directly, but it does install Git, which includes a certificate bundle (`curl-ca-bundle.crt`). For simplicity, this example configures the SDK for Ruby to use the Git certificate bundle, but you can also install your own and configure the SDK accordingly.

3: 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 [AWS 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 \(p. 157\)](#) to the stack with the following settings.

- **Name** – S3Download
 - **Short name** – s3download
3. [Add a 24/7 instance \(p. 168\)](#) with default settings to the S3Download layer and [start it \(p. 178\)](#).

You can now install and run the recipe

To run the recipe

1. [Edit the stack to enable custom cookbooks \(p. 249\)](#), 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 \(p. 133\)](#), 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`.

Tip

You typically have AWS OpsWorks Stacks [run your recipes automatically \(p. 255\)](#) 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 \(p. 221\)](#) 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 \(p. 214\)](#) and examine the contents of `c:\chef`.

Installing Windows Software

Note

These examples assume that you have already done the [Running a Recipe on a Windows Instance \(p. 416\)](#) 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 Server (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 \(p. 452\)](#)
- [Installing a Package on a Windows Instance \(p. 454\)](#)

[Installing a Windows Feature: IIS](#)

Windows features are a set of optional system components, including the .NET frameworks and Internet Information Server (IIS). This topic describes how to implement a cookbook to install a commonly used feature, Internet Information Server (IIS).

Tip

[Installing a Package \(p. 454\)](#) 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 \(p. 416\)](#) 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.

Tip

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).

Tip

For a complete list of available Windows features, [use RDP to log in to the instance \(p. 214\)](#), 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 \(p. 235\)](#).

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 [AWS 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 \(p. 157\)](#) to the stack with the following settings.

- **Name** – `IIS`
- **Short name** – `iis`

3. [Add a 24/7 instance \(p. 168\)](#) with default settings to the IIS layer and [start it \(p. 178\)](#).

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 \(p. 249\)](#), 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 \(p. 133\)](#), 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.

Tip

This example uses **Execute Recipes** for convenience, but you typically have AWS OpsWorks Stacks [run your recipes automatically \(p. 255\)](#) 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 \(p. 221\)](#) to trigger Deploy and Undeploy events.

4. To verify the installation, [use RDP to connect to the instance \(p. 214\)](#) 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

Note

This example assumes that you have already done the [Running a Recipe on a Windows Instance \(p. 416\)](#) 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 \(p. 452\)](#).

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 \(p. 389\)](#).

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 \(p. 397\)](#).

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 \(p. 235\)](#).

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 [AWS 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 \(p. 157\)](#) to the stack with the following settings.
 - **Name** – Python
 - **Short name** – python
 3. [Add a 24/7 instance \(p. 168\)](#) with default settings to the Python layer and [start it \(p. 178\)](#).

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 \(p. 249\)](#), 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 \(p. 133\)](#), 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`.

Tip

This example uses **Execute Recipes** for convenience, but you typically have AWS OpsWorks Stacks [run your recipes automatically \(p. 255\)](#) 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 \(p. 221\)](#) to trigger Deploy and Undeploy events.

4. To verify the installation, [use RDP to connect to the instance \(p. 214\)](#) 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

Note

This topic applies only to Linux stacks. You cannot override built-in attributes on Windows stacks.

AWS 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 \(p. 348\)](#). For a general discussion of how to override attributes, see [Overriding Attributes \(p. 346\)](#).

Tip

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 \(p. 459\)](#).

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
...
```

Tip

For more information about commonly used built-in attributes, see [Built-in Cookbook Attributes \(p. 532\)](#).

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 AWS 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 \(p. 347\)](#).

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 \(p. 235\)](#).

To use the custom attribute, create a stack and install the cookbook.

To use the custom attribute

1. Open the [AWS 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.

Tip

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 \(p. 484\)](#) with default settings to the stack.
4. [Add a 24/7 instance \(p. 168\)](#) 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 \(p. 212\)](#). 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.erb` 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

Note

This topic applies only to Linux stacks. You cannot override built-in templates on Windows stacks.

The AWS 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 \(p. 456\)](#). 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 \(p. 351\)](#).

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 AWS OpsWorks Stacks updates the built-in template, the custom template becomes out of sync and might not work correctly. AWS OpsWorks Stacks doesn't make such changes often, and when a template does change, AWS OpsWorks Stacks lists the changes and gives you the option of upgrading to a new version. We recommend that you monitor the [AWS 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.

Tip

If you have already completed the [Overriding Built-In Attributes \(p. 456\)](#) 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 and Ubuntu 12.04 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 \(p. 235\)](#).

Tip

For simplicity, this example adds a hardcoded error message to the template. To change it, you must modify the template and [reinstall the cookbook \(p. 251\)](#). To give yourself greater flexibility, you can [define a default custom attribute \(p. 456\)](#) 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 \(p. 348\)](#), 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 [AWS 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 \(p. 484\)](#) to the stack with default settings.
4. [Add a 24/7 instance \(p. 168\)](#) 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 \(p. 212\)](#). 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

AWS 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 AWS 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 \(p. 351\)](#). 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 AWS OpsWorks Stacks

Because AWS OpsWorks Stacks is based on Chef, migrating from Chef Server to AWS OpsWorks Stacks is relatively straightforward. This topic provides guidelines for modifying Chef Server code to work with AWS 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 \(p. 464\)](#)
- [Using Data Bags \(p. 464\)](#)
- [Using Chef Search \(p. 465\)](#)
- [Managing Cookbooks and Recipes \(p. 465\)](#)
- [Using Chef Environments \(p. 467\)](#)

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 [AWS OpsWorks Stacks layer \(p. 138\)](#) 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.

AWS OpsWorks Stacks includes a set of [built-in layers \(p. 138\)](#) 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 \(p. 484\)](#) layer is a blueprint for creating instances that host a Tomcat server.

To migrate to AWS 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 \(p. 138\)](#) and [AWS OpsWorks Stacks Layer Reference \(p. 467\)](#). To examine the built-in recipes, see the [AWS 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 \(p. 346\)](#) or [templates \(p. 351\)](#).

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 \(p. 352\)](#). 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 \(p. 356\)](#).

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.

AWS 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.

- AWS 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 \(p. 442\)](#).

For more information, see [Using Data Bags \(p. 242\)](#).

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. AWS 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, AWS OpsWorks Stacks adds a set of [stack configuration and deployment attributes \(p. 376\)](#) 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 AWS OpsWorks Stacks. Because Chef search operates on the node object, which includes the stack configuration and deployment attributes, search queries in AWS 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 AWS 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 AWS 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 \(p. 241\)](#).

For compatibility with Chef Server, AWS 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

AWS 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 AWS OpsWorks Stacks:

- AWS 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 AWS OpsWorks Stacks cookbooks in a remote repository, such as an S3 bucket or a Git repository.

For more information, see [Storing Cookbooks \(p. 466\)](#).

- You can [execute recipes manually \(p. 255\)](#), but you typically have AWS OpsWorks Stacks execute recipes for you in response to a set of [lifecycle events \(p. 253\)](#) that occur at key points during an instance's lifecycle.

For more information, see [Executing Recipes \(p. 466\)](#).

- AWS 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 \(p. 243\)](#).

Topics

- [Storing Cookbooks \(p. 466\)](#)
- [Executing Recipes \(p. 466\)](#)

Storing Cookbooks

With Chef Server, you store your cookbooks on the server and deploy them from the server to the instances. With AWS OpsWorks Stacks, you store cookbooks in a repository—an S3 or HTTP archive or a Git or Subversion repository. You specify the information that AWS OpsWorks Stacks needs to download the code from the repository to a stack's instances when you [install cookbooks \(p. 217\)](#).

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 \(p. 235\)](#).

Executing Recipes

In AWS OpsWorks Stacks, each layer has a set of [lifecycle events \(p. 253\)](#)—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, AWS 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 \(p. 133\)](#). 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 AWS OpsWorks Stacks console, you can use the [AWS CLI](#) or [SDKs](#) to execute recipes. These tools support all of the [AWS 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 \(p. 354\)](#).

- Add a task to your continuous integration pipeline that uses the `create-deployment` CLI command to deploy the app.

Using Chef Environments

AWS OpsWorks Stacks does not support Chef environments; `node.chef_environment` always returns `_default`.

AWS OpsWorks Stacks Layer Reference

Every instance that AWS 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 AWS OpsWorks Stacks to create and manage layers, see [Layers \(p. 138\)](#).

Each layer description includes a list of the built-in recipes that AWS 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 AWS 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 \(p. 635\)](#).

Topics

- [HAProxy Layer Reference \(p. 467\)](#)
- [HAProxy AWS OpsWorks Stacks Layer \(p. 469\)](#)
- [MySQL Layer Reference \(p. 473\)](#)
- [MySQL OpsWorks Layer \(p. 474\)](#)
- [Application Server Layers Reference \(p. 475\)](#)
- [Application Server Layers \(p. 482\)](#)
- [ECS Cluster Layer Reference \(p. 497\)](#)
- [Custom Layer Reference \(p. 499\)](#)
- [Other Layers Reference \(p. 500\)](#)
- [Other Layers \(p. 502\)](#)

HAProxy Layer Reference

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, AWS 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:

- AWS 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 AWS OpsWorks Stacks Layer

Note

This layer is available only for Chef 11 and earlier Linux-based stacks.

The AWS OpsWorks Stacks HAProxy layer is an AWS OpsWorks Stacks layer that provides a blueprint for instances that host an [HAProxy](#) server—a reliable high-performance TCP/HTTP load balancer. 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://DNSNameStatisticsPath`, 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 \(p. 145\)](#). 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 AWS 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 \(p. 120\)](#).

Add layer

OpsWorks ECS RDS

Layer type **HAProxy**

An HAProxy layer is a blueprint for instances that expose a single IP address to represent a set of application servers. It receives incoming requests, distributes them across the application server instances, and returns responses to the caller. [Learn more](#).

HAProxy statistics **Yes**

Statistics URL	/haproxy?stats
Statistics user name	opsworks
Statistics password	dzrfk9y66r
Health check URL	/
Health check method	OPTIONS

Need further support? [Let us know.](#)

[Cancel](#) [Add layer](#)

Note

Record the password for later use; AWS 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.

Layer HAProxy

[General Settings](#) Recipes Network EBS Volumes Security

Settings

HAProxy statistics **Yes**

Statistics URL	/haproxy?stats
Statistics user name	opsworks
Statistics password	Update password
Health check URL	/
Health check method	OPTIONS

Instance shutdown timeout 120

Auto healing enabled **Yes**

Custom JSON **Optional**

Enter custom JSON that is passed to your Chef recipes for all instances in this layer. You can use this to override and customize built-in recipes or pass variables to your own recipes. [Learn more](#).

[Cancel](#) [Save](#)

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, AWS 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.

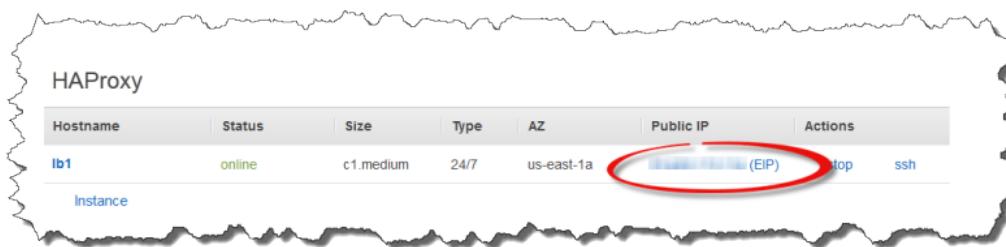
If the method does not return within a specified timeout period, the instance is presumed to have failed and HAProxy stops routing requests to the instance. AWS OpsWorks Stacks also provides a way to automatically replace failed instances. For more information, see [Using Auto Healing \(p. 145\)](#). You can change the health check method when you create the layer.

- Collects statistics and optionally displays them on a web page.

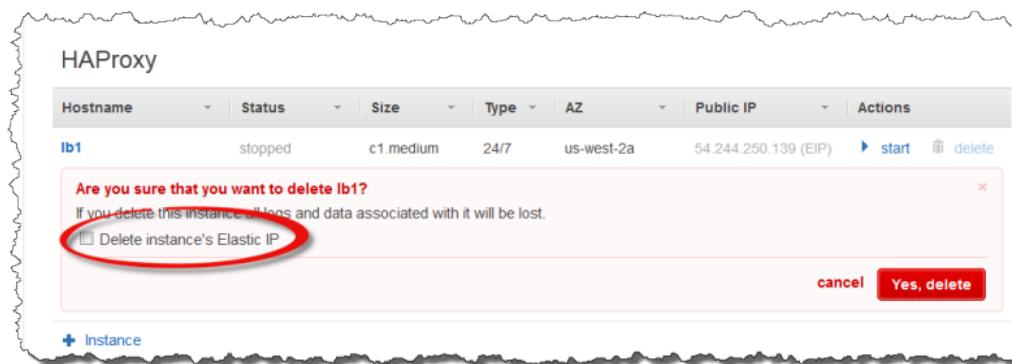
Important

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, AWS 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, AWS 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, AWS 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, AWS OpsWorks Stacks assigns the address to the new instance.
- Otherwise, AWS 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 \(p. 181\)](#) or [auto healing \(p. 145\)](#)—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, AWS OpsWorks Stacks triggers a Configure [lifecycle event \(p. 253\)](#). 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, AWS 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.

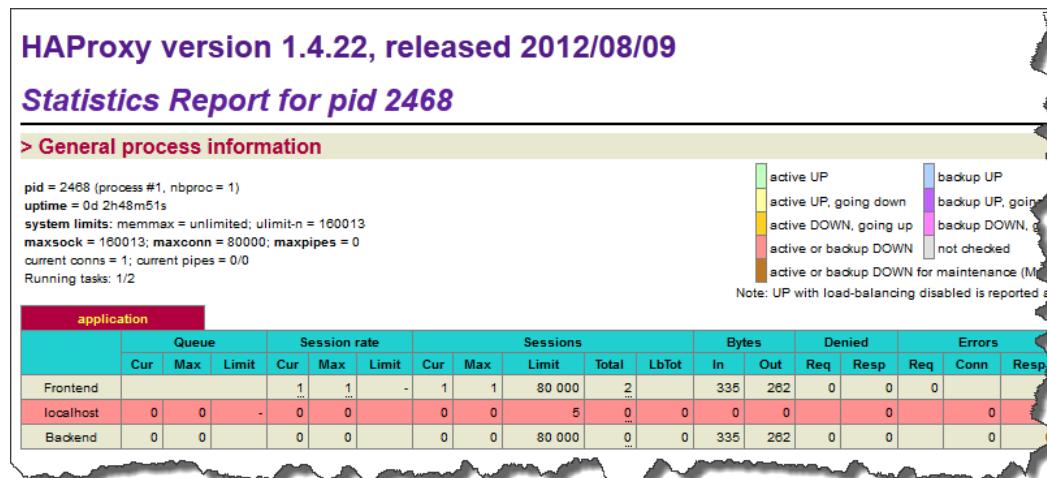
Finally, you can also use a custom domain with the Haproxy layer. For more information, see [Using Custom Domains \(p. 229\)](#).

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 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

Note

This layer is available only for Linux-based stacks.

The MySQL layer supports MySQL, a widely used relational database management system. AWS 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:

- AWS 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

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**, AWS 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 \(p. 120\)](#).

Add layer

OpsWorks ECS RDS

Layer type: MySQL

A MySQL Master layer is a blueprint for instances that function as MySQL relational database servers. [Learn more.](#)

MySQL root user password: 93gpv3rf6r

Set root user password on every instance: Yes

Need further support? [Let us know.](#)

Cancel Add layer

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 \(p. 217\)](#), 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 \(p. 224\)](#).

Application Server Layers Reference

AWS OpsWorks Stacks supports several different application and static web page servers.

Topics

- [AWS Flow \(Ruby\) Layer Reference \(p. 475\)](#)
- [Java App Server Layer Reference \(p. 476\)](#)
- [Node.js App Server Layer Reference \(p. 477\)](#)
- [PHP App Server Layer Reference \(p. 479\)](#)
- [Rails App Server Layer Reference \(p. 480\)](#)
- [Static Web Server Layer Reference \(p. 481\)](#)

AWS Flow (Ruby) Layer Reference

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 AWS 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

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](#)

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](#).

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 AWS OpsWorks Stacks, you must explicitly specify port 80, as follows:

```
app.set('port', 80);
```

[PHP App Server Layer Reference](#)

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:

- AWS 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](#)

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:

- AWS 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](#)

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

Note

These layers are available only for Chef 11 and earlier Linux-based stacks.

AWS OpsWorks Stacks supports several different application servers, where "application" includes static web pages. Each type of server has a separate AWS 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 (p. 139) one of the available **App Server** layer types.
2. Add one or more instances (p. 168) to the layer.
3. Create apps and deploy them to the instances. For more information, see [Apps \(p. 216\)](#).
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 AWS OpsWorks Stacks Layer \(p. 469\)](#).

Topics

- [AWS Flow \(Ruby\) Layer \(p. 483\)](#)
- [Java App Server AWS OpsWorks Stacks Layer \(p. 484\)](#)
- [Node.js App Server AWS OpsWorks Stacks Layer \(p. 491\)](#)
- [PHP App Server AWS OpsWorks Stacks Layer \(p. 492\)](#)
- [Rails App Server AWS OpsWorks Stacks Layer \(p. 493\)](#)
- [Static Web Server AWS OpsWorks Stacks Layer \(p. 497\)](#)

[AWS Flow \(Ruby\) Layer](#)

Note

This layer is available only for Linux-based stacks.

An AWS Flow (Ruby) layer is an AWS 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 AWS 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 [Using IAM to Manage Access to Amazon SWF Resources](#).

Creating a profile for AWS Flow (Ruby) instances

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Click **Policies** in the navigation pane and click **Create Policy** to create a new customer-managed policy.
3. Click **Select** next to **Policy Generator** and then specify the following policy generator settings:
 - **Effect – Allow.**
 - **AWS Service – Amazon Simple Workflow Service.**
 - **Actions – All Actions (*)**.
 - **Amazon Resource Name (ARN)** – An ARN that specifies which Amazon SWF domains the workers can access. Type * to provide access to all domains.

When you are finished, click **Add Statement, Next Step**, and then **Create Policy**.

4. Click **Roles** in the navigation pane and click **Create New Role**.
5. Specify the role name and click **Next Step**. You cannot change the name after the role has been created.
6. Select **AWS Service Roles** and then select **Amazon EC2**.
7. Click **Customer Managed Policies** from the **Policy Type** list and select the policy that you created earlier.
8. Specify this profile when you create an AWS Flow (Ruby) layer in AWS OpsWorks Stacks.

Tip

For more information on how to use the AWS Flow (Ruby) layer, including a detailed walkthrough that describes how to deploy a sample application, see [Tutorial: Hello AWS OpsWorks!](#).

Java App Server AWS OpsWorks Stacks Layer

Note

This layer is available only for Linux-based stacks.

The Java App Server layer is an AWS 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](#). AWS 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; AWS 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 \(p. 635\)](#).

Custom security groups

This setting appears if you chose to not automatically associate a built-in AWS 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 \(p. 120\)](#).

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 \(p. 147\)](#).

You can specify other configuration settings by using custom JSON or a custom attributes file. For more information, see [Custom Configuration \(p. 486\)](#).

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 \(p. 485\)](#).

Topics

- [Disabling SSLv3 for Apache Servers \(p. 485\)](#)
- [Custom Configuration \(p. 486\)](#)
- [Deploying Java Apps \(p. 486\)](#)

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 \(p. 351\)](#).

Amazon Linux and Ubuntu 12.04 LTS

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, 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, 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, TLSv1.1, 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, TLSv1.1, TLSv1.2
# SSL v2 is no longer supported
SSLProtocol all -SSLv3 -SSLv2
...
```

Custom Configuration

AWS 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 \(p. 135\)](#).

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 \(p. 346\)](#). For a list of built in attributes, see [opsworks_java Attributes \(p. 552\)](#).

Deploying Java Apps

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 AWS 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 \(p. 353\)](#).

```
:Dir.glob(::File.join(release_path, '*.war')) do |archive_file|
  execute "unzip_#{archive_file}" do
    command "unzip #{archive_file}"
    cwd release_path
  end
end
```

Tip

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 \(p. 487\)](#)
- [Deploying a JSP App with a Back-End Database \(p. 489\)](#)

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 \(p. 217\)](#). 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 \(p. 218\)](#).

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 AWS OpsWorks Stacks, you should first see [Getting Started with Chef 11 Linux Stacks \(p. 313\)](#).

To deploy a JSP page from an Amazon S3 archive

1. [Create a stack \(p. 120\)](#) with a Java App Server layer, [add a 24/7 instance \(p. 168\)](#) to the layer, and [start it \(p. 178\)](#).
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 AWS 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 \(p. 217\)](#) 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. It should look something like `https://s3.amazonaws.com/jsp_example/simplejsp.zip`.

Use the default values for the remaining settings and then click **Add App** to create the app.

5. [Deploy the app \(p. 221\)](#) 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 AWS 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 AWS 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>
```

AWS 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 \(p. 120\)](#) with a Java App Server layer, [add 24/7 instances \(p. 168\)](#) to each layer, and [start it \(p. 178\)](#).
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 \(p. 474\)](#) to the stack, [add a 24/7 instance \(p. 168\)](#) to the layer, and [start it \(p. 178\)](#).

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. AWS OpsWorks Stacks creates these security groups for you when you create your first stack in the region.
- Create a database named `simplejspdbs`.
- 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 \(p. 366\)](#).

- [Add a MySQL driver \(p. 224\)](#) to the Java App Server layer.
- [Register the RDS instance \(p. 151\)](#) with your stack.

For more information about how to use Amazon RDS instances with AWS OpsWorks Stacks, see [Amazon RDS Service Layer \(p. 150\)](#).

3. Copy the example code to a file named `simplejspdbs.jsp`, put the file in a folder named `simplejspdbs`, 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 \(p. 218\)](#).
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 \(p. 217\)](#) 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** – `simplejspdbs`.
 - **Repository type** – `Http Archive`

- **Repository URL** – the Amazon S3 URL of your archive file. It should look something like `https://s3.amazonaws.com/jsp_example/simplejspdb.zip`.

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"  
    }  
  }  
}
```

AWS 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 \(p. 135\)](#).

7. [Deploy the app \(p. 221\)](#) 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 \(p. 487\)](#).

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 app's 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/myjspsdb",  
      ...  
    }  
  }  
}
```

[Node.js App Server AWS OpsWorks Stacks Layer](#)

Note

This layer is available only for Linux-based stacks.

The Node.js App Server layer is an AWS OpsWorks Stacks layer that provides a blueprint for instances that function as [Node.js](#) application servers. AWS 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 [AWS OpsWorks Stacks Operating Systems \(p. 159\)](#).

Custom security groups

This setting appears if you chose to not automatically associate a built-in AWS 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 \(p. 120\)](#).

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 AWS OpsWorks Stacks and deploy it to a stack, see [Creating Your First Node.js Stack \(p. 335\)](#). In general, Node.js applications for AWS 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 \(p. 144\)](#). Do not modify inbound rules by editing the built-in security group. Each time you create a stack, AWS OpsWorks Stacks overwrites the built-in security groups with the standard settings, so any changes that you make will be lost.

Tip

AWS 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 \(p. 221\)](#), you must specify the key and certificates. AWS 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 AWS OpsWorks Stacks Layer

Note

This layer is available only for Linux-based stacks.

The PHP App Server layer is an AWS 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 \(p. 159\)](#) you specify for the layer's instances.

Operating System	PHP Version	Apache Version
Amazon Linux 2015.03	5.3	2.2
Amazon Linux 2014.09	5.3	2.2
Ubuntu 14.04 LTS	5.5	2.4
Ubuntu 12.04 LTS	5.3	2.2

Installation: AWS 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 AWS 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 \(p. 120\)](#).

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 \(p. 346\)](#). For a list of Apache attributes that can be overridden, see [apache2 Attributes \(p. 532\)](#).

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 \(p. 313\)](#).

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 \(p. 485\)](#).

Rails App Server AWS OpsWorks Stacks Layer

Note

This layer is available only for Linux-based stacks.

The Rails App Server layer is an AWS OpsWorks Stacks layer that provides a blueprint for instances that function as Rails application servers.

Installation: AWS 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 2.0.0.

You can also specify your preferred Ruby version by [overriding the \[:opsworks\] \[:ruby_version\] \(p. 346\)](#) attribute.

Note

AWS OpsWorks Stacks installs a separate Ruby package to be used by recipes and the instance agent. For more information, see [Ruby Versions \(p. 248\)](#).

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 4.0.46.

Rubygems Version

The default [Rubygems](#) version is 2.2.2.

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 1.5.3.

Custom security groups

This setting appears if you chose to not automatically associate a built-in AWS 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 \(p. 120\)](#).

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 \(p. 346\)](#). For a list of Apache, Nginx, Phusion Passenger, and Unicorn attributes that can be overridden, see [Built-in Cookbook Attributes \(p. 532\)](#).

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 \(p. 495\)](#).

Topics

- [Disabling SSLv3 for Rails Servers \(p. 495\)](#)
- [Connecting to a Database \(p. 495\)](#)
- [Deploying Ruby on Rails Apps \(p. 496\)](#)

Disabling SSLv3 for Rails Servers

To disable SSLv3 for Rails servers, update the layer's **Ruby Version** setting to 2.1, which installs Ruby 2.1.4 as the version that applications use.

- Update the layer's **Ruby Version** setting to 2.1, which installs Ruby 2.1.4 as the version that applications use.
- 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 \(p. 485\)](#).

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 TLSv1.1 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 \(p. 351\)](#).

Connecting to a Database

When you deploy an app, AWS OpsWorks Stacks creates a new `database.yml` file using information from the app's [deploy attributes \(p. 380\)](#). If you [attach a MySQL or Amazon RDS instance \(p. 220\)](#) to the app, AWS 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, AWS 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 AWS OpsWorks Stacks generates from the app name. The values you specify in custom JSON override any default settings. For more information, see [Adding Apps \(p. 217\)](#).

AWS OpsWorks Stacks incorporates the following `[:...][:database]` (p. 526) attribute values into `database.yml`. The required attributes depend on the particular database, but you must have a `host` attribute or AWS 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 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 \(p. 135\)](#). 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 \(p. 216\)](#). 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 \(p. 120\)](#) with a Rails App Server layer with Apache/Passenger as the Rails stack, [add a 24/7 instance \(p. 168\)](#) to the layer, and [start it \(p. 178\)](#).
2. After the instance is online, [add an app \(p. 217\)](#) to the stack and specify following settings:

- **Name** – Any name you prefer; the example uses `PhotoPoll`.

AWS 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 \(p. 376\)](#). 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 \(p. 221\)](#) 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 AWS OpsWorks Stacks Layer

Note

This layer is available only for Linux-based stacks.

The Static Web Server layer is an AWS 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 AWS 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 \(p. 120\)](#).

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 \(p. 346\)](#). For a list of Apache attributes that can be overridden, see [nginx Attributes \(p. 549\)](#).

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 TLSv1.1 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 \(p. 351\)](#).

ECS Cluster Layer Reference

Note

This layer is available only for Linux-based stacks.

An ECS Cluster layer represents an [Amazon EC2 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:

- AWS 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 AWS OpsWorks Stacks do not include Amazon ECS error log information. For more information on how to handle Amazon ECS errors, see [Amazon ECS Troubleshooting](#).

Custom Layer Reference

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 AWS 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

AWS OpsWorks Stacks also supports the following layers.

Topics

- [Ganglia Layer Reference \(p. 500\)](#)
- [Memcached Layer Reference \(p. 501\)](#)

Ganglia Layer Reference

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.

AWS 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](#)

Note

This layer is available only for Linux-based stacks.

Memcached is a distributed memory-caching system for arbitrary data. It speeds 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, AWS 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:

- AWS OpsWorks Stacks uses the instance's package installer to install Memcached and its log files in their default locations.

Other Layers

Note

These layers are available only for Chef 11 and earlier Linux-based stacks.

AWS OpsWorks Stacks also supports the Ganglia and Memcached layers.

Topics

- [Ganglia Layer \(p. 503\)](#)
- [Memcached \(p. 504\)](#)

Ganglia Layer

Note

This layer is available only for Chef 11 and earlier Linux-based stacks.

AWS 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 AWS 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.

Tip

Record the password for later use; AWS 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 AWS 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 \(p. 120\)](#).

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 \(p. 485\)](#).

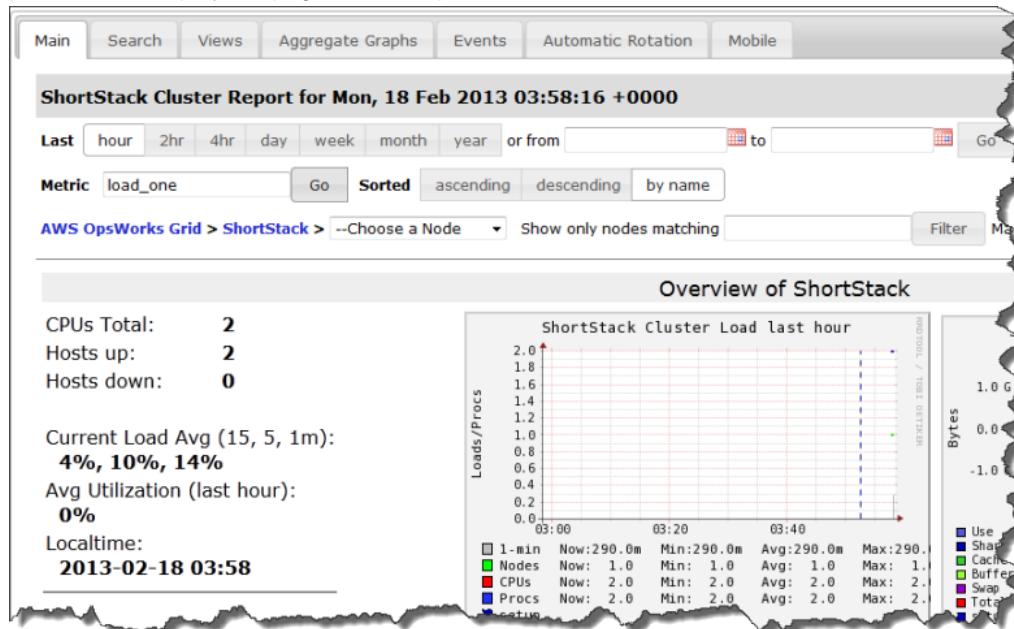
[View the Ganglia Statistics](#)

AWS 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

Note

This layer is available only for Chef 11 and earlier Linux-based stacks.

A Memcached layer is an AWS 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 AWS 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 \(p. 120\)](#).

Cookbook Components

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 \(p. 505\)](#)
- [Templates \(p. 507\)](#)
- [Recipes \(p. 508\)](#)

Attributes

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 \(p. 348\)](#).

For more information about attributes and how to override them, see [Overriding Attributes \(p. 346\)](#).

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
...
```

AWS 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. AWS 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 AWS OpsWorks Stacks attribute values.

Note

Chef supports additional types, which aren't necessary for AWS 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 \(p. 346\)](#), 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](#). For examples of working attribute files, see the AWS OpsWorks Stacks built-in cookbooks at <https://github.com/aws/opsworks-cookbooks>.

Templates

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 \(p. 346\)](#).

For more information on templates, including how to incorporate Ruby code, see [About Templates](#).

Recipes

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 AWS OpsWorks Stacks execute recipes automatically when a [lifecycle event \(p. 253\)](#) occurs on the instance but you can also run them explicitly at any time by using the [Execute Recipes stack command \(p. 252\)](#). 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 \(p. 507\)](#).

```
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

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 \(p. 376\)](#).

Tip

On Linux instances, AWS 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 \(p. 652\)](#).

Topics

- [opsworks Attributes \(p. 510\)](#)
- [opsworks_custom_cookbooks Attributes \(p. 523\)](#)
- [dependencies Attributes \(p. 523\)](#)
- [ganglia Attributes \(p. 524\)](#)
- [mysql Attributes \(p. 524\)](#)
- [passenger Attributes \(p. 524\)](#)
- [opsworks_bundler Attributes \(p. 525\)](#)
- [deploy Attributes \(p. 525\)](#)
- [Other Top-Level Attributes \(p. 531\)](#)

opsworks Attributes

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 \(p. 510\)](#)
- [instance Attributes \(p. 511\)](#)
- [layers Attributes \(p. 513\)](#)
- [rails_stack Attributes \(p. 516\)](#)
- [stack Attributes \(p. 517\)](#)
- [Other Top-level opsworks Attributes \(p. 522\)](#)

applications

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

The `instance` attribute contains a set of attributes that specify the configuration of this instance.

architecture (p. 511)	availability_zone (p. 512)	backends (p. 512)
aws_instance_id (p. 512)	hostname (p. 512)	id (p. 512)
instance_type (p. 512)	ip (p. 512)	layers (p. 512)
private_dns_name (p. 512)	private_ip (p. 513)	public_dns_name (p. 513)
region (p. 513)		

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 AWS 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

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](#) (p. 513)
- [instances](#) (p. 514)
- [name](#) (p. 516)

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 (p. 514)	aws_instance_id (p. 514)	backends (p. 514)
booted_at (p. 514)	created_at (p. 514)	elastic_ip (p. 515)
instance_type (p. 515)	ip (p. 515)	private_ip (p. 515)
public_dns_name (p. 515)	private_dns_name (p. 515)	region (p. 515)
status (p. 515)		

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-ddThh: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-ddThh: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](#)

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

stack attributes specify some aspects of the stack configuration, such as service layer configurations.

- [elb-load-balancers \(p. 517\)](#)
- [id \(p. 517\)](#)
- [name \(p. 517\)](#)
- [rds_instances \(p. 517\)](#)
- [vpc_id \(p. 522\)](#)

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 (p. 518)	allocated_storage (p. 518)	arn (p. 518)
auto_minor_version_upgrade (p. 519)	availability_zone (p. 519)	backup_retention_period (p. 519)
db_instance_class (p. 519)	db_instance_identifier (p. 519)	db_instance_status (p. 519)
db_name (p. 519)	db_parameter_groups (p. 519)	db_security_groups (p. 520)
db_user (p. 520)	engine (p. 520)	instance_create_time (p. 520)
license_model (p. 520)	multi_az (p. 520)	option_group_memberships (p. 521)
port (p. 521)	preferred_backup_window (p. 521)	preferred_maintenance_window (p. 521)
publicly_accessible (p. 521)	read_replica_db_instance_identifier (p. 521)	read_replica_db_instance_identifier (p. 521)
status_infos (p. 522)	vpc_security_groups (p. 522)	

address

The instances URL, such as `opsinstance.ccdvt3hwogla.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

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). For more information, see [deploy](#). 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 [\["ruby"\] \(p. 558\)](#) attribute to specify the patch version. For more information about how to specify a version, including examples, see [Ruby Versions \(p. 248\)](#). For complete details on how AWS 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 AWS OpsWorks Stacks-generated GUID that uniquely identifies the deployment (string). Otherwise the attribute is set to null.

```
node[ "opsworks" ][ "deployment" ]
```

[opsworks_custom_cookbooks Attributes](#)

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](#)

Contains several attributes that are related to the [update_dependencies](#) [stack command \(p. 133\)](#).

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

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

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

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](#)

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](#)

If the attributes are associated with a [Deploy event \(p. 253\)](#) or an [Execute Recipes stack command \(p. 133\)](#), 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 (p. 526)	application_type (p. 526)	auto_bundle_on_deploy (p. 526)
database (p. 526)	deploy_to (p. 527)	domains (p. 527)
document_root (p. 527)	environment_variables (p. 527)	group (p. 528)
keep_releases (p. 528)	memcached (p. 528)	migrate (p. 528)
mounted_at (p. 528)	purge_before_symlink (p. 528)	rails_env (p. 529)
restart_command (p. 529)	scm (p. 529)	ssl_certificate (p. 530)

ssl_certificate_ca (p. 530)	ssl_certificate_key (p. 530)	ssl_support (p. 530)
stack (p. 530)	symlink_before_migrate (p. 530)	symlinks (p. 530)
user (p. 530)		

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, AWS 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 \(p. 217\)](#). 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 AWS 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 \(p. 539\)](#), 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 \(p. 228\)](#), 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

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 AWS OpsWorks Stacks and depends on the instance type, but you can [override it \(p. 346\)](#) 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 IAM users that have been granted SSH permissions. Each attribute is named with a user's Unix ID. AWS OpsWorks Stacks generates a unique ID for each user in the 2000 range, such as `"2001"`, and creates a user with that ID on every instance. Each attribute contains the following attributes:

email

The IAM user's e-mail address (string).

```
node[ "ssh_users" ][ "UnixID" ][ "email" ]
```

public_key

The IAM user's public SSH key (string).

```
node[ "ssh_users" ][ "UnixID" ][ "public_key" ]
```

sudoer

Whether the IAM user has sudo permissions (Boolean).

```
node[ "ssh_users" ][ "UnixID" ][ "sudoer" ]
```

name

The IAM user name (string).

```
node[ "ssh_users" ][ "UnixID" ][ "name" ]
```

Built-in Cookbook Attributes

Note

Most of these attributes are available only on Linux stacks.

Most of the built-in recipes have one or more [attributes files \(p. 505\)](#) 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 AWS 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 \(p. 532\)](#)
- [deploy Attributes \(p. 539\)](#)
- [haproxy Attributes \(p. 540\)](#)
- [memached Attributes \(p. 543\)](#)
- [mysql Attributes \(p. 544\)](#)
- [nginx Attributes \(p. 549\)](#)
- [opsworks_berkshelf Attributes \(p. 552\)](#)
- [opsworks_java Attributes \(p. 552\)](#)
- [passenger_apache2 Attributes \(p. 556\)](#)
- [ruby Attributes \(p. 558\)](#)
- [unicorn Attributes \(p. 560\)](#)

apache2 Attributes

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 \(p. 346\)](#).

binary (p. 533)	contact (p. 533)	deflate_types (p. 533)
dir (p. 533)	document_root (p. 534)	group (p. 534)
hide_info_headers (p. 534)	icondir (p. 534)	init_script (p. 534)
keepalive (p. 534)	keepaliverequests (p. 534)	keepalivetimeout (p. 535)

lib_dir (p. 535)	libexecdir (p. 535)	listen_ports (p. 535)
log_dir (p. 535)	logrotate Attributes (p. 535)	pid_file (p. 536)
prefork Attributes (p. 536)	serversignature (p. 537)	servertokens (p. 537)
timeout (p. 538)	traceenable (p. 538)	user (p. 538)
version (p. 538)	worker Attributes (p. 538)	

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']
```

Caution

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, RHEL, or Ubuntu 12.04 LTS. If your instances are running Ubuntu 14.04 LTS, use [maxrequestworkers \(p. 537\)](#).

```
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, RHEL, or Ubuntu 12.04 LTS, use [maxclients \(p. 536\)](#).

```
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 12.04 LTS: 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

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 \(p. 369\)](#). For more information on how to override built-in attributes to specify custom values, see [Overriding Attributes \(p. 346\)](#).

deploy_keep_releases

A global setting for the number of app deployments that AWS 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

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 \(p. 346\)](#).

balance (p. 540)	check_interval (p. 541)	client_timeout (p. 541)
connect_timeout (p. 541)	default_max_connections (p. 541)	global_max_connections (p. 541)
health_check_method (p. 541)	health_check_url (p. 541)	queue_timeout (p. 541)
http_request_timeout (p. 541)	maxcon_factor_nodejs_app (p. 542)	maxcon_factor_nodejs_app_ssl (p. 542)
maxcon_factor_php_app (p. 542)	maxcon_factor_php_app_ssl (p. 542)	maxcon_factor_rails_app (p. 543)
maxcon_factor_rails_app_ssl (p. 543)	maxcon_factor_static (p. 543)	maxcon_factor_static_ssl (p. 543)
retries (p. 542)	server_timeout (p. 542)	stats_url (p. 542)
stats_user (p. 542)		

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 \(p. 512\)](#). For example, suppose you have a Rails app server on a small instance with a `backend` value of 4, which means that AWS 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 ]
```

memached Attributes

Note

These attributes are available only on Linux stacks.

The [memached attributes](#) specify the [Memcached](#) server configuration. For more information on how to override built-in attributes to specify custom values, see [Overriding Attributes \(p. 346\)](#).

memory (p. 543)	max_connections (p. 543)	pid_file (p. 544)
port (p. 544)	start_command (p. 544)	stop_command (p. 544)
user (p. 544)		

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

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 \(p. 346\)](#).

basedir (p. 544)	bind_address (p. 545)	clients (p. 545)
conf_dir (p. 545)	confd_dir (p. 545)	datadir (p. 545)
grants_path (p. 545)	mysql_bin (p. 545)	mysqladmin_bin (p. 545)
pid_file (p. 545)	port (p. 546)	root_group (p. 546)
server_root_password (p. 546)	socket (p. 546)	tunable Attributes (p. 546)

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 (p. 546)	innodb_additional_mem_pool_size (p. 547)	innodb_buffer_pool_size (p. 547)
innodb_flush_log_at_trx_commit (p. 547)	innodb_lock_wait_timeout (p. 547)	key_buffer (p. 547)
log_slow_queries (p. 547)	long_query_time (p. 547)	max_allowed_packet (p. 547)
max_connections (p. 548)	max_heap_table_size (p. 548)	net_read_timeout (p. 548)
net_write_timeout (p. 548)	query_cache_limit (p. 548)	query_cache_size (p. 548)
query_cache_type (p. 548)	thread_cache_size (p. 548)	thread_stack (p. 549)
wait_timeout (p. 549)	table_cache (p. 549)	

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 AWS OpsWorks Stacks and depends on the instance type, but you can [override it \(p. 346\)](#) 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

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 \(p. 346\)](#).

binary (p. 549)	dir (p. 549)	gzip (p. 549)
gzip_comp_level (p. 550)	gzip_disable (p. 550)	gzip_http_version (p. 550)
gzip_proxied (p. 550)	gzip_static (p. 550)	gzip_types (p. 551)
gzip_vary (p. 551)	keepalive (p. 551)	keepalive_timeout (p. 551)
log_dir (p. 551)	user (p. 551)	server_names_hash_bucket_size (p. 551)
worker_processes (p. 551)	worker_connections (p. 552)	

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'.

Caution

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

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 \(p. 346\)](#).

debug

Whether to include Berkshelf debugging information in the Chef log (Boolean). The default value is `false`.

```
node[ 'opsworks_berkshelf'][ 'debug']
```

opsworks_java Attributes

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 \(p. 346\)](#).

datasources (p. 552)	java_app_server_version (p. 553)	java_shared_lib_dir (p. 553)
jvm_pkg Attributes (p. 553)	custom_pkg_location_url_debian (p. 553)	java_home_basedir (p. 553)
custom_pkg_location_url_rhel (p. 553)	use_custom_pkg_location (p. 553)	jvm_options (p. 553)
jvm_version (p. 554)	tomcat Attributes (p. 554)	

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 \(p. 489\)](#). 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 \(p. 489\)](#).

```
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 (p. 554)	apache_tomcat_bind_mod (p. 554)	apache_tomcat_bind_path (p. 554)
auto_deploy (p. 554)	connection_timeout (p. 554)	mysql_connector_jar (p. 555)
port (p. 555)	secure_port (p. 555)	shutdown_port (p. 555)
threadpool_max_threads (p. 555)	threadpool_min_spare_threads (p. 555)	unpack_wars (p. 555)
uri_encoding (p. 555)	use_ssl_connector (p. 555)	use_threadpool (p. 556)
userdatabase.pathname (p. 556)		

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

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 \(p. 346\)](#).

friendly_error_pages (p. 556)	gem_bin (p. 556)	gems_path (p. 556)
high_performance_mode (p. 557)	root_path (p. 557)	max_instances_per_app (p. 557)
max_pool_size (p. 557)	max_requests (p. 557)	module_path (p. 557)
pool_idle_time (p. 557)	rails_app_spawner_idle_time (p. 558)	rails_framework_spawner_idle_time (p. 558)
rails_spawn_method (p. 558)	ruby_bin (p. 558)	ruby_wrapper_bin (p. 558)
stat_throttle_rate (p. 558)	version (p. 558)	

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 (p. 535)']}/mod_passenger.so"
- Ubuntu: "#{passenger[:root_path (p. 557)]}/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

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 \(p. 248\)](#). For complete details on how AWS 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 \(p. 346\)](#).

full_version

The full version number (string). You should not override this attribute. Instead, use [\[:opsworks\] \[:ruby_version\] \(p. 523\)](#) 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 [\[:opsworks\] \[:ruby_version\] \(p. 523\)](#) to specify the major version.

```
[ :ruby][ :major_version]
```

minor_version

The minor version number (string). You should not override this attribute. Instead, use [\[:opsworks\] \[:ruby_version\] \(p. 523\)](#) 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

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 \(p. 346\)](#).

accept_filter (p. 560)	backlog (p. 560)	delay (p. 560)
tcp_nodelay (p. 560)	tcp_nopush (p. 560)	preload_app (p. 560)
timeout (p. 560)	tries (p. 561)	version (p. 561)
worker_processes (p. 561)		

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

Note

For additional troubleshooting information, see [Debugging and Troubleshooting Guide \(p. 633\)](#).

Chef Logs for Chef 11.10 and Earlier Versions for Linux

AWS 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 \(p. 635\)](#)
- [Viewing a Chef Log with the CLI or API \(p. 635\)](#)
- [Interpreting a Chef Log \(p. 638\)](#)
- [Common Chef Log Errors \(p. 640\)](#)

Using AWS OpsWorks Stacks with Other AWS Services

You can have application servers running in an AWS OpsWorks Stacks stack use a variety of AWS services that are not directly integrated with AWS 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 \(p. 376\)](#) 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 AWS 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 \(p. 235\)](#) and [Customizing AWS OpsWorks Stacks \(p. 345\)](#).

Topics

- [Using a Back-end Data Store \(p. 562\)](#)
- [Using ElastiCache Redis as an In-Memory Key-Value Store \(p. 566\)](#)
- [Using an Amazon S3 Bucket \(p. 574\)](#)
- [Using AWS CodePipeline with AWS OpsWorks Stacks \(p. 583\)](#)

Using a Back-end Data Store

Application server stacks commonly include a database server to provide a back-end data store. AWS OpsWorks Stacks provides integrated support for MySQL servers through the [MySQL layer \(p. 474\)](#) and for several types of database servers through the [Amazon Relational Database Service \(Amazon RDS\) layer \(p. 150\)](#). 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 \(p. 313\)](#) 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](#).

Tip

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 \(p. 562\)](#)
- [How to Connect an Application Server Instance to Amazon RDS \(p. 564\)](#)

How to Set up a Database Connection

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 \(p. 376\)](#) that AWS OpsWorks Stacks installs on every instance.

For example, Step 2 of [Getting Started with Chef 11 Linux Stacks \(p. 313\)](#) 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, AWS 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 AWS 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 \(p. 253\)](#). When you deploy SimplePHPApp, AWS 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
```

The variables that characterize the connection—`host`, `user`, and so on—are set the corresponding values from the [deploy JSON's \(p. 380\)](#) `[: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 \(p. 135\)](#). AWS 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 \(p. 346\)](#).

How to Connect an Application Server Instance to Amazon RDS

This section describes how to customize MyStack from [Getting Started with Chef 11 Linux Stacks \(p. 313\)](#) to have the PHP application server connect to an RDS instance.

Topics

- [Create an Amazon RDS MySQL Database \(p. 564\)](#)
- [Customize the Stack to Connect to the RDS Database \(p. 565\)](#)

Create an Amazon RDS MySQL Database

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 `rdsexampled`
 - 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 `rdsexampledbs` 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

Once you have [created an RDS instance \(p. 564\)](#) to use as a back-end database for the PHP application server, you can customize MyStack from [Getting Started with Chef 11 Linux Stacks \(p. 313\)](#).

To connect the PHP App Server to an RDS database

1. Open the AWS 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 \(p. 313\)](#). This stack uses version1 of SimplePHPApp, which does not use a database connection.
2. [Update the stack configuration \(p. 132\)](#) 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": "rdsexampledbs",
        "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 `rdsexampledbs`.

- **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 \(p. 224\)](#) 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 \(p. 140\)](#) to set up the database connection by adding `phpapp::appsetup` to the layer's Deploy recipes.
6. [Deploy the new SimplePHPApp version \(p. 221\)](#).
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, AWS 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 \(p. 221\)](#). For more information on overriding AWS OpsWorks Stacks built-in attributes, see [Overriding Attributes \(p. 346\)](#).

Using ElastiCache Redis as an In-Memory Key-Value Store

Note

This topic is based on a Linux stack, but Windows stacks can also use Amazon ElastiCache (ElastiCache). For an example 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. AWS OpsWorks Stacks provides built-in support for [Memcached \(p. 504\)](#). 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 \(p. 567\)](#)

- Step 2: Set up a Rails Stack (p. 568)
- Step 3: Create and Deploy a Custom Cookbook (p. 569)
- Step 4: Assign the Recipe to a LifeCycle Event (p. 572)
- Step 5: Add Access Information to the Stack Configuration JSON (p. 573)
- Step 6: Deploy and run the App (p. 574)

Step 1: Create an ElastiCache Redis Cluster

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**.

The screenshot shows the 'Launch Cache Cluster Wizard' interface. The title bar says 'Launch Cache Cluster Wizard'. Below it, there are three tabs: 'CACHE CLUSTER DETAILS' (which is selected), 'ADDITIONAL CONFIGURATION', and 'REVIEW'. A 'Cancel' button is in the top right corner. The main area has a heading 'To get started, provide the details for your Cache Cluster below.' It contains several input fields:

- Name:** OpsWorks-Redis
- Engine:** redis
- Cache Engine Version:** 2.6.13
- Node Type:** cache.m1.small (1.3 GB me...)
- Number of Nodes:** 1
- Cache Port:** 6379 (e.g. 11211)
- Cache Subnet Group:** Not in VPC
- Preferred Zone:** No Preference
- Topic for SNS Notification:** Disable Notifications (with a 'Manual ARN input' link)
- S3 Snapshot Location:** myBucket/myFolder/o/
- Auto Minor Version Upgrade:** Yes No

A note at the bottom states: 'Note: "Auto Minor Version Upgrade" only applies to the Cache Engine software. Critical System Software patches (e.g. security related) may be applied irrespective of this selection.' At the bottom right, there is a 'Continue' button with a yellow arrow icon, and a note '* Required'.

3. On the **Additional Configuration** page, accept the defaults and click **Continue**.

Cache Security Group(s): default

Cache Parameter Group: default.redis2.6

Maintenance Window: No Preference Select Window

Continue

- 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](#).

- 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.

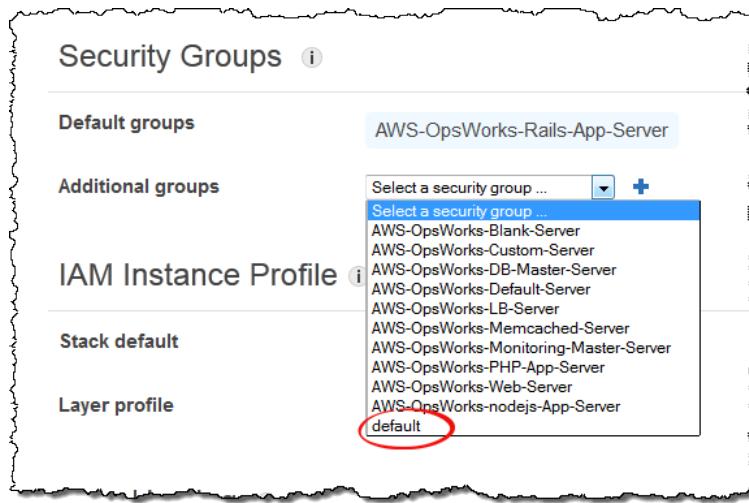
Cache Cluster: opsws-redis						
Description		Nodes				
Add Nodes	Remove Nodes	Delete Nodes	Copy Node Endpoint(s)			
0001	available	Thu Sep 05 16:32:45 GMT-700 2013	6379	opsws-redis.b47jtf.0001.use1.cache.amazonaws.com	in-sync	

Step 2: Set up a Rails Stack

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

- 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 \(p. 120\)](#) and [Creating an OpsWorks Layer \(p. 139\)](#).
- On the **Layers** page, for Rails App Server, click **Security** and then click **Edit**.
- 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.



- 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 \(p. 168\)](#).

Step 3: Create and Deploy a Custom Cookbook

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 AWS OpsWorks Stacks run the recipe on every server. Chef recipes are specialized Ruby applications that AWS 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 \(p. 235\)](#) 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 AWS OpsWorks Stacks [stack configuration and deployment JSON \(p. 376\)](#) 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 \(p. 525\)](#). 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.

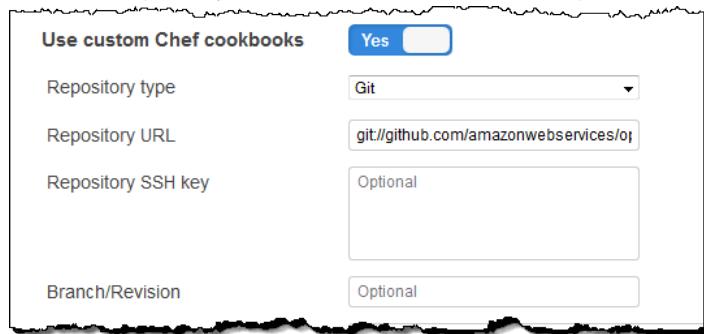
The `[:redis]` attribute's values are set by using custom JSON, as described later; it is not one of the standard app attributes.

- 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 AWS 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.

Run Command

The screenshot shows the 'Run Command' interface. The 'Command' dropdown is set to 'Update Custom Cookbooks'. The 'Comment' field is optional and empty. Below the form, under 'Instances', there is a list of selected instances: 'Rails App Server' and 'rails-app1'. At the bottom right are 'Cancel' and 'Update Custom Cookbooks' buttons.

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 \(p. 249\)](#).

Step 4: Assign the Recipe to a LifeCycle Event

You can run custom recipes [manually \(p. 255\)](#), but the best approach is usually to have AWS OpsWorks Stacks run them automatically. Every layer has a set of built-in recipes assigned each of five [Lifecycle events \(p. 253\)](#)—Setup, Configure, Deploy, Undeploy, and Shutdown. Each time an event occurs for an instance, AWS 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, AWS 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 AWS 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. AWS 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 \(p. 255\)](#).

To assign a recipe to the Rails App Server layer's Deploy event

1. On the AWS 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

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 AWS 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 AWS 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 `elasticsearch_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 \(p. 567\)](#).

```
{
  "deploy": {
    "elasticsearch_redis_example": {
      "redis": {
        "host": "mycluster.XXXXXXXX.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. AWS 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 \(p. 569\)](#). The next time you deploy an app, AWS 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 AWS 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 \(p. 346\)](#).

Step 6: Deploy and run the App

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 \(p. 217\)](#) to represent your application and [deploy it \(p. 221\)](#) 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

Applications often use an Amazon Simple Storage Service (Amazon S3) bucket to store large items such as images or other media files. Although AWS 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.

Topics

- [Step 1: Create an Amazon S3 Bucket \(p. 575\)](#)
- [Step 2: Create a PHP App Server Stack \(p. 576\)](#)
- [Step 3: Create and Deploy a Custom Cookbook \(p. 577\)](#)
- [Step 4: Assign the Recipes to LifeCycle Events \(p. 579\)](#)
- [Step 5: Add Access Information to the Stack Configuration and Deployment Attributes \(p. 580\)](#)
- [Step 6: Deploy and Run PhotoApp \(p. 581\)](#)

Step 1: Create an Amazon S3 Bucket

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 click **Create Stack**.
3. In the **Stack Name** box, enter the stack name.

This example assumes that the name is `AppServer`.

4. Click **Upload template file**, click **Browse**, select the `appserver.template` file that you created in Step 1, and click **Next Step**.
5. On the **Specify Parameters** page, select **I acknowledge that this template may create IAM resources**, then click **Next Step** on each page of the wizard until you reach the end. Click **Create**.
6. After the **AppServer** stack reaches **CREATE_COMPLETE** status, select it and click its **Outputs** tab.

You might need to click refresh a few times to update the status.

7. On the **Outputs** tab, record the **BucketName** and **InstanceProfileName** values for later use.

Note

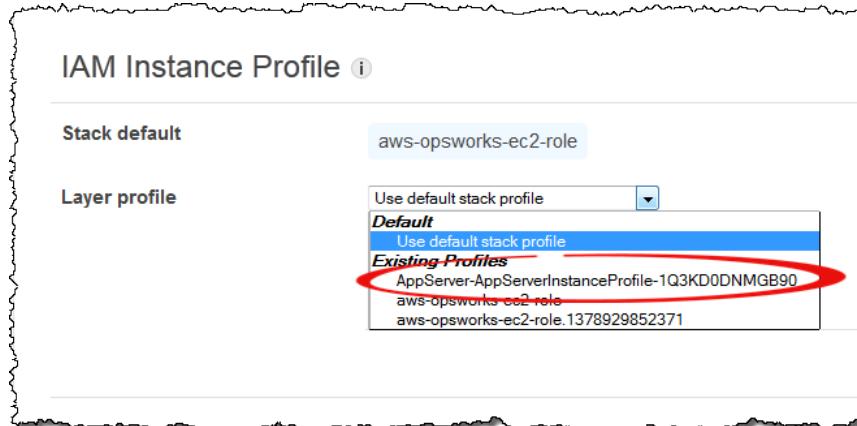
AWS CloudFormation uses the term *stack* to refer to the collection of resources that are created from a template; it is not the same as an AWS OpsWorks Stacks stack.

Step 2: Create a PHP App Server Stack

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.

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 \(p. 120\)](#) and [Creating an OpsWorks Layer \(p. 139\)](#).
2. On the **Layers** page, for PHP App Server, click **Security** and then click **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`. AWS 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 \(p. 168\)](#).
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 AWS OpsWorks Stacks instance profile, which is selected by default.

Step 3: Create and Deploy a Custom Cookbook

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 AWS OpsWorks Stacks run the recipe automatically on the appropriate instances. Chef recipes are specialized Ruby applications that AWS 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 \(p. 235\)](#) 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 \(p. 326\)](#); see that topic for a detailed description. The primary difference is the variables that are passed to the template, which represent the access information.

```
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[:photoapp][:dbtable] rescue nil),
  :s3bucket =>  (node[:photobucket] rescue nil)
)
```

The first four attributes define database connection settings, and are automatically defined by AWS 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 \(p. 505\)](#)

For more information on attributes, see [Attributes \(p. 505\)](#).

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 \(p. 324\)](#); see that topic for a detailed description.

```
node[:deploy].each do |app_name, deploy|
  execute "mysql-create-table" do
    command "/usr/bin/mysql -u#{deploy[:database][:username]} -p#{deploy[:database][:password]} #{deploy[:database][:database]} -e'CREATE TABLE #{node[:photoapp][:dbtable]}(id INT UNSIGNED NOT NULL AUTO_INCREMENT, url VARCHAR(255) NOT NULL, caption VARCHAR(255), PRIMARY KEY (id))'"
    not_if "/usr/bin/mysql -u#{deploy[:database][:username]} -p#{deploy[:database][:password]} #{deploy[:database][:database]} -e'SHOW TABLES' | grep #{node[:photoapp][:dbtable]}"
    action :run
  end
end
```

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. AWS OpsWorks Stacks will then run the cached recipes when the appropriate lifecycle event occurs, as described later.

To deploy the photoapp cookbook

1. On the AWS OpsWorks Stacks **Stack** page, click **Stack Settings** and then **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, 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.

Run Command

The screenshot shows the 'Run Command' configuration page. The 'Command' dropdown is set to 'Update Custom Cookbooks', which is highlighted with a red circle. The 'Comment' field is labeled 'Optional' and contains no text. In the 'Instances' section, there is one instance selected: 'Rails App Server' with ID 'rails-app1'. A note below says 'Click to select instances in this layer'. At the bottom right are 'Cancel' and 'Update Custom Cookbooks' buttons.

Step 4: Assign the Recipes to LifeCycle Events

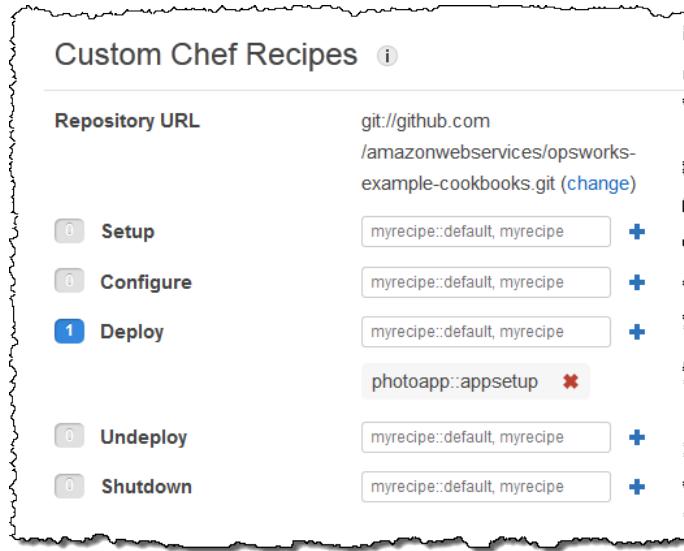
You can run custom recipes [manually \(p. 255\)](#), but the best approach is usually to have AWS OpsWorks Stacks run them automatically. Every layer has a set of built-in recipes assigned to each of five [lifecycle events \(p. 253\)](#)—Setup, Configure, Deploy, Undeploy, and Shutdown. Each time an event occurs on an instance, AWS 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, AWS OpsWorks Stacks triggers a Setup event to run the Setup recipes, which typically handle tasks such as installing and configuring packages .

You can have AWS OpsWorks Stacks run custom recipes on a layer's instances by assigning each recipe to the appropriate lifecycle event. AWS 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. AWS 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 \(p. 255\)](#).

To assign custom recipes to the layer's Deploy event

1. On the AWS OpsWorks Stacks **Layers** page, for the PHP App Server click **Recipes** and then click **Edit**.
2. Under **Custom Chef Recipes**, add the recipe name to the deploy event and click **+**. 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 click **Save** to update the layer configuration.



3. On the **Layers** page, click **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

The `appsetup.rb` recipe depends on data from the AWS OpsWorks Stacks [stack configuration and deployment attributes \(p. 376\)](#), 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 \(p. 525\)](#). 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:

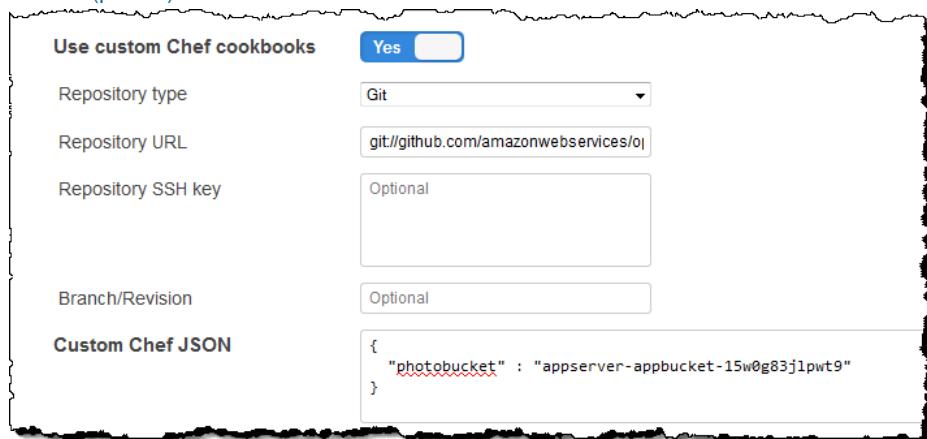
- The database connection attributes, such as `[:deploy][:database][:host]`, are defined by AWS OpsWorks Stacks when it creates the MySQL layer.
- The table name attribute, `[:photoapp][:dbtable]`, is defined in the custom cookbook's attributes file, and is set to `foto`.
- You must define the bucket 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 AWS 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:

```
{  
  "photobucket" : "yourbucketname"  
}
```

Replace `yourbucketname` with the bucket name that you recorded in [Step 1: Create an Amazon S3 Bucket \(p. 575\)](#).



AWS 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 \(p. 346\)](#) to provide a new bucket name.

Step 6: Deploy and Run PhotoApp

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 click **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`.
- Click **Add App** to accept the defaults for the other settings.

Add App

Settings

Name	PhotoApp
App type	PHP
Document root	web

Application Source

Repository type	Git
Repository URL	git://github.com/amazonwebservices/opsworks-demo-php-photo-share-app.git
Repository SSH key	Optional
Branch/Revision	Optional

3. On the **Apps** page, click **deploy** in the PhotoApp app's **Actions** column

Apps

An app represents code stored in a repository that you want to install on application server instances. When you deploy the app, OpsWorks downloads the code from the repository to the specified server instances. [Learn more](#).

Name	Type	Last Deployment	Actions
PhotoApp	PHP	2013-09-27 17:38:35 UTC	deploy
+ App			

4. Accept the defaults and click **Deploy** to deploy the app to the server.

To run PhotoApp, go to the **Instances** page and click the PHP App Server instance's public IP address.

Hostname	Status	Size	Type	AZ	Public IP	Actions
php-app1	online	c1.medium	24/7	us-east-1a	54.242.100.116	
+ Instance						

You should see the following user interface. Click **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 AWS OpsWorks Stacks

AWS CodePipeline lets you create continuous delivery pipelines that track code changes from sources such as AWS CodeCommit, Amazon Simple Storage Service (Amazon S3), or GitHub. You can use AWS CodePipeline to automate the release of your Chef cookbooks and application code to AWS 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 AWS CodePipeline as a deployment tool for code that you run on AWS OpsWorks Stacks layers.

Note

AWS CodePipeline and AWS OpsWorks Stacks integration is not supported for deploying to Chef 11.4 and older stacks.

Topics

- [AWS CodePipeline with AWS OpsWorks Stacks - Chef 12 Stacks \(p. 583\)](#)
- [AWS CodePipeline with AWS OpsWorks Stacks - Chef 11 Stacks \(p. 603\)](#)

AWS CodePipeline with AWS OpsWorks Stacks - Chef 12 Stacks

AWS CodePipeline lets you create continuous delivery pipelines that track code changes from sources such as AWS CodeCommit, Amazon Simple Storage Service (Amazon S3), or GitHub. The example in this topic describes how to create and use a simple pipeline from AWS CodePipeline as a deployment tool for code that you run on AWS OpsWorks Stacks layers. In this example, you create a pipeline for a simple [Node.js app](#), and then instruct AWS 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 AWS OpsWorks Stacks - Chef 11 Stacks \(p. 603\)](#).

Topics

- [Prerequisites \(p. 584\)](#)
- [Other Supported Scenarios \(p. 584\)](#)
- [Step 1: Create a stack, layer, and an instance in AWS OpsWorks Stacks \(p. 585\)](#)
- [Step 2: Configure your stack and layer to use custom cookbooks \(p. 591\)](#)
- [Step 3: Upload app code to an Amazon S3 bucket \(p. 592\)](#)
- [Step 4: Add your app to AWS OpsWorks Stacks \(p. 593\)](#)
- [Step 5: Create a pipeline in AWS CodePipeline \(p. 595\)](#)
- [Step 6: Verifying the app deployment in AWS OpsWorks Stacks \(p. 597\)](#)
- [Step 7 \(Optional\): Update the app code to see AWS CodePipeline redeploy your app automatically \(p. 599\)](#)
- [Step 8 \(Optional\): Clean up resources \(p. 602\)](#)

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 user accounts.

For more information about creating a security group in IAM and assigning permissions to the group, see [Creating Your First IAM User and Administrators Group](#). For more information about managing AWS OpsWorks Stacks permissions, see [Best Practices: Managing Permissions](#). For more information about managing AWS CodePipeline permissions, see [AWS CodePipeline Access Permissions Reference](#) in the AWS CodePipeline documentation.

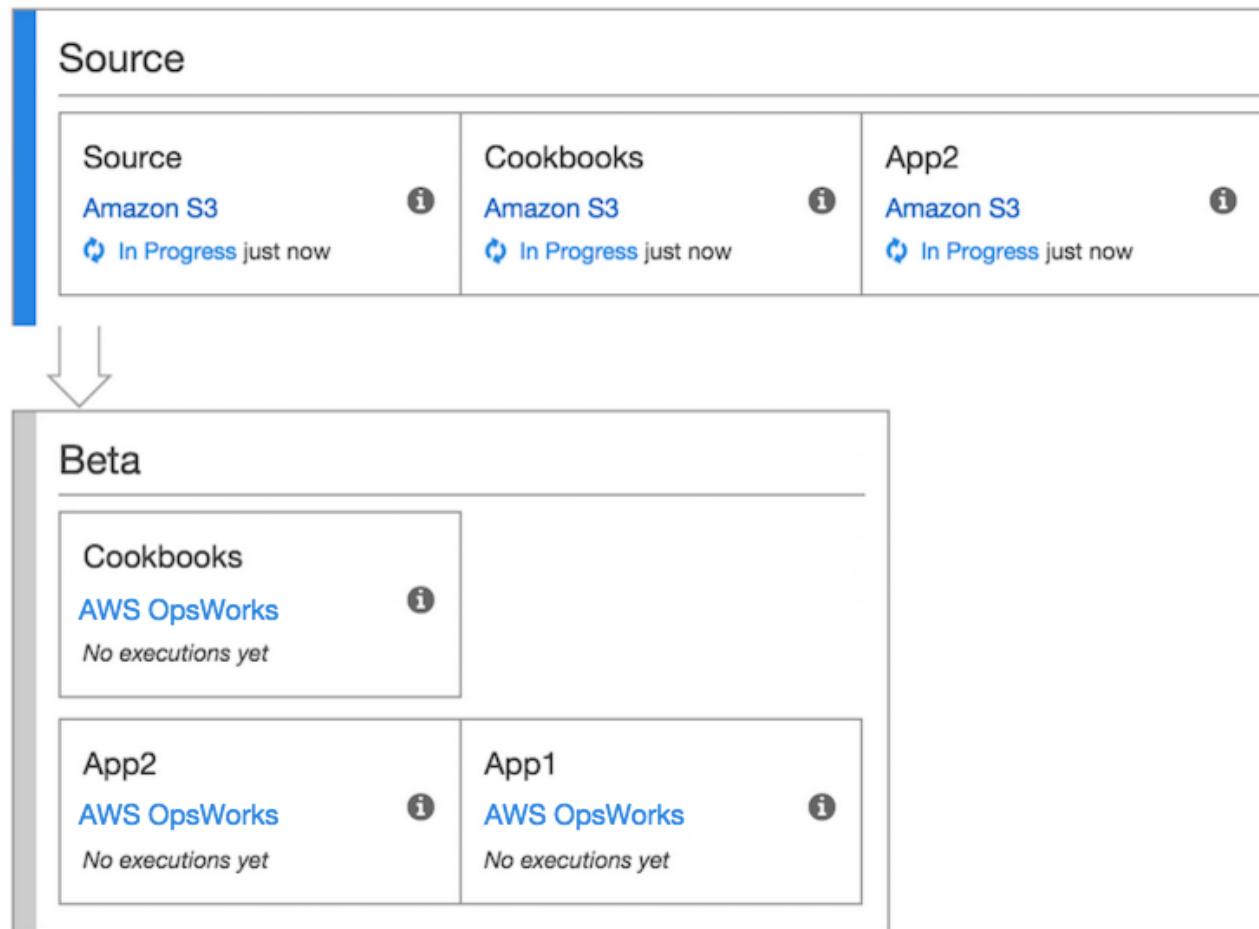
Permissions	Recommended Policy to Attach to Group
Create and edit stacks, layers, and instances in AWS OpsWorks Stacks.	AWSOpsWorksFullAccess
Create, edit, and run templates in AWS CloudFormation.	AmazonCloudFormationFullAccess
Create, edit, and access Amazon S3 buckets.	AmazonS3FullAccess
Create, edit, and run pipelines in AWS CodePipeline, especially pipelines that use AWS OpsWorks Stacks as the provider.	AWSCodePipelineFullAccess

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 AWS 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 AWS OpsWorks Stacks stack. The pipeline tracks changes to both the application and cookbook sources, and redeploy when you have made changes. The following shows an example of a similar, complex pipeline:



For more information about working with AWS CodePipeline, see the [AWS CodePipeline User Guide](#).

Step 1: Create a stack, layer, and an instance in AWS OpsWorks Stacks

To use AWS 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 AWS 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 AWS CodePipeline. At this time, AWS CodePipeline supports the AWS 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 the EC2 key pair to use for the instance."}
    }
}
```

```

        "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**.

Select Template

Select the template that describes the stack that you want to create. A stack is a group of related resources that you manage as a single unit.

The screenshot shows the 'Select Template' step of the AWS CloudFormation 'Create stack' wizard. It has two main sections:

- Design a template:** A link to use AWS CloudFormation Designer to create or modify an existing template, with a 'Learn more' link.
- Choose a template:** A section for uploading a template from Amazon S3. It includes:
 - A radio button for 'Select a sample template' with a dropdown menu.
 - A selected radio button for 'Upload a template to Amazon S3' with a 'Browse...' button and a file path 'NewOpsWorksStack.template'.
 - A radio button for 'Specify an Amazon S3 template URL' with a text input field.

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.

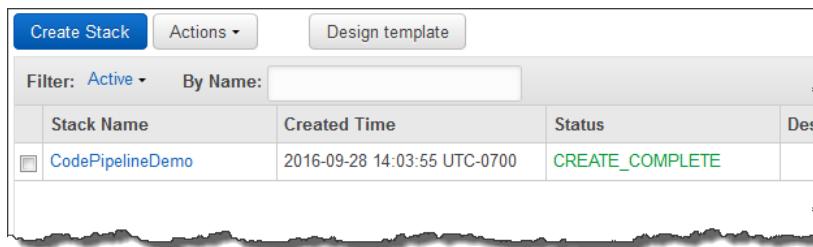
7. In the **Parameters** area, provide the name of an EC2 key pair that you want to use to access your AWS OpsWorks Stacks instance after it has been created. Choose **Next**.
8. On the **Options** page, choose **Next**. (Settings on this page are not required for this walkthrough.)
9. The AWS CloudFormation 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.

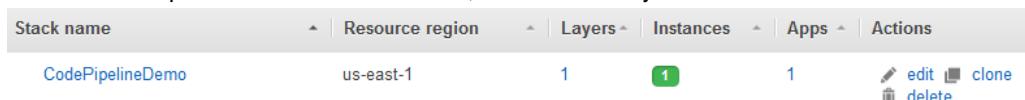
10. On the AWS CloudFormation dashboard, you can view the progress of the creation of the stack. Before you continue to the next step, wait until **CREATE_COMPLETE** is displayed in the **Status** column.



Stack Name	Created Time	Status	Des...
CodePipelineDemo	2016-09-28 14:03:55 UTC-0700	CREATE_COMPLETE	

To verify stack creation in AWS OpsWorks Stacks

1. Open the AWS OpsWorks console at <https://console.aws.amazon.com/opsworks/>.
2. On the AWS OpsWorks Stacks dashboard, view the stack you created.



Stack name	Resource region	Layers	Instances	Apps	Actions
CodePipelineDemo	us-east-1	1	1	1	edit clone delete

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

Chef 12 stacks in AWS 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 AWS OpsWorks Stacks \(p. 593\)](#). The Chef recipe that you specify in this step runs every time a new version of your application is deployed by AWS CodePipeline.

1. In the AWS OpsWorks Stacks console, open the stack that you created in [Step 1: Create a stack, layer, and an instance in AWS OpsWorks Stacks \(p. 585\)](#). 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 AWS CodePipeline and AWS OpsWorks, your cookbook source must be S3.
4. For **Repository URL**, specify <https://s3.amazonaws.com/opsworks-codepipeline-demo/opsworks-nodejs-demo-cookbook.zip>. Your settings should resemble the following.

Use custom Chef cookbooks	<input checked="" type="checkbox"/>
Repository type	S3 Archive
Repository URL	/opsworks-nodejs-demo-cookbook.zip
Access key ID	Optional
Secret access key	Optional

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 AWS OpsWorks Stacks \(p. 585\)](#).
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**.

Your recipe settings should resemble the following:

Layer Node.js App Server

General Settings **Recipes** Network EBS Volumes Security

Custom Chef Recipes i

Repository URL: https://s3.amazonaws.com/opsworks-codepipeline-demo/opsworks-nodejs-demo-cookbook.zip

Deploy: nodejs_demo

Setup, Configure, Undeploy, Shutdown

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

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 AWS CodePipeline can use code directly from GitHub or AWS CodeCommit as sources, this walkthrough demonstrates how to use an Amazon S3 bucket. Although the sample Node.js app already is stored in an Amazon S3 bucket, in this walkthrough, you upload it 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 AWS 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 AWS CodePipeline documentation.

Important

The Amazon S3 bucket must be in the same region in which you will later create your pipeline. At this time, AWS CodePipeline supports the AWS OpsWorks Stacks provider in the US East (N. Virginia) Region (us-east-1) only. All resources in this walkthrough should be created in the

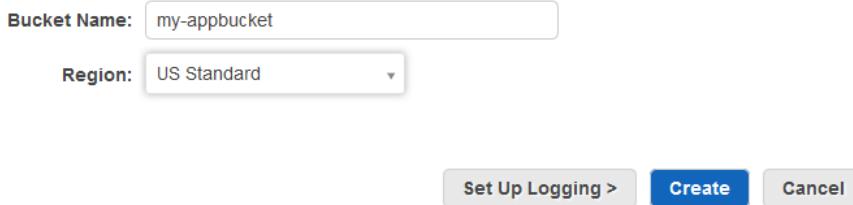
US East (N. Virginia) Region. The bucket must also be versioned because AWS CodePipeline requires a versioned source. For more information, see [Using Versioning](#).

To upload your app to an Amazon S3 bucket

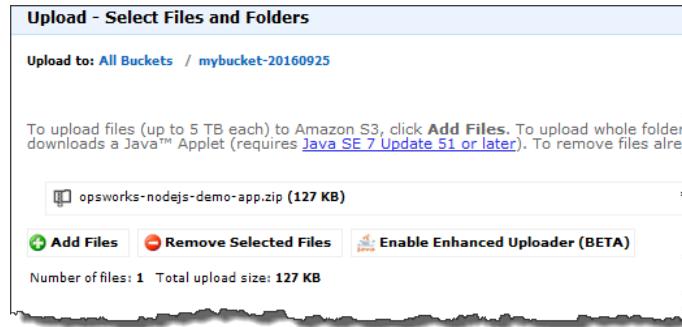
1. Download the ZIP file of the AWS OpsWorks Stacks sample, [Node.js app](#), and save it to a convenient location on your local computer.
2. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
3. Choose **Create Bucket**.
4. On the **Create a Bucket - Select a Bucket Name and Region** 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`.



A bucket is a container for objects stored in Amazon S3. When creating a bucket, you can choose a Region to optimize for latency, minimize costs, or address regulatory requirements. For more information regarding bucket naming conventions, please visit the [Amazon S3 documentation](#).



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 AWS OpsWorks Stacks

Before you create a pipeline in AWS CodePipeline, add the Node.js test app to AWS OpsWorks Stacks. When you create the pipeline, you will need to select the app that you've added to AWS 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 AWS OpsWorks Stacks

1. In the AWS 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 \(p. 592\)](#). Your form should be similar to the following:

Add App

All app attributes are stored in Chef data bags. [Learn more](#).

Settings

Name	<input type="text" value="Node.js Demo App"/>
Document root	<input type="text" value="opsworks-nodejs-demo-app"/>

Data Sources

Data source type RDS None

Application Source

Repository type	<input type="text" value="S3 Archive"/>
Repository URL	<input type="text" value="160925/opsworks-nodejs-demo-app.zip"/>
Access key ID	<input type="text" value="Optional"/>
Secret access key	<input type="text" value="Optional"/>

Environment Variables

<input type="text" value="KEY"/>	<input type="text" value="VALUE"/>	<input type="checkbox"/> Protected value
----------------------------------	------------------------------------	--

Add Domains

Domain name	<input type="text" value="Optional"/>	<input type="button" value="+"/>
-------------	---------------------------------------	----------------------------------

SSL Settings

Enable SSL No

[Cancel](#) [Add App](#)

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 AWS CodePipeline \(p. 595\)](#).

Step 5: Create a pipeline in AWS CodePipeline

After you have a stack with a layer and at least one instance configured in AWS OpsWorks Stacks, create a pipeline in AWS CodePipeline with AWS OpsWorks Stacks as the provider to deploy apps or Chef cookbooks to your AWS OpsWorks Stacks resources.

To create a pipeline

1. Open the AWS CodePipeline console at <https://console.aws.amazon.com/codepipeline/>.
2. Choose **Create pipeline**.
3. On the **Getting started with AWS 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 \(p. 592\)](#). In this walkthrough, the path is `s3://my-appbucket/opsworks-nodejs-demo-app.zip`. Choose **Next step**.

Source location

Specify where your source code is stored. Choose the provider, and then provide connection details for that provider.

Source provider*

Amazon S3 details

Specify your Amazon S3 location, such as `s3://my-bucket/path/to/object.zip`.

Amazon S3 location*

* Required

6. On the **Build** page, choose **No Build** from the drop-down list, and then choose **Next step**.
7. On the **Deploy** page, choose **AWS OpsWorks Stacks** as the deployment provider.

Beta

Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.

Deployment provider*

AWS OpsWorks (i)

Choose one of your existing stacks.

Stack*

Choose the layer that your taget instances belong to.

Layer

Choose the app that you want to update and deploy, or [create a new one in AWS OpsWorks](#).

App*

The application source that you specified for 'Node.js Demo App' in AWS OpsWorks will use a new Amazon S3 archive, and the repository URL will point to the version of the artifact that you are deploying.
[Learn more](#)

* Required

Cancel

Previous

Next step

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 AWS OpsWorks Stacks \(p. 585\)](#).
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 AWS OpsWorks Stacks \(p. 585\)](#).
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 \(p. 592\)](#), 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.

▼ Hide Details

Role Summary

Role Description Provides read and write access to AWS services and resources.

IAM Role AWS-CodePipeline-Service

Policy Name Create a new Role Policy

▼ Hide Policy Document

Edit

```
{
  "Statement": [
    {
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:GetBucketVersioning"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

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 AWS 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 AWS OpsWorks Stacks

To verify that AWS 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 AWS OpsWorks Stacks \(p. 585\)](#). You should be able to see and use the Node.js web app.

To verify the app deployment in your AWS OpsWorks Stacks instance

1. Open the AWS OpsWorks console at <https://console.aws.amazon.com/opsworks/>.
2. On the AWS OpsWorks Stacks dashboard, choose **CodePipelineDemo**, and then choose **Node.js App Server**.

- 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 screenshot shows the AWS OpsWorks Instances page. At the top, there is a summary bar with the following counts: total 1, online 1, setting up 0, shutting down 0, stopped 0, and errors 0. To the right of the summary is a button labeled "Stop All Instances". Below the summary is a descriptive text box: "An instance represents a server. It can belong to one or more layers, that define the instance's settings, resources, installed packages, profiles and security groups. When you start the instance, OpsWorks uses the associated layer's blueprint to create and configure a corresponding EC2 instance. [Learn more](#)." The main table below has columns: Hostname, Status, Size, Type, AZ, Public IP, and Actions. There is one row for the instance "nodejs-server1", which is currently "online". The Actions column for this row contains links for "stop" and "ssh". A "Search for instances in this layer by name, status, size, type, AZ or IP" input field is located at the top left of the table area. A "Instance" button is located at the bottom left of the table.

Hostname	Status	Size	Type	AZ	Public IP	Actions
nodejs-server1	online	c3.large	24/7	us-east-1a	[Public IP]	stop ssh

The app will be displayed in a new browser tab.

Congratulations!



You just deployed your first app with AWS OpsWorks.

!!! Deployed with CodePipeline !!!

[Twitter](#) Tweet [Follow @AWSOpsWorks](#)



This app runs on app11 (Linux). Your request came from Mozilla/5.0 [REDACTED]. The system time is 9/28/2016, 6:06:43 PM. Page rendered using Node.js version v4.1.1.

Leave a comment

[Send](#)

So cool!

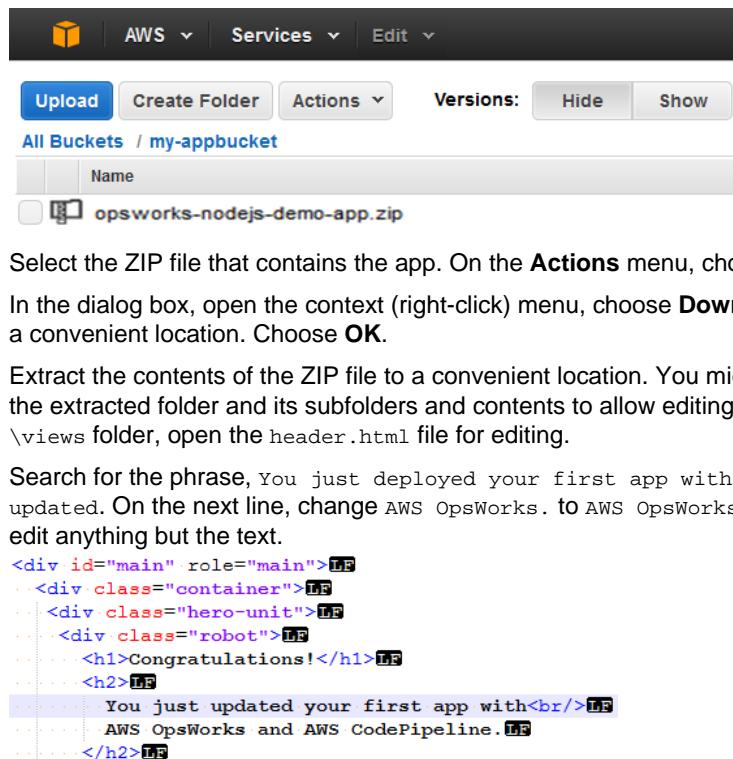
9/28/2016, 12:40:20 AM

Step 7 (Optional): Update the app code to see AWS CodePipeline redeploy your app automatically

When you make changes to code in apps or cookbooks that you have deployed by using AWS CodePipeline, the updated artifacts will be deployed automatically by AWS CodePipeline to your target instances (in this case, to a target AWS 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 `AWS OpsWorks` to `AWS OpsWorks and AWS CodePipeline`. Do not edit anything but the text.

```
<div id="main" role="main">
..<div class="container">
...<div class="hero-unit">
...<div class="robot">
....<h1>Congratulations!</h1>
....<h2>
.....You just updated your first app with<br/>
.....AWS OpsWorks and AWS CodePipeline.<br/>
....</h2>
```

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 AWS CodePipeline console, and open your AWS OpsWorks Stacks pipeline (**MyOpsWorksPipeline**). Choose **Release Change**.

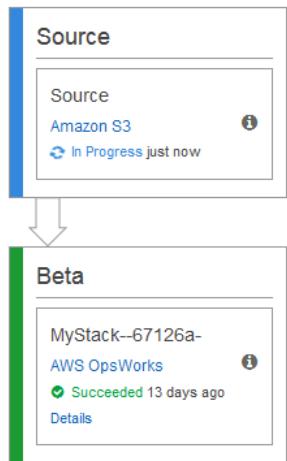
(You can wait for AWS CodePipeline to detect the code change from the updated version of the app in your Amazon S3 bucket. To save you time, this walkthrough instructs you to simply choose **Release Change**.)

11. Observe as AWS CodePipeline runs through the stages of the pipeline. First, AWS CodePipeline detects changes to the source artifact.

MyOpsWorksPipeline

View progress and manage your pipeline.

[Edit](#) [Release change](#)

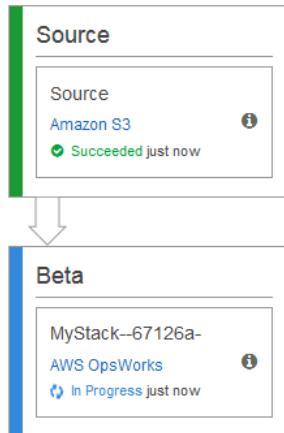


AWS CodePipeline pushes the updated code to your stack in AWS OpsWorks Stacks.

MyOpsWorksPipeline

View progress and manage your pipeline.

[Edit](#) [Release change](#)



12. When both stages of the pipeline have been successfully completed, open your stack in AWS 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.

Congratulations!



You just updated your first app with AWS OpsWorks and AWS CodePipeline.

!!! Deployed with CodePipeline !!!

[Tweet](#) [Follow @AWSOpsWorks](#)



This app runs on app11 (Linux). Your request came from Mozilla/5.0 [REDACTED] [REDACTED]. The system time is 9/28/2016, 6:06:43 PM. Page rendered using Node.js version v4.1.1.

Leave a comment

[Send](#)

So cool!

9/28/2016, 12:40:20 AM

Step 8 (Optional): Clean up resources

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 AWS OpsWorks Stacks stack, the IAM role and instance profile, and the pipeline that you created in AWS CodePipeline. However, you might want to keep using these AWS resources as you continue to learn more about AWS OpsWorks Stacks and AWS 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 AWS 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**. AWS 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 AWS CodePipeline console.
2. In the AWS 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 AWS OpsWorks Stacks - Chef 11 Stacks

[AWS CodePipeline](#) lets you create continuous delivery pipelines that track code changes from sources such as AWS CodeCommit, Amazon Simple Storage Service (Amazon S3), or [GitHub](#). The example in this topic describes how to create and use a simple pipeline from AWS CodePipeline as a deployment tool for code that you run on AWS OpsWorks Stacks layers. In this example, you create a pipeline for a simple [PHP app](#), and then instruct AWS 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 AWS OpsWorks Stacks - Chef 12 Stacks \(p. 583\)](#).

Topics

- [Prerequisites \(p. 603\)](#)
- [Other Supported Scenarios \(p. 604\)](#)
- [Step 1: Create a stack, layer, and an instance in AWS OpsWorks Stacks \(p. 605\)](#)
- [Step 2: Upload app code to an Amazon S3 bucket \(p. 611\)](#)
- [Step 3: Add your app to AWS OpsWorks Stacks \(p. 612\)](#)
- [Step 4: Create a pipeline in AWS CodePipeline \(p. 614\)](#)
- [Step 5: Verifying the app deployment in AWS OpsWorks Stacks \(p. 617\)](#)
- [Step 6 \(Optional\): Update the app code to see AWS CodePipeline redeploy your app automatically \(p. 618\)](#)
- [Step 7 \(Optional\): Clean up resources \(p. 620\)](#)

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 user accounts.

For more information about creating a security group in IAM and assigning permissions to the group, see [Creating Your First IAM User and Administrators Group](#). For more information about managing AWS OpsWorks Stacks permissions, see [Best Practices: Managing Permissions](#). For more information about managing AWS CodePipeline permissions, see [AWS CodePipeline Access Permissions Reference](#) in the AWS CodePipeline documentation.

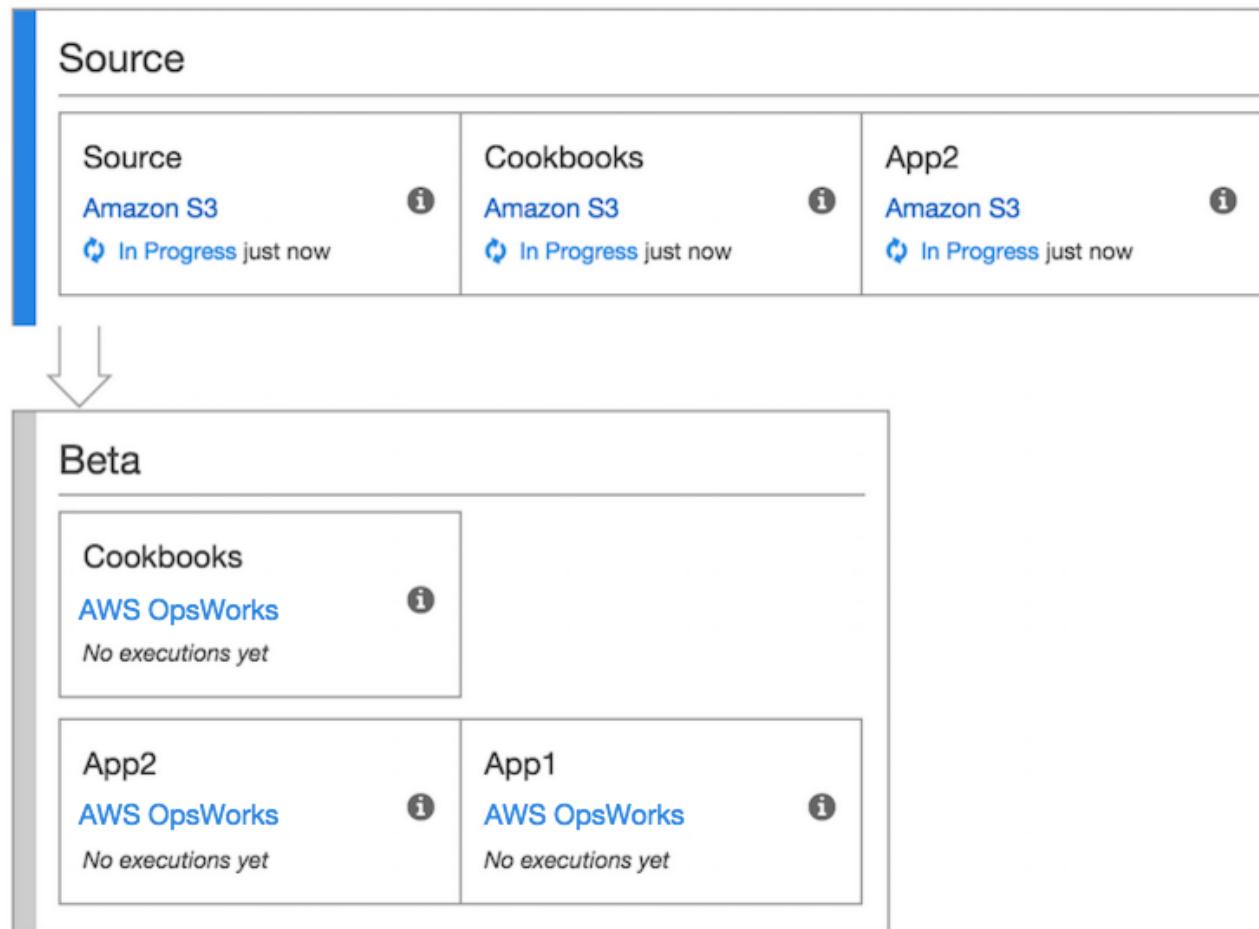
Permissions	Recommended Policy to Attach to Group
Create and edit stacks, layers, and instances in AWS OpsWorks Stacks.	AWSOpsWorksFullAccess
Create, edit, and run templates in AWS CloudFormation.	AmazonCloudFormationFullAccess
Create, edit, and access Amazon S3 buckets.	AmazonS3FullAccess
Create, edit, and run pipelines in AWS CodePipeline, especially pipelines that use AWS OpsWorks Stacks as the provider.	AWSCodePipelineFullAccess

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 AWS 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 AWS 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 AWS CodePipeline, see the [AWS CodePipeline documentation](#).

Step 1: Create a stack, layer, and an instance in AWS OpsWorks Stacks

To use AWS 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 AWS 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 AWS CodePipeline. At this time, AWS CodePipeline supports the AWS 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"
                    },
                    "LayerName": "MyLayer"
                }
            }
        }
    }
}

```

```

    "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**.

Select Template

Select the template that describes the stack that you want to create. A stack is a group of related resources that you manage as a single unit.

<p>Design a template</p> <p>Use AWS CloudFormation Designer to create or modify an existing template. Learn more.</p> <p>Design template</p>	<p>Choose a template</p> <p>A template is a JSON-formatted text file that describes your stack's resources and their properties. Learn more.</p> <p><input type="radio"/> Select a sample template <input checked="" type="radio"/> Upload a template to Amazon S3 <input type="radio"/> Specify an Amazon S3 template URL</p> <p>Browse... NewOpsWorksStack.template</p>
--	---

- 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.
 - In the **Parameters** area, provide the name of an EC2 key pair that you want to use to access your AWS OpsWorks Stacks instance after it has been created. Choose **Next**.
 - On the **Options** page, choose **Next**. (Settings on this page are not required for this walkthrough.)

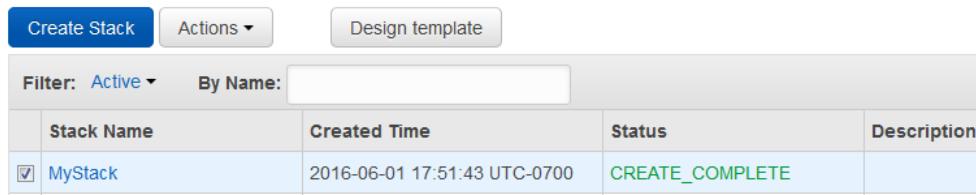
- The AWS CloudFormation template that you use in this walkthrough creates IAM roles, an instance profile, and an instance.

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.

- 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.



A screenshot of the AWS CloudFormation stacks list. At the top, there are three buttons: 'Create Stack', 'Actions ▾', and 'Design template'. Below these are two filters: 'Filter: Active' and 'By Name:'. A table lists one stack:

	Stack Name	Created Time	Status	Description
<input checked="" type="checkbox"/>	MyStack	2016-06-01 17:51:43 UTC-0700	CREATE_COMPLETE	

To verify stack creation in AWS OpsWorks Stacks

- Open the AWS OpsWorks console at <https://console.aws.amazon.com/opsworks/>.
- On the AWS OpsWorks Stacks dashboard, view the stack you created.



- 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

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 AWS CodePipeline can use code directly from GitHub or AWS CodeCommit as sources, this walkthrough demonstrates how to use an Amazon S3 bucket. The Amazon S3 bucket enables AWS 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 AWS CodePipeline as described in [Simple Pipeline Walkthrough \(Amazon S3 Bucket\)](#) in the AWS CodePipeline documentation.

Important

The Amazon S3 bucket must be in the same region in which you later create your pipeline. At this time, AWS CodePipeline supports the AWS 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 AWS CodePipeline requires a versioned source. For more information, see [Using Versioning](#).

To upload your app to an Amazon S3 bucket

- From the [GitHub website](#), download a ZIP file of the AWS OpsWorks Stacks sample PHP app, and save it to a convenient location on your local computer.
- 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`.

Create a Bucket - Select a Bucket Name and Region

A bucket is a container for objects stored in Amazon S3. When creating a bucket, you can choose a Region to optimize for latency, minimize costs, or address regulatory requirements. For more information regarding bucket naming conventions, please visit the [Amazon S3 documentation](#).

Bucket Name:	<input type="text" value="my-appbucket"/>	Cancel
Region:	<input style="width: 100%; height: 100%; border: 1px solid #ccc; padding: 2px; font-size: small;" type="button" value="US Standard"/>	▼
		Set Up Logging > <input style="background-color: #0072bc; color: white; border: 1px solid #0072bc; padding: 2px 10px; font-weight: bold; border-radius: 5px;" type="button" value="Create"/> Cancel

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**.

Upload - Select Files and Folders

Upload to: [All Buckets](#) / [my-appbucket](#)

To upload files (up to 5 TB each) to Amazon S3, click **Add Files**. To upload whole folders to it as it downloads a Java™ Applet (requires [Java SE 7 Update 51 or later](#)). To remove files and folders, click **Remove Selected Files**.

[opsworks-demo-php-simple-app-version1.zip \(33.9 KB\)](#)

Add Files
Remove Selected Files
Enable Enhanced Uploader (BETA)

Number of files: 1 Total upload size: 33.9 KB

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 AWS OpsWorks Stacks

Before you create a pipeline in AWS CodePipeline, add the PHP test app to AWS OpsWorks Stacks. When you create the pipeline, you will need to select the app that you've added to AWS 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 AWS OpsWorks Stacks

1. In the AWS 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 \(p. 611\)](#). Your form should be similar to the following:

Add App

Settings

Name	<input type="text" value="PHPTestApp"/>
Type	<input type="text" value="PHP"/> <input type="button" value="▼"/>
Document root	<input type="text" value="Optional"/>

Data Sources

Data source type	<input checked="" type="radio"/> RDS <input type="radio"/> OpsWorks <input checked="" type="radio"/> None
------------------	---

Application Source

Repository type	<input type="text" value="S3 Archive"/> <input type="button" value="▼"/>
Repository URL	<input type="text" value="eks-demo-php-simple-app-version1.zip"/>
Access key ID	<input type="text" value="Optional"/>
Secret access key	<input type="text" value="Optional"/>

Environment Variables

KEY	VALUE	<input type="checkbox"/> Protected value
-----	-------	--

Add Domains

Domain name	<input type="text" value="Optional"/> <input type="button" value="+"/>
-------------	--

SSL Settings

Enable SSL	<input type="checkbox"/> No
------------	-----------------------------

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 AWS CodePipeline \(p. 614\)](#).

Step 4: Create a pipeline in AWS CodePipeline

After you have a stack with a layer and at least one instance configured in AWS OpsWorks Stacks, create a pipeline in AWS CodePipeline with AWS OpsWorks Stacks as the provider to deploy apps or Chef cookbooks to your AWS OpsWorks Stacks resources.

To create a pipeline

1. Open the AWS CodePipeline console at <https://console.aws.amazon.com/codepipeline/>.
2. Choose **Create pipeline**.
3. On the **Getting started with AWS 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 \(p. 611\)](#). In this walkthrough, the path is `s3://my-appbucket/opsworks-demo-php-simple-app-version1.zip`. Choose **Next step**.

Source location 

Specify where your source code is stored. Choose the provider, and then provide connection details for that provider.

Source provider* 

Amazon S3 details

Specify your Amazon S3 location, such as `s3://my-bucket/path/to/object.zip`.

Amazon S3 location* 

* Required [Cancel](#) [Previous](#) [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 **AWS OpsWorks Stacks** as the deployment provider.

Beta 

Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.

Deployment provider* 

AWS OpsWorks 

Choose one of your existing stacks.

Stack* 

Choose the layer that your target instances belong to.

Layer 

Choose the app that you want to update and deploy, or [create a new one in AWS OpsWorks](#).

App* 

The application source that you specified for 'PHPTestApp' in AWS OpsWorks will use a new Amazon S3 archive, and the repository URL will point to the version of the artifact that you are deploying.
[Learn more](#)

* Required

[Cancel](#)

[Previous](#)

[Next step](#)

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 AWS OpsWorks Stacks \(p. 605\)](#).
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 AWS OpsWorks Stacks \(p. 605\)](#).
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 \(p. 611\)](#), 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.

▼ Hide Details

Role Summary 

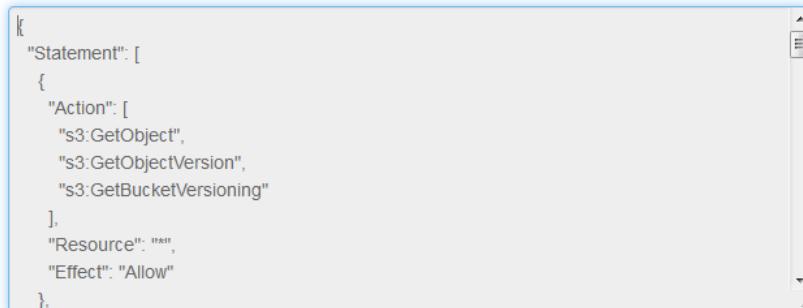
Role Description Provides read and write access to AWS services and resources.

IAM Role AWS-CodePipeline-Service

Policy Name Create a new Role Policy

▼ Hide Policy Document

Edit



The screenshot shows a JSON policy document in a code editor window. The policy defines a single statement allowing S3 actions on all resources with an effect of 'Allow'. The JSON is as follows:

```
{  
  "Statement": [  
    {  
      "Action": [  
        "s3:GetObject",  
        "s3:GetObjectVersion",  
        "s3:GetBucketVersioning"  
      ],  
      "Resource": "*",  
      "Effect": "Allow"  
    },  
    ...  
  ]  
}
```

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 AWS 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 AWS OpsWorks Stacks

To verify that AWS 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 AWS OpsWorks Stacks \(p. 605\)](#). You should be able to see and use the PHP web app.

To verify the app deployment in your AWS OpsWorks Stacks instance

1. Open the AWS OpsWorks console at <https://console.aws.amazon.com/opsworks/>.
2. On the AWS 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.

Instances  Stop All Instances

MyLayer

Search for instances in this layer by name, status, size, type, AZ or IP						
Hostname	Status	Size	Type	AZ	Public IP	Actions
php-app1	online	c3.large	24/7	us-east-1a	54.242.188.34	 stop  ssh

The app will be displayed in a new browser tab.

Simple PHP App

Congratulations!

Your PHP application is now running on the host "php-app1" in your own dedicated environment in the AWS Cloud.

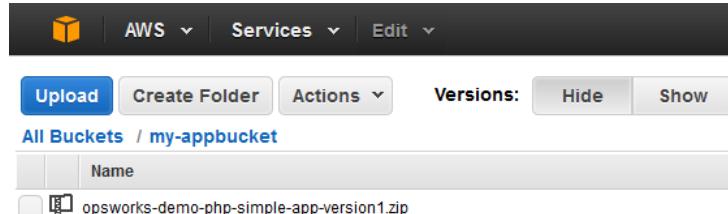
This host is running PHP version 5.3.29.

Step 6 (Optional): Update the app code to see AWS CodePipeline redeploy your app automatically

When you make changes to code in apps or cookbooks that you have deployed by using AWS CodePipeline, the updated artifacts will be deployed automatically by AWS CodePipeline to your target instances (in this case, to a target AWS 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>

```

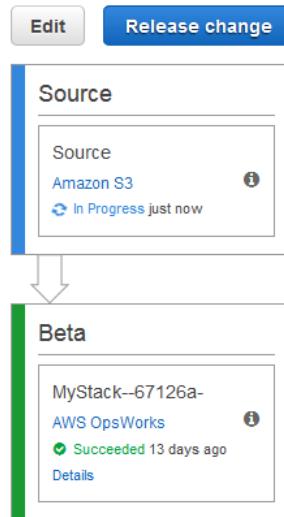
7. Save and close the `index.php` file.
8. Zip the `opsworks-demo-php-simple-app-version1` folder, and save the ZIP file to a convenient location. Do not change the name of the ZIP file.
9. Upload the new ZIP file to your Amazon S3 bucket. In this walkthrough, the name of the bucket is `my-appbucket`.
10. Open the AWS CodePipeline console, and open your AWS OpsWorks Stacks pipeline (**MyOpsWorksPipeline**). Choose **Release Change**.

(You can wait for AWS 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 AWS CodePipeline runs through the stages of the pipeline. First, AWS CodePipeline detects changes to the source artifact.

MyOpsWorksPipeline

View progress and manage your pipeline.

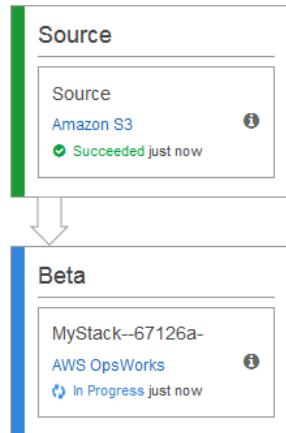


AWS CodePipeline pushes the updated code to your stack in AWS OpsWorks Stacks.

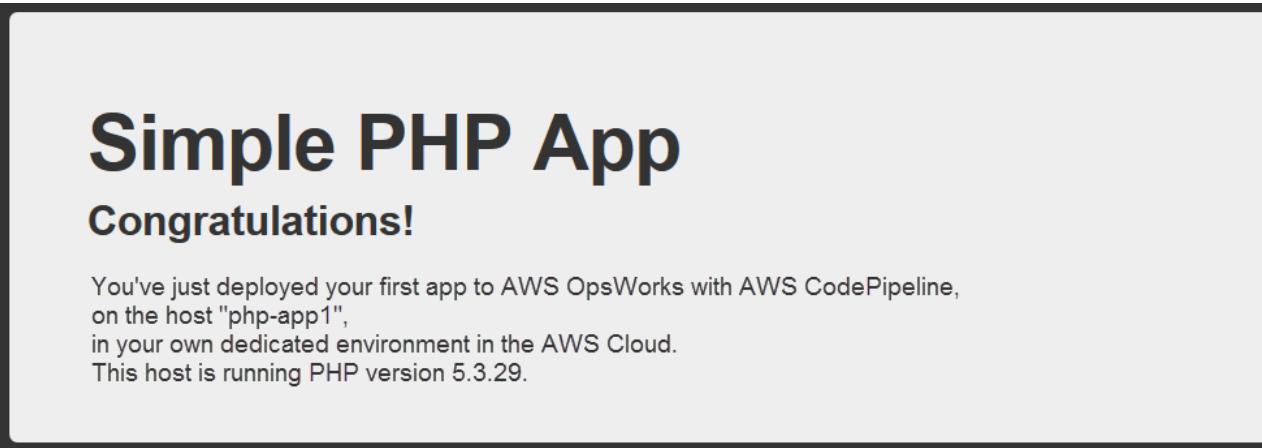
MyOpsWorksPipeline

View progress and manage your pipeline.

[Edit](#) [Release change](#)



12. When both stages of the pipeline have been successfully completed, open your stack in AWS 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

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 AWS OpsWorks Stacks stack, the IAM role and instance profile, and the pipeline that you created in AWS CodePipeline. However, you might want to keep using these AWS resources as you continue to learn more about AWS OpsWorks Stacks and AWS 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 AWS 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**. AWS 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 AWS CodePipeline console.
2. In the AWS 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 AWS OpsWorks Stacks CLI

The AWS OpsWorks Stacks command line interface (CLI) provides the same functionality as the console and can be used for a variety of tasks. The AWS 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 [AWS OpsWorks Stacks reference](#).

Tip

If you are using a Windows-based workstation, you can also run the AWS Tools for Windows PowerShell to perform AWS OpsWorks Stacks operations from the command line. For more information, see [AWS Tools for Windows PowerShell](#).

AWS OpsWorks Stacks commands have the following general format:

```
C:\>aws --region us-east-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-east-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 [AWS OpsWorks Stacks reference](#).

Tip

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 AWS 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 CLI to use a default regional endpoint. For more information, see [Configuring the AWS Region](#).

Region name	Command code
US East (N. Virginia) Region	us-east-1
US East (Ohio) Region	us-east-2
US West (N. California) Region	us-west-1
US West (Oregon) Region	us-west-2
EU (Ireland) Region	eu-west-1
EU (London) Region	eu-west-2
EU (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 AWS OpsWorks Stacks permissions, see [Managing User Permissions \(p. 282\)](#). To determine the permissions required for a particular command, see the command's reference page in the [AWS OpsWorks Stacks reference](#).

The following sections describe how to use the AWS OpsWorks Stacks CLI to perform a variety of common tasks.

Topics

- [Create an Instance \(create-instance\) \(p. 623\)](#)
- [Deploy an App \(create-deployment\) \(p. 624\)](#)
- [List a Stack's Apps \(describe-apps\) \(p. 625\)](#)
- [List a Stack's Commands \(describe-commands\) \(p. 626\)](#)
- [List a Stack's Deployments \(describe-deployments\) \(p. 627\)](#)
- [List a Stack's Elastic IP Addresses \(describe-elastic-ips\) \(p. 627\)](#)
- [List a Stack's Instances \(describe-instances\) \(p. 628\)](#)
- [List an Account's Stacks \(describe-stacks\) \(p. 629\)](#)
- [List a Stack's Layers \(describe-layers\) \(p. 630\)](#)
- [Execute a Recipe \(create-deployment\) \(p. 632\)](#)
- [Install Dependencies \(create-deployment\) \(p. 633\)](#)
- [Update the Stack Configuration \(update-stack\) \(p. 633\)](#)

Create an Instance (create-instance)

Use the [create-instance](#) command to create an instance on a specified stack.

Topics

- [Create an Instance with a Default Host Name \(p. 623\)](#)
- [Create an Instance with a Themed Host Name \(p. 623\)](#)
- [Create an Instance with a Custom AMI \(p. 624\)](#)

Create an Instance with a Default Host Name

```
C:\>aws opsworks --region us-east-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-aefc-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 \(p. 120\)](#). To create the instance, first call [get-hostname-suggestion](#) to generate a name. For example:

```
C:\>aws opsworks get-hostname-suggestion --region us-east-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 \(p. 120\)](#).

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-east-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 AWS OpsWorks Stacks, see [Using Custom AMIs \(p. 172\)](#).

```
C:\>aws opsworks create-instance --region us-east-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-aeeef-28a5376002cd"  
}
```

Deploy an App (create-deployment)

Use the `create-deployment` command to deploy an app to a specified stack.

Topics

- [Deploy an App \(p. 625\)](#)

Deploy an App

```
aws opsworks --region us-east-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"  
}
```

Tip

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)

Use the `describe-apps` command to list a stack's apps or get details about specified apps.

```
aws opsworks --region us-east-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": "https://s3-us-west-2.amazonaws.com/opsworks-tomcat/simplejsp.zip",  
                "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)

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-east-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/008cia91-
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": "008cia91-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-content-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)

Use the [describe-deployments](#) command to list a stack's deployments or get details about specified deployments.

```
aws opsworks --region us-east-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)

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)

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)

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          \"datasources\": {\n            \"ROOT\": {\n              \"jdbc/mydb\"\n            }\n          }\n        },\n        \"Region\": \"us-west-2\",  
        \"DefaultInstanceProfileArn\": \"arn:aws:iam::444455556666:instance-profile/aws-opsworks-ec2-role\",  
        \"CustomCookbooksSource\": {\n          \"Url\": \"git://github.com/example-repo/tomcustom.git\",  
          \"Type\": \"git\"\n        },  
        \"DefaultAvailabilityZone\": \"us-west-2a\",  
        \"HostnameTheme\": \"Layer_Dependent\",  
        \"Attributes\": {\n          \"Color\": \"rgb(45, 114, 184)\"\n        },  
        \"DefaultOs\": \"Amazon Linux\",  
        \"CreatedAt\": \"2013-08-01T22:53:42+00:00\"\n      },  
      {  
        "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": {\n          \"Color\": \"rgb(45, 114, 184)\"\n        },  
        \"DefaultOs\": \"Amazon Linux\",  
      }  
    ]  
}
```

```
        "CreatedAt": "2013-10-25T19:24:30+00:00"
    ]
}
```

List a Stack's Layers (describe-layers)

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)

Use the [create-deployment](#) command to execute [stack commands \(p. 133\)](#) and [deployment commands \(p. 221\)](#). The following example executes a stack command to run a custom recipe on a specified stack.

```

aws opsworks --region us-east-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": "5cbaar7b9-4e09-4e53-aa1b-314fb106038"  
}
```

Install Dependencies (create-deployment)

Use the [create-deployment](#) command to execute [stack commands \(p. 133\)](#) and [deployment commands \(p. 221\)](#). The following example runs the `update_dependencies` stack command to update the dependencies on a stack's instances.

```
aws opsworks --region us-east-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)

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 \(p. 135\)](#).

```
aws opsworks --region us-east-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

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 \(p. 645\)](#) for your specific issue.
2. Search the [AWS OpsWorks Stacks Forum](#) to see if the issue has been discussed there.

The Forum includes many experienced users and is monitored by the AWS OpsWorks Stacks team.
3. For issues with recipes, see [Debugging Recipes \(p. 634\)](#).
4. Contact AWS OpsWorks Stacks support or post your issue on the [AWS OpsWorks Stacks Forum](#).

The following section provides guidance for debugging recipes. The final section describes common debugging and troubleshooting issues and their solutions.

Tip

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 \(p. 635\)](#).

Topics

- [Debugging Recipes \(p. 634\)](#)
- [Common Debugging and Troubleshooting Issues \(p. 645\)](#)

Debugging Recipes

When a lifecycle event occurs, or you run the [Execute Recipes stack command \(p. 133\)](#), AWS OpsWorks Stacks issues a command to the [agent \(p. 641\)](#) 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 \(p. 634\)](#)
- [Chef Logs \(p. 635\)](#)
- [Using the AWS OpsWorks Stacks Agent CLI \(p. 641\)](#)

Logging in to a Failed Instance

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 AWS 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 AWS OpsWorks Stacks built-in support for [SSH \(p. 212\)](#) and [RDP \(p. 214\)](#) 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 \(p. 302\)](#) 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 \(p. 216\)](#). You cannot use a [personal SSH key pair \(p. 302\)](#) to retrieve an Administrator password.

Chef Logs

Chef logs are one of your key troubleshooting resources, especially for debugging recipes. AWS 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 `stderr`. If a recipe fails, the log includes the Chef stack trace.

AWS 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 \(p. 643\)](#).

Topics

- [Viewing a Chef Log with the Console \(p. 635\)](#)
- [Viewing a Chef Log with the CLI or API \(p. 635\)](#)
- [Viewing a Chef Log on an Instance \(p. 636\)](#)
- [Interpreting a Chef Log \(p. 638\)](#)
- [Common Chef Log Errors \(p. 640\)](#)

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 \(p. 133\)](#) command. The following shows an instance's **Logs** section, with **configure** and **setup** commands, which correspond to Configure and Setup lifecycle events.

Logs			
Created at	Command	Duration	Log
✓ 2013-10-02 21:06:56 UTC	configure	00:01:04	show
✓ 2013-10-02 21:01:15 UTC	setup	00:05:40	show

Click **show** in the appropriate command's **Log** column to view the corresponding Chef log. If an error occurs, AWS 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 AWS 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 AWS 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 AWS 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

AWS 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](#) (p. 659), stored as JSON-formatted files.) You need [sudo privileges](#) (p. 282) to access this directory. The log for each run is in a file named `chef.log` inside of the individual run's subdirectory.

AWS 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 AWS 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-
a87b428fffc2b		
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 ### attrs.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-appl.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-appl.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
```

Tip

You can use the agent CLI to display the log's tail during or after the run. For more information, see [Displaying Chef Logs \(p. 643\)](#).

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`.

AWS OpsWorks Stacks runs recipes from the instance's cookbook cache. It downloads cookbooks from your repository to this cache when the instance starts. However, AWS 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 \(p. 133\)](#) 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 `STDERR` output, which should help you determine why the package installation failed.

Using the AWS OpsWorks Stacks Agent CLI

Note

The agent CLI is available only on Linux instances.

On every online instance, AWS OpsWorks Stacks installs an agent, which communicates with the service. The AWS OpsWorks Stacks service in turn sends commands to the agent to perform 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 \(p. 212\)](#). 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 \(p. 376\)](#).

For more information on how to set up an SSH connection to an instance, see [Logging In with SSH \(p. 212\)](#). You must also have [SSH and sudo permissions \(p. 282\)](#) for the stack.

This section describes how to use the agent CLI for troubleshooting. For more information and a complete command reference, see [AWS OpsWorks Stacks Agent CLI \(p. 651\)](#).

Topics

- [Executing Recipes \(p. 642\)](#)
- [Displaying Chef Logs \(p. 643\)](#)
- [Displaying the Stack Configuration and Deployment JSON \(p. 644\)](#)

Executing Recipes

The agent CLI `run_command` ([p. 656](#)) 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 \(p. 133\)](#) 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

AWS OpsWorks Stacks runs recipes from the instance's cookbook cache, not the cookbook repository. AWS 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 \(p. 133\)](#) 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` ([p. 656](#)), 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 \(p. 376\)](#) 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 /path/to/valid/json.file
```

Displaying Chef Logs

The agent CLI [show_log \(p. 656\)](#) 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 \(p. 656\)](#), which returns a list of cached commands and the time that they were performed. For an example, see [Executing Recipes \(p. 642\)](#).

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.

Tip

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 \(p. 642\)](#).
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 \(p. 376\)](#), 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, AWS 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 \(p. 376\)](#).

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 \(p. 652\)](#) 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

This section describes some commonly encountered debugging and troubleshooting issues and their solutions.

Topics

- [Troubleshooting AWS OpsWorks Stacks \(p. 645\)](#)
- [Troubleshooting Instance Registration \(p. 650\)](#)

Troubleshooting AWS OpsWorks Stacks

This section contains some commonly encountered AWS OpsWorks Stacks issues and their solutions.

Topics

- [Unable to Manage Instances \(p. 645\)](#)
- [After a Chef Run, Instances Won't Boot \(p. 646\)](#)
- [A Layer's Instances All Fail an Elastic Load Balancing Health Check \(p. 646\)](#)
- [Can't Communicate with an Elastic Load Balancing Load Balancer \(p. 646\)](#)
- [An Imported On-premises Instance Fails to Finish Volume Setup After a Restart \(p. 647\)](#)
- [An EBS Volume Does Not Reattach After a Reboot \(p. 647\)](#)
- [Can't Delete AWS OpsWorks Stacks Security Groups \(p. 647\)](#)
- [Accidentally Deleted an AWS OpsWorks Stacks Security Group \(p. 648\)](#)
- [Chef Log Terminates Abruptly \(p. 648\)](#)
- [Cookbook Does Not Get Updated \(p. 648\)](#)
- [Instances are Stuck at Booting Status \(p. 648\)](#)
- [Instances Unexpectedly Restart \(p. 648\)](#)
- [opsworks-agent Processes are Running on Instances \(p. 649\)](#)
- [Unexpected execute_recipes Commands \(p. 649\)](#)

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 AWS 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 AWS OpsWorks Stacks. This causes a communication failure between the AWS OpsWorks agent that is installed on the instance, and the AWS OpsWorks Stacks service. The IAM 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 EIP manually. AWS OpsWorks reconfigures an assigned Elastic Load Balancing load balancer each time an instance enters or leaves the online state. AWS OpsWorks only considers instances it knows about to be valid members; instances that are added outside of AWS 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 AWS OpsWorks to manage the load balancer or EIP memberships of AWS OpsWorks instances. When you are registering an instance, to help prevent problems managing registered instances in the event that the IAM 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 AWS 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 \(p. 313\)](#) 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 AWS OpsWorks Stacks, and the AWS OpsWorks Stacks console displays **failed** as the instance status. This can occur on either Chef 11 or Chef 12 instances.

Cause: AWS OpsWorks Stacks might be unable to attach a volume to your instance during the setup process. One possible cause is that AWS 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 AWS 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: AWS 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 AWS 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 AWS 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 \(p. 256\)](#).

Can't Delete AWS OpsWorks Stacks Security Groups

Problem: After you delete a stack, there are a number of AWS 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 AWS OpsWorks Stacks Security Group

Problem: You deleted one of the AWS OpsWorks Stacks security groups and need to recreate it.

Cause: These security groups are usually deleted by accident.

Solution: The recreated group must be 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 AWS OpsWorks Stacks perform the task for you. Just create a new stack in the same AWS region—and VPC, if present—and AWS 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.

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 \(p. 642\)](#).

Cookbook Does Not Get Updated

Problem: You updated your cookbooks but the stack's instances are still running the old recipes.

Cause: AWS OpsWorks Stacks caches cookbooks on each instance, and runs recipes from the cache, not the repository. When you start a new instance, AWS OpsWorks Stacks downloads your cookbooks from the repository to the instance's cache. However, if you subsequently modify your custom cookbooks, AWS OpsWorks Stacks does not automatically update the online instances' caches.

Solution: Run the [Update Cookbooks stack command \(p. 133\)](#) to explicitly direct AWS 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 AWS 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 AWS OpsWorks Stacks service. Because AWS OpsWorks Stacks can no longer communicate with the instance [agent \(p. 641\)](#), it treats the instance as failed and [auto heals \(p. 145\)](#) it. However, without a connection, AWS OpsWorks Stacks cannot install an instance agent on the healed instance. Without an agent, AWS 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 \(p. 145\)](#) for your instances, AWS 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 AWS OpsWorks Stacks-managed instance by using the Amazon

EC2 console, API, or CLI, AWS 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 AWS OpsWorks Stacks console, API, or CLI. If you use AWS 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 \(p. 178\)](#) and [Deleting AWS OpsWorks Stacks Instances \(p. 211\)](#).

Cause 2: Instances can fail for a variety of reasons. If you have auto healing enabled, AWS OpsWorks Stacks automatically restarts failed instances.

Solution: This is normal operation; there is no need to do anything unless you do not want AWS 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, AWS OpsWorks Stacks runs `execute_recipes` to update the stack's instances and also triggers a Configure event. Another source of `execute_recipes` commands is AWS 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.

Ran command `execute_recipes`

[Repeat](#)

Status	successful	User	OpsWorks
Created at	2014-02-21 17:15:40 UTC	Recipes	ssh_users
Completed at	2014-02-21 17:16:32 UTC		
Duration	00:00:52		
<hr/>			
Hostname		SSH	Layers
php-app1		PHP App Server	00:00:52 show

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`. AWS 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.

Tip

If you are having registration problems, run `register` with the `--debug` argument, which provides additional debugging information.

Topics

- [EC2User Is Not Authorized to Perform: ... \(p. 650\)](#)
- [Credential Should Be Scoped to a Valid Region \(p. 650\)](#)

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 IAM user 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 \(p. 194\)](#).

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 AWS OpsWorks Stacks region. For a list of supported regions, see [Region Support \(p. 34\)](#). 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; AWS 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 AWS OpsWorks Stacks.

Solution: Explicitly set `--region` to a supported AWS OpsWorks Stacks region, or edit your AWS CLI `config` file to change the default region to a supported AWS OpsWorks Stacks region. For more information, see [Configuring the AWS Command Line Interface](#).

AWS OpsWorks Stacks Agent CLI

Note

This feature is available only on Linux instances.

The agent that AWS OpsWorks Stacks installs on every instance exposes a command line interface (CLI). If you [use SSH to log in \(p. 212\)](#) to the instance, you can use the CLI to the following:

- Access log files for Chef runs.
- Access AWS 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 \(p. 652\)](#)
- [get_json \(p. 652\)](#)
- [instance_report \(p. 655\)](#)
- [list_commands \(p. 656\)](#)
- [run_command \(p. 656\)](#)
- [show_log \(p. 656\)](#)
- [stack_state \(p. 657\)](#)

activity

(Optional) Used as an argument with some commands to specify a particular AWS OpsWorks Stacks activity: `setup`, `configure`, `deploy`, `undeploy`, `start`, `stop`, or `restart`.

date

(Optional) Used as an argument with some commands to specify a particular AWS 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 AWS OpsWorks Stacks command was executed, run [list_commands \(p. 656\)](#).

Note

If the agent has executed the same AWS 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 \(p. 652\)](#)
- [get_json \(p. 652\)](#)
- [instance_report \(p. 655\)](#)
- [list_commands \(p. 656\)](#)
- [run_command \(p. 656\)](#)
- [show_log \(p. 656\)](#)
- [stack_state \(p. 657\)](#)

agent_report

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

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 \(p. 656\)](#).

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 \(p. 656\)](#).

-i, --internal

Displays information that AWS 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 AWS OpsWorks Stacks creates on the instance.

For more information, see the [AWS OpsWorks Stacks Data Bag Reference \(p. 659\)](#).

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 AWS 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-0d4f07cd1EX",
    "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-0d4f07cd1EX"
            ],
            "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...g50tgQ==\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/opsworks-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

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

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

Runs an AWS OpsWorks Stacks command, which is a JSON file containing a Chef run-list that contains the information necessary to execute an AWS 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 \(p. 656\)](#). This option is intended only for development purposes, so AWS 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 AWS OpsWorks Stacks command. Use the following options to specify a particular command.

activity

Run a specified AWS 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 \(p. 656\)](#).

file

Run the specified command JSON file. To get a command's file path, run [get_json \(p. 652\)](#).

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

Returns 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 \(p. 656\)](#).

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

Displays information that AWS 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 AWS OpsWorks Stacks creates on the instance.

For more information, see the [AWS OpsWorks Stacks Data Bag Reference \(p. 659\)](#).

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-0d4f07cd1EX"
    ]
}
```

```

],
"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...g50tgQ==\n-----END DSA PRIVATE KEY-----\n",
"ssh_host_rsa_key_private": "-----BEGIN RSA PRIVATE KEY-----\nMIIEowIB...78kpriIw\n-----END RSA PRIVATE KEY-----\n",
},
"layers": [
{
"layer_id": "93f50d83-1e73-45c4-840a-0d4f07cdalEX",
"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/opsworks-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"
    ]
}
```

AWS OpsWorks Stacks Data Bag Reference

AWS 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. AWS OpsWorks Stacks stores its data bags on each stack's instances. On Linux instances, AWS 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 AWS 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.

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. AWS OpsWorks Stacks provides the following search indexes:

- [aws_opsworks_app](#) ([p. 661](#)), which represents a set of deployed apps for a stack.
- [aws_opsworks_command](#) ([p. 663](#)), which represents a set of commands that were run on a stack.
- [aws_opsworks_ecs_cluster](#) ([p. 665](#)), which represents a set of Amazon EC2 Container Service (Amazon ECS) cluster instances for a stack.
- [aws_opsworks_elastic_load_balancer](#) ([p. 665](#)), which represents a set of Elastic Load Balancing load balancers for a stack.
- [aws_opsworks_instance](#) ([p. 666](#)), which represents a set of instances for a stack.
- [aws_opsworks_layer](#) ([p. 669](#)), which represents a set of layers for a stack.
- [aws_opsworks_rds_db_instance](#) ([p. 670](#)), which represents a set of Amazon Relational Database Service (Amazon RDS) instances for a stack.
- [aws_opsworks_stack](#) ([p. 671](#)), which represents a stack.
- [aws_opsworks_user](#) ([p. 673](#)), 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-chef-12-linux-demo/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 AWS 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 \(aws_opsworks_app\) \(p. 661\)](#)
- [Command Data Bag \(aws_opsworks_command\) \(p. 663\)](#)
- [Amazon ECS Cluster Data Bag \(aws_opsworks_ecs_cluster\) \(p. 665\)](#)
- [Elastic Load Balancing Data Bag \(aws_opsworks_elastic_load_balancer\) \(p. 665\)](#)
- [Instance Data Bag \(aws_opsworks_instance\) \(p. 666\)](#)
- [Layer Data Bag \(aws_opsworks_layer\) \(p. 669\)](#)
- [Amazon RDS Data Bag \(aws_opsworks_rds_db_instance\) \(p. 670\)](#)
- [Stack Data Bag \(aws_opsworks_stack\) \(p. 671\)](#)
- [User Data Bag \(aws_opsworks_user\) \(p. 673\)](#)

App Data Bag (aws_opsworks_app)

For a [Deploy event \(p. 253\)](#) or an [Execute Recipes stack command \(p. 133\)](#), 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 (p. 662)	app_source (p. 662)	data_sources (p. 662)
---------------------------------	-------------------------------------	---------------------------------------

deploy (p. 663)	attributes (p. 662)	domains (p. 663)
enable_ssl (p. 663)	environment (p. 663)	name (p. 663)
shortname (p. 663)	ssl_configuration (p. 663)	type (p. 663)

app_id

The app ID (string). A GUID that identifies the app.

app_source

A set of content that specifies the information that AWS 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 \(p. 228\)](#) 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, AWS 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 \(p. 217\)](#). 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 AWS 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)

Represents settings for a command that AWS 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 (p. 664)	command_id (p. 664)	iam_user_arn (p. 664)
instance_id (p. 664)	sent_at (p. 664)	type (p. 664)

args

Arguments for the command (string).

command_id

The command's random unique identifier, assigned by AWS 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 AWS 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)

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 \(p. 665\)](#)

[ecs_cluster_name \(p. 665\)](#)

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)

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 (p. 666)	dns_name (p. 666)
-------------------------------------	-------------------

layer_id (p. 666)

elastic_load_balancer_name

The load balancer's name (string).

dns_name

The load balancer's DNS name (string).

layer_id

The AWS OpsWorks Stacks ID of the layer that the load balancer is assigned to (string).

Instance Data Bag (aws_opsworks_instance)

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 AWS 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 (p. 667)	architecture (p. 667)	auto_scaling_type (p. 667)
availability_zone (p. 667)	created_at (p. 667)	ebs_optimized (p. 667)
ec2_instance_id (p. 667)	elastic_ip (p. 667)	hostname (p. 667)
instance_id (p. 667)	instance_type (p. 668)	layer_ids (p. 668)
os (p. 668)	private_dns (p. 668)	private_ip (p. 668)
public_dns (p. 668)	public_ip (p. 668)	root_device_type (p. 668)
root_device_volume_id (p. 668)	self (p. 668)	ssh_host_dsa_key_fingerprint (p. 668)
ssh_host_dsa_key_private (p. 668)	ssh_host_dsa_key_public (p. 669)	ssh_host_rsa_key_fingerprint (p. 669)
ssh_host_rsa_key_private (p. 669)	ssh_host_rsa_key_public (p. 669)	status (p. 669)
subnet_id (p. 669)	virtualization_type (p. 669)	

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-ddThh: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 AWS 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, 307ut73c-c7e4-40cc-52f0-67d5k1f8882c.

os

The instance's operating system (string). Valid values include:

- "Amazon Linux 2016.09"
- "Amazon Linux 2015.03"
- "Amazon Linux 2015.09"
- "Amazon Linux 2016.03"
- "Custom"
- "Microsoft Windows Server 2012 R2 Base"
- "CentOS 7"
- "Red Hat Enterprise Linux 7"
- "Ubuntu 12.04 LTS"
- "Ubuntu 14.04 LTS"
- "Ubuntu 16.04 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 AWS 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)

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 (p. 670)	layer_id (p. 670)	name (p. 670)
packages (p. 670)	shortname (p. 670)	type (p. 670)
volume_configurations (p. 670)		

ecs_cluster_arn

If the layer has an Amazon ECS cluster assigned, the Amazon ECS cluster's Amazon Resource Name (ARN) (string).

layer_id

The layer ID, which is a GUID that is generated by AWS 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, or provisioned IOPS.

Amazon RDS Data Bag (aws_opsworks_rds_db_instance)

A set of data bag content that specifies an Amazon Relational Database Service (Amazon RDS) instance's configuration as follows:

address (p. 671)	db_instance_identifier (p. 671)	db_password (p. 671)
db_user (p. 671)	engine (p. 671)	rds_db_instance_arn (p. 671)

[region \(p. 671\)](#)

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)

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 (p. 672)	custom_cookbooks_source (p. 672)	name (p. 672)
region (p. 672)	stack_id (p. 672)	use_custom_cookbooks (p. 672)
vpc_id (p. 672)		

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](#) (p. 228) 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)

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 (p. 673)	iam_user_arn (p. 673)	remote_access (p. 673)
ssh_public_key (p. 673)	unix_user_id (p. 673)	username (p. 673)

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 AWS OpsWorks Stacks console or API (string).

unix_user_id

The user's Unix ID (number).

username

The user name (string).

OpsWorks Agent Changes

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
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

Agent Version	Description	Release Date
4020	<ul style="list-style-type: none"> Update Chef client version to 12.16.42 	December 8, 2016
4019	<ul style="list-style-type: none"> Source proxy variables during agent installation Red Hat Enterprise Linux 7 now uses <code>systemd</code> instead of <code>monit</code> Don't setup EPEL on Red Hat Enterprise Linux 7 Use <code>flock</code> instead of <code>lockrun.c</code> for process locking Avoid odd output of <code>ps -p1</code> when checking for <code>systemd</code> 	October 19, 2016
4018	<ul style="list-style-type: none"> Update Chef client version to 12.13.37 Add support for Amazon Linux 2016.09 	August 25, 2016
4017	<ul style="list-style-type: none"> Update Chef client version to 12.12.15 	August 10, 2016
4016	<ul style="list-style-type: none"> Fix agent uninstallation on systems where <code>monit</code> is not used 	June 23, 2016
4015	<ul style="list-style-type: none"> Fix ECS setup for Amazon Linux 2016.03 	June 17, 2016
4011	<ul style="list-style-type: none"> Update Chef client version to 12.10.24 Improve log upload handling 	May 19, 2016
4008	<ul style="list-style-type: none"> Add support for Amazon Linux 2016.03 Add timeout to bundle install Add <code>xfs</code> to <code>/etc/filesystems</code> if it exists 	March 16, 2016
4007	<ul style="list-style-type: none"> 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
4006	<ul style="list-style-type: none"> Update Chef client version to 12.6.0 Don't install <code>libxml2-devel/libxml2-dev</code> and <code>libxslt-devel/libxslt-dev</code> packages on agent install 	January 21, 2016
4005	<ul style="list-style-type: none"> Fix <code>ec2</code> import by always enabling <code>ec2</code> data in <code>ohai</code> for <code>ec2</code> infrastructure 	December 17, 2015
4004	<ul style="list-style-type: none"> AWS OpsWorks Stacks support for Chef 12 Linux- Chef Client 12.5.1 	December 3, 2015

AWS OpsWorks Stacks Resources

The following related resources can help you as you work with this service.

Topics

- [Reference Guides, Tools, and Support Resources \(p. 675\)](#)
- [AWS Software Development Kits \(p. 676\)](#)
- [Open Source Software \(p. 676\)](#)

Reference Guides, Tools, and Support Resources

Several helpful guides, forums, contact info, and other resources are available from AWS OpsWorks Stacks and Amazon Web Services.

- **[AWS OpsWorks Stacks API Reference](#)** – Descriptions, syntax, and usage examples about AWS OpsWorks Stacks actions and data types, including common parameters and error codes.
- **[AWS OpsWorks Stacks Technical FAQ](#)** – Top questions developers have asked about this product.
- **[AWS 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.
- **[AWS OpsWorks Stacks Command Line Reference](#)** – AWS OpsWorks Stacks-specific commands for use at a command line prompt.

- **[Classes & Workshops](#)** – Links to role-based and specialty courses as well as self-paced labs to help sharpen your AWS skills and gain practical experience.
- **[AWS Developer Tools](#)** – Links to developer tools, SDKs, IDE toolkits, and command line tools for developing and managing AWS applications.
- **[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.
- **AWS Support** – The primary web page for information about AWS 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 AWS 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** – [Setup](#) and [other documentation](#)
- **AWS SDK for Ruby** – [Documentation](#)
- **AWS SDK for JavaScript in Node.js** – [Setup](#) and [other documentation](#)
- **AWS SDK for Python (Boto)** – [Setup](#) and [other documentation](#)

Open Source Software

AWS 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, go to the [OpsWorks Linux Agent Attributions Document](#).

History

The following table describes the important changes to the documentation in this release of AWS OpsWorks Stacks and AWS OpsWorks for Chef Automate.

- **AWS OpsWorks for Chef Automate API version:** 2016-11-01
- **AWS OpsWorks Stacks API version:** 2016-03-08
- **Latest documentation update:** December 1, 2016

Change	Description	Date Changed
New Feature and New Documentation	Added the new AWS OpsWorks for Chef Automate service and documentation.	December 1, 2016
New Feature	Added support for the US East (Ohio) Region regional endpoint.	October 12, 2016
New Feature	Added support for stacks and instances that run the Amazon Linux 2016.09 operating system.	September 30, 2016
New Feature	Added support for the Asia Pacific (Seoul) Region and nine additional regional endpoints.	August 15, 2016
New Feature	Added support for Node.js 0.12.15 and Ruby 2.3 in built-in layers.	July 6, 2016
New Feature and New Documentation	Added support for the Asia Pacific (Mumbai) Region.	June 28, 2016
New Feature	Added support for stacks and instances that run the CentOS 7 operating system.	June 22, 2016
New Feature and New Documentation	Added walkthrough describing AWS CodePipeline and AWS OpsWorks Stacks integration.	June 2, 2016
New Feature	Added support for stacks and instances that run the Ubuntu 16.04 LTS operating system.	June 1, 2016
New Feature and New Documentation	Added Chef 12 Linux support and related documentation.	December 3, 2015

Change	Description	Date Changed
New Documentation	Added Node.js walkthrough to Getting Started.	July 14, 2015
New Documentation	Added two new cookbook examples to Cookbooks 101.	July 14, 2015
New Feature	Added support for agent version management.	June 23, 2015
New Feature	Added support for managing the agent version.	June 24, 2015
New Feature	Added support for custom Windows AMIs.	June 22, 2015
New Documentation	Added three new Best Practices topics.	June 11, 2015
New Feature	Added support for Windows stacks.	May 18, 2015
New Documentation	Added a Best Practices chapter.	Dec. 15, 2014
New Feature	Added support for Elastic Load Balancing connection draining and custom Shutdown timeouts.	Dec. 15, 2014
New Feature	Added support for registering instances created outside of AWS OpsWorks Stacks.	Dec 9, 2014
New Feature	Added support for Amazon SWF.	Sept. 4, 2014
New Feature	Added support for associating environment variables with apps and extended Cookbooks 101.	July 16, 2014
New Documentation	Added Cookbooks 101, a tutorial introduction to implementing cookbooks.	July 16, 2014
New Feature	Added support for CloudTrail.	June 4, 2014
New Feature	Added support for Amazon RDS.	May 14, 2014
New Feature	Added support for Chef 11.10 and Berkshelf.	Mar. 27, 2014
New Feature	Added support for Amazon EBS PIOPS volumes.	Dec. 16, 2013
New Feature	Added resource-based permissions.	Dec. 5, 2013
New Feature	Added resource management.	Oct. 7, 2013
New Feature	Added support for VPCs.	Aug. 29, 2013
New Features	Added support for custom AMIs and Chef 11.4.	July 24, 2013
New Feature	Added console support for multiple layers per instance.	July 1, 2013
New Features	Added support for Amazon EBS-backed instances, Elastic Load Balancing, and Amazon CloudWatch monitoring.	May 14, 2013
Initial Release	Initial release of the AWS OpsWorks Stacks User Guide.	February 18, 2013