



Ansible for AWS

A simple way to provision and manage
your Amazon Cloud infrastructure

Yan Kurniawan

Packt

Ansible for AWS

A simple way to provision and manage your Amazon Cloud infrastructure

Yan Kurniawan



BIRMINGHAM - MUMBAI

Ansible for AWS

Copyright © 2016 Yan Kurniawan

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: October, 2016

Production reference: 1201016

Published by Packt Publishing Ltd.

Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78646-919-9

www.packtpub.com

Credits

Author

Yan Kurniawan

Technical Editors

Abhishek Kotian

Pratish Shetty

Acquisition Editor

Frank Pohlmann

Indexer

Tejal Daruwale Soni

Content Development Editor

Rohit Singh

Production Coordinator

Melwyn Dsa

About the Author

Yan Kurniawan is a Sydney-based Cloud Engineer who is working at Flux7 Labs Inc. Yan has worked on many AWS projects and he has helped companies from startups to Fortune 100 companies. He is an AWS Certified Solutions Architect Professional and a Puppet Certified Professional.

www.PacktPub.com

For support files and downloads related to your book, please visit www.PacktPub.com.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<https://www.packtpub.com/mapt>

Get the most in-demand software skills with Mapt. Mapt gives you full access to all Packt books and video courses, as well as industry-leading tools to help you plan your personal development and advance your career.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Table of Contents

Preface	1
<hr/>	
Chapter 1: Getting Started with AWS	7
What is Amazon Web Services?	7
EC2 – Elastic Compute Cloud	9
VPC – Virtual Private Cloud	9
RDS – Relational Database Service	11
S3 – Simple Storage Service	11
ELB – Elastic Load Balancing	12
Route 53	13
Setting up your AWS account	15
AWS management console	23
Create your first EC2 instance	27
Create a key pair	28
Create a security group	31
Connect to your instance	38
Connect from Mac or Linux using an SSH client	40
Terminate your instance	40
<hr/>	
Chapter 2: Getting Started with Ansible	43
What you'll need	43
Installing Ansible	44
SSH keys	45
Inventory	46
Host variables	47
Group variables	48
Your first commands	49
Playbooks	50
The hosts section	52
The variables section	54
The tasks section	55
Handlers	56
Your first playbook	56
Roles and include statements	58

Chapter 3: EC2 Provisioning and Configuration Management with Ansible	61
Python boto	61
AWS access keys	64
Cloud modules	66
Key pair	67
Security groups	71
EC2 provisioning	76
Elastic IP address (EIP)	84
Configuration management	87
Dynamic inventory	92
Elastic Block Store – EBS	100
Instance profile	103
Chapter 4: Project 1 - A WordPress Site	106
Provisioning playbook	106
Variables file	109
Roles directory	109
Common roles	110
Web roles	110
mysql roles	110
wordpress roles	111
site.yml	114
Directory structure	115
Playbook run	115
Chapter 5: Route 53 Management with Ansible	117
Hosted zones	117
DNS records management	119
Chapter 6: VPC Provisioning with Ansible	124
The default VPC	124
Getting started with VPC	126
Route tables	134
Main route table	134
Custom route tables	135
VPC provisioning	137
VPC security groups	142
EC2-VPC provisioning	148
NAT instance	150

Multi-AZ deployment	153
Ansible in VPC	156
OpenVPN server	160
Configure OpenVPN server	163
Connect client	165
Getting VPC and subnet ID	166
Chapter 7: RDS Provisioning with Ansible	169
Introduction	169
DB instances	170
Regions and availability zones	171
Security groups	171
DB parameter groups	172
DB option groups	172
How you are charged for Amazon RDS	172
Getting started with RDS	173
The <code>rds_subnet_group</code> module	179
The <code>rds_param_group</code> module	182
The <code>rds</code> module	185
Chapter 8: S3 Management with Ansible	192
Bucket	192
Objects	194
Ansible's S3 module	195
Creating S3 bucket	198
Virtual directory	199
Uploading File	200
Sharing a file	202
Downloading file	202
Deleting S3 bucket	203
Backup storage	203
Using <code>s3cmd</code> with Ansible	205
Chapter 9: Using AWS CLI with Ansible	207
Installing the AWS CLI	210
Configuring the AWS CLI	211
Configuration settings and precedence	211
Configuring credentials	211
Configuring the AWS Region	213
Using the AWS CLI	213
Getting help with the AWS CLI	214

AWS CLI in Ansible playbook	215
Getting VPC information based on name tag	216
Create a DHCP options set and associate it with a VPC	218
Idempotent EC2 instance creation based on Name tag	219
Start or stop an instance with particular Name tag	220
Chapter 10: Project 2 - A Multi-Tier WordPress Site	222
Multi-AZ RDS provisioning	224
Master WordPress instance	226
Next steps	231
Chapter 11: Amazon Machine Images - AMI	232
The ec2_ami module	233
Creating custom AMI	235
Deleting custom AMI	236
Chapter 12: Auto Scaling and Elastic Load Balancing – ELB	238
What is Auto Scaling?	238
How Auto Scaling works?	239
Availability zones and regions	240
The architecture	241
VPC update	244
Security groups update	245
Getting started with Auto Scaling	247
Step 1 – Create a launch configuration	247
Step 2 – Create an Auto Scaling group	251
Step 3 – Verify your Auto Scaling group	253
Scaling the size of your Auto Scaling group	255
Scaling manually using the console	258
Load Balance your Auto Scaling group	259
What is Elastic Load Balancing?	260
Features of Elastic Load Balancing	260
Getting started with ELB	261
Step 1 – Define load balancer	261
Step 2 – Assign security groups to your load balancer	264
Step 3 – Configure security settings	264
Step 4 – Configure health checks for your EC2 instances	265
Step 5 – Register EC2 instances with your load balancer	266
Step 6 – Tag your load balancer (Optional)	266
Step 7 – Create and verify your load balancer	267
Attaching load balancer to ASG	268

Verify your Load Balancer	269
Scheduled scaling	271
Considerations for scheduled actions	271
Create a scheduled action using the console	272
Update a scheduled action	273
Dynamic scaling	273
Scaling adjustment types	274
Scaling policy types	275
Simple scaling policies	276
Step scaling policies	276
Step adjustments	276
Instance warm-up	278
Scaling based on metrics	279
Create an Auto Scaling group with scaling policies	279
Add a scaling policy to an Auto Scaling group	283
More about Auto Scaling	284
Scaling based on amazon SQS	284
Auto Scaling cooldowns	285
Auto Scaling instance termination	285
Auto Scaling lifecycle hooks	286
Temporarily removing instances from your Auto Scaling group	286
Suspending and resuming Auto Scaling processes	287
Latest AWS Auto Scaling user guide	287
Chapter 13: ELB and Auto Scaling with Ansible	288
The ec2_elb_lb module	288
The ec2_lc module	291
The ec2_asg module	294
The ec2_scaling_policy module	297
The ec2_metric_alarm module	299
The Playbook	300
Chapter 14: Identity and Access Management (IAM)	304
Accessing IAM	305
Getting started with IAM	306
IAM users	306
IAM policy	307
IAM group	307
IAM best practices	308
The iam module	311
The iam_policy module	313

Example playbooks	314
Instance profile	318
5 ddYbXJI 5 : Installing Ansible from Source	XXXXXXXXXX 321
5 ddYbXJI 6 : What's New in Ansible v2	HG3
Index	325

Preface

Since the CFEngine project by Mark Burgess began in 1993, configuration management tools have been revolutionizing IT operations. Followed by the emergence of Puppet and Chef, which later gain more popularity, there are now many choices available to do IT automation. The new generation of configuration management tools can build servers in seconds and automate your entire infrastructure.

Ansible, first released in 2012, is one of the newer tools in the IT automation space. While other tools like Puppet and Chef focused on completeness and configurability, Ansible focused on simplicity and low learning curve, without sacrificing security and reliability.

Amazon Web Services (AWS) began offering IT infrastructure services to businesses in 2006, in the form of web services – now commonly known as cloud computing. One of the key benefits of cloud computing is the opportunity to replace up-front capital infrastructure expenses with low variable costs that scale with your business. With the Cloud, businesses no longer need to plan for and procure servers and other IT infrastructure weeks or months in advance. Instead, they can instantly spin up hundreds or thousands of servers in minutes and deliver results faster.

This book will show you how to use Ansible's cloud modules to easily provision and manage AWS resources including EC2, VPC, RDS, S3, ELB, Auto Scaling, IAM and Route 53. This book takes you beyond the basics of Ansible, showing you real-world examples of AWS infrastructure automation and management using Ansible, with detailed steps, complete codes, and screen captures from AWS console.

The example projects will help you grasp the concepts quickly. From a single WordPress site, to a highly available and scalable WordPress site, Ansible will help you automate all tasks.

What this book covers

Chapter 1, *Getting Started with AWS*, shows you how to sign up for AWS (Amazon Web Services), set up your account, get familiar with AWS console, create your first Amazon EC2 (Elastic Compute Cloud) instance, and connect to your EC2 instance using SSH.

Chapter 2, *Getting Started with Ansible*, teaches you the basics of Ansible, how to build an inventory, how to use modules, and create Ansible playbooks to manage your hosts.

Chapter 3, *EC2 Provisioning and Configuration Management with Ansible*, teaches you how to use Ansible playbook to configure and launch EC2 instances, and use dynamic inventory to manage EC2 instances.

Chapter 4, *Project 1 - A WordPress Site*, gives you an example project to provision a simple WordPress site in AWS cloud.

Chapter 5, *Route 53 Management with Ansible*, teaches you how to create and delete Domain Name System (DNS) records in Amazon Route 53 DNS web service using Ansible.

Chapter 6, *VPC Provisioning with Ansible*, delves deeper into AWS cloud infrastructure and teaches you how to use Ansible to create and manage Virtual Private Cloud (VPC), VPC subnets, VPC routing tables, and VPC Security Groups, also how to use Ansible to launch EC2 instances in VPC subnet and attach VPC security groups to the instances.

Chapter 7, *RDS Provisioning with Ansible*, teaches you how to use Ansible to provision Amazon Relational Database Service (RDS), replicate RDS database, take snapshot, and restore backup.

Chapter 8, *S3 Management with Ansible*, teaches you how to manage files in an Amazon Simple Storage Service (S3) bucket using Ansible.

Chapter 9, *Using AWS CLI with Ansible*, shows you how to use AWS CLI to extend Ansible functionality in AWS environment.

Chapter 10, *Project 2 - A Multi-Tier WordPress Site*, an example project to build a highly available and scalable WordPress site.

Chapter 11, *Amazon Machine Images (AMI)*, teaches you how to create AMI from an EC2 instance.

Chapter 12, *Auto Scaling and Elastic Load Balancing (ELB)*, introduces you to Elastic Load Balancing and Auto Scaling.

Chapter 13, *ELB and Auto Scaling with Ansible*, teaches you how to use Ansible to provision ELB and Auto Scaling Groups.

Chapter 14, *Identity and Access Management (IAM)*, shows you how to use Ansible to manage IAM users, groups, roles and keys.

Appendix A, *Installing Ansible from Source*.

Appendix B, *What's New in Ansible v2*.

What you need for this book

To run the examples in this book, you will need a computer with Linux operating system and an Internet connection. To use the services in Amazon Web Services, you will need to setup an account and register your credit card with Amazon.

Who this book is for

The book assumes that the reader has a little experience of Linux systems administration, including familiarity with the command line, file system, and text editing. It is expected that the reader has basic knowledge of Amazon Web Services and a little experience of Ansible usage.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Copy the external inventory script `ec2.py` and `ec2.ini` file to this project directory"

A block of code is set as follows:

```
---  
ansible_user: ec2-user  
ansible_port: 5300  
ansible_ssh_private_key_file: /home/yan/.ssh/keypair1.pem
```

Any command-line input or output is written as follows:

```
$ chmod 400 yan-key-pair-ap-sydney.pem
```

New terms and important words are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "Amazon **Virtual Private Cloud** (Amazon **VPC**) lets you provision a logically isolated section of the AWS Cloud where you can launch AWS resources in a virtual network that you define."

Warnings or important notes appear in a box like this.





Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of. To send us general feedback, simply e-mail feedback@packtpub.com, and mention the book's title in the subject of your message. If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for this book from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box.
5. Select the book for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this book from.
7. Click on **Code Download**.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the book is also hosted on GitHub at <https://github.com/yankurniawan/ansible-for-aws>. We also have other code bundles from our rich catalog of books and videos available at <https://github.com/PacktPublishing/>. Check them out!

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from http://www.packtpub.com/sites/default/files/downloads/AnsibleforAWS_ColorImages.pdf.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books-maybe a mistake in the text or the code-we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at questions@packtpub.com, and we will do our best to address the problem.

1

Getting Started with AWS

This chapter will give you introduction to Amazon Web Services (AWS), show you how to set up your AWS account, create your first EC2 instance, and connect to your EC2 instance using SSH. You can skip this chapter if you already have an AWS account and experience with EC2 instance.

What is Amazon Web Services?

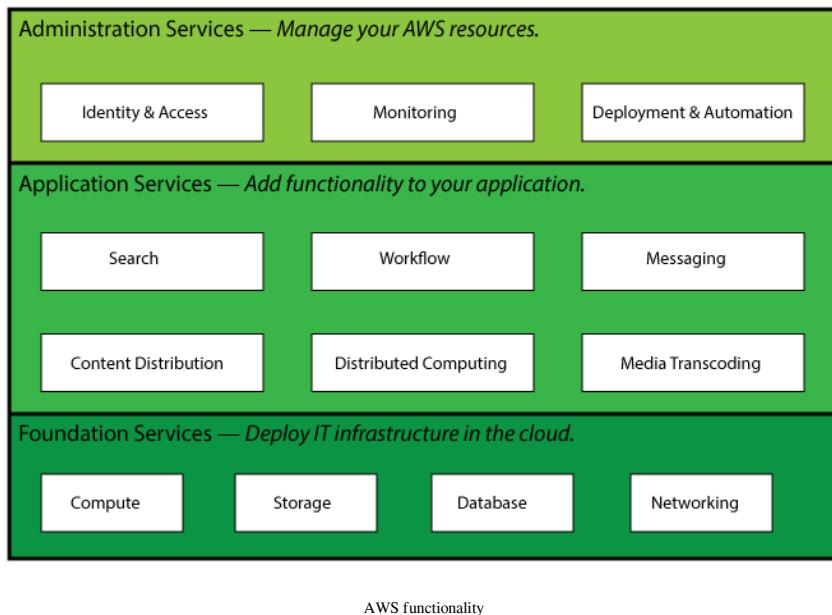
Amazon Web Services (AWS) provides computing resources and services that you can use to build applications within minutes at pay-as-you-go pricing. For example, you can rent a server on AWS that you can connect to, configure, secure, and run just as you would a physical server. The difference is the virtual server runs on top of a planet-scale network managed by AWS.

You pay for your virtual server only while it runs, with no up-front purchase costs or ongoing maintenance costs. Backed by the AWS network, your virtual server can do things no physical server can, such as automatically scaling into multiple servers when demand for your application increases.

Using AWS to build your Internet application is like purchasing electricity from a power company instead of running your own generator, and it provides many of the same benefits: capacity exactly matches your need, you pay only for what you use, economies of scale result in lower costs, and the service is provided by a vendor experienced in running large-scale networks. In addition, AWS can offer significant cost savings, up to 80%, compared to the equivalent on-premises deployments.

You can run nearly anything on AWS that you would run on physical hardware: websites, applications, databases, mobile apps, e-mail campaigns, distributed data analysis, media storage, and private networks. The services we provide are designed to work together so that you can build complete solutions. There are currently dozens of services, with more being added each year (for more information on AWS visit <http://docs.aws.amazon.com/gettingstarted/latest/awsgsg-intro/gsg-aws-intro.html>).

The following diagram shows the categories of functionality offered by AWS:



In each category, there are one or more services. For example, AWS offers five database services, each one optimized for a certain type of use. With so many offerings, you can design an AWS solution that is tailored to your needs.

Amazon has produced a set of short videos to help you understand AWS basics:

- What is Cloud Computing
(<https://www.youtube.com/watch?v=j0hbTAU4OPI&feature=youtu.be>)
- What is Amazon Web Services
(https://www.youtube.com/watch?v=mZ5H8sn_2ZI&feature=youtu.be)

In this book we will only use following AWS services from **Foundation Services** category:

- **EC2 (Elastic Compute Cloud)**
- **VPC (Virtual Private Cloud)**
- **RDS (Relational Database Service)**
- **S3 (Simple Storage Service)**
- **ELB (Elastic Load Balancing)**
- **Auto Scaling**
- **Route 53**

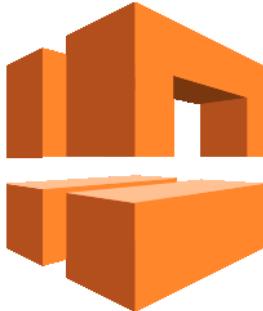
EC2 – Elastic Compute Cloud



Amazon **Elastic Compute Cloud** (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers.

Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change. Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you actually use. Amazon EC2 provides developers the tools to build failure resilient applications and isolate themselves from common failure scenarios (for more information on EC2 visit <https://aws.amazon.com/ec2/>).

VPC – Virtual Private Cloud



Amazon Virtual Private Cloud (Amazon VPC) lets you provision a logically isolated section of the AWS Cloud where you can launch AWS resources in a virtual network that you define. You have complete control over your virtual networking environment, including selection of your own IP address range, creation of subnets, and configuration of route tables and network gateways.

You can easily customize the network configuration for your Amazon VPC. For example, you can create a public-facing subnet for your webservers that has access to the Internet, and place your backend systems such as databases or application servers in a private-facing subnet with no Internet access. You can leverage multiple layers of security, including security groups and network access control lists, to help control access to Amazon EC2 instances in each subnet (for more information on VPC visit <https://aws.amazon.com/vpc/>).

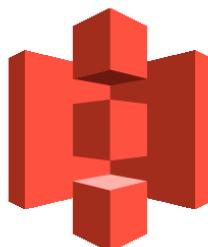
RDS – Relational Database Service



Amazon Relational Database Service (Amazon **RDS**) is a web service that makes it easy to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks, freeing you up to focus on your applications and business.

Amazon RDS gives you access to the capabilities of a familiar MySQL, Oracle, Microsoft SQL Server, or PostgreSQL database engine. This means that the code, applications, and tools you already use today with your existing databases can be used with Amazon RDS. Amazon RDS automatically patches the database software and backs up your database, storing the backups for a user-defined retention period and enabling point in time recovery. You benefit from the flexibility of being able to scale the compute resources or storage capacity associated with your Database Instance (DB Instance) via a single API call (for more information on RDS visit <https://aws.amazon.com/rds/>).

S3 – Simple Storage Service



Amazon S3 is storage for the Internet. It is designed to make web-scale computing easier for developers.

Amazon S3 provides a simple web-services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. It gives any developer access to the same highly scalable, reliable, secure, fast, inexpensive infrastructure that Amazon uses to run its own global network of web sites. The service aims to maximize benefits of scale and to pass those benefits on to developers.

Amazon S3 provides a highly durable and available store for a variety of content, ranging from web applications to media files. It allows you to offload your entire storage infrastructure onto the cloud, where you can take advantage of Amazon S3's scalability and pay as you go pricing to handle your growing storage needs. You can distribute your content directly from Amazon S3 or use Amazon S3 as an origin store for pushing content to your Amazon CloudFront edge locations (for more information on S3 visit <https://aws.amazon.com/s3/>).

ELB – Elastic Load Balancing



Elastic Load Balancing automatically distributes incoming application traffic across multiple Amazon EC2 instances.

It enables you to achieve greater levels of fault tolerance in your applications, seamlessly providing the required amount of load balancing capacity needed to distribute application traffic.

Elastic Load Balancing automatically scales its request handling capacity to meet the demands of application traffic. Additionally, Elastic Load Balancing offers integration with Auto Scaling to ensure that you have back-end capacity to meet varying levels of traffic levels without requiring manual intervention (for more information on Elastic Load Balancing visit <https://aws.amazon.com/elasticloadbalancing/>).

Route 53



Amazon Route 53 is a highly available and scalable **Domain Name System (DNS)** web service. It is designed to give developers and businesses an extremely reliable and cost effective way to route end users to Internet applications by translating names like `www.example.com` into the numeric IP addresses like `192.0.2.1` that computers use to connect to each other.

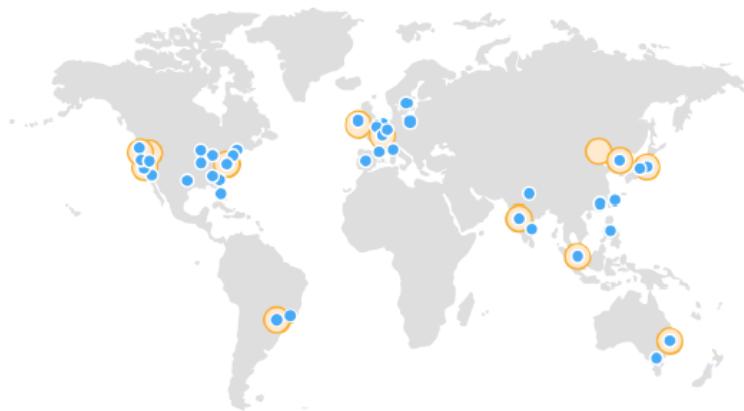
Route 53 effectively connects user requests to infrastructure running in AWS—such as Amazon EC2 instances, Elastic Load Balancers, or Amazon S3 buckets—and can also be used to route users to infrastructure outside of AWS.

Route 53 is designed to be fast, easy to use, and cost-effective. It answers DNS queries with low latency by using a global network of DNS servers. Queries for your domain are automatically routed to the nearest DNS server, and thus answered with the best possible performance. With Route 53, you can create and manage your public DNS records with the AWS Management Console or with an easy-to-use API. It's also integrated with other Amazon Web Services. For instance, by using the AWS **Identity and Access Management (IAM)** service with Route 53, you can control who in your organization can make changes to your DNS records. Like other Amazon Web Services, there are no long-term contracts or minimum usage requirements for using Route 53—you pay only for managing domains through the service and the number of queries that the service answers (for more information on Route53 visit <https://aws.amazon.com/route53/>).



AWS Global Infrastructure

Whether you are a large global company or small start-up, you may have potential customers around the world. With traditional infrastructure, it's hard to deliver great performance to a broadly distributed user base and most companies focus on one geographic region at a time to save costs and time. With Cloud Computing, the game changes—you can easily deploy your application in any or all of the AWS regions around the world. This means you can provide a lower latency and better experience for your customers at minimal cost (for more information on Global Reach visit <https://aws.amazon.com/what-is-cloud-computing/#global-reach>).

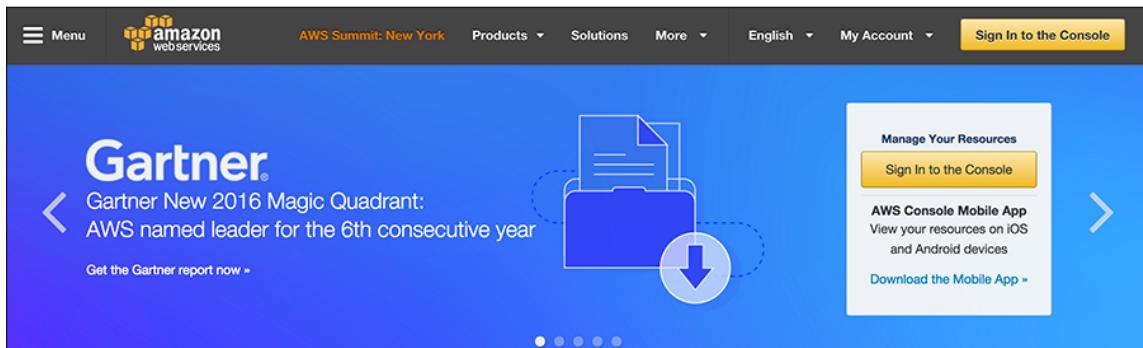


For more information on Regional Product Services visit <https://aws.amazon.com/about-aws/global-infrastructure/regional-product-services/>.

Setting up your AWS account

If you don't already have an Amazon Web Services account, open your web browser on your computer and go to <http://aws.amazon.com>. Follow the steps:

1. Click the **Sign In to the Console** button.



2. On the next screen, select the **I am a new user** radio button, fill in your e-mail address in the given field, and then click the **Sign in using our secure server** button. Technically, if you have Amazon retail account, you can sign in using your Amazon.com account, but it is recommended that you set up a new AWS account.

Sign In or Create an AWS Account

What is your email (phone for mobile accounts)?

E-mail or mobile number:

I am a new user.

I am a returning user
and my password is:

Sign in using our secure server

[Forgot your password?](#)

3. On the next page, enter your name, type your e-mail address again, and enter your password (twice), then click **Create account** button.

Login Credentials

Use the form below to create login credentials that can be used for AWS as well as Amazon.com.

My name is: Yan Kurniawan

My e-mail address is: [redacted]

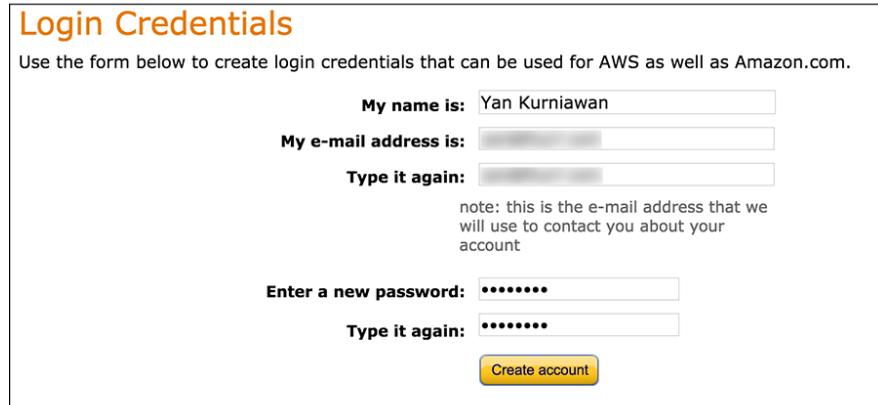
Type it again: [redacted]

note: this is the e-mail address that we will use to contact you about your account

Enter a new password: [redacted]

Type it again: [redacted]

Create account



4. On the next screen, select **Personal Account** (or **Company Account**) if you want to create an AWS account for your company), enter the required information on the **Contact Information** form, type the **Security Check** characters, confirm your acceptance of the AWS customer agreement, and then click the **Securely Submit** button.

— Contact Information —

Company Account Personal Account

* Required Fields

Full Name* Yan Kurniawan

Country* Australia

Address*
Apartment, suite, unit, building, floor, etc.

City*

State / Province or Region*

Postal Code*

Phone Number*

Security Check 

[Refresh Image](#)

Please type the characters as shown above

AWS Customer Agreement

Check here to indicate that you have read and agree to the terms
of the [AWS Customer Agreement](#)

Securely Submit

5. The next page asks you for a credit card number and your billing address information. Enter the required information and click **Continue** button.

Payment Information

Please enter your payment information below. You will be able to try a broad set of AWS products for free via the Free Tier. We will only bill your credit or debit card for usage that is not covered by our Free Tier.

[Frequently Asked Questions](#)

Credit/Debit Card Number	Expiration Date
<input type="text" value="XXXX-XXXX-XXXX-XXXX"/>	<input type="button" value="MM"/> <input type="button" value="YY"/>
Cardholder's Name	
<input type="text" value="Yan Kumlawan"/>	
<input checked="" type="radio"/> Use my contact address <input type="radio"/> Use a new address	
<input type="button" value="Continue"/>	

6. On the next page, Amazon wants to confirm your identity. Enter your valid phone or mobile number and click the **Call Me Now** button.

Identity Verification

You will be called immediately by an automated system and prompted to enter the PIN number provided.

1. Provide a telephone number
Please enter your information below and click the "Call Me Now" button.

Country Code	Phone Number	Ext
Australia (+61)	<input type="text"/>	<input type="text"/>

Call Me Now

2. Call in progress

3. Identity verification complete

Answer the phone and enter the displayed PIN code on the telephone keypad, or you can say the PIN numbers.

Identity Verification

You will be called immediately by an automated system and prompted to enter the PIN number provided.

1. Provide a telephone number ✓

2. Call in progress

Please follow the instructions on the telephone and key in the following Personal Identification Number (PIN) on your telephone when prompted.

PIN: 6607

If you have not yet received a call at the number indicated above please wait. This page will automatically update with what you need to do next.

3. Identity verification complete

After the identity verification completed successfully, click the **Continue to select your Support Plan** button.

Identity Verification

You will be called immediately by an automated system and prompted to enter the PIN number provided.

1. Provide a telephone number ✓

2. Call in progress ✓

3. Identity verification complete
Your identity has been verified successfully

[Continue to select your Support Plan](#)

7. On the next page, choose your support plan and click the **Continue** button.

Support Plan

AWS Support offers a selection of plans to meet your needs. All plans provide 24x7 access to customer service, AWS documentation, whitepapers, and support forums. For access to technical support and additional resources to help you plan, deploy, and optimize your AWS environment, we recommend selecting a support plan that best aligns with your AWS usage.

Please Select One

Basic
Description: Customer Service for account and billing questions and access to the AWS Community Forums.
Price: Included

Developer
Use case: Experimenting with AWS
Description: One primary contact may ask technical questions through Support Center and get a response within 12–24 hours during local business hours.
Price: Starts at \$29/month (scales based on usage)

Business
Use case: Production use of AWS
Description: 24x7 support by phone and chat, 1-hour response to urgent support cases, and help with common third-party software. Full access to AWS Trusted Advisor for optimizing your AWS infrastructure, and access to the AWS Support API for automating your support cases and retrieving Trusted Advisor results.
Price: Starts at \$100/month (scales based on usage)

Enterprise
Use case: Mission-critical use of AWS
Description: All the features of the Business support plan, plus an assigned Technical Account Manager (TAM) who provides proactive guidance and best practices to help plan, develop, and run your AWS solutions, a Support Concierge who provides billing and account analysis and assistance, access to Infrastructure Event Management to support product launches, seasonal promotions/events, and migrations, and 15-minute response to critical support cases with prioritized case handling.
Price: Starts at \$15,000/month (scales based on usage)
If you select this option, customer support will contact you within 48 hours to discuss your needs and finalize the signup. Support resources will be available when signup is finalized, and no charges will be incurred until that time.

To explore all features and benefits of AWS Support, including plan comparisons and pricing samples, [click here](#).

Continue

8. Setup is now complete, you'll get an e-mail confirming your account setup.



You have given AWS your credit card information to pay for the resources you use. Be careful about how much AWS resource you use and try to understand the pricing scheme for each service.

EC2 Pricing Scheme (<https://aws.amazon.com/ec2/pricing/>)

S3 Pricing Scheme (<https://aws.amazon.com/s3/pricing/>)



Your initial account sign-up provides free usage tier for a year. For a complete list of services that you can use for free, check out AWS Free Usage Tier page (for more information visit <https://aws.amazon.com/free/>).



Amazon provides an online calculator (<http://calculator.s3.amazonaws.com/index.html>) to estimate your monthly AWS bill.

AWS management console

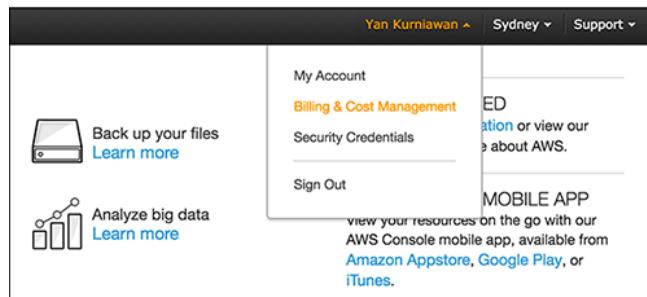
Amazon provides a web-based AWS Management Console. You can access and manage Amazon Web Services resources through a simple and intuitive web-based user interface. The AWS Management Console is a single destination for managing all your AWS resources, from EC2 instances to DynamoDB tables. Use the console to perform any number of tasks, from deploying new applications to monitoring the health of your application.

The AWS Management Console also enables you to manage all aspects of your AWS account, including accessing your monthly spending by service, managing security credentials, or even setting up new IAM Users.

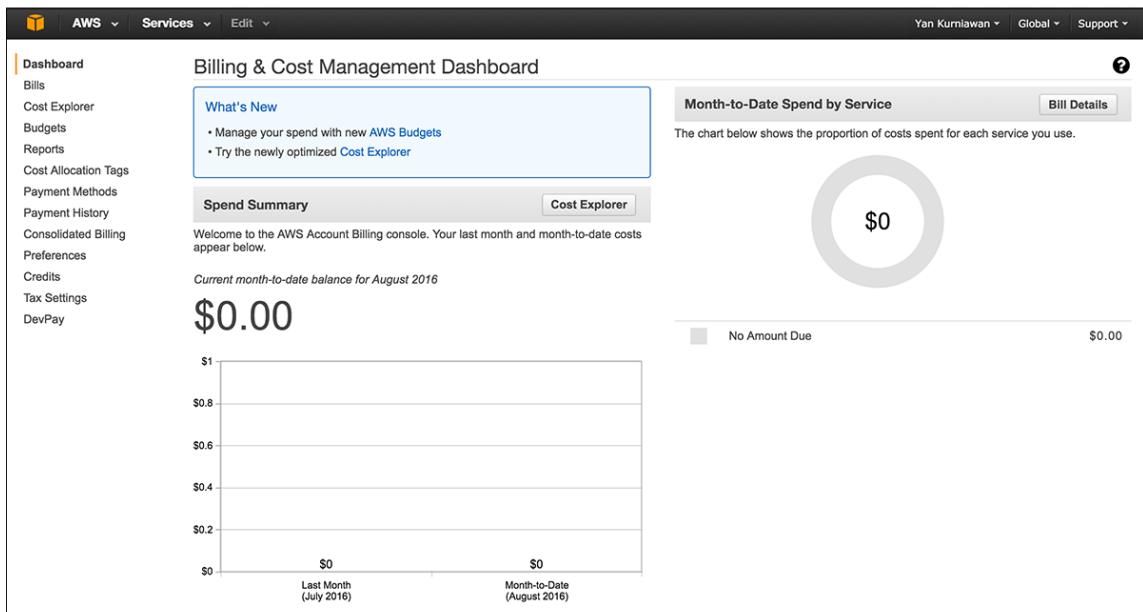
The screenshot shows the AWS Management Console homepage. At the top, there's a navigation bar with the AWS logo, a services dropdown, and user information (Yan Kurniawan, Select a Region, Support). Below the navigation is a "Quick Starts" section with six cards: "Build a web app" (Start now), "Launch a Virtual Machine (EC2 Instance)", "Back up your files", "Build a back end for your mobile app" (Start now), "Host a static website", and "Analyze big data". Underneath is a "Shortcuts and Recently Viewed Services" section with icons for EC2, ElastiCache, Elasticsearch Service, CloudWatch, and Route 53. The main area is titled "AWS Services" with a "SHOW CATEGORIES" link. It's organized into several sections: COMPUTE (EC2, EC2 Container Service, Elastic Beanstalk, Lambda), DEVELOPER TOOLS (CodeCommit, CodeDeploy, CodePipeline), MANAGEMENT TOOLS (CloudWatch, CloudFormation, CloudTrail, Config, OpsWorks, Service Catalog), INTERNET OF THINGS (AWS IoT), GAME DEVELOPMENT (GameLift), and MOBILE SERVICES (Mobile Hub, Cognito, Device Farm, Mobile Analytics, SNS). On the right side, there's a "GETTING STARTED" section with links to documentation and training, an "AWS CONSOLE MOBILE APP" section with links to the Appstore, Google Play, and iTunes, an "AWS MARKETPLACE" section with a link to find and buy software, a "FEEDBACK" section with a link to tell AWS what you think, and a "Service Health" section showing that all services are operating normally as of August 10, 2016, at 11:20:01 GMT+1000.

The console home

To see your monthly billing, you can choose **Billing & Cost Management** from the pull down menu under your account name on top right of the page.



Billing and cost management



The billing dashboard

You can customize one-click navigation on the menu bar. Click on the **Edit** pull down menu and drag your selected service to/from the menu bar.

The screenshot shows the AWS Management Console interface. At the top, there's a dark header bar with the AWS logo, a "Services" dropdown, and several service icons: S3, EC2, and Edit. Below the header, a message says: "To customize one-click navigation shortcuts simply drag your services to and from the menu bar above." The main area contains three columns of service icons and names, each enclosed in a white box:

API Gateway	DynamoDB	OpsWorks
AppStream	EC2	RDS
AWS IoT	EC2 Container Service	Redshift
Certificate Manager	Elastic Beanstalk	Route 53
CloudFormation	Elastic File System	S3
CloudFront	Elastic Transcoder	Service Catalog
CloudSearch	ElastiCache	SES
CloudTrail	Elasticsearch Service	Snowball
CloudWatch	EMR	SNS
CodeCommit	GameLift	SQS
CodeDeploy	Glacier	Storage Gateway
CodePipeline	IAM	SWF
Cognito	Inspector	Trusted Advisor
Config	Kinesis	VPC
Data Pipeline	Lambda	WAF
Device Farm	Machine Learning	WorkDocs
Direct Connect	Mobile Analytics	WorkMail
Directory Service	Mobile Hub	WorkSpaces
DMS		

Customize one-click navigation

Each service has its own console, which you can access from the AWS Management Console.

Navigation bar

Navigation pane

Current page

Region selector

EC2 console



For a complete guide to AWS Management Console, visit AWS Management Console Getting Started Guide page at <http://docs.aws.amazon.com/awsconsolehelpdocs/latest/gsg/getting-started.html>.

Create your first EC2 instance

EC2—the Elastic Compute Cloud, is the most widely used AWS service. EC2 is the most revolutionary of the AWS services because it has transformed the way of provisioning servers. EC2 provides virtual servers in a matter of minutes with a few clicks on the AWS Management Console.

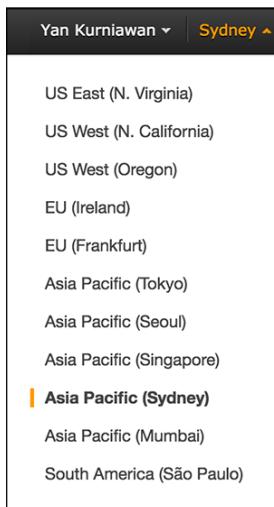
Let's create our first EC2 instance.

Create a key pair

First, we need to create a key pair. AWS uses public key cryptography to secure the login information for your instance. A Linux instance has no password; you use a key pair to log in to your instance securely. You specify the name of the key pair when you launch your instance, then provide the private key when you log in. Key pairs are specific to a region; for example, if you plan to launch an instance in the Asia Pacific (Sydney) Region, you must create a key pair for the instance in the Asia Pacific (Sydney) Region.

To create a key pair:

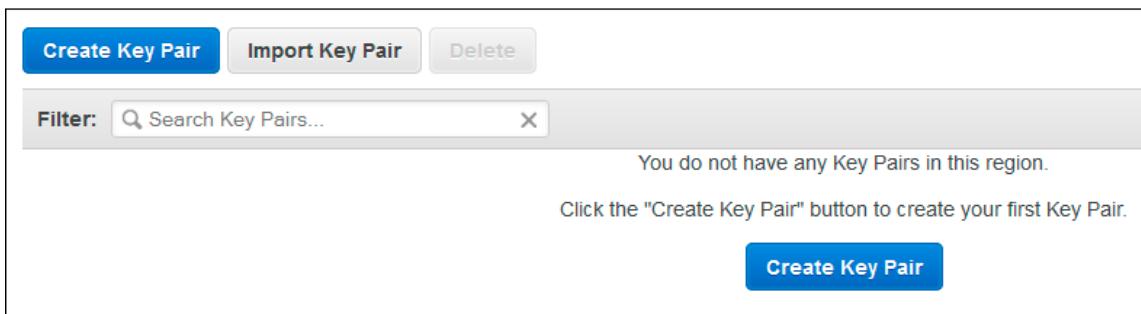
1. Open the Amazon EC2 Dashboard <https://console.aws.amazon.com/ec2>.
2. From the navigation bar, select a region for the key pair.



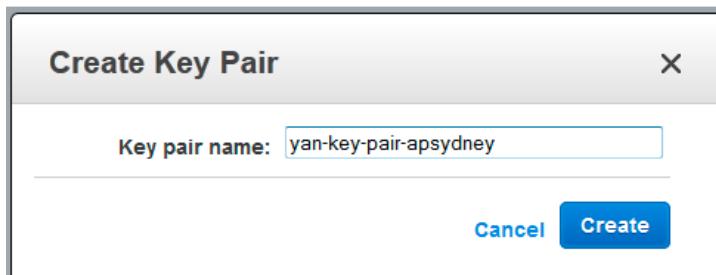
3. Click **Key Pairs** in the navigation pane.



4. Click **Create Key Pair**.



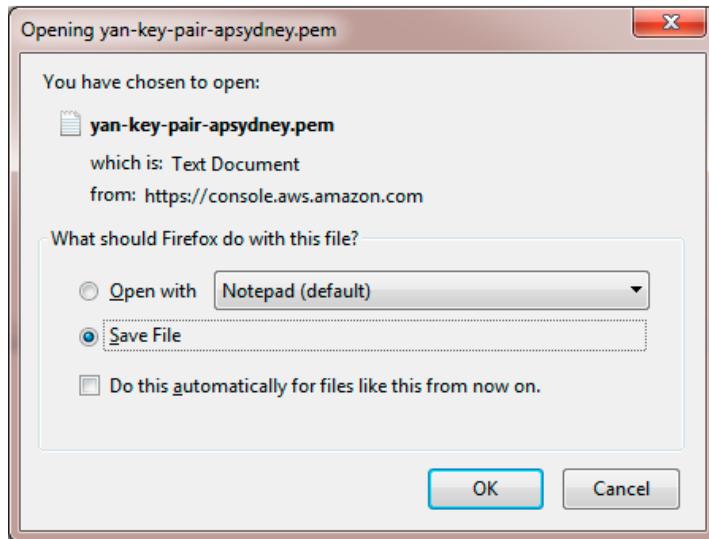
5. Enter a name for the new key pair in the **Key pair name** field of the **Create Key Pair** dialog box, and then click **Create**. Choose a name that is easy for you to remember, such as your name, followed by `-key-pair`, plus the region name. For example, `yan-key-pair-apsydney`.



6. If you use Google Chrome as your browser, the private key file is automatically downloaded by your browser. The base file name is the name you specified as the name of your key pair, and the file name extension is `.pem`. If you use Firefox, the browser might ask you to open/save the file. Save the private key file in a safe place.



This is the only chance for you to save the private key file. You'll need to provide the name of your key pair when you launch an instance and the corresponding private key each time you connect to the instance.



7. If you will use an SSH client on a Mac or Linux computer to connect to your Linux instance, use the `chmod` command to set the permissions of your private key file so that only you can read it.

```
$ chmod 400 yan-key-pair-ap Sydney.pem
```

If you'll connect to your Linux instance from a computer running Windows, you can use **PuTTY** or **MindTerm**. If you use PuTTY, you'll need to install it and use the following procedure to convert the `.pem` file to a `.ppk` file.

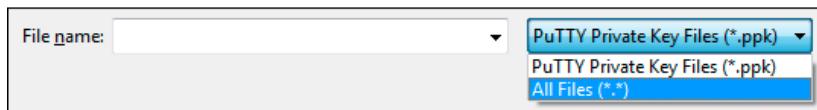
To prepare to connect to a Linux instance from Windows using PuTTY:

1. Download and install PuTTY from <http://www.chiark.greenend.org.uk/~sgtatham/putty>. Be sure to install the entire suite (Download the installer file under *A Windows installer for everything except PuTTYtel* section).
2. Start **PuTTYgen** (for example, from the **Start** menu, click **All Programs | PuTTY | PuTTYgen**).

3. Under **Type of key to generate**, select **SSH-2 RSA**.



4. Click **Load**. By default, PuTTYgen displays only files with the extension .ppk. To locate your .pem file, select the option to display files of all types.



5. Select the private key file that you created in the previous procedure and click **Open**. Click on **OK** to dismiss the confirmation dialog box.



6. Click **Save private key**. PuTTYgen displays a warning about saving the key without a passphrase. Click on **Yes**.
7. Specify the same name for the key that you used for the key pair. PuTTY automatically adds the .ppk file extension.

Create a security group

Security groups act as a firewall for associated instances, controlling both inbound and outbound traffic at the instance level. You must add rules to a security group that enable you to connect to your instance from your IP address using SSH. You can also add rules that allow inbound and outbound HTTP and HTTPS access from anywhere.

Note that if you plan to launch instances in multiple regions, you'll need to create a security group in each region.



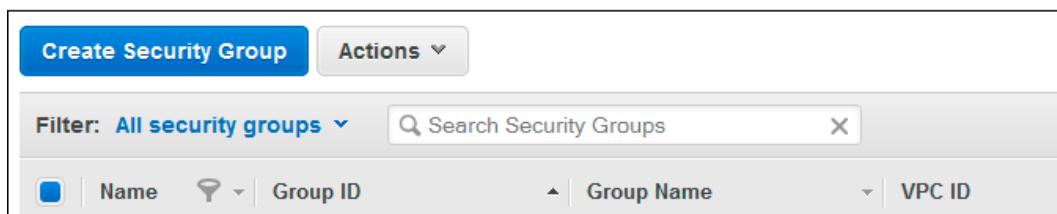
You'll need the public IP address of your local computer, which you can get using a service from Amazon AWS <http://checkip.amazonaws.com>. If you are connecting through an Internet service provider (ISP) or from behind a firewall without a static IP address, you need to find out the range of IP addresses used by client computers.

To create a security group:

1. Open the Amazon EC2 Dashboard <https://console.aws.amazon.com/ec2>.
2. From the navigation bar, select a region for the security group. Security groups are specific to a region; for example, if you plan to launch an instance in the Asia Pacific (Sydney) Region, you must create a security group for the instance in the Asia Pacific (Sydney) Region.
3. Click **Security Groups** in the navigation pane.



4. Click **Create Security Group**.



5. Enter a name for the new security group and a description. Choose a name that is easy for you to remember, such as SG_ plus the region name. For example, SG_apsydney. On the **Inbound** tab, create the following rules (click **Add Rule** for each new rule), and click **Create** when you're done:
- Select **HTTP** from the **Type** list, and make sure that **Source** is set to **Anywhere (0.0.0.0/0)**.
 - Select **HTTPS** from the **Type** list, and make sure that **Source** is set to **Anywhere (0.0.0.0/0)**.
 - Select **SSH** from the **Type** list. In the **Source** box, ensure **Custom IP** is selected, and specify the public IP address of your computer or network in CIDR notation. To specify an individual IP address in CIDR notation, add the routing prefix /32. For example, if your IP address is 203.0.100.2, specify 203.0.100.2/32. If your company allocates addresses from a range, specify the entire range, such as 203.0.100.0/24. For security reasons, Amazon doesn't recommend that you allow SSH access from all IP addresses (0.0.0.0/0) to your instance, except for testing purposes and only for a short time.

Create Security Group

Security group name	SG_apsydney
Description	base security group - sydney
VPC	vpc- [172.31.0.0/16] *

* denotes default VPC

Security group rules:

Inbound **Outbound**

Type	Protocol	Port Range	Source
HTTP	TCP	80	Anywhere 0.0.0.0/0
HTTPS	TCP	443	Anywhere 0.0.0.0/0
SSH	TCP	22	Custom IP 123.123.123.123/32

Add Rule

Create

Create security group



You can select **My IP** in the **Source** box to allow traffic from your IP address.

The following procedure is intended to help you launch your first instance quickly and doesn't go through all possible options. For more information about the advanced options see AWS Documentation on Launching an Instance documentation at <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/launching-instance.html>.

To launch an instance:

1. Open the Amazon EC2 Dashboard <https://console.aws.amazon.com/ec2>.
2. From the console dashboard, click **Launch Instance**.

Create Instance

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Launch Instance

Note: Your instances will launch in the Asia Pacific (Sydney) region

3. The **Choose an Amazon Machine Image (AMI)** page displays a list of basic configurations called **Amazon Machine Images (AMI)** that serve as templates for your instance. Select the 64-bit Amazon Linux AMI. Notice that this configuration is marked **Free tier eligible**. Click on the **Select** button.

Step 1: Choose an Amazon Machine Image (AMI)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Quick Start

- My AMIs
- AWS Marketplace **Amazon Linux** Free tier eligible
- Community AMIs
- Free tier only (i)

Amazon Linux AMI 2016.03.3 (HVM), SSD Volume Type - ami-dc361ebf

The Amazon Linux AMI is an EBS-backed, AWS-supported image. The default image includes AWS command line tools, Python, Ruby, Perl, and Java. The repositories include Docker, PHP, MySQL, PostgreSQL, and other packages.

Root device type: ebs Virtualization type: hvm

Select

64-bit

- On the **Choose an Instance Type** page, you can select the hardware configuration of your instance. The **t2.micro** (or **t1.micro**, depends on the AMI virtualization type) instance is selected by default. Click **Review and Launch** to let the wizard complete other configuration settings for you, so you can get started quickly.

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. [Learn more](#) about instance types and how they can meet your computing needs.

Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)						
	Family	Type	vCPUs <small>(i)</small>	Memory (GiB)	Instance Storage (GB) <small>(i)</small>	EBS-Optimized Available <small>(i)</small>
<input type="checkbox"/>	General purpose	t2.nano	1	0.5	EBS only	-
<input checked="" type="checkbox"/>	General purpose	t2.micro <small>Free tier eligible</small>	1	1	EBS only	-
<input type="checkbox"/>	General purpose	t2.small	1	2	EBS only	-

Cancel **Previous** **Review and Launch** **Next: Configure Instance Details**

5. On the **Review Instance Launch** page, you can review the settings for your instance. Under **Security Groups**, you'll see that the wizard created and selected a security group for you. Instead, select the security group that you created when getting set up using the following steps:
- Click on **Edit** security groups.
 - On the **Configure Security Group** page, ensure the **Select an existing security group** option is selected.
 - Select your security group from the list of existing security groups, and click **Review and Launch**.

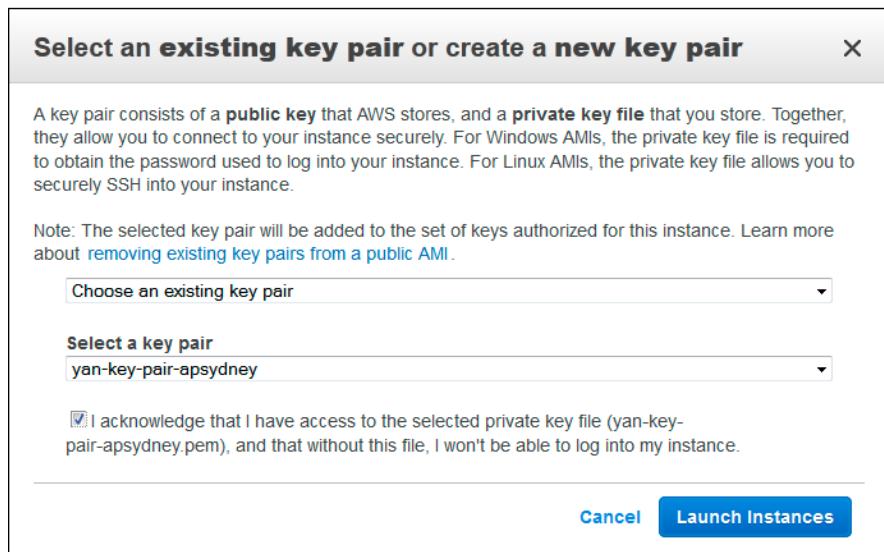
The screenshot shows the 'Assign a security group' section of the AWS Configure Security Group dialog. It has two radio button options: 'Create a new security group' (unchecked) and 'Select an existing security group' (checked). Below this is a table showing existing security groups. A row for 'SG_apsydney' is selected, indicated by a checked checkbox in the first column. The table columns are: Security Group ID, Name, Description, and Actions (with 'Copy to new' links). Below the table is another table showing port mappings: Type (SSH, HTTP, HTTPS), Protocol (TCP), Port Range (22, 80, 443), and Source (/32). At the bottom are 'Cancel', 'Previous', and a large blue 'Review and Launch' button.

Security Group ID	Name	Description	Actions
sg-[REDACTED]	default	default VPC security group	Copy to new
<input checked="" type="checkbox"/> sg-[REDACTED]	SG_apsydney	base security group - sydney	Copy to new

Type	Protocol	Port Range	Source
SSH	TCP	22	123. /32
HTTP	TCP	80	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0

[Cancel](#) [Previous](#) [Review and Launch](#)

6. On the **Review Instance Launch** page, click on **Launch**.
7. In the **Select an existing key pair or create a new key pair** dialog box, select **Choose an existing key pair**, then select the key pair you created when getting set up. Select the acknowledgment check box, and then click on **Launch Instances**.



8. A confirmation page lets you know that your instance is launching. Click on **View Instances** to close the confirmation page and return to the console.
9. On the **Instances** screen, you can view the status of your instance. It takes a short time for an instance to launch. When you launch an instance, its initial state is pending. After the instance starts, its state changes to running, and it receives a public DNS name. (If the **Public DNS** column is hidden, click on the **Show/Hide** icon and select **Public DNS**). View Instances.

Filter by tags and attributes or search by keyword								
Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS	
	i- [REDACTED]	t2.micro	ap-southeast-2c	pending		Initializing	None	

The screenshot shows the AWS Management Console interface for the EC2 service. At the top, there's a search bar and navigation links. Below it, a table lists instances. One instance is selected, showing its details in a modal window.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS
i-	i-	t2.micro	ap-southeast-2c	running	2/2 checks ...	None	ec2-

Below the table, the instance details are shown:

Description		Status Checks	Monitoring	Tags
Instance ID	i-	Public DNS	.ap-southeast-2.compute.amazonaws.com	
Instance state	running	Public IP		
Instance type	t2.micro	Elastic IPs		
Private DNS	.ap-southeast-2.compute.internal	Availability zone	ap-southeast-2c	
Private IPs		Security groups	SG_apsydney . view rules	
Secondary private IPs		Scheduled events	No scheduled events	
VPC ID	vpc-	AMI ID	amzn-ami-hvm-2016.03.3.x86_64-gp2 (ami-dc361ebf)	
Subnet ID	subnet-	Platform	-	
Network interfaces	eth0	IAM role	-	



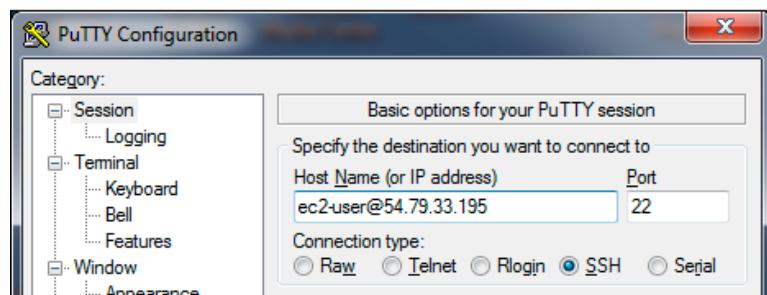
To learn more about Amazon EC2 instance type, see <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html>.

Connect to your instance

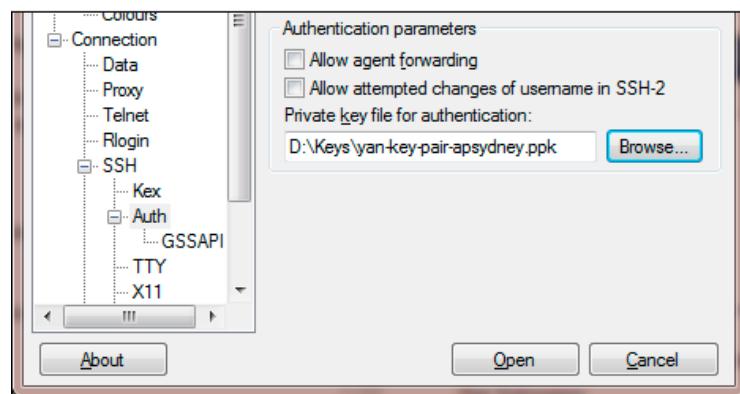
If your computer uses Windows operating system, you will need to install PuTTY to connect to your Linux EC2 instance.

To connect to your Linux instance using PuTTY:

1. Start PuTTY (from the **Start** menu, click **All Programs** | **PuTTY** | **PuTTY**).
2. In the Category pane, select **Session** and complete the following fields:
 - In the **Host Name (or IP address)** box, enter `ec2-user@public_ip`. You can use either **Public IP** address or **Public DNS** name of your instance.
 - Under **Connection type**, select **SSH**.
 - Ensure that **Port** is **22**.



3. In the **Category** pane, expand **Connection**, expand **SSH**, and then select **Auth**. Complete the following:
- Click on **Browse**
 - Select the **.ppk** file that you generated for your key pair, and then click on **Open**
 - Click on **Open** to start the PuTTY session.



4. If this is the first time you have connected to this instance, PuTTY displays a security alert dialog box that asks whether you trust the host you are connecting to. Click on **Yes**. A window opens and you are connected to your instance.

Connect from Mac or Linux using an SSH client

If you are using Mac or Linux computer to connect to your instance, your computer most likely includes an SSH client by default. You can check for an SSH client by typing `ssh` at the command line. If your computer doesn't recognize the command, the OpenSSH project provides a free implementation of the full suite of SSH tools. For more information, see <http://www.openssh.org>.

Open your command shell and run the following command:

```
$ ssh -i /path/key_pair.pem ec2-user@public_ip
```



For Amazon Linux, the default user name is `ec2-user`. For RHEL5, the user name is often `root` but might be `ec2-user`. For Ubuntu, the user name is `ubuntu`. For SUSE Linux, the user name is `root`. Otherwise, check with your AMI provider.

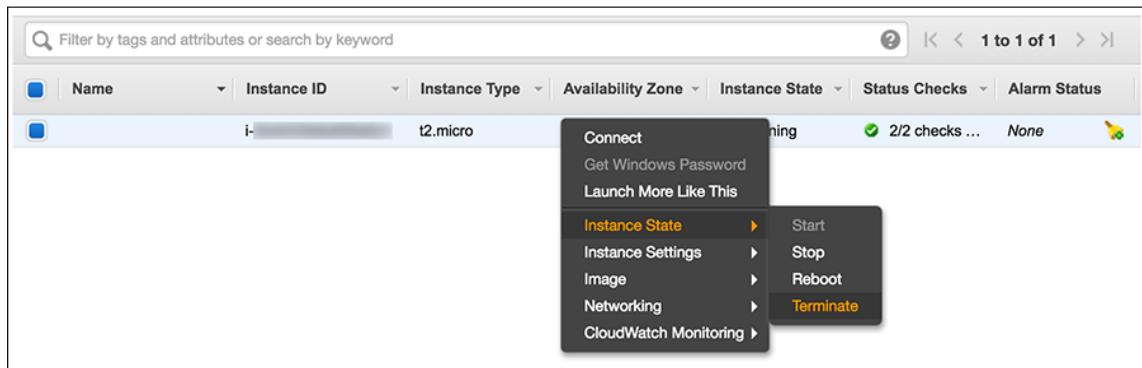
Terminate your instance

The purpose of the tutorial in this chapter is to show you how to launch EC2 instance from AWS Management Console, so you can get basic understanding of EC2, key pair, and security groups. After you've finished with the instance that you created for this chapter, you should clean up, terminate the instance.

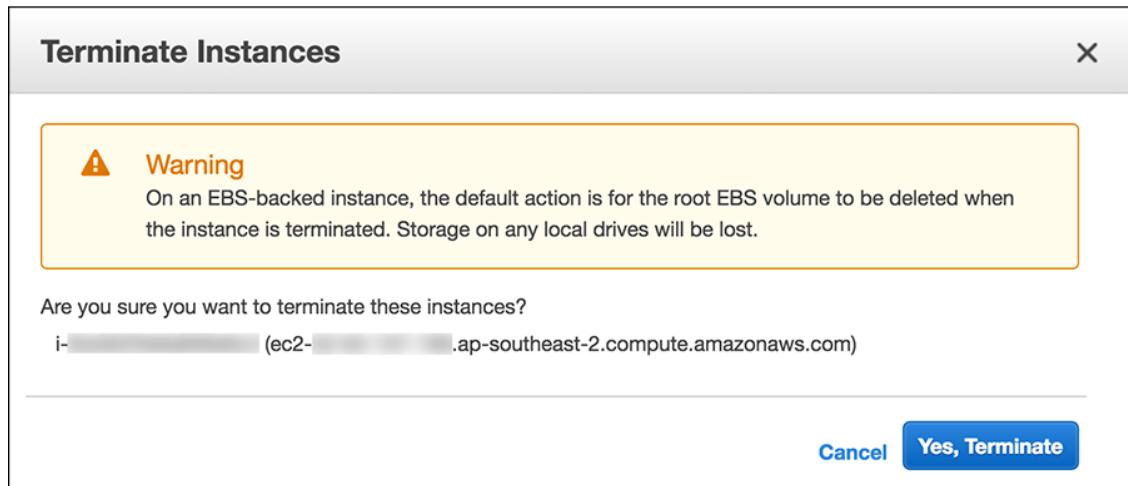
Terminating an instance effectively deletes it because you can't reconnect to an instance after you've terminated it. This differs from stopping the instance; when you stop an instance, it is shut down and you are not billed for hourly usage or data transfer. Also, you can restart a stopped instance at any time.

To terminate the instance:

1. Locate your instance in the list of instances on the **Instances** page. If you can't find your instance, verify that you have selected the correct region.



2. Right-click the instance, select **Instance State** and then click **Terminate**.



3. Click **Yes, Terminate** when prompted for confirmation.



Most parts of this chapter is based on Amazon AWS Online Documentation. This chapter only covers the basics of AWS and EC2. If you want to learn more about EC2, see Amazon EC2 User Guide (<http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>). For a complete list of Amazon AWS Documentation, visit AWS Documentation (<https://aws.amazon.com/documentation/>). You can also watch Introduction to AWS videos here:
https://aws.amazon.com/training/intro_series/.

2

Getting Started with Ansible

Ansible is a radically simple IT orchestration engine that automates configuration management, application deployment, and many other IT needs. Ansible models your IT infrastructure by looking at the comprehensive architecture of how all of your systems interrelate, rather than just managing one system at a time. It uses no agents and no additional custom security infrastructure, so it's easy to deploy – and most importantly, it uses a very simple language (YAML, in the form of Ansible playbooks) that allows you to describe your automation jobs in a way that approaches plain English.

Ansible works by connecting to your nodes and pushing out small programs, called **Ansible Modules** to them. These programs are written to be resource models of the desired state of the system. Ansible then executes these modules (over SSH by default), and removes them when finished. Your library of modules can reside on any machine, and there are no servers, daemons, or databases required. For more information, visit <http://www.ansible.com/how-ansible-works>



YAML is a recursive acronym for *YAML Ain't Markup Language*. YAML is a human-friendly data serialization standard for all programming languages. To learn more about YAML, visit <http://www.yaml.org>.

Unlike Chef and Puppet, Ansible uses an agentless architecture. You only need to install Ansible on the machines that you use to manage your infrastructure. Managed nodes are not required to install and run background daemons to connect with a controlling machine and pull configuration, thus reduces the overhead on the network.

In this chapter you'll learn how to install Ansible, build an inventory file, use Ansible from the command line, write a simple playbook, and use Ansible modules.

What you'll need

To follow the examples in this book, you'll need a computer, preferably running Linux, connected to the Internet. You'll also need to be able to run commands in a terminal and do simple editing of text files.

Throughout this book I will use the CentOS 6.5 (minimal install) distribution of Linux to run Ansible. Ansible runs on a number of different platforms, but I'm not going to provide detailed instructions for all of them. All EC2 instances provisioned by Ansible in this book's examples will use the Amazon Linux **Amazon Machine Images (AMI)** which is based on Red Hat distribution, similar to CentOS.

Installing Ansible

Ansible is written in Python. To install the latest version of Ansible, we will use **pip**. Pip is a tool used to manage packages of Python software and libraries. Ansible releases are pushed to pip as soon as they are released. To install Ansible via pip follow the following steps:

1. Install RHEL EPEL repository. The **Extra Packages for Enterprise Linux (EPEL)** repository is a package repository for **Red Hat Enterprise Linux (RHEL)** or CentOS, maintained by people from Fedora Project community, to provide add-on packages from Fedora, which are not included in the commercially supported Red Hat product line.

```
$ sudo yum -y update  
$ sudo yum -y install wget  
$ wget http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release  
6-8.noarch.rpm  
$ sudo rpm -Uvh epel-release-6*.rpm
```

2. Install **Development tools** group. The Development tools are a yum group, which is a predefined bundle of software that can be installed at once, instead of having to install each application separately. The Development tools will allow you to build and compile software from source code. Tools for building RPMs are also included, as well as source code management tools like Git, SVN, and CVS.

```
$ sudo yum groupinstall -y 'development tools'
```

3. Install `python-pip` and `python-devel`

```
$ sudo yum -y install python-pip python-devel
```

4. Upgrade setuptools

```
$ sudo pip install setuptools --upgrade
```

5. Install Ansible via pip

```
$ sudo pip install ansible
```

After the installation has completed successfully, you will be able to run this command to show your Ansible's version number:

```
$ ansible --version
ansible 2.1.1.0
```

To upgrade Ansible to the latest version available in pip repository:

```
$ sudo pip install ansible --upgrade
```

SSH keys

Ansible communicates with remote machines over SSH. By default, Ansible 1.3 and later will try to use native OpenSSH for remote communication when possible. It is recommended that you use SSH keys for SSH authentication, so Ansible won't have to ask password to communicate with remote hosts. To enable SSH keys authentication use the instructions which are shown as follows:

1. Create public and private keys using ssh-keygen

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/yan/.ssh/id_rsa): [press ENTER
key]
Enter passphrase (empty for no passphrase) : [press ENTER key]
Enter same passphrase again: [press ENTER key]
Your identification has been saved in /home/yan/.ssh/id_rsa.
Your public key has been saved in /home/yan/.ssh/id_rsa.pub.
The key fingerprint is:
```

2. Copy the public key to remote host using ssh-copy-id (For this example, I will use localhost as the remote host)

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub localhost yan@localhost's
password: [enter PASSWORD]
Now try logging into the machine, with "ssh 'localhost'", and check in:
.ssh/authorized_keys to make sure we haven't added extra keys that you
```

weren't expecting.

3. ssh-copy-id appends the keys to the remote host's .ssh/authorized_key
4. Login to remote host without entering the password

```
$ ssh localhost
Last login: Sun Jul 31 10:17:35 2016 from ::1
$ exit
logout
Connection to localhost closed.
```

Inventory

Ansible works against multiple nodes in your infrastructure at the same time, by selecting portions of nodes listed in Ansible's inventory file. By default, Ansible uses /etc/ansible/hosts for the inventory file. You can have another path for hosts file and specify the path using -i option when running ansible or ansible-playbook command. The format for the host inventory file is an INI format and looks like this:

```
one.example.com

[webservers] web1.example.com web2.example.com two.example.com
[dbservers] db1.example.com db2.example.com two.example.com
```

The words in brackets are group names, which are used in classifying nodes and deciding what hosts you are controlling in an Ansible task. One node can be a member of more than one group, like two.example.com in the example we just saw. To add a lot of hosts with similar patterns you can specify range like this:

```
[webservers]
web[01:50].example.com
[databases]db-[a:f].example.com
```

It is also possible to make groups of groups:

```
[sydney]
host1
host2
[singapore]
host3
host4

[asiapacific:children]
sydney
singapore
```

Host variables

You can specify variables in hosts file that will be used later in Ansible playbooks:

```
[webservers]
web1.example.com      ansible_user=ec2-user
web2.example.com      ansible_user=ubuntu
```

Assuming the inventory file path is /home/yan/ansible4aws/hosts, you can store host variables, in YAML format, in individual files:

```
/home/yan/ansible4aws/host_vars/web1.example.com
/home/yan/ansible4aws/host_vars/web2.example.com
```

For example, the data in the host variables file

/home/yan/ansible4aws/host_vars/web1.example.com might look like:

```
---
ansible_user: ec2-user
ansible_port: 5300
ansible_ssh_private_key_file: /home/yan/.ssh/keypair1.pem
```

The following variables control how Ansible interacts with remote hosts:

- `ansible_host`: The name of the host to connect to, if different from the alias you wish to give to it.
- `ansible_port`: The ssh port number, if not 22.
- `ansible_user`: The default ssh user name to use.
- `ansible_ssh_pass`: The ssh password to use (this is insecure, it's strongly recommended to use `--ask-pass` or SSH keys).
- `ansible_ssh_private_key_file`: Private key file used by ssh. Useful if using multiple keys and you don't want to use SSH agent.
- `ansible_ssh_common_args`: This setting is always appended to the default command line for sftp, scp, and ssh. Useful to configure a `ProxyCommand` for a certain host (or group).
- `ansible_sftp_extra_args`: This setting is always appended to the default sftp command line.
- `ansible_scp_extra_args`: This setting is always appended to the default scp command line.
- `ansible_ssh_extra_args`: This setting is always appended to the default ssh command line.
- `ansible_ssh_pipelining`: Determines whether or not to use SSH pipelining.

This can override the pipelining setting in `ansible.cfg`.

Privilege escalation:

- `ansible_become`: Equivalent to `ansible_sudo` or `ansible_su`, allows to force privilege escalation
- `ansible_become_method`: Allows to set privilege escalation method
- `ansible_become_user`: Equivalent to `ansible_sudo_user` or `ansible_su_user`, allows to set the user you become through privilege escalation
- `ansible_become_pass`: Equivalent to `ansible_sudo_pass` or `ansible_su_pass`, allows you to set the privilege escalation password

Remote host environment parameters:

- `ansible_shell_type`: The shell type of the target system. You should not use this setting unless you have set the `ansible_shell_executable` to a non-Bourne (sh) compatible shell. By default commands are formatted using sh-style syntax. Setting this to csh or fish will cause commands executed on target systems to follow those shell's syntax instead.
- `ansible_python_interpreter`: The target host python path. This is useful for systems with more than one Python or not located at `/usr/bin/python` such as *BSD, or where `/usr/bin/python` is not a 2.X series Python.
- `ansible_*_interpreter`: Works for anything such as ruby or perl and works just like `ansible_python_interpreter`. This replaces shebang of modules which will run on that host.

You can find more documentation on host variables here: http://docs.ansible.com/ansible/intro_inventory.html#host-variables

Group variables

Variables can be applied to an entire group at once:

```
[sydney]
node1
node2
[sydney:vars]
ntp_server=ntp.sydney.example.com
proxy_server=proxy.sydney.example.com
```

It is also possible to assign variables to groups of groups:

```
[sydney]
host1
host2

[singapore]
host3
host4

[asiapacific:children]
sydney
singapore

[asiapacific:vars]
db_server=db1.asiapacific.example.com
```

Assuming the inventory file path is /home/yan/ansible4aws/hosts, you can store group variables, in YAML format, in individual files:

```
/home/yan/ansible4aws/group_vars/sydney
```

For example, the data in the group variables file

/home/yan/ansible4aws/group_vars/sydney might look like:

```
---
ntp_server: ntp.sydney.example.com
proxy_server: proxy.sydney.example.com
```

Your first commands

It's time to get started with some Ansible basic commands. Let's create our Ansible inventory file in your home directory /home/yan/ansible4aws/hosts. If the directory /home/yan/ansible4aws does not exist, create the directory first.

```
$ mkdir /home/yan/ansible4aws && cd /home/yan/ansible4aws
$ vi hosts

[local]
localhost

# You can add more hosts
# [mygroup]
# 192.168.1.10
# 192.168.1.11
```

Now ping all your nodes:

```
$ ansible -i hosts all -m ping
```

Ansible will attempt to connect to remote machines using your current user name, just like SSH would. To override the remote user name, use the `-u` parameter.

We can also run a live command on all of our nodes:

```
$ ansible -i hosts all -a "/bin/echo hello"
```

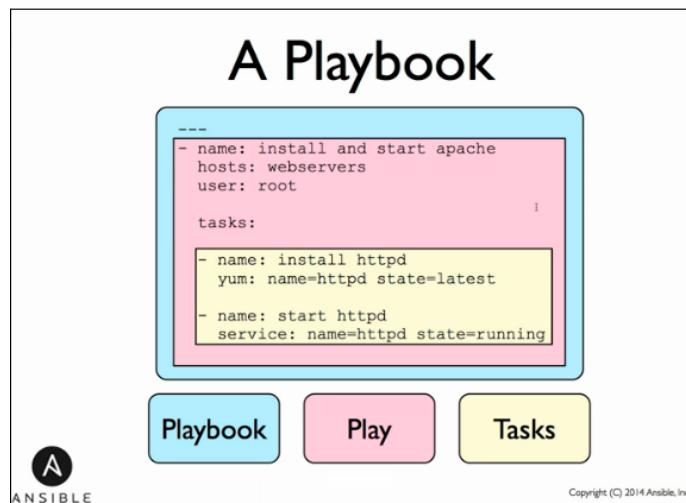
The above examples show how to use `/usr/bin/ansible` for running ad-hoc tasks. An ad-hoc command can be used to do quick things that you might not necessarily want to write a full playbook for. For example, if you want to restart some services on remote hosts.

Playbooks

Playbooks are Ansible's configuration, deployment, and orchestration language. They are a completely different way to use Ansible than in ad-hoc task execution mode.

Playbooks are written in YAML format and have a minimum of syntax, which intentionally tries to not be a programming language or script, but rather a model of a configuration or a process. Writing in YAML format allows you to describe your automation jobs in a way that approaches plain English. It is easy to learn and easy to understand for new Ansible users, but it is also powerful for expert users.

Each playbook is composed of one or more *plays* in a list. A play maps a group of hosts to some well defined roles, represented by ansible *tasks*. A task is a call to an Ansible module.



Ansible playbook

A module can control system resources, like services, packages, or files, or anything on remote hosts. Modules can be executed from the command line using `/usr/bin/ansible`, or by writing a playbook and run it using `/usr/bin/ansible-playbook` command. Each module supports taking arguments. Nearly all modules take key=value arguments, space delimited. Some modules take no arguments, and the command/shell modules simply take the string of the command you want to run.

Examples of executing modules from the command line:

```
$ ansible webservers -m service -a "name=httpd state=started"
$ ansible webservers -m ping
$ ansible webservers -m command -a "/sbin/reboot -t now"
```

Executing modules from playbooks:

```
- name: reboot the servers
  command: /sbin/reboot -t now
```

Ansible ships with hundreds of modules. Some examples of Ansible modules:

- Package management: yum, apt
- Remote execution: command, shell
- Service management: service
- File handling: copy, template
- SCM: git, subversion

- Database: mysql_db, redis
- Cloud: digital_ocean, ec2, gce

For a complete list of Ansible modules, see Ansible Module Index (http://docs.ansible.com/ansible/modules_by_category.html) page.

Most modules are *idempotent*, they will only make changes in order to bring the system to the desired state. It is safe to rerun the same playbook multiple times. It won't make changes if the desired state has been achieved.



YAML Syntax

For Ansible playbook, nearly every YAML file starts with a list. Each item in the list is a list of key/value pairs, commonly called a **hash** or a **dictionary**. All YAML files should begin with ---, which indicates the start of a document.

All members of a list are lines beginning at the same indentation level starting with a - (dash) character:

```
---  
- hosts: webservers  
  tasks:  
    - name: ensure apache is installed  
      yum: name=httpd state=present  
- hosts: databases  
  tasks:  
    - name: ensure mysql server is installed  
      yum: name=mysql-server state=present
```

A play in playbook consists of three sections: the hosts section, the variables section, and the tasks section. You can include as many plays as you like in a single playbook.

The hosts section

The hosts section defines hosts on which the play will be run, and how it will be run. In this section you can set the SSH username or other SSH-related settings needed to connect to the targeted hosts.

```
- hosts: webservers  
  remote_user: root
```

The hosts that a play will be run on must be set in the value of `hosts`. This value uses the same host-pattern-matching syntax as the one used in Ansible command line:

- The following patterns target all hosts in inventory:

```
all  
*
```

- To address a specific host or set of hosts by name or IP addresses and wildcards:

```
one.example.com  
one.example.com:two.example.com  
192.168.1.1  
192.168.1.*  
*.example.com
```

- The following patterns address one or more groups. Groups separated by a colon indicate an **OR** configuration (the host may be in either one group or the other):

```
webservers  
webservers:dbservers
```

- To exclude groups:

```
webservers:!sydney
```

- All hosts must be in the webservers group but not in the sydney group
- Group intersection:

```
webservers:&staging
```

- All hosts must be in the webservers group and also in the staging group
- You can also use regular expressions. Start the pattern with a ~:

```
~(web|db).*\.\example\.com
```

In this section you can provide some parameters:

- **become**: Set to `true` or `yes` to activate privilege escalation.
- **become_user**: Set to user with desired privileges, the user you *become*, NOT the user you login as. Does NOT imply *become: yes*, to allow it to be set at host level.
- **become_method**: At play or task level overrides the default method set in `ansible.cfg`, set to `sudo`, `su`, `pbrun`, `pfexec`, `doas`, or `dzdo`
- **connection**: Tells Ansible what connection method to use to connect to the remote hosts. You can use `ssh`, `paramiko`, or `local`.
- **gather_facts**: If set to `no` Ansible will not run the `setup` module to collect facts from remote hosts.

The variables section

In this section you can define variables that apply to the entire play on all hosts. You can also tell Ansible to prompt for variables. This section allows you to have all the configuration for the play stored at the top, so you can easily read and modify.

```
- hosts: webservers
  vars:
    http_port: 80
    region: ap-southeast-2
```

To use the variable in the tasks section, use `{{ variable }}` syntax:

```
- hosts: webservers
  vars:
    region: ap-southeast-2
  tasks:
    - name: create key pair
      local_action:
        module: ec2_key
        region: "{{ region }}"
```

Variables can also be loaded from external YAML files. This is done using the `vars_files` directive:

```
- hosts: webservers
  vars_files:
    - /vars/external_vars.yml
```

You can instruct Ansible to prompt for variables using the `vars_prompt` directive:

```
vars_prompt:
  - name: 'vpc_subnet_id'
    prompt: 'Enter the VPC subnet ID: '
```

It is also possible to send variables over the Ansible command line. For example:

```
---
- hosts: '{{ hosts }}'
  remote_user: '{{ user }}'

  tasks:
    - ...
```

Run the playbook and pass the variables:

```
$ ansible-playbook -i hosts site.yml --extra-vars "hosts=dbservers  
user=ec2-user"
```

The tasks section

Each play contains a list of tasks. Tasks are executed in order, one at a time, against all hosts matched by the host pattern, before moving on to the next task. When running the playbook, which runs top to bottom, hosts with failed tasks are taken out of the execution for the entire playbook, and error messages for the hosts will be displayed. If the run failed, simply correct the playbook and rerun.

Every task should have a name. The name should have a good description of what the task will do. It will be included in the output messages from running the playbook. Below the name line, you can declare the action. Tasks can be declared using the older `action:module options` format, but it is recommended to use the `module:options` format.

```
tasks:  
  - name: make sure apache is running  
    service: name=httpd state=running
```

The `command` and `shell` modules are the only modules that just take a list of arguments and don't use the key = value form:

```
tasks:  
  - name: disable selinux  
    command: /sbin/setenforce 0
```

The `command` and `shell` modules will return error code. If the exit code is not zero you can ignore the error:

```
tasks:  
  - name: run some command and ignore the return code  
    shell: /usr/bin/somecommand  
    ignore_errors: yes
```

If the action line is getting too long, you can break it into separate lines, indent any continuation lines:

```
tasks:  
  - name: Copy somefile to remote host  
    copy: src=/home/somefile dest=/etc/somefile  
    owner=root group=root mode=0644
```

Handlers

Handlers are lists of tasks, referenced by name, called by `notify` directive. Notify actions are triggered when the task made a change on the remote system. If many tasks in a play notify one handler, it will run only once, after all tasks completed in a particular play.

```
tasks:  
  - name: Configure ntp file  
    template: src=ntp.conf.j2 dest=/etc/ntp.conf  
    notify: restart ntp  
handlers:  
  - name: restart ntp  
    service: name=ntpd state=restarted
```

The service `ntpd` will be restarted only if the template module made changes to remote hosts file `/etc/ntp.conf`.

Your first playbook

Let's create our first playbook and run it.

Make sure you have created the inventory file `hosts`:

```
$ cd /home/yan/ansible4aws  
$ vi hosts  
  
[local]  
localhost
```

Create a playbook file `site.yml`:

```
$ vi site.yml  
  
---  
- hosts: localhost  
  tasks:  
    - name: ensure apache is at the latest version  
      yum: name=httpd state=latest  
    - name: ensure apache is running  
      service: name=httpd state=started
```

Save the file and run the playbook:

```
$ ansible-playbook -i hosts site.yml
```

```
PLAY [localhost] ****
GATHERING FACTS ****
ok: [localhost]

TASK: [ensure apache is at the latest version] ****
changed: [localhost]

TASK: [ensure apache is running] ****
changed: [localhost]

PLAY RECAP ****
localhost : ok=3    changed=2    unreachable=0    failed=0
```

Running our first playbook

Apache is now installed and running.

```
# ps ax | grep httpd
2357 ? Ss 0:00 /usr/sbin/httpd
2360 ? S 0:00 /usr/sbin/httpd
2361 ? S 0:00 /usr/sbin/httpd
2362 ? S 0:00 /usr/sbin/httpd
2363 ? S 0:00 /usr/sbin/httpd
2364 ? S 0:00 /usr/sbin/httpd
2365 ? S 0:00 /usr/sbin/httpd
$ yum info httpd
Installed Packages
Name        : httpd
Arch        : x86_64
Version     : 2.2.15
Release     : 30.el6.centos
Size        : 2.9 M
Repo        : installed
From repo   : updates
Summary     : Apache HTTP Server
URL         : http://httpd.apache.org/
License     : ASL 2.0
```

Now, let's uninstall Apache using our playbook

```
$ vi site.yml

---
- hosts: localhost
  tasks:
    - name: ensure apache is absent
      yum: name=httpd state=absent
```

Save the file and run the playbook:

```
$ ansible-playbook -i hosts site.yml
```

```
PLAY [localhost] ****
GATHERING FACTS ****
ok: [localhost]

TASK: [ensure apache is absent] ****
changed: [localhost]

PLAY RECAP ****
localhost : ok=2    changed=1    unreachable=0    failed=0
```

Ensure Apache was absent

Apache is successfully uninstalled from localhost.

Roles and include statements

It is possible to write everything in a single playbook, list all tasks in one very large file for your entire infrastructure, but it will be very hard to read and to maintain. Roles and include statements can help you organize things.

At a basic level, `include` directive can be used to break up bits of configuration policy into smaller files. You can write tasks in a separate file and include it in your play. Playbooks can also include plays from other playbook files.

A task include file simply contains a flat list of tasks, for example:

```
---
# saved as tasks/task1.yml

- name: task one      command: /bin/commandone
- name: task two      command: /bin/commandtwo
```

Include directives look like this, and can be mixed in with regular tasks in a playbook:

```
tasks:
  - include: tasks/task1.yml
```

Roles are a better way to organize your playbooks. Roles are ways of automatically loading certain variables, tasks, and handlers based on a known file structure. Grouping content using roles makes it easier to share roles with other users. Example project structure:

```
site.yml
webservers.yml
dbservers.yml
roles/
  common/
    files/
    templates/
    tasks/
    handlers/
    vars/
    meta/
  webservers/
    files/
    templates/
    tasks/
    handlers/
    vars/
    meta/
```

In a playbook, it would look like this:

```
---
- hosts: webservers
  roles:
    - common
    - webservers
```

This designates the following behaviors, for each role *x*:

- If `roles/x/tasks/main.yml` exists, tasks listed therein will be added to the play
- If `roles/x/handlers/main.yml` exists, handlers listed therein will be added to the play
- If `roles/x/vars/main.yml` exists, variables listed therein will be added to the play
- If `roles/x/meta/main.yml` exists, any role dependencies listed therein will be added to the list of roles
- Any copy tasks can reference files in `roles/x/files/` without having to path them relatively or absolutely
- Any script tasks can reference scripts in `roles/x/files/` without having to path them relatively or absolutely

- Any template tasks can reference files in `roles/x/templates/` without having to path them relatively or absolutely
- Any include tasks can reference files in `roles/x/tasks/` without having to path them relatively or absolutely



Ansible is a young rising project and may be rapidly changing. To update your knowledge of Ansible, visit Ansible documentations page (<http://docs.ansible.com/>).



Ansible provides a nice quick start video. You can find it here <http://www.ansible.com/resources>.

3

EC2 Provisioning and Configuration Management with Ansible

Ansible makes configuration management, application deployment and continuous delivery in AWS easy, with no agents or special coding skills required. In this chapter you will learn about AWS Elastic Compute Cloud (EC2) provisioning and configuration management using Ansible.

This chapter will also show you how to use the EC2 inventory plugin to dynamically manage configuration of your AWS EC2 instances, without having to list all EC2 instances DNS name or IP addresses in the inventory file.

Python boto

To interact with Amazon Web Services, Ansible uses **boto**, a Python interface to Amazon Web Services. At the moment, boto supports (for more information visit <http://boto.readthedocs.org/en/latest/#currently-supported-services>):

1. Compute
 - Elastic Compute Cloud (EC2)
 - Elastic Map Reduce (EMR)
 - AutoScaling
 - Kinesis
 - Lambda

- EC2 Container Service (ECS)
2. Content Delivery
- CloudFront
3. Database
- Relational Data Service (RDS)
 - DynamoDB
 - SimpleDB
 - ElastiCache
 - Redshift
4. Deployment and Management
- Elastic Beanstalk
 - CloudFormation
 - Data Pipeline
 - Opsworks
 - CloudTrail
 - CodeDeploy
5. Identity & Access
- Identity and Access Management (IAM) Security Token Service (STS)
 - Key Management Service (KMS)
 - Config
 - CloudHSM
6. Application Services
- CloudSearch
 - CloudSearch Domain
 - Elastic Transcoder
 - Simple Workflow Service (SWF)
 - Simple Queue Service (SQS)
 - Simple Notification Server (SNS)
 - Simple Email Service (SES)
 - Amazon Cognito Identity
 - Amazon Cognito Sync
 - Amazon Machine Learning

7. Monitoring

- CloudWatch
- CloudWatch Logs

8. Networking

- Route53
- Route53 Domain
- Virtual Private Cloud (VPC)
- Elastic Load Balancing (ELB)
- AWS Direct Connect

9. Payments and Billing

- Amazon Flexible Payment Service (FPS)

10. Storage

- Simple Storage Service (S3)
- Amazon Glacier
- Amazon Elastic Block Store (EBS)
- Google Cloud Storage

11. Workforce

- Amazon Mechanical Turk

12. Other

- Marketplace Web Services
- AWS Support

To install boto:

```
$ sudo pip install boto
...
Successfully installed boto
Cleaning up...
```

AWS access keys

To use AWS API, you'll need to obtain your AWS access keys first. AWS access keys consists of an access key ID and a secret access key.

To create new access keys:

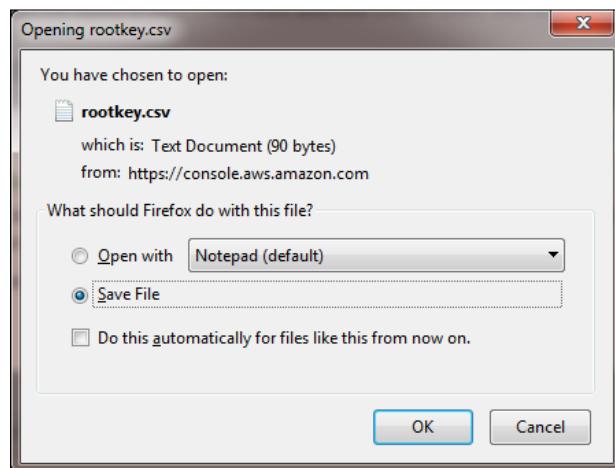
1. Go to **Your Security Credentials** page (https://console.aws.amazon.com/iam/home?#security_credential).

The screenshot shows the AWS IAM 'Your Security Credentials' page. On the left, there's a sidebar with links: Dashboard, Details, Groups, Users, Roles, Identity Providers, and Password Policy. The main content area has a title 'Your Security Credentials'. It includes a note about managing credentials for AWS Identity and Access Management (IAM) users. Below this, there are three expandable sections: 'Password', 'Multi-Factor Authentication (MFA)', and 'Access Keys (Access Key ID and Secret Access Key)'. The 'Access Keys' section is expanded, showing a note that you can have a maximum of two access keys at a time. A table lists existing access keys: one created on Jun 27th 2014 and deleted, and another created on Jun 13th 2014 and deleted. At the bottom of this section is a 'Create New Access Key' button.

2. Click **Create New Access Key** button.

The screenshot shows a 'Create Access Key' dialog box. It displays a green checkmark and the message 'Your access key (access key ID and secret access key) has been created successfully.' It also says to download the key file now. Below this, there's a note about protecting the secret access key. At the bottom, there are 'Show Access Key', 'Download Key File', and 'Close' buttons.

3. Click **Download Key File** button and click **OK** to save the key file.



4. Click on **Close** button.



You can retrieve the access key ID from the Security Credentials page, but you cannot retrieve the secret access key. If the secret access key is lost or forgotten, you need to create new access keys.



In this example, we created root access keys, which provide unlimited access to your AWS resources. To help secure your account, follow an AWS best practice by creating and using **AWS Identity and Access Management (IAM)** users with limited permissions. Learn more about creating IAM users here: http://docs.aws.amazon.com/IAM/latest/UserGuide/Using_SettingUpUser.html.

Now we need to configure boto to use our AWS access keys. To configure boto, create `~/.boto` file with your AWS credentials:

```
$ vi ~/.boto
```

```
[Credentials]
aws_access_key_id = YOURACCESSKEY
aws_secret_access_key = YOURSECRETKEY
```

Cloud modules

Ansible ships with a number of cloud modules to provision and manage cloud resources. You can see the full list of cloud modules at

http://docs.ansible.com/ansible/list_of_cloud_modules.html. Ansible contributors have written cloud modules for Amazon Web Services, Microsoft Azure, Digital Ocean, Google Cloud, and Rackspace. To check out for new updates and new modules, read Ansible changelog file <https://github.com/ansible/ansible/blob/devel/CHANGELOG.md>.

The following core modules are written for Amazon Web Services (Ansible version 2.1):

- `cloudformation`: Creates a CloudFormation stack.
- `ec2`: Creates, terminates, starts or stops an instance in EC2, return `instanceid`.
- `ec2_ami`: Creates or destroys an image in EC2, return `imageid`.
- `ec2_ami_find`: Searches for AMIs to obtain the AMI ID and other information.
- `ec2_asg`: Creates or deletes AWS AutoScaling Groups.
- `ec2_eip`: Associates an EC2 elastic IP with an instance.
- `ec2_elb`: De-registers or registers instances from EC2 ELBs.
- `ec2_elb_lb`: Creates or destroys Amazon ELB. Returns information about the load balancer.
- `ec2_facts`: Gathers facts about remote hosts within EC2.
- `ec2_group`: Maintains an EC2 VPC security group.
- `ec2_key`: Maintains an EC2 key pair.
- `ec2_lc`: Creates or deletes AWS AutoScaling Launch Configurations.
- `ec2_metric_alarm`: Creates/updates or deletes AWS Cloudwatch metric alarms.
- `ec2_scaling_policy`: Creates or deletes AWS scaling policies for AutoScaling groups.
- `ec2_snapshot`: Creates a snapshot from an existing volume.
- `ec2_tag`: Creates and removes tag(s) to EC2 resources.
- `ec2_vol`: Creates and attaches a volume, return volume id and device map.
- `ec2_vpc`: Configures AWS virtual private clouds.
- `ec2_vpc_net`: Configures AWS virtual private clouds.
- `elasticache`: Manages cache clusters in Amazon Elasticache.
- `elasticache_subnet_group`: Manage Elasticache subnet groups.
- `iam`: Manage IAM users, groups, roles and keys.

- `iam_cert`: Manage server certificates for use on ELBs and CloudFront.
- `iam_policy`: Manage IAM policies for users, groups, and roles.
- `rds`: Creates, deletes, or modifies an Amazon RDS instance.
- `rds_param_group`: Manages RDS parameter groups.
- `rds_subnet_group`: Manages RDS database subnet groups.
- `route53`: Adds or deletes entries in Amazons Route53 DNS service.
- `s3`: Manages S3 buckets and files.

Core modules are the ones that the core ansible team maintains and will always ship with Ansible itself. They will also receive slightly higher priority for all requests than those in the *extras* repos.

Some *extra modules* for AWS are available to use. These modules are currently shipped with Ansible, but might be shipped separately in the future. They are also mostly maintained by the community. Non-core modules are still fully usable, but may receive slightly lower response rates for issues and pull requests.

You can find the complete module list here: http://docs.ansible.com/ansible/list_of_cloud_modules.html.

Key pair

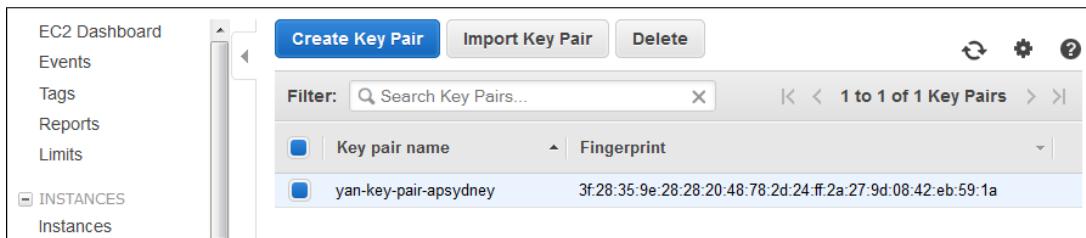
AWS uses public-key cryptography to secure the login information for your instance. A Linux instance has no password; you use a key pair to log in to your instance securely. You specify the name of the key pair when you launch your instance, then provide the private key when you log in. Key pairs are specific to a region; for example, if you plan to launch an instance in the Asia Pacific (Sydney) Region, you must create a key pair for the instance in the Asia Pacific (Sydney) Region.

In the following examples we will use `ec2_key` module to create or delete EC2 key pairs.

Options for the `ec2_key` module (http://docs.ansible.com/ec2_key_module.html):

parameter	required	default	choices	comments
<code>aws_access_key</code>	no			AWS access key. If not set then the value of the <code>AWS_ACCESS_KEY</code> environment variable is used
<code>aws_secret_key</code>	no			AWS secret key. If not set then the value of the <code>AWS_SECRET_KEY</code> environment variable is used
<code>ec2_url</code>	no			URL to use to connect to EC2 or your Eucalyptus cloud (by default the module will use EC2 endpoints). Must be specified if region is not used. If not set then the value of the <code>EC2_URL</code> environment variable, if any, is used
<code>key_material</code>	no			Public key material.
<code>name</code>	yes			Name of the key pair.
<code>profile</code>	no			Uses a boto profile. Only works with <code>boto >= 2.24.0</code> (added in Ansible 1.6)
<code>region</code>	no			The EC2 region to use
<code>security_token</code>	no			Security token to authenticate against AWS (added in Ansible 1.6)
<code>state</code>	no	present		Create or delete key pair
<code>validate_certs</code>	no	yes	yes no	When set to "no", SSL certificates will not be validated for boto versions $\geq 2.6.0$. (added in Ansible 1.5)
<code>wait</code>	no			Wait for the specified action to complete before returning. (added in Ansible 1.6)
<code>wait_timeout</code>	no	300		How long before wait gives up, in seconds (added in Ansible 1.6)

Let's try to delete the key pair we created in Chapter 1, *Getting Started with AWS*, example, using Ansible playbook.



Key Pair List

```
$ cd /home/yan/ansible4aws
```

Add localhost to the inventory file hosts:

```
$ vi hosts
```

```
[local]
localhost
```

Create the playbook file keypair.yml:

```
---
- hosts: localhost
  connection: local
  gather_facts: no
  vars:
    #your region
    region: ap-southeast-2
  tasks:
    - name: remove key pair
      ec2_key:
        region: "{{ region }}" #your key pair name
        name: yan-key-pair-apsydney
        state: absent
```

Execute the playbook:

```
$ ansible-playbook -i hosts keypair.yml
```

```
PLAY [localhost] *****
TASK: [remove key pair] *****
changed: [localhost]

PLAY RECAP *****
localhost : ok=1    changed=1    unreachable=0    failed=0
```

Execute the playbook

The key pair is successfully removed:



The Key Pair is removed

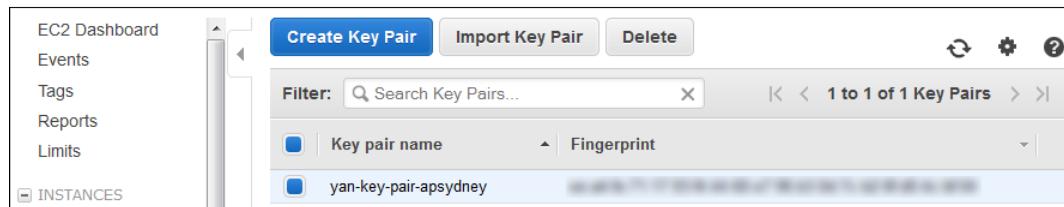
Now, let's create another key pair and save it to a file.

```
---
- hosts: localhost
  connection: local
  gather_facts: no
  vars:
    region: ap-southeast-2
    keyname: yan-key-pair-apsydne
  tasks:
    - name: create key pair
      ec2_key:
        region: "{{ region }}"
        name: "{{ keyname }}"
        register: mykey
    - debug: var=mykey
    - name: write to file
      copy: content="{{ mykey.key.private_key }}" dest="~/.ssh/{{ keyname }}.pem" mode=0600
      when: mykey.changed
```

Execute the playbook:

```
$ ansible-playbook -i hosts keypair.yml
```

New key pair is created and the private key is saved to `~/.ssh/yan-key-pair-apsydne.pem`.



The `register` keyword stores the output/result of a given module in a task into a variable, so you can access it later in another task.



The `when` keyword targets certain condition. Ansible will only run the `copy` module if the value of `changed` key inside `mykey` dictionary is `true`.

If you'll connect to your Linux instance from a computer running Windows using PuTTY, you'll need to provide `.ppk` private key file. To convert the `.pem` private key file to PuTTY `.ppk` file, follow the instruction in [Chapter 1, Getting Started with AWS](#), under the *Connect to Your Instance* section.

Security groups

Security groups act as a firewall for associated instances, controlling both inbound and outbound traffic at the instance level. You can add an inbound rule to a security group that enables you to connect to your instance from your IP address using SSH. You can also add other rules that allow inbound or outbound access for certain ports and IP addresses range. Note that if you plan to launch instances in multiple regions, you'll need to create a security group in each region.

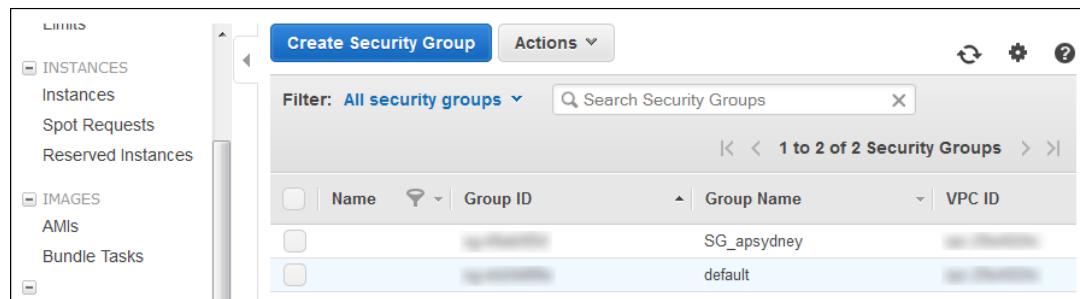
In the following examples we will use `ec2_group` module to create or delete EC2 security groups.

Options for the `ec2_group` module (http://docs.ansible.com/ec2_group_module.html):

parameter	required	default	choices	comments
<code>aws_access_key</code>	no			AWS access key. If not set then the value of the <code>AWS_ACCESS_KEY</code> environment variable is used
<code>aws_secret_key</code>	no			AWS secret key. If not set then the value of the <code>AWS_SECRET_KEY</code> environment variable is used
<code>description</code>	yes			Description of the security group.
<code>ec2_url</code>	no			URL to use to connect to EC2 or your Eucalyptus cloud (by default the module will use EC2 endpoints). Must be specified if region is not used. If not set then the value of the <code>EC2_URL</code> environment variable, if any, is used
<code>name</code>	yes			Name of the security group.
<code>profile</code>	no			Uses a boto profile. Only works with <code>boto >= 2.24.0</code> (added in Ansible 1.6)
<code>purge_rules</code>	no	true		Purge existing rules on security group that are not found in rules
<code>purge_rules_egress</code>	no	true		Purge existing rules_egress on security group that are not found in rules_egress
<code>region</code>	no			The EC2 region to use
<code>rules</code>	no			List of firewall inbound rules to enforce in this group
<code>rules_egress</code>	no			List of firewall outbound rules to enforce in this group (added in Ansible 1.6)
<code>security_token</code>	no			Security token to authenticate against AWS (added in Ansible 1.6)
<code>state</code>	no	present		Create or delete security group

parameter	required	default	choices	comments
validate_certs	no	yes	yes no	When set to "no", SSL certificates will not be validated for boto versions >= 2.6.0. (added in Ansible 1.5)
vpc_id	no			ID of the VPC to create the group in

Let's try to delete the security group we created in Chapter 1, *Getting Started with AWS*, example, using Ansible playbook.



Security Groups list

Create the playbook file /home/yan/ansible4aws/sg.yml.

```
---
- hosts: localhost
  connection: local
  gather_facts: no
  vars:
    #your region
    region: ap-southeast-2
  tasks:
    - name: remove security group
      ec2_group:
        region: "{{ region }}"
        #your security group name
        name: SG_apsydney
        description: sg apsydney
        state: absent
```

Execute the playbook:

```
$ ansible-playbook -i hosts sg.yml
```

```
PLAY [localhost] ****
TASK: [remove security group] ****
changed: [localhost]

PLAY RECAP ****
localhost : ok=1    changed=1    unreachable=0    failed=0
```

Execute the playbook

The security group SG_apsydney is successfully removed.

The screenshot shows the AWS Lambda function configuration page. At the top, there is a 'Create Function' button and a 'Actions' dropdown. Below that is a 'Filter: All functions' dropdown and a search bar labeled 'Search Functions'. The main area displays a table with one row. The columns are 'Name', 'Group ID', 'Group Name', and 'VPC ID'. The row contains the value 'default' under 'Group Name'. There are also 'Edit' and 'Delete' buttons for this row.

The security group is removed



You'll need the public IP address of your local computer, which you can get using a service from Amazon AWS <http://checkip.amazonaws.com>. If you are connecting through an Internet service provider (ISP) or from behind a firewall without a static IP address, you need to find out the range of IP addresses used by client computers.

Now, let's create a new security group, which allow SSH access from your local computer's IP address and allow HTTP access from anywhere:

```
$ vi sg_webserver.yml
```

```
---
- hosts: localhost
  connection: local
  gather_facts: no
  vars:
```

```
#your region
region: ap-southeast-2
#your ip address
allowed_ip: 123.xxx.xxx.xxx/32
tasks:
  name: create security group
  ec2_group:
    region: "{{ region }}" #your security group name
    name: sg_webserver_apsydney
    description: security group for apsydney webserver host
    rules:
      # allow ssh access from your ip address
      - proto: tcp
        from_port: 22
        to_port: 22
        cidr_ip: "{{ allowed_ip }}"
      # allow http access from anywhere
      - proto: tcp
        from_port: 80
        to_port: 80
        cidr_ip: 0.0.0.0/0
    rules_egress:
      - proto: all
        cidr_ip: 0.0.0.0/0
```

Execute the playbook:

```
$ ansible-playbook -i hosts sg_webserver.yml
```

The security group sg_webserver_apsydney is successfully created:

A screenshot of the AWS Management Console showing the 'Create Security Group' interface. The table lists two security groups: 'sg_webserver_apsydney' and 'default'. The 'sg_webserver_apsydney' row is highlighted with a blue selection bar.

	Name	Group ID	Group Name	VPC ID
<input checked="" type="checkbox"/>	sg-[REDACTED]		sg_webserver_apsydney	vpc-[REDACTED]
<input type="checkbox"/>	sg-[REDACTED]		default	vpc-[REDACTED]

New security group is created

Now select our newly created security group and select the **Inbound** tab to see the inbound rules:

Type	Protocol	Port Range	Source
SSH	TCP	22	123.***.***.*/32
HTTP	TCP	80	0.0.0.0/0

Inbound rules

The `ec2_group` module is idempotent. You can run the playbook again and Ansible won't make any changes. Later on you can change the `rules` or `rules_egress` in the playbook, for example you can add inbound rule to allow HTTPS access (TCP port 443) from anywhere, then run the playbook again. Ansible will make changes to the `sg_webserver_apsydne` security group.

EC2 provisioning

Ansible `ec2` module allows us to create, terminate, start, or stop an EC2 instance. It returns the instance ID.

Options for the `ec2` module (http://docs.ansible.com/ec2_module.html):

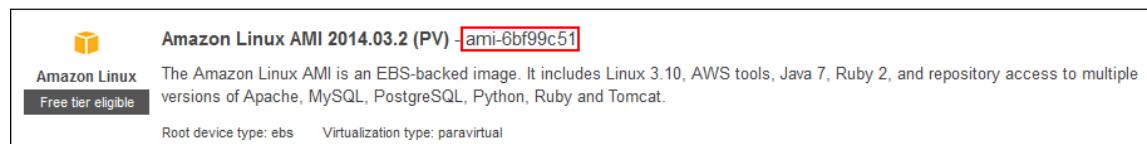
parameter	required	default	choices	comments
assign_public_ip (added in 1.5)	no			When provisioning within VPC, assign a public IP address.
aws_access_key	no			AWS access key. If not set then the value of the AWS_ACCESS_KEY environment variable is used aliases: ec2_access_key, access_key
aws_secret_key	no			AWS secret key. If not set then the value of the AWS_SECRET_KEY environment variable is used aliases: ec2_secret_key, secret_key
count	no	1		Number of instances to launch
count_tag (added in 1.5)	no			Used with ‘exact_count’ to determine how many nodes based on a specific tag criteria should be running. For instance, one can request 25 servers that are tagged with “class=webserver”.
ebs_optimized (added in 1.6)	no			Whether instance is using optimized EBS
ec2_url	no			URL to use to connect to EC2 or your Eucalyptus cloud (by default the module will use EC2 endpoints). Must be specified if region is not used. If not set then the value of the EC2_URL environment variable, if any, is used
exact_count (added in 1.5)	no			An integer value which indicates how many instances that match the ‘count_tag’ parameter should be running. Instances are either created or terminated based on this value.
group	no			Security group (or list of groups) to use with the instance aliases: groups
group_id	no			Security group id (or list of ids) to use with the instance
image	yes			ami ID to use for the instance
instance_ids	no			List of instance ids, currently used for states:

parameter	required	default	choices	comments
(added in 1.3)			absent, running, stopped aliases: <code>instance_id</code>	
<code>instance_profile_name</code> (added in 1.3)	no			Name of the IAM instance profile to use.
<code>instance_tags</code>	no			A hash/dictionary of tags to add to the new instance; <code>{“key”：“value”}</code> and <code>{“key”：“value”, “key”：“value”}</code>
<code>instance_type</code>	yes			Instance type to use for the instance
<code>kernel</code>	no			Kernel eki to use for the instance
<code>key_name</code>	no			Key pair to use on the instance aliases: <code>keypair</code>
<code>monitoring</code>	no			Enable detailed monitoring (CloudWatch) for instance
<code>network_interfaces</code> (added in 2.0)	no			A list of existing network interfaces to attach to the instance at launch. When specifying existing network interfaces, none of the <code>assign_public_ip</code> , <code>private_ip</code> , <code>group</code> , <code>vpc_subnet_id</code> , or <code>group_id</code> parameters may be used. (Those parameters are for creating a new network interface at launch). aliases: <code>network_interface</code>
<code>placement_group</code> (added in 1.3)	no			Placement group for the instance when using EC2 Clustered Compute
<code>private_ip</code>	no			The private IP address to be assigned to the instance (from the VPC subnet)
<code>profile</code> (added in 1.6)	no			Uses a boto profile. Only works with <code>boto >= 2.24.0</code>
<code>ramdisk</code>	no			ramdisk eri to use for the instance
<code>region</code>	no			The AWS region to use. Must be specified if <code>ec2_url</code> is not used. If not specified then the value of the <code>EC2_REGION</code> environment variable, if any, is used. ³⁶ aliases: <code>aws_region</code> , <code>ec2_region</code>
<code>security_token</code> (added in 1.6)	no			AWS STS security token. If not set then the value of the <code>AWS_SECURITY_TOKEN</code> or

parameter	required	default	choices	comments
				EC2_SECURITY_TOKEN environment variable is used. aliases: access_token
source_dest_check (added in 1.6)	no	True		Enable or Disable the Source/Destination checks (for NAT instances and Virtual Routers)
spot_launch_group (added in 2.1)	no			Launch group for spot request ³⁷
spot_price (added in 1.5)	no			Maximum spot price to bid, If not set a regular on-demand instance is requested. A spot request is made with this maximum bid. When it is filled the instance is started.
spot_type (added in 2.0)	no	one-time		Type of spot request; one of “one-time” or “persistent”. Defaults to “one-time” if not supplied.
spot_wait_timeout (added in 1.5)	no	600		How long to wait for the spot instance request to be fulfilled
state	no	present		Create or terminate instances
tenancy (added in 1.9)	no	default		An instance with a tenancy of “dedicated” runs on single-tenant hardware and can only be launched into a VPC. Note that to use dedicated tenancy you MUST specify a vpc_subnet_id as well. Dedicated tenancy is not available for EC2 “micro” instances.
termination_protection (added in 2.0)	no	yes no		Enable or Disable the Termination Protection
user_data	no			Opaque blob of data which is made available to the ec2 instance
validate_certs (added in 1.5)	no	yes no		When set to “no”, SSL certificates will not be validated for boto versions >= 2.6.0.
volumes (added in 1.5)	no			A list of volume dicts, each containing device name and optionally ephemeral id or snapshot id. Size and type (and number of iops for io device type) must be specified for a new volume or a root volume, and may be passed for a snapshot volume. For any volume, a volume size less than 1 will be interpreted as a request not to create the volume.

parameter	required	default	choices	comments
vpc_subnet_id	no			The subnet ID in which to launch the instance (VPC)
wait	no	no	yes no	Wait for the instance to be in state ‘running’ before returning
wait_timeout	no	300		How long before wait gives up, in seconds
zone	no			AWS availability zone in which to launch the instance aliases: aws_zone, ec2_zone

Let's start with a basic provisioning example. We will use the key pair and security group we created previously in this chapter. We'll also need to know the AMI ID we want to launch on the instance. We will use the free tier eligible Amazon Linux AMI for all examples in this chapter. To see the AMI ID, go to your EC2 dashboard, select your region, and then click on **Launch Instance** button, then note the AMI ID of the free tier eligible Amazon Linux AMI with Root device type: ebs and Virtualization type: paravirtual.



The AMI ID

The following playbook will launch one EC2 instance in ap-southeast-2 region, using key pair yan-key-pair-apsydney and sg_webserver_apsydney security group:

```
$ vi launch_ec2.yml

---
- hosts: localhost
  connection: local
  gather_facts: no
  vars:
    #your region
    region: ap-southeast-2
  tasks:
    - name: EC2 basic provisioning
      ec2:
        region: "{{ region }}"
```

```
key_name: yan-key-pair-apsydney
instance_type: t2.micro
image: ami-dc361ebf
wait: yes
group: sg_webserver_apsydney
```

Run the playbook:

```
$ ansible-playbook -i hosts launch_ec2.yml
```

In the navigation pane, click **Instances** and see your newly launched instance.

The screenshot shows the AWS CloudWatch Metrics console interface. At the top, there are filters for 'All instances' and 'All instance types', a search bar, and a breadcrumb trail indicating '1 to 1 of 1 Instances'. Below this, a table displays a single instance: i-9e1e18a1, which is a t1.micro instance running in the ap-southeast-2a availability zone. The instance state is shown as 'running' with a green status indicator. A detailed view of the instance is shown below the table, with tabs for 'Description', 'Status Checks', 'Monitoring', and 'Tags'. The 'Description' tab is selected, showing the following details:

Attribute	Value	Attribute	Value
Instance ID	i-9e1e18a1	Public DNS	ec2-54-79-47-129.ap-southeast-2.compute.amazonaws.com
Instance state	running	Public IP	54.79.47.129
Instance type	t1.micro	Elastic IP	-
Private DNS	ip-172-31-8-82.ap-southeast-2.compute.internal	Availability zone	ap-southeast-2a
Private IPs	172.31.8.82	Security groups	sg_webserver_apsydney. view rules

New instance launched

The preceding playbook example is not idempotent. If you run the playbook again, another new instance will be launched. To ensure idempotency operation of the EC2 provisioning operation, we can use the combination of `count_tag` and `exact_count` options.

EC2 provisioning example, using the combination of `count_tag` and `exact_count`:

```
$ vi launch_ec2_count.yml

---
- hosts: localhost
  connection: local
  gather_facts: no
  vars:
    #your region
    region: ap-southeast-2
  tasks:
    - name:
        EC2 basic provisioning ec2:
          region: "{{ region }}"
          key_name: yan-key-pair-apsydney
          instance_type: t2.micro
          image: ami-dc361ebf
          wait: yes
          group: sg_webserver_apsydney
          instance_tags:
            group: webserver
          exact_count: 5
          count_tag:
            group: webserver
```

Run the playbook:

```
$ ansible-playbook -i hosts launch_ec2_count.yml
```

The preceding example launched five instances with tags `group=webserver`.

	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks
	i-9e1e18a1	t1.micro	ap-southeast-2a	green running	green 2/2 checks ...	
	i-f61b1dc9	t1.micro	ap-southeast-2a	green running	green 2/2 checks ...	
<input checked="" type="checkbox"/>	i-36272109	t1.micro	ap-southeast-2a	green running	yellow Initializing	
	i-0b272134	t1.micro	ap-southeast-2a	green running	yellow Initializing	
	i-0a272135	t1.micro	ap-southeast-2a	green running	yellow Initializing	
	i-09272136	t1.micro	ap-southeast-2a	green running	yellow Initializing	
	i-08272137	t1.micro	ap-southeast-2a	green running	yellow Initializing	

Key	Value
group	webserver

Launch instances with count_tag

If we run the playbook again, Ansible will not make any changes because there are already five instances with tags group=webserver.

```
$ ansible-playbook -i hosts launch_ec2_count.yml
```

```
PLAY [localhost] ****
TASK: [EC2 basic provisioning] ****
ok: [localhost]

PLAY RECAP ****
localhost                  : ok=1      changed=0      unreachable=0      failed=0
```

Idempotent Operation

We can use the exact_count option to enforce the number of running instances based on specific tag criteria. Try to change the exact_count from 5 to 3 in the playbook:

```
instance_tags:
  group: webserver
  exact_count: 3
```

```
count_tag:  
  group: webserver
```

Run the playbook again:

```
$ ansible-playbook -i hosts launch_ec2_count.yml
```

Two instances with tags group=webserver should be terminated by Ansible to match the parameter exact_count: 3.

If you have finished learning the examples, let's terminate all instances using a playbook:

```
$ vi terminate_ec2.yml  
  
---  
- hosts: localhost  
  connection: local  
  gather_facts: no  
  vars:  
    region: ap-southeast-2  
  tasks:  
    - name: terminate instances  
      ec2:  
        region: "{{ region }}"  
        wait: yes  
        # List all of your running instance IDs here  
        instance_ids: ['i-9e1e18a1', 'i-f61b1dc9', 'i-36272109',  
                      'i-0b272134', 'i-0a272135']  
        state: absent  
$ ansible-playbook -i hosts terminate_ec2.yml
```



Check the EC2 dashboard and make sure that all instances have been terminated. The AWS Free Tier only allows 750 hours per month of Amazon Linux Micro Instance usage.

Elastic IP address (EIP)

An **Elastic IP address (EIP)** is a static IP address designed for dynamic cloud computing. With an EIP, you can mask the failure of an instance or software by rapidly remapping the address to another instance in your account. Your EIP is associated with your AWS account, not a particular instance, and it remains associated with your account until you choose to explicitly release it (for more information visit <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/elastic-ip-addresses-eip.html>).

When you launch an EC2 instance, a public IP address is assigned to your instance from Amazon's pool of public IP addresses, and is not associated with your AWS account. When a public IP address is disassociated from your instance, it is released back into the public IP address pool, and you cannot reuse it.

Amazon releases the public IP address for your instance when it's stopped or terminated. Your stopped instance receives a new public IP address when it's restarted. If you require a persistent public IP address that can be associated to and from instances as you require, use an EIP instead. You can allocate your own EIP, and associate it to your instance.

Amazon releases the public IP address for your instance when you associate an EIP with your instance, or when you associate an EIP with the primary network interface (eth0) of your instance in a VPC. When you disassociate the EIP from your instance, it receives a new public IP address.

If the public IP address of your instance in a VPC has been released, it will not receive a new one if there is more than one network interface attached to your instance (for more information visit

<http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-instance-addressing.html>).

We will use Ansible ec2_eip module to allocate and associate AWS EIP with instances. Options for this module are (for more information visit http://docs.ansible.com/ec2_eip_module.html).

parameter	required	default	choices	comments
aws_access_key	no			AWS access key. If not set then the value of the AWS_ACCESS_KEY environment variable is used aliases: ec2_access_key, access_key
aws_secret_key	no			AWS secret key. If not set then the value of the AWS_SECRET_KEY environment variable is used aliases: ec2_secret_key, secret_key
device_id (added in 2.0)	no			The id of the device for the EIP. Can be an EC2 Instance or Elastic Network Interface (ENI) id. aliases: instance_id
ec2_url	no			URL to use to connect to EC2 or your Eucalyptus cloud (by default the module will use EC2 endpoints). Must be specified if region is not used. If not set then the value of the EC2_URL

parameter	required	default	choices	comments
				environment variable, if any, is used.
in_vpc	no			Allocate an EIP inside a VPC or not.
profile (added in 1.6)	no			Uses a boto profile. Only works with boto >= 2.24.0
public_ip	no			The elastic IP address to associate with the instance. If absent, allocate a new address.
region	no			The AWS EC2 region to use.
release_on_ disassociation (added in 2.0)	no			whether or not to automatically release the EIP when it is disassociated
reuse_existing _ip_allowed (added in 1.6)	no			Reuse an EIP that is not associated to an instance (when available), instead of allocating a new one.
security_token (added in 1.6)	no			Security token to authenticate against AWS
state	no	present	present absent	If present, associate the IP with the instance. If absent, disassociate the IP with the instance
validate_certs (added in 1.5)	no	yes	yes no	When set to "no", SSL certificates will not be validated for boto versions >= 2.6.0.

This following playbook will launch an EC2 instance, allocate new Elastic IP address, and associate the EIP with the instance:

```
$ vi launch_ec2_eip.yml

---
- hosts: localhost
  connection: local
  gather_facts: no
  vars:
    region: ap-southeast-2
  tasks:
    - name: launch instance ec2:
      region: "{{ region }}"
      key_name: yan-key-pair-apsydney
      instance_type: t2.micro
      image: ami-dc361ebf
      wait: yes
      group: sg_webserver_apsydney
```

```
register: ec2
- name: associate new EIP for the instance
  ec2_eip:
    region: "{{ region }}"
    instance_id: "{{ item.id }}"
  with_items: ec2.instances
```

Run the playbook:

```
$ ansible-playbook -i hosts launch_ec2_eip.yml
```

A new EC2 instance should be created with an EIP associated to the instance:

Instance: i-dac3c2e5 Elastic IP: 54.79.109.14	
Description	
Instance ID	i-dac3c2e5
Public DNS	ec2-54-79-109-14.ap-southeast-2.compute.amazonaws.com
Instance state	running
Public IP	54.79.109.14
Instance type	t1.micro
Elastic IP	54.79.109.14
Private DNS	ip-172-31-5-100.ap-southeast-2.compute.internal
Availability zone	ap-southeast-2a
Private IPs	172.31.5.100
Security groups	sg_webserver_apsydney.

New instance with EIP

Configuration management

Using Ansible, you can easily manage software packages, users, configuration files, services, and any other resources on your EC2 instances. To do this, first we have to list all nodes to be managed in Ansible's inventory.

In the following example we will use the /home/yan/ansible4aws/hosts inventory file to list our managed nodes in AWS. We will talk about **dynamic inventory** later in this chapter.

Add the public IP address of your EC2 instance to the Ansible hosts file:

```
$ vi /home/yan/ansible4aws/hosts
[local]
localhost
[webservers]
54.79.109.14 ansible_user=ec2-user
ansible_ssh_private_key_file=~/ssh/yan-key-pair-apsydne.pem
```

Run the ping module to test Ansible connection to all hosts. The module returns pong on successful connection.

```
$ ansible all -m ping
```

You should receive a success message as shown below if the Ansible connection is success:

```
paramiko: The authenticity of host '54.79.109.14' can't be established.
The ssh-rsa key fingerprint is 621a3eb85140063966269e3cef3ab0bf.
Are you sure you want to continue connecting (yes/no)?
yes
localhost | success >> {
    "changed": false,
    "ping": "pong"
}

54.79.109.14 | success >> {
    "changed": false,
    "ping": "pong"
}
```

Test Ansible connection

Let's create Ansible roles and playbook to install and configure **Network Time Protocol (NTP)** service on the instance. NTP service is installed and configured by default on Amazon Linux instances, but it's OK, you'll learn about roles and template usage in this example.

```
$ cd /home/yan/ansible4aws
$ mkdir roles
$ mkdir group_vars
$ mkdir roles/common
$ mkdir roles/common/tasks
$ mkdir roles/common/handlers
$ mkdir roles/common/templates
```

List the tasks here:

```
$ vi roles/common/tasks/main.yml

---
- name: install ntp
  yum: name=ntp state=present
  tags: ntp
- name: configure ntp file
  template: src=ntp.conf.j2 dest=/etc/ntp.conf
  tags: ntp
  notify: restart ntp
- name: start the ntp service
  service: name=ntpd state=started enabled=true
  tags: ntp
```

Ansible template module will create a file on remote node based on a template in Ansible template directory. Options for the template module (http://docs.ansible.com/template_module.html):

parameter	required	default	choices	comments
backup	no	no	yes no	Create a backup file including the timestamp information so you can get the original file back
dest	yes			Location to render the template to on the remote machine
force	no	yes		The default is yes, which will replace the remote file when contents are different than the source. If no, the file will only be transferred if the destination does not exist.
group	no			Name of the group that should own the file or directory, as would be fed to <i>chown</i>
mode	no			Mode the file or directory should be. For those used to /usr/bin/chmod remember that modes are actually octal numbers (like 0644). Leaving off the leading zero will likely have unexpected

parameter	required	default	choices	comments
				results. As of version 1.8, the mode may be specified as a symbolic mode (for example, <code>u+rwx</code> or <code>u=rw,g=r,o=r</code>).
owner	no			name of the user that should own the file or directory, as would be fed to <code>chown</code>
selevel	no	s0		level part of the SELinux file context. This is the MLS/MCS attribute, sometimes known as the range. <code>_default</code> feature works as for <code>seuser</code> .
serole	no			role part of SELinux file context, <code>_default</code> feature works as for <code>seuser</code> .
setype	no			type part of SELinux file context, <code>_default</code> feature works as for <code>seuser</code> .
seuser	no			user part of SELinux file context. Will default to system policy, if applicable. If set to <code>_default</code> it will use the <code>user</code> portion of the policy if available.
src	yes			Path of a Jinja2 formatted template on the local server. This can be a relative or absolute path.
validate	no			The validation command to run before copying into place. The path to the file to validate is passed in via ' <code>%s</code> ' which must be present as in the visudo

Write the handler to restart ntp service:

```
$ vi roles/common/handlers/main.yml  
---  
- name: restart ntp  
  service: name=ntpd state=restarted
```

Write the configuration file template:

```
$ vi roles/common/templates/ntp.conf.j2  
driftfile /var/lib/ntp/drift  
restrict 127.0.0.1  
restrict -6 ::1  
server {{ ntpserver }}  
includefile /etc/ntp/crypto/pw  
keys /etc/ntp/keys
```



Templates are processed by the Jinja2 templating language <http://jinja.pocoo.org/docs>. Documentation on the template formatting can be found in the Template Designer Documentation <http://jinja.pocoo.org/docs/templates>.

You can configure the ntpserver variable here:

```
$ vi group_vars/all  
---  
# Variables here are applicable to all host groups  
ntpserver: 0.au.pool.ntp.org
```

Finally, the playbook for our site:

```
$ vi site.yml  
---  
- hosts: webservers  
  become: yes  
  roles:  
    - common
```

Run the playbook:

```
$ ansible-playbook -i hosts site.yml
```

```
PLAY [webservers] *****  
  
GATHERING FACTS *****  
ok: [54.79.109.14]  
  
TASK: [common | install ntp] *****  
ok: [54.79.109.14]  
  
TASK: [common | configure ntp file] *****  
changed: [54.79.109.14]  
  
TASK: [common | start the ntp service] *****  
ok: [54.79.109.14]  
  
NOTIFIED: [common | restart ntp] *****  
changed: [54.79.109.14]  
  
PLAY RECAP *****  
54.79.109.14 : ok=5     changed=2     unreachable=0     failed=0
```

Install and configure ntp service

The problem with using static inventory file is that not every instance in our AWS infrastructure uses Elastic IP address. Some instances will have AWS dynamic IP addressing scheme. The IP address will be released if we stop the instance and a new IP address will be allocated when we restart the instance.

If you have finished learning this example, terminate the instance and release the EIP. Go to your EC2 dashboard. In the navigation pane, select **Instances**, right click on the running instance and select **Action – Terminate** and click on **Yes, Terminate** button. Then select **Elastic IPs** in the navigation pane, right click on the IP address and select **Release Addresses**, then click on **Yes, Release** button.



AWS impose an hourly charge if an EIP is not associated with a running instance, or if it is associated with a stopped instance or an unattached network interface.

Dynamic inventory

If you use AWS EC2, maintaining an inventory file might not be the best approach, because hosts may come and go over time, be managed by external applications, or you might even be using AWS autoscaling. We will use the Ansible EC2 external inventory script to provide mappings to our instances.

The EC2 external inventory provides mappings to instances from several groups (for more information visit http://docs.ansible.com/intro_dynamic_inventory.html):

- **Global:** All instances are in group `ec2`.
- **Instance ID:** These are groups of one since instance IDs are unique, for example, `i-00112233 i-a1b1c1d1`.
- **Region:** A group of all instances in an AWS region, for example, `us-east-1 us-west-2`.
- **Availability Zone:** A group of all instances in an availability zone, for example, `us-east-1a us-east-1b`.
- **Security Group:** Instances belong to one or more security groups. A group is created for each security group, with all characters except alphanumerics, dashes (-) converted to underscores (_). Each group is prefixed by `security_group_`, for example, `security_group_default` `security_group_web servers` `security_group_Pete_s_Fancy_Group`.

- **Tags:** Each instance can have a variety of key/value pairs associated with it called **Tags**. The most common tag key is `Name`, though anything is possible. Each key/value pair is its own group of instances, again with special characters converted to underscores, in the format `tag_KEY--VALUE`, for example, `tag_Name_Web tag_Name_redis-master-001 tag_aws_cloudformation_logicalid_WebServerGroup`.

We will use the Tags grouping to classify our instances. In this following example we will create 2 instances in webserver group and 1 instance in database group. The provisioning playbook will add tags to the instances to classify them.

First, download the `ec2.py` script and `ec2.ini` file to your Ansible project directory `/home/yan/ansible4aws`:

```
$ cd /home/yan/ansible4aws
$ wget
https://raw.github.com/ansible/ansible/devel/plugins/inventory/ec2.py
$ wget
https://raw.github.com/ansible/ansible/devel/plugins/inventory/ec2.ini
$ chmod +x ec2.py
```

Create the security group for database server:

```
$ vi sg_database.yml

---
- hosts: localhost
  connection: local
  gather_facts: no
  vars:
    region: ap-southeast-2
    allowed_ip: 123.xxx.xxx.xxx/32
  tasks:
    - name: create database security group
      ec2_group:
        region: "{{ region }}"
        name: sg_database_apsydney
        description: security group for apsydney database host
        rules:
          # allow ssh access from your ip address
          - proto: tcp
            from_port: 22
            to_port: 22
            cidr_ip: "{{ allowed_ip }}"
          # allow mysql access from webserver group
          - proto: tcp
```

```
    from_port: 3306
    to_port: 3306
    group_name: sg_webserver_apsydney
    rules_egress:
      - proto: all
        cidr_ip: 0.0.0.0/0
```

Run the playbook:

```
$ ansible-playbook -i hosts sg_database.yml
```

Provision new instances with tags:

```
$ vi launch_ec2_tags.yml

---
- hosts: localhost
  connection: local
  gather_facts: no
  vars:
    region: ap-southeast-2
    instance_type: t2.micro
    image: ami-d9fe9be3
    key: yan-key-pair-apsydney
  tasks:
    - name: launch ec2 with tags webserver staging
      ec2:
        region: "{{ region }}"
        key_name: "{{ key }}"
        instance_type: "{{ instance_type }}"
        image: "{{ image }}"
        wait: yes
        group: sg_webserver_apsydney
        instance_tags:
          Name: staging-webserver-1
          class: webserver
          environment: staging
    - name: launch ec2 with tags webserver production
      ec2:
        region: "{{ region }}"
        key_name: "{{ key }}"
        instance_type: "{{ instance_type }}"
        image: "{{ image }}"
        wait: yes
        group: sg_webserver_apsydney
        instance_tags:
          Name: production-webserver-1
          class: webserver
```

```
environment: production
- name: launch ec2 with tags database staging
  ec2:
    region: "{{ region }}"
    key_name: "{{ key }}"
    instance_type: "{{ instance_type }}"
    image: "{{ image }}"
    wait: yes
    group: sg_database_apsydne
    instance_tags:
      Name: staging-database-1
      class: database
      environment: staging
```

Run the playbook:

```
$ ansible-playbook -i hosts launch_ec2_tags.yml
```

We've created three instances with different tags:

	Name	Instance ID	Instance Type	Availability Zone	Instance State
<input checked="" type="checkbox"/>	staging-webserver-1	i-8d11b7b3	t2.micro	ap-southeast-2b	running
<input type="checkbox"/>	staging-database-1	i-ee13b5d0	t2.micro	ap-southeast-2b	running
<input type="checkbox"/>	production-webserver-1	i-ec13b5d2	t2.micro	ap-southeast-2b	running

Description **Status Checks** **Monitoring** **Tags**

Add/Edit Tags

Key	Value	
Name	staging-webserver-1	Hide Column
class	webserver	Show Column
environment	staging	Show Column

Instances with tags



As of this writing, Amazon has just launched new T2 instance type. From now on I will use the t2.micro instance in this book's examples. The AMI used is the free tier eligible Amazon Linux AMI with Root device type: ebs and Virtualization type: hvm.

Let's see how the Ansible EC2 external inventory script's mappings work.

First, add the SSH user and key file configuration in the group variables file `group_vars/all`:

```
$ vi group_vars/all
# Variables here are applicable to all host groups
ntpserver: 0.au.pool.ntp.org
ansible_user: ec2-user
ansible_ssh_private_key_file: ~/.ssh/yan-key-pair-apsydney.pem
```

Now, use Ansible with the EC2 external inventory script to ping matched instances:

Ping all instances:

```
$ ansible -i ec2.py all -m ping
```

If the `ec2.py` script returns `ImportError: No module named argparse` message, install the `argparse` module:

```
# easy_install argparse
```

Ping all instances:

```
$ ansible -i ec2.py ec2 -m ping
```

Ping instances in `ap-southeast-2` region:

```
$ ansible -i ec2.py ap-southeast-2 -m ping
```

Ping instances with `sg_webserver_apsydney` security group:

```
$ ansible -i ec2.py security_group_sg_webserver_apsydney -m ping
```

Ping instances with tags class=database:

```
$ ansible -i ec2.py tag_class_database -m ping
```

Ping instances with tags class=webserver and environment=staging:

```
$ ansible -i ec2.py "tag_class_webserver:&tag_environment_staging" -m ping
```

The following example will show you how to use this script with Ansible playbook.

Create roles to install and start apache:

```
$ mkdir roles/apache  
$ mkdir roles/apache/tasks
```

List the tasks here:

```
$ vi roles/apache/tasks/main.yml  
  
---  
- name: install apache  
  yum: name=httpd state=present  
  tags: apache  
- name: start the httpd service  
  service: name=httpd state=started enabled=true  
  tags: apache
```

Create roles to install and start mysql:

```
$ mkdir roles/mysql  
$ mkdir roles/mysql/tasks
```

List the tasks here:

```
$ vi roles/mysql/tasks/main.yml  
  
---  
- name: install mysql server  
  yum: name=mysql-server state=present  
  tags: mysql  
- name: start the mysql service  
  service: name=mysqld state=started enabled=true  
  tags: mysql
```

Playbook for our site:

```
$ vi site.yml

---
# install, configure, and start ntp on all ec2 instances
- hosts: ec2
  become: yes
  roles:
    - common
# install and start mysql server on instance with tags class=database
- hosts: tag_class_database
  become: yes
  roles:
    - mysql
# install and start apache on instance
# with tags class=webserver and environment=staging
- hosts: tag_class_webserver:&tag_environment_staging
  become: yes
  roles:
    - apache
```

Run the playbook:

```
$ ansible-playbook -i ec2.py site.yml
```

```
PLAY [ec2] ****
GATHERING FACTS ****
ok: [54.79.108.27]
ok: [54.79.40.16]
ok: [54.79.108.171]

TASK: [common | install ntp] ****
ok: [54.79.108.27]
ok: [54.79.40.16]
ok: [54.79.108.171]

TASK: [common | configure ntp file] ****
changed: [54.79.108.27]
changed: [54.79.40.16]
changed: [54.79.108.171]

TASK: [common | start the ntp service] ****
ok: [54.79.108.27]
ok: [54.79.108.171]
ok: [54.79.40.16]

NOTIFIED: [common | restart ntp] ****
changed: [54.79.108.27]
changed: [54.79.108.171]
changed: [54.79.40.16]

PLAY [tag_class_database] ****
GATHERING FACTS ****
ok: [54.79.108.171]

TASK: [mysql | install mysql server] ****
changed: [54.79.108.171]

TASK: [mysql | start the mysql service] ****
changed: [54.79.108.171]

PLAY [tag_class_webserver:&tag_environment_staging] ****
GATHERING FACTS ****
ok: [54.79.108.27]

TASK: [apache | install apache] ****
changed: [54.79.108.27]

TASK: [apache | start the httpd service] ****
changed: [54.79.108.27]

PLAY RECAP ****
54.79.108.171      : ok=8    changed=4    unreachable=0    failed=0
54.79.108.27      : ok=8    changed=4    unreachable=0    failed=0
54.79.40.16       : ok=5    changed=2    unreachable=0    failed=0
```

Run site.yml playbook

You can see from the preceding example how easy it is to manage EC2 instances configuration with Ansible dynamic inventory, without even knowing the IP address or DNS name of the instances, and without having to install any agent on the managed nodes. You can easily group your managed nodes based on region, availability zones, security group, or any type of classification scheme you define with the instance tagging.

If you have finished with the example, don't forget to terminate all instances to avoid any cost.



You can also mix both dynamic and statically managed inventory sources in the same Ansible run. To do this, create a directory, for example `/home/yan/ansible4aws/hostfiles/`, and move your inventory file(s) and external inventory script(s) to this directory, then run `ansible` or `ansible-playbook` using `-i /home/yan/ansible4aws/hostfiles/` option.

Elastic Block Store – EBS

An Amazon EBS volume is a durable, block-level storage device that you can attach to a single EC2 instance. You can use Amazon EBS volumes as primary storage for data that requires frequent updates, such as the system drive for an instance or storage for a database application. Amazon EBS volumes persist independently from the running life of an EC2 instance. After a volume is attached to an instance, you can use it like any other physical hard drive (for more information visit

<http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSVolumes.html>)

Amazon EBS provides the following volume types: General Purpose (SSD), Provisioned IOPS (SSD), and Magnetic. They differ in performance characteristics and price, allowing you to tailor your storage performance and cost to the needs of your applications (for more information visit

<http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSVolumeTypes.html>).

- General Purpose (SSD) volumes offer cost-effective storage that is ideal for a broad range of workloads. These volumes deliver single-digit millisecond latencies, the ability to burst to 3,000 IOPS for extended periods of time, and a base performance of 3 IOPS/GB. General Purpose (SSD) volumes can range in size from 1 GB to 1 TB.

- Provisioned IOPS (SSD) volumes are designed to meet the needs of I/O-intensive workloads, particularly database workloads, that are sensitive to storage performance and consistency in random access I/O throughput. You specify an IOPS rate when you create the volume, and Amazon EBS delivers within 10 percent of the provisioned IOPS performance 99.9 percent of the time over a given year.
A Provisioned IOPS (SSD) volume can range in size from 10 GB to 1 TB and you can provision up to 4,000 IOPS per volume. The ratio of IOPS provisioned to the volume size requested can be a maximum of 30; for example, a volume with 3,000 IOPS must be at least 100 GB. You can stripe multiple volumes together in a RAID configuration for larger size and greater performance.
- Magnetic volumes provide the lowest cost per gigabyte of all Amazon EBS volume types. Magnetic volumes are backed by magnetic drives and are ideal for workloads performing sequential reads, workloads where data is accessed infrequently, and scenarios where the lowest storage cost is important. These volumes deliver approximately 100 IOPS on average, with burst capability of up to hundreds of IOPS, and they can range in size from 1 GB to 1 TB. Magnetic volumes can be striped together in a RAID configuration for larger size and greater performance.

For detailed pricing information on these volume types, see <http://aws.amazon.com/ebs/pricing>.



IOPS (Input/Output Operations Per Second, pronounced eye-ops) is a common performance measurement used to benchmark computer storage devices like hard disk drives (HDD), solid state drives (SSD), and storage area networks (SAN), for more information visit <http://en.wikipedia.org/wiki/IOPS>.

Ansible `ec2` module has the `volume` option to set the parameter of EBS we want to attach to the instance. The `volume` option is a list of volume dictionary, containing:

- `device_name`: Device name to override device mapping.
- `snapshot`: Snapshot ID on which to base the volume.
- `device_type`: Type of volume, valid options are `standard` (magnetic volume), `gp2` (general purpose SSD volume), and `io1` (provisioned IOPS SSD volume).
- `iops`: The provisioned IOPS you want to associate with this volume (integer).
- `volume_size`: Size of volume (in GB)
- `delete_on_termination`: If set to `true` or `yes` will delete the volume on instance termination.

The following is a playbook to launch an EC2 instance with a 100 GB general purpose SSD volume:

```
$ vi ec2_vol_1.yml

---
- hosts: localhost
  gather_facts: no
  vars:
    #your region
    region: ap-southeast-2
  tasks:
    - name: EC2 provisioning with provisioned IOPS EBS volume
      ec2:
        region: "{{ region }}"
        key_name: yan-key-pair-apsydney
        instance_type: t2.micro
        image: ami-dc361ebf
        group: sg_webserver_apsydney
        volumes:
          - device_name: /dev/sda1
            device_type: gp2
            volume_size: 100
            delete_on_termination: true
```

And the following is a playbook to launch an EC2 instance with a 500 GB provisioned IOPS SSD volume (provisioned at 1000 IOPS):

```
$ vi ec2_vol_2.yml

---
- hosts: localhost
  gather_facts: no
  vars:
    #your region
    region: ap-southeast-2
  tasks:
    - name: EC2 provisioning with provisioned IOPS EBS volume
      ec2:
        region: "{{ region }}"
        key_name: yan-key-pair-apsydney
        instance_type: t2.micro
        image: ami-dc361ebf
        group: sg_webserver_apsydney
        volumes:
          - device_name: /dev/sda1
            device_type: io1
            iops: 1000
```

```
volume_size: 500  
delete_on_termination: true
```

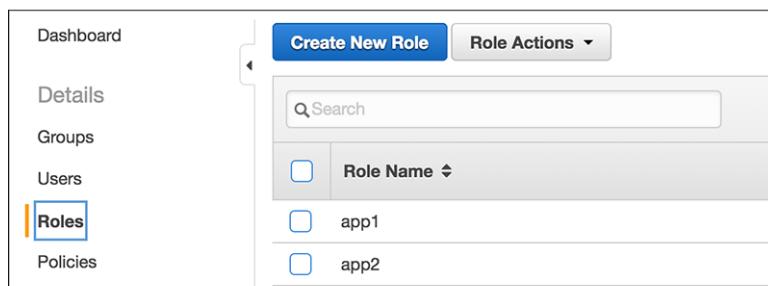
Instance profile

An instance profile is a container for an *IAM role* and enable you to pass role information to an Amazon EC2 instance when the instance starts (for more information visit <http://docs.aws.amazon.com/IAM/latest/UserGuide/roles-usingrole-instanceprofile.html>). An *IAM role* is a set of permissions that grant access to actions and resources in AWS. You define the permissions for a role in an *IAM policy*, which is a JSON document written in the *IAM Policy Language* (for more information visit <http://docs.aws.amazon.com/IAM/latest/UserGuide/roles-usingrole-instanceprofile.html>).

With *IAM roles*, applications run in an EC2 instance can securely make API requests, without requiring you to manage the security credentials that the applications use. Instead of creating and distributing your AWS credentials, you can delegate permission to make API requests using IAM roles. For example, you can use IAM roles to grant permissions to applications running on your instances that need to use a bucket in Amazon S3 (for more information visit <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html>).

The following steps will create an IAM role and instance profile to allow read access to your S3 bucket:

1. Open the IAM console <https://console.aws.amazon.com/iam>.
2. In the navigation pane, click on **Roles**.



3. Click on **Create New Role**, enter the role name, and click on **Next Step**.

<p>Create Role</p> <p>Step 1: Set Role Name</p> <p>Step 2: Select Role Type</p> <p>Step 3: Establish Trust</p> <p>Step 4: Attach Policy</p> <p>Step 5: Review</p>	<h3>Set Role Name</h3> <p>Enter a role name. You cannot edit the role name after the role is created.</p> <p>Role Name <input type="text" value="S3ReadAccess"/> Maximum 64 characters. Use alphanumeric and '+-=,@-_ ' characters</p>
--	---

4. Select **Amazon EC2**.

<p>Create Role</p> <p>Step 1: Set Role Name</p> <p>Step 2: Select Role Type</p> <p>Step 3: Establish Trust</p> <p>Step 4: Attach Policy</p> <p>Step 5: Review</p>	<h3>Select Role Type</h3> <p><input checked="" type="radio"/> AWS Service Roles</p> <p>› Amazon EC2 Allows EC2 instances to call AWS services on your behalf. <input type="button" value="Select"/></p> <p>› AWS Directory Service Allows AWS Directory Service to manage access for existing directory users and groups to AWS services. <input type="button" value="Select"/></p>
---	---

5. In the Attach Policy **Search** pane, type `s3`, select **AmazonS3ReadOnlyAccess** policy, and click on **Next Step**.

<p>Create Role</p> <p>Step 1: Set Role Name</p> <p>Step 2: Select Role Type</p> <p>Step 3: Establish Trust</p> <p>Step 4: Attach Policy</p> <p>Step 5: Review</p>	<h3>Attach Policy</h3> <p>Select up to two policies to attach to the role.</p> <p>Filter: Policy Type ▾ <input type="text" value="s3"/></p> <table border="1"><thead><tr><th></th><th>Policy Name ▾</th><th>Attached Entities ▾</th></tr></thead><tbody><tr><td><input type="checkbox"/></td><td>AmazonS3FullAccess</td><td>0</td></tr><tr><td><input checked="" type="checkbox"/></td><td>AmazonS3ReadOnlyAccess</td><td>0</td></tr></tbody></table>		Policy Name ▾	Attached Entities ▾	<input type="checkbox"/>	AmazonS3FullAccess	0	<input checked="" type="checkbox"/>	AmazonS3ReadOnlyAccess	0
	Policy Name ▾	Attached Entities ▾								
<input type="checkbox"/>	AmazonS3FullAccess	0								
<input checked="" type="checkbox"/>	AmazonS3ReadOnlyAccess	0								

6. Click on **Create Role**.

If you created your IAM role using the console, the instance profile was created for you and given the same name as the role (`S3ReadAccess`).

To launch an instance with this role using the `ec2` module, set the `instance_profile_name` parameter to `S3ReadAccess`.



New modules for IAM have been added in Ansible v2, `iam` module can be used to manage IAM users, groups, roles and keys, and `iam_policy` module can be used to manage IAM policies for users, groups, and roles. Find out more about these modules in Chapter 14, *Identity and Access Management (IAM)*.

4

Project 1 - A WordPress Site

In this chapter we will create a simple WordPress blogging site in AWS, using Ansible. We will install everything in a single EC2 instance with Elastic IP address. We will create complete Ansible playbooks to do everything from provisioning to configuration management.

First, let's create our WordPress project directory:

```
$ cd /home/yan/ansible4aws  
$ mkdir wordpress
```

Copy the external inventory script `ec2.py` and `ec2.ini` file to this project directory:

```
$ cp ec2* wordpress/
```

Add localhost to the inventory file `hosts`:

```
$ cd wordpress$ vi hosts
```

```
[local]  
localhost
```

Provisioning playbook

Playbook for provisioning:

```
$ vi provisioning.yml  
  
---  
- hosts: localhost  
  connection: local  
  gather_facts: no
```

```
vars:
  #your region
  region: ap-southeast-2
  keyname: wordpress-apsydney
  #your ip address
  allowed_ip: 123.xxx.xxx.xxx/32
  instance_type: t2.micro
  image: ami-dc361ebf
tasks:
- name: create key pair
  tags: keypair
  ec2_key:
    region: "{{ region }}"
    name: "{{ keyname }}"
  register: mykey

- name: write the private key to file
  copy: content="{{ mykey.key.private_key }}" dest="~/.ssh/{{ keyname }}.pem"
- mode=0600
  when: mykey.changed
- name: create security group
  tags: sg
  ec2_group:
    region: "{{ region }}"
    #your security group name
    name: sg_wordpress_apsydney
    description: security group for apsydney webserver host
    rules:
      # allow ssh access from your ip address
      - proto: tcp
        from_port: 22
        to_port: 22
        cidr_ip: "{{ allowed_ip }}"
      # allow http access from anywhere
      - proto: tcp
        from_port: 80
        to_port: 80
        cidr_ip: 0.0.0.0/0
      # allow https access from anywhere
      - proto: tcp
        from_port: 443
        to_port: 443
        cidr_ip: 0.0.0.0/0
    rules_egress:
      - proto: all
        cidr_ip: 0.0.0.0/0
- name: launch ec2 instance
```

```
tags: ec2
ec2:
  region: "{{ region }}"
  key_name: "{{ keyname }}"
  instance_type: "{{ instance_type }}"
  image: "{{ image }}"
  wait: yes
  group: sg_wordpress_apsydney
  id: wordpress_launch_1
  instance_tags:
    Name: wordpress-1
    class: wordpress
register: ec2
- name: associate new EIP for the instance
  tags: eip
  ec2_eip:
    region: "{{ region }}"
    instance_id: "{{ item.id }}"
  with_items: ec2.instances
  when: item.id is defined
```

Run the playbook:

```
# ansible-playbook -i hosts provisioning.yml
```

The `tags` keyword allows you to run specific tasks without running the whole playbook. For example, if you modify the security groups rules and you want to run only the security groups part from the preceding playbook:

ansible-playbook -i hosts provisioning.yml --tags "sg".
To run the playbook *without* the key pair tasks you could do this:
ansible-playbook -i hosts provisioning.yml --skip-tags
"keypair".

The `ec2_eip` module in the preceding example is not idempotent. If you run the playbook again, a new Elastic IP will be allocated for the instance. Check your AWS EC2 console for any disassociated EIP and release it to avoid any cost.



Variables file

Create variables file:

```
$ cd /home/yan/ansible4aws/wordpress
$ mkdir group_vars
$ vi group_vars/all

ansible_ssh_user: ec2-user
ansible_ssh_private_key_file: ~/.ssh/wordpress-apsydney.pem

# Which version of WordPress to deploy
wp_version: 4.5.3

# These are the WordPress database settings
wp_db_name: wordpress
wp_db_user: wordpress
# Set your database password here
wp_db_password: secret

# WordPress settings

# Disable All Updates
# By default automatic updates are enabled, set this value to true to
# disable all automatic updates
auto_up_disable: false

#Define Core Update Level
#true = Development, minor, and major updates are all enabled
#false = Development, minor, and major updates are all disabled
#minor = Minor updates are enabled, development, and major updates are
disabled
core_update_level: true
```

Roles directory

Create directory for roles:

```
$ cd /home/yan/ansible4aws/wordpress
$ mkdir roles && cd roles
$ mkdir common web mysql wordpress
$ cd common && mkdir tasks && cd ..
$ cd web && mkdir tasks && cd ..
$ cd mysql && mkdir handlers tasks templates && cd ..
$ cd wordpress && mkdir tasks templates && cd ..
```

Common roles

Tasks for common roles:

```
$ vi roles/common/tasks/main.yml  
---  
- name: install the 'Development tools' package group  
  yum: name="@Development tools" state=present update_cache=yes
```

Web roles

Tasks for web roles (Apache, PHP, PHP-MySQL):

```
$ vi roles/web/tasks/main.yml  
---  
- name: install apache, php, and php-mysql  
  yum: name={{ item }} state=present  
  with_items:  
    - httpd  
    - php  
    - php-mysql  
- name: start and enable httpd  
  service: name=httpd state=started enabled=yes
```

mysql roles

Handler for mysql roles:

```
$ vi roles/mysql/handlers/main.yml  
---  
- name: restart mysql  
  service: name=mysqld state=restarted
```

Template for mysql configuration:

```
$ vi roles/mysql/templates/my.cnf.j2
```

```
[mysqld]  
datadir=/var/lib/mysql  
socket=/var/lib/mysql/mysql.sock  
user=mysql
```

```
# Disabling symbolic-links is recommended to prevent assorted security
risks
symbolic-links=0 port=3306

[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

Tasks for mysql roles:

```
$ vi roles/mysql/tasks/main.yml

---
- name: install mysql server
  yum: name={{ item }} state=present
  with_items:
    - mysql-devel
    - mysql-server

- name: install mysql-python
  pip: name=mysql-python state=present

- name: create mysql configuration file
  template: src=my.cnf.j2 dest=/etc/my.cnf
  notify: restart mysql

- name: start mysql service
  service: name=mysqld state=started enabled=true
```

wordpress roles

Template for wordpress configuration:

```
$ vi roles/wordpress/templates/wp-config.php

<?php
/**
 * The base configurations of the WordPress.
 *
 * This file has the following configurations: MySQL settings, Table
Prefix,
 * Secret Keys, WordPress Language, and ABSPATH. You can find more
information
 * by visiting {@link http://codex.wordpress.org/Editing_wp-config.php
Editing
 * wp-config.php} Codex page. You can get the MySQL settings from your web
host.
```

```
*  
* This file is used by the wp-config.php creation script during the  
* installation. You don't have to use the web site, you can just copy this  
file  
* to "wp-config.php" and fill in the values. @package WordPress  
*/  
  
// ** MySQL settings - You can get this info from your web host ** //  
/** The name of the database for WordPress */  
define('DB_NAME', '{{ wp_db_name }}');  
  
/** MySQL database username */  
define('DB_USER', '{{ wp_db_user }}');  
  
/** MySQL database password */  
define('DB_PASSWORD', '{{ wp_db_password }}');  
  
/** MySQL hostname */  
define('DB_HOST', 'localhost');  
  
/** Database Charset to use in creating database tables. */  
define('DB_CHARSET', 'utf8');  
  
/** The Database Collate type. Don't change this if in doubt. */  
define('DB_COLLATE', '');  
  
/**#@+  
 * Authentication Unique Keys and Salts.  
 *  
 * Change these to different unique phrases!  
 * You can generate these using the {@link  
https://api.wordpress.org/secret-key/1.1/salt/ WordPress.org secret-key  
service}  
 * You can change these at any point in time to invalidate all existing  
cookies. This will force all users to have to log in again.  
 *  
 * @since 2.6.0  
*/  
  
<?php  
{{ wp_salt.stdout }}  
  
/**#@-*/  
  
/**  
 * WordPress Database Table prefix.  
 *  
 * You can have multiple installations in one database if you give each a  
unique
```

```
* prefix. Only numbers, letters, and underscores please!
*/
$table_prefix = 'wp_';

/**
 * WordPress Localized Language, defaults to English.

 * Change this to localize WordPress. A corresponding MO file for the
chosen
 * language must be installed to wp-content/languages. For example, install
 * de_DE.mo to wp-content/languages and set WPLANG to 'de_DE' to enable
German
 * language support.
 */
define('WPLANG', '');

/**
 * For developers: WordPress debugging mode.

 * Change this to true to enable the display of notices during development.
 * It is strongly recommended that plugin and theme developers use WP_DEBUG
 * in their development environments.
 */
define('WP_DEBUG', false);

/* That's all, stop editing! Happy blogging. */

/** Absolute path to the WordPress directory. */
if ( !defined('ABSPATH') )
    define('ABSPATH', dirname(FILE) . '/');

/** Sets up WordPress vars and included files. */
require_once(ABSPATH . 'wp-settings.php');

/** Disable Automatic Updates Completely */
define( 'AUTOMATIC_UPDATER_DISABLED', {{auto_up_disable}} );

/** Define AUTOMATIC Updates for Components. */
define( 'WP_AUTO_UPDATE_CORE', {{core_update_level}} );
```

Tasks for wordpress roles:

```
$ vi roles/wordpress/tasks/main.yml

---
- name: download wordpress
  get_url: url=http://wordpress.org/wordpress-{{ wp_version }}.tar.gz
  dest=~/wordpress-{{ wp_version }}.tar.gz
```

```
- name: extract wordpress archive
  command: chdir=~ /bin/tar xvf wordpress-{{ wp_version }}.tar.gz
  creates=~/wordpress

- name: copy wordpress to apache root directory
  shell: cp -r ~/wordpress/* /var/www/html

- name: fetch random salts for wordpress config
  local_action: command curl https://api.wordpress.org/secret-key/1.1/salt/
  register: "wp_salt"

- name: create wordpress database
  mysql_db: name={{ wp_db_name }} state=present

- name: create wordpress database user
  mysql_user: name={{ wp_db_user }} password={{ wp_db_password }} priv={{ wp_db_name }}.*:ALL host='localhost' state=present

- name: copy wordpress config file
  template: src=wp-config.php dest=/var/www/html/

- name: change ownership of wordpress installation
  file: path=/var/www/html/ owner=apache group=apache state=directory
  recurse=yes
```

site.yml

The playbook for our WordPress site's configuration management:

```
$ vi site.yml

---

- name: install apache, php, mysql server, wordpress
  hosts: tag_class_wordpress
  become: yes

  roles:
    - web
    - mysql
    - wordpress
```

Directory structure

Our WordPress project's directory structure should look like this:

```
/home/
  yan/
    ansible4aws/
      wordpress/
        ec2.ini
        ec2.py
        provisioning.yml
        site.yml
        group_vars/
          all
      roles/
        mysql/
          handlers/
            main.yml
          tasks/
            main.yml
          templates/
            my.cnf.j2
      web/
        tasks/
          main.yml
    wordpress/
      tasks/
        main.yml
      templates/
        wp-config.php
```

Playbook run

Run our `site.yml` playbook:

```
$ ansible-playbook -i hosts -i ec2.py site.yml
```

By the end of this playbook run, you will have a WordPress site up and running. Start your web browser and type the public IP address of your WordPress EC2 instance. You'll land on WordPress `wp-admin/install.php` page where you can set your admin username and password.

Enter your **Site Title**, **Username**, **Password** (twice) and your e-mail address. Also displayed is a check box asking if you would like your blog to appear in search engines like Google and Technorati. Leave the box checked if you would like your blog to be visible to everyone, including search engines, and uncheck the box if you want to block search engines, but allow normal visitors. Note all this information can be changed later in your **Administration Panels**.

5

Route 53 Management with Ansible

Amazon Route 53 is a scalable **Domain Name System (DNS)** web service. It provides secure and reliable routing to your infrastructure that uses Amazon Web Services (AWS) products, such as Amazon **Elastic Compute Cloud** (Amazon EC2), Elastic Load Balancing, or Amazon **Simple Storage Service** (Amazon S3). You can also use Amazon Route 53 to route users to your infrastructure outside of AWS.

Amazon Route 53 is an authoritative DNS service, meaning it translates friendly domains names like `www.example.com` into IP addresses like `192.0.2.1`. Amazon Route 53 responds to DNS queries using a global network of authoritative DNS servers, which reduces latency. For more information, visit <http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/HowDoesRoute53Work.html>

Hosted zones

A hosted zone is a collection of resource record sets hosted by Amazon Route 53. Like a traditional DNS zone file, a hosted zone represents a collection of resource record sets that are managed together under a single domain name. Each hosted zone has its own metadata and configuration information.

The resource record sets contained in a hosted zone must share the same suffix. For example, the `example.com` hosted zone can contain resource record sets for `www.example.com` and `www.aws.example.com` subdomains, but cannot contain resource record sets for a `www.example.ca` subdomain. For more information, visit <http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/AboutHostedZones.html>



The AWS free usage tier does not include Route 53 service. You will be charged a small monthly fee to use this service. See <http://aws.amazon.com/route53/pricing> for details.

To use Route 53 DNS service, first you have to register your domain name with a domain name registrar, for example, <http://www.godaddy.com>.

To create a hosted zone in the Route 53 console:

1. Open the Route 53 console <https://console.aws.amazon.com/route53>.



2. Click **Create Hosted Zone**.
3. In the **Create Hosted Zone** pane, enter a domain name and, optionally, a comment. For more information about a setting, pause the mouse pointer over its label to see a tool tip.

Create Hosted Zone

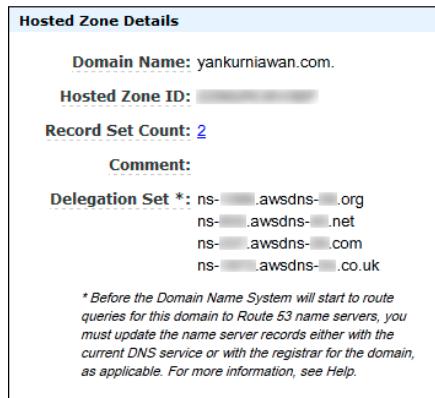
Domain Name: yankurniawan.com
Example: example.com

Comment:

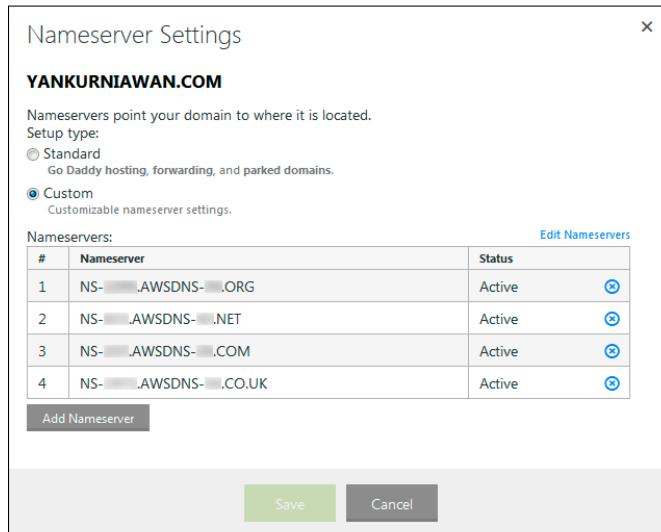
Optional comment about the hosted zone. You can't change the comment after you create a hosted zone. Maximum of 256 characters.

4. Click **Create**.

- When you create a hosted zone, Amazon Route 53 automatically creates 4 **Name Server (NS)** records for the zone. Login to your domain name registrar's account, go to your domain name management page, update the nameserver settings with these 4 name server records.



Hosted Zone Details



GoDaddy Nameserver Settings

DNS records management

We will use Ansibleroute 53 module to manage DNS records in Amazon Route 53 service. The options are (You can visit http://docs.ansible.com/route53_module.html for more information):

parameter	required	default	choices	comments
alias (added in 1.9)	no		True False	Indicates if this is an alias record.
alias_ evaluate_ target_health (added in 2.1)	no			Whether or not to evaluate an alias target health. Useful for aliases to Elastic Load Balancers.
alias_hosted_ zone_id (added in 1.9)	no			The hosted zone identifier.
aws_access_key	no			AWS access key. If not set then the value of the AWS_ACCESS_KEY environment variable is used aliases: ec2_access_key, access_key
aws_secret_key	no			AWS secret key. If not set then the value of the AWS_SECRET_KEY environment variable is used aliases: ec2_secret_key, secret_key
command	yes		get create delete	Specifies the action to take
ec2_url	no			URL to use to connect to EC2 or your Eucalyptus cloud (by default the module will use EC2 endpoints). Must be specified if region is not used. If not set then the value of the EC2_URL environment variable, if any, is used
failover (added in 2.0)	no			Failover resource record sets only. Whether this is the primary or secondary resource record set. Allowed values are PRIMARY and SECONDARY
health_check (added in 2.0)	no			Health check to associate with this record
hosted_zone_id (added in 2.0)	no			The Hosted Zone ID of the DNS zone to modify
identifier (added in 2.0)	no			Have to be specified for Weighted, latency-based and failover resource record sets only. An identifier that differentiates among multiple

parameter	required	default	choices	comments
				resource record sets that have the same combination of DNS name and type.
overwrite	no			Whether an existing record should be overwritten on create if values do not match
private_zone (added in 1.9)	no			If set to true, the private zone matching the requested name within the domain will be used if there are both public and private zones. The default is to use the public zone.
profile (added in 1.6)	no			Uses a boto profile. Only works with boto >= 2.24.0
record	yes			The full DNS record to create or delete
region (added in 2.0)	no			Latency-based resource record sets only. Among resource record sets that have the same combination of DNS name and type, a value that determines which region this should be associated with for the latency-based routing
retry_interval (added in 2.0)	no	500		In the case that route53 is still servicing a prior request, this module will wait and try again after this many seconds. If you have many domain names, the default of 500 seconds may be too long
security_token (added in 1.6)	no			AWS STS security token. If not set then the value of the AWS_SECURITY_TOKEN or EC2_SECURITY_TOKEN environment variable is used. aliases: access_token
ttl	no	3600		The TTL for the new record (in seconds)
type	yes		A CNAME MX AAAA TXT PTR SRV SPF NS SOA	The type of DNS record to create
validate_certs (added in 1.5)	no	yes	yes no	When set to "no", SSL certificates will not be validated for boto versions >= 2.6.0.
value	no			The new value when creating a DNS record.

parameter	required	default	choices	comments
				Multiple comma-spaced values are allowed. When deleting a record all values for the record must be specified or Route 53 will not delete it.
vpc_id (added in 2.0)	no			When used in conjunction with private_zone: true this will only modify records in the private hosted zone attached to this VPC. This allows you to have multiple private hosted zones, all with the same name, attached to different VPCs.
wait (added in 2.1)	no			Wait until the changes have been replicated to all Amazon Route 53 DNS servers.
wait_timeout (added in 2.1)	no	300		How long to wait for the changes to be replicated, in seconds.
weight (added in 2.0)	no			Weighted resource record sets only. Among resource record sets that have the same combination of DNS name and type, a value that determines what portion of traffic for the current resource record set is routed to the associated location.
zone	yes			The hosted DNS zone to modify

For this module to work, the request timestamp from Ansible host must be within 300 seconds of Route 53 server. Install NTP on the Ansible host to synchronize the system clock with a public NTP server. Change the host timezone to UTC:

```
# ln -sf /usr/share/zoneinfo/UTC /etc/localtime
```

The following is a playbook to create a DNS A record:

```
$ cd /home/yan/ansible4aws$ vi route53.yml
$ vi route53.yml

---
- hosts: localhost
  connection: local
  gather_facts: no
  tasks:
    - name: create record
      route53:
        command: create
        zone: yankurniawan.com.
        record: yankurniawan.com.
        type: A
```

```
# Public IP address of wordpress server from chapter 4  
value: 54.79.34.239
```

To change the record setting, we need to specify `overwrite: yes`, for example to change the TTL value from the default 3600s to 7200s:

```
---  
- hosts: localhost  
  connection: local  
  gather_facts: no  
  tasks:  
    - name: modify record  
      route53:  
        command: create  
        zone: yankurniawan.com.  
        record: yankurniawan.com.  
        type: A  
        ttl: 7200  
        overwrite: yes  
        # Public IP address of wordpress server from chapter 4  
        value: 54.79.34.239
```

To delete the record:

```
---  
- hosts: localhost  
  connection: local  
  gather_facts: no  
  tasks:  
    - name: delete record  
      route53:  
        command: delete  
        zone: yankurniawan.com.  
        record: yankurniawan.com.  
        type: A  
        value: 54.79.34.239
```

6

VPC Provisioning with Ansible

Amazon **Virtual Private Cloud** (Amazon VPC) enables you to launch **Amazon Web Services (AWS)** resources into a virtual network that you've defined. This virtual network closely resembles a traditional network that you'd operate in your own data center, with the benefits of using the scalable infrastructure of AWS. For more information, visit http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Introduction.html.

Using Ansible for Amazon VPC provisioning allows you to manage your AWS infrastructure as a code base. It means that you are writing code to define your infrastructure. By applying **Version Control System (VCS)** like git, your infrastructure is subject to version control. You will be able to easily recreate your whole infrastructure, revert back to the previous version of your infrastructure, and setup a consistent development, testing, and production environment.

In this chapter, you will learn about AWS VPC basics and how to use Ansible to provision AWS VPC; including VPC subnets, VPC routing tables, and VPC security groups. You will learn how to use Ansible to launch EC2 instances in VPC subnet and attach VPC security groups to the instances. I will also show you how to provision NAT instance in your VPC.

The default VPC

When you launch an EC2 instance without creating and specifying a non-default VPC, Amazon launches the instance into your default VPC. All examples from the previous chapters launched EC2 instances into the default VPC. You can see the default VPC on your Amazon VPC console <https://console.aws.amazon.com/vpc>.

Amazon creates a default VPC with the following set up:

- A default subnet in each Availability Zone
- An Internet gateway connected to your default VPC
- The main route table for your default VPC with a rule that sends all traffic destined for the Internet to the Internet gateway
- A default security group associated with your default VPC
- A default network **Access Control List (ACL)** associated with your VPC
- A default DHCP options set for your AWS account associated with your default VPC

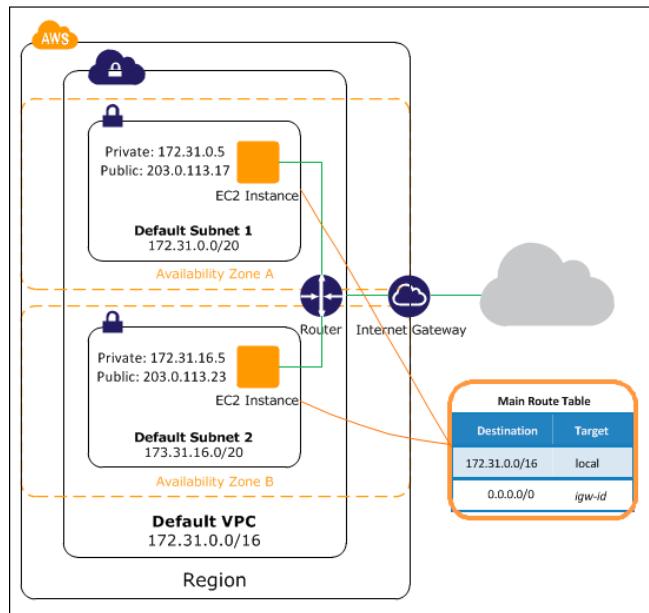


Amazon has multiple locations worldwide. These locations are composed of regions and Availability Zones. Each region is a separate geographic area. Each region has multiple, isolated locations known as **Availability Zones**. Amazon provides you the ability to place resources, such as instances, and data in multiple locations. Resources aren't replicated across regions unless you do so specifically.



A *network access control list* is an optional layer of security that acts as a firewall for controlling traffic in and out of a subnet. You might set up network ACLs with rules similar to your security groups in order to add an additional layer of security to your VPC.

The following figure illustrates the key components that Amazon set up for a default VPC (Refer <http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/default-vpc.html> for more information):



The CIDR block for a default VPC is always 172.31.0.0/16, which provides up to 65,536 private IP addresses. A default subnet has a /20 subnet mask, which provides up to 4,096 addresses per subnet. Some addresses are reserved for Amazon's use.

By default, a default subnet is connected to the Internet through the Internet gateway. Instances that you launch into a default subnet receive both a private IP address and a public IP address.

Getting started with VPC

A VPC is a virtual network dedicated to your AWS account. It is logically isolated from other virtual networks in the AWS cloud. You can launch your AWS resources, such as Amazon EC2 instances, into your VPC. You can visit http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Subnets.html for more information.

When you create a VPC, you specify the range of IP addresses for the VPC in the form of a **Classless Inter-Domain Routing (CIDR)** block; for example, 10.0.0.0/16. For more information about CIDR notation and what /16 means, go to https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing.

You must use a private address space for the VPC CIDR block. For more information about private address, go to https://en.wikipedia.org/wiki/Private_network.

You can create a VPC that spans multiple Availability Zones. After creating a VPC, you can add one or more subnets in each Availability Zone. When you create a subnet, you specify the CIDR block for the subnet, which is a subset of the VPC CIDR block. Each subnet must reside entirely within one Availability Zone and cannot span zones. Availability Zones are distinct locations that are engineered to be isolated from failures in other Availability Zones. By launching instances in separate Availability Zones, you can protect your applications from the failure of a single location.

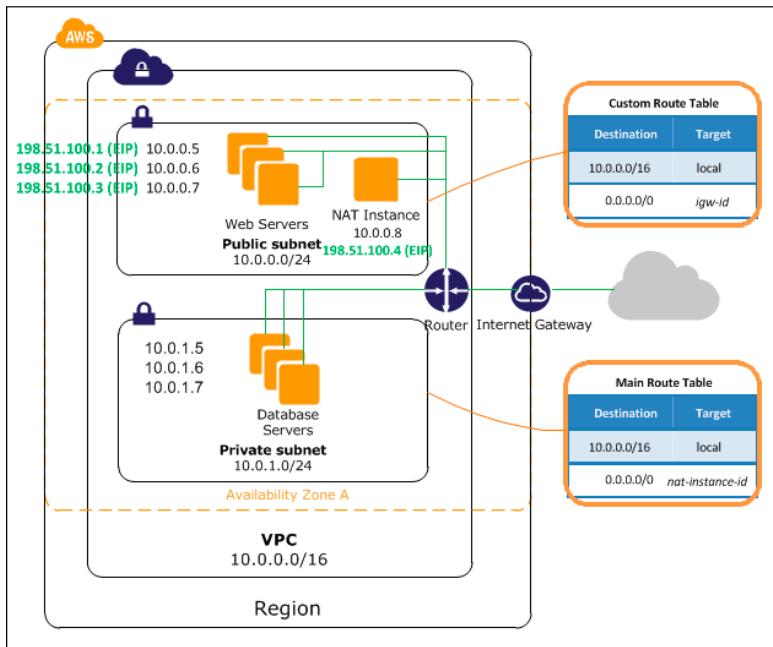
For more information about VPC and subnet sizing go to http://docs.aws.amazon.com/AWSVPC/latest/UserGuide/VPC_Subnets.html.

There are many tools available to help you calculate subnet CIDR blocks; for example, see <http://www.subnet-calculator.com/cidr.php>.

To get started with Amazon VPC, let's create a VPC and subnets from the VPC console. If you have done this before, you can skip this section. We will create a VPC for a multi-tier website scenario, with the web servers in a public subnet and the database servers in a private subnet. This scenario will be used for all examples in this chapter.

The instances in the public subnet can receive inbound traffic directly from the Internet, whereas the instances in the private subnet can't. The instances in the public subnet can send outbound traffic directly to the Internet, whereas the instances in the private subnet can't. Instead, the instances in the private subnet can access the Internet by using a **Network Address Translation (NAT)** instance that you launch into the public subnet. AWS also provides managed NAT gateway service you can launch in your VPC, but for this book I will not use that. For your production environment you will definitely need to use AWS managed NAT service to provide better availability and higher bandwidth.

The following diagram shows the key components of the configuration for the scenario (Refer http://docs.aws.amazon.com/AWSVPC/latest/UserGuide/VPC_Scenario2.html):



VPC Diagram

VPC Sizing

The allowed CIDR block size for a VPC is between a /28 netmask and /16 netmask, therefore the VPC can contain from 16 to 65,536 IP addresses.



You can't change the size of a VPC after you create it. If your VPC is too small to meet your needs, you must terminate all the instances in the VPC, delete the VPC, and then create a new, larger VPC.



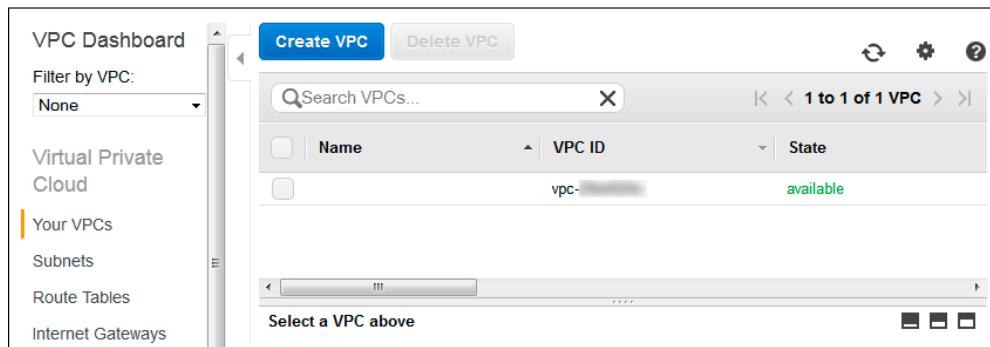
The following list describes the basic components presented in the configuration diagram for this scenario:

- A virtual private cloud (VPC) of size /16 (CIDR: 10.0.0.0/16). This provides 65,536 private IP addresses.
- A public subnet of size /24 (CIDR: 10.0.0.0/24). This provides 256 private IP addresses.
- A private subnet of size /24 (CIDR: 10.0.1.0/24). This provides 256 private IP addresses.

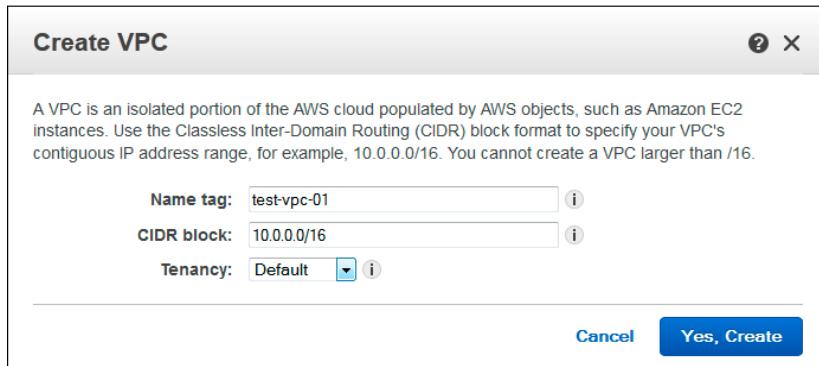
- An Internet gateway. This connects the VPC to the Internet and to other AWS products, such as Amazon **Simple Storage Service** (Amazon S3).
- Instances with private IP addresses in the subnet range (examples: 10.0.0.5, 10.0.1.5), which enables them to communicate with each other and other instances in the VPC. Instances in the public subnet also have Elastic IP addresses (example: 198.51.100.1), which enable them to be reached from the Internet. Instances in the private subnet are backend servers that don't need to accept incoming traffic from the Internet; however, they can send requests to the Internet using the NAT instance.
- A NAT instance with its own Elastic IP address. This enables instances in the private subnet to send requests to the Internet (for example, for software updates).
- A custom route table associated with the public subnet. This route table contains an entry that enables instances in the subnet to communicate with other instances in the VPC, and an entry that enables instances in the subnet to communicate directly with the Internet.
- The main route table associated with the private subnet. The route table contains an entry that enables instances in the subnet to communicate with other instances in the VPC, and an entry that enables instances in the subnet to communicate with the Internet through the NAT instance.

To create a VPC:

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, click **Your VPCs**.



3. Click **Create VPC**.
4. Enter the desired **Name tag** and **CIDR block** for the VPC and select **Tenancy:Default**.



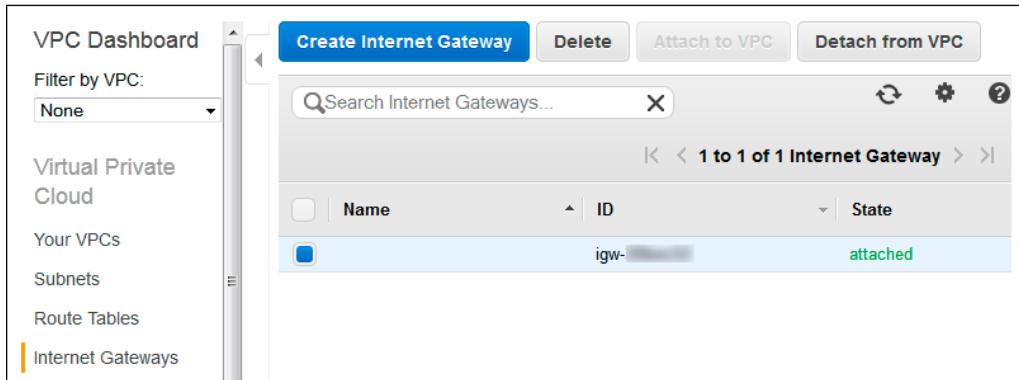
5. Click **Yes, Create** button.

Each VPC has a related instance tenancy attribute. You can't change the instance tenancy of a VPC after you create it. A dedicated VPC tenancy attribute means that all instances launched into the VPC are Dedicated Instances, regardless of the value of the tenancy attribute for the instance. If you set this to default, then the tenancy attribute will follow the tenancy attribute setting on each instance. A default tenancy instance runs on shared hardware. A dedicated tenancy instance runs on single-tenant hardware. Dedicated Instances are Amazon EC2 instances that run in a **Virtual Private Cloud (VPC)** on hardware that's dedicated to a single customer. Your Dedicated Instances are physically isolated at the host hardware level from your instances that aren't Dedicated Instances and from instances that belong to other AWS accounts. To see the pricing for dedicated instances go to <http://aws.amazon.com/ec2/purchasing-options/dedicated-instances/>.

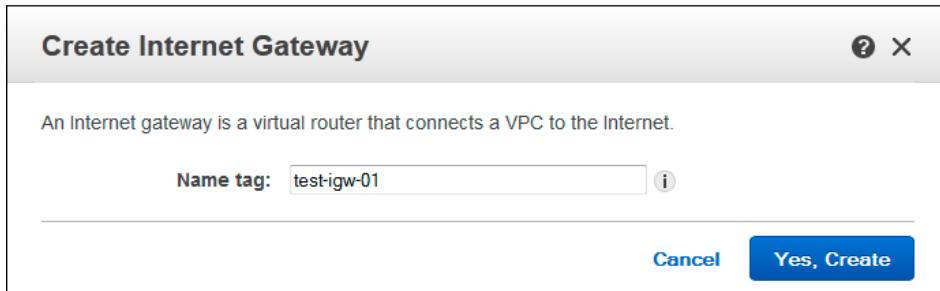


To create an Internet gateway and attach it to a VPC:

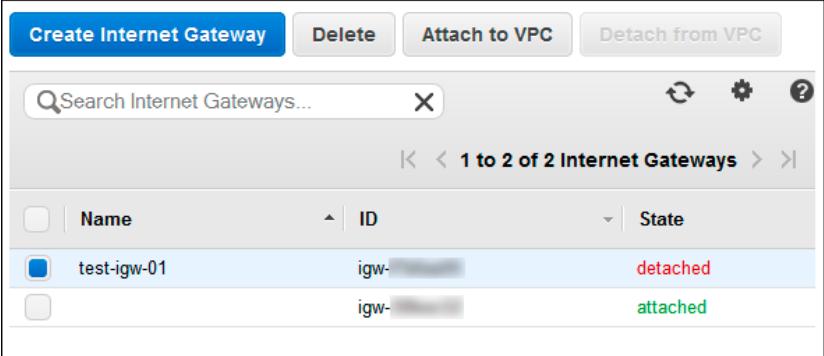
1. In the VPC Dashboard navigation pane, click **Internet Gateways**.



2. Click **Create Internet Gateway**.
3. Enter the desired **Name tag** for this Internet gateway.



4. Click **Yes, Create** button.
5. Select the new Internet gateway and click **Attach to VPC**.



Internet Gateways		
<input type="checkbox"/>	Name	ID
Actions		
<input checked="" type="checkbox"/>	test-igw-01	igw- [REDACTED]
<input type="checkbox"/>		igw- [REDACTED]

6. Select the desired VPC and click **Yes, Attach**.



Attach to VPC

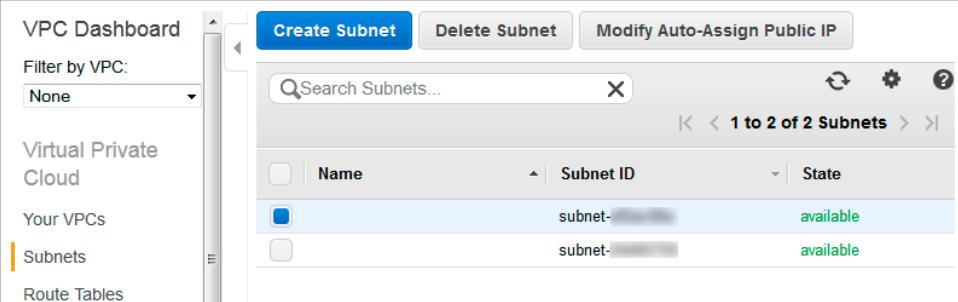
Attach an Internet gateway to a VPC to enable communication with the Internet.

VPC: vpc- [REDACTED] (10.0.0.0/16) | test-vpc-01

Cancel Yes, Attach

To create subnets on the VPC:

1. In the VPC Dashboard navigation pane, click **Subnets**.



Subnets		
<input type="checkbox"/>	Name	Subnet ID
Actions		
<input checked="" type="checkbox"/>	subnet- [REDACTED]	available
<input type="checkbox"/>		available

2. Click **Create Subnet**.
3. Enter the desired **Name tag** and **CIDR block** for the public subnet, select **VPC** and **Availability Zone**.

Create Subnet

Use the CIDR format to specify your subnet's IP address block (e.g., 10.0.0.0/24). Note that block sizes must be between a /16 netmask and /28 netmask. Also, note that a subnet can be the same size as your VPC.

Name tag: test-subnet-public [i](#)

VPC: vpc- [REDACTED] (10.0.0.0/16) | test-vpc-01 [i](#)

Availability Zone: ap-southeast-2a [i](#)

CIDR block: 10.0.0.0/24 [i](#)

[Cancel](#) [Yes, Create](#)

4. Click **Yes, Create**.
5. Click **Create Subnet**. Enter the desired **Name tag** and **CIDR block** for the private subnet, select **VPC** and **Availability Zone**.

Create Subnet

Use the CIDR format to specify your subnet's IP address block (e.g., 10.0.0.0/24). Note that block sizes must be between a /16 netmask and /28 netmask. Also, note that a subnet can be the same size as your VPC.

Name tag: test-subnet-private [i](#)

VPC: vpc- [REDACTED] (10.0.0.0/16) | test-vpc-01 [i](#)

Availability Zone: ap-southeast-2a [i](#)

CIDR block: 10.0.1.0/24 [i](#)

[Cancel](#) [Yes, Create](#)

6. Click **Yes, Create**.

Route tables

The following are the basic things that you need to know about route tables (For more information, please visit

http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Route_Tables.html:

- Your VPC has an implicit router.
- Your VPC automatically comes with the main route table that you can modify.
- You can create additional custom route tables for your VPC.
- Each subnet must be associated with a route table, which controls the routing for the subnet. If you don't explicitly associate a subnet with a particular route table, the subnet uses the main route table.
- You can replace the main route table with a custom table that you've created (so that this table is the default table each new subnet is associated with).
- Each route in a table specifies a destination CIDR and a target.

Main route table

When you create a VPC, it automatically has the main route table. To see the main route tables for your VPC, in the VPC Dashboard navigation pane, click **Route Tables**.

The screenshot shows the AWS VPC Dashboard. On the left, there's a sidebar with navigation links: VPC Dashboard, Filter by VPC (set to None), Virtual Private Cloud, Your VPCs, Subnets, Route Tables (which is selected and highlighted in orange), Internet Gateways, DHCP Options Sets, Elastic IPs, and Peering Connections. At the top right, there are buttons for Create Route Table, Delete Route Table, Set As Main Table, and other options. The main area displays a table for route tables. A search bar at the top of the table area contains the text 'rtb-'. The table has columns: Name, Route Table ID, Associated With, and Subnets. One row is visible, showing 'rtb-' in all four columns. Below the table, there are tabs for Summary, Routes (which is selected and highlighted in orange), Subnet Associations, Route Propagation, and Tags. Under the 'Routes' tab, there's a sub-table titled 'Edit' with columns: Destination, Target, Status, and Propagated. One row is shown: '10.0.0.0/16' with 'local' as the target, 'Active' as the status, and 'No' as the propagation status.

The main route table

Initially, the main route table (and every route table in a VPC) contains only a single route: a local route that enables communication within the VPC. You can't modify the local route in a route table. Whenever you launch an instance in the VPC, the local route automatically covers that instance; you don't need to add the new instance to a route table.

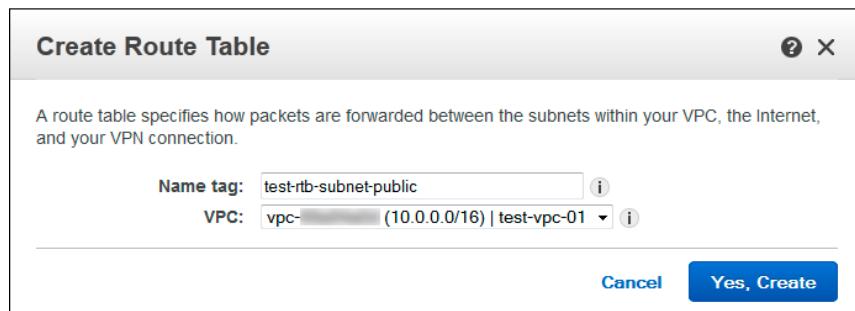
If you don't explicitly associate a subnet with a route table, the subnet is implicitly associated with the main route table. However, you can still explicitly associate a subnet with the main route table. You might do that if you change which table is the main route table. The console shows the number of subnets associated with each table. Only explicit associations are included in that number.

For the scenario in this section, the main route tables should contain an entry that enables instances in the private subnet to communicate with the Internet through the NAT instance, but I'm not going to provide instructions for manually creating a NAT instance here. If you want to learn about this, follow the instruction here: http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_NAT_Instance.html and then you could edit the main route table and add an entry: destination 0.0.0.0/0, target nat-instance-id. I will show you later in this chapter how to provision a NAT instance using Ansible.

Custom route tables

To create a custom route table for the public VPC:

1. In the VPC Dashboard navigation pane, click **Route Tables**.
2. Click **Create Route Table**.
3. Enter the desired **Name tag** and select **VPC**.



4. Click **Yes, Create**.

To edit the custom route table and associate it to the public subnet:

1. In the VPC Dashboard navigation pane, click **Route Tables**. Select the custom route table we created for public subnet and select the **Routes** tab.

Name	Route Table ID	Associated With	Main
test-rtb-subnet-public	rtb-[REDACTED]	0 Subnets	No
	rtb-[REDACTED]	0 Subnets	Yes
	rtb-[REDACTED]	0 Subnets	Yes

rtb-[REDACTED] | test-rtb-subnet-public

Summary Routes Subnet Associations Route Propagation Tags

Edit

Destination	Target	Status	Propagated
10.0.0.0/16	local	Active	No

2. Click **Edit** button.
3. Enter 0.0.0.0/0 CIDR block in the **Destination** field and select the Internet gateway from the **Target** list.

Destination	Target	Status	Propagated	Remove
10.0.0.0/16	local	Active	No	
0.0.0.0/0	igw-[REDACTED]	No		X

Add another route

4. Click **Save**.

5. Select **Subnet Associations** tab.

The screenshot shows the AWS Route Table configuration for 'rtb- [REDACTED] | test-rtb-subnet-public'. The 'Subnet Associations' tab is selected. A blue 'Edit' button is visible. Below it, a message states: 'You do not have any subnet associations.' Another message below says: 'The following subnets have not been associated with any route tables and are therefore using the main table routes:' A table lists two subnets:

Subnet	CIDR
subnet- [REDACTED] (10.0.0.0/24) test-subnet-public	10.0.0.0/24
subnet- [REDACTED] (10.0.1.0/24) test-subnet-private	10.0.1.0/24

6. Click **Edit**.
7. Select the **Associate** check box for public subnet and click **Save**.

The screenshot shows the 'Subnet Associations' tab for the same route table. The 'Save' button is now highlighted in blue. A table shows the association status:

Associate	Subnet	CIDR	Current Route Table
<input checked="" type="checkbox"/>	subnet- [REDACTED] (10.0.0.0/24) test-subnet-public	10.0.0.0/24	Main
<input type="checkbox"/>	subnet- [REDACTED] (10.0.1.0/24) test-subnet-private	10.0.1.0/24	Main

VPC provisioning

We will use Ansible `ec2_vpc` module to create or terminate AWS VPC. Options for this module are listed in the following table:

parameter	required	default	choices	comments
aws_access_key	no			AWS access key. If not set then the value of the AWS_ACCESS_KEY environment variable is used aliases: ec2_access_key, access_key
aws_secret_key	no			AWS secret key. If not set then the value of the AWS_SECRET_KEY environment variable is used aliases: ec2_secret_key, secret_key
cidr_block	yes			The cidr block representing the VPC, e.g. 10.0.0.0/16, required when state is 'present'
dns_hostnames	no	yes	yes no	Toggles the "Enable DNS hostname support for instances" flag
dns_support	no	yes	yes no	Toggles the "Enable DNS resolution" flag
ec2_url	no			URL to use to connect to EC2 or your Eucalyptus cloud (by default the module will use EC2 endpoints). Must be specified if region is not used. If not set then the value of the EC2_URL environment variable, if any, is used
instance_tenancy	no	default	default dedicated	The supported tenancy options for instances launched into the VPC
internet_gateway	no	no	yes no	Toggles whether there should be an Internet gateway attached to the VPC
profile (added in 1.6)	no			Uses a boto profile. Only works with boto >= 2.24.0
region	no			Region in which the resource exists
resource_tags	yes			A dictionary array of resource tags of the form: { tag1: value1, tag2: value2 }. Tags in this list are used in conjunction with CIDR block to uniquely identify a VPC in lieu of vpc_id. Therefore, if CIDR/Tag combination does not exist, a new VPC will be created. VPC tags not on this list will be ignored.
route_tables	no			A dictionary array of route tables to add of the form: { subnets: [172.22.2.0/24, routes:

parameter	required	default	choices	comments
				[{ dest: 0.0.0.0/0, gw: igw},] }. Where the subnets list is those subnets the route table should be associated with, and the routes list is a list of routes to be in the table. The special keyword for the gw of igw specifies that the route should go through the internet gateway attached to the VPC. gw also accepts instance-ids. This module is currently unable to affect the “main” route table due to some limitations in boto, so you must explicitly define the associated subnets or they will be attached to the main table implicitly.
security_token (added in 1.6)	no			AWS STS security token. If not set then the value of the AWS_SECURITY_TOKEN or EC2_SECURITY_TOKEN environment variable is used. aliases: access_token
state	yes	present		Create or terminate the VPC
subnets	no			A dictionary array of subnets to add of the form { cidr: ... , az: ... , resource_tags: ... } . Where az is the desired availability zone of the subnet, but it is not required. All VPC subnets not in this list will be removed
validate_certs	no	yes	yes no	When set to “no”, SSL certificates will not be validated for boto versions >= 2.6.0. (added in Ansible 1.5)
vpc_id	no			A VPC id to terminate when state=absent
wait	no	no	yes no	Wait for the VPC to be in state ‘available’ before returning
wait_timeout	no	300		How long before wait gives up, in seconds

The following playbook will show you how to create VPC, subnets, and route tables, using the same public and private subnets scenario as used in preceding section.

```
$ cd /home/yan/ansible4aws$ vi vpc_create.yml

---
- hosts: localhost
  connection: local
```

```
gather_facts: no
vars:
    region: ap-southeast-2
    # prefix for naming
    prefix: staging
    # availability zone
    az: ap-southeast-2a
tasks:
- name: create vpc
  ec2_vpc:
    region: "{{ region }}"
    cidr_block: 10.0.0.0/16
    resource_tags: '{"Name": "{{ prefix }}vpc"}'
    subnets:
      - cidr: 10.0.0.0/24
        az: "{{ az }}"
        resourcetags: '{"Name": "{{ prefix }}subnetpublic"}'
      - cidr: 10.0.1.0/24
        az: "{{ az }}"
        resource_tags: '{"Name": "{{ prefix }}subnetprivate"}'
    internet_gateway: yes
    route_tables:
      - subnets:
          - 10.0.0.0/24
        routes:
          - dest: 0.0.0.0/0
            gw: igw
    register: vpc
- name: write vpc id to {{ prefix }}vpcinfo file
  shell: echo "{{ prefix }}vpc:" "{{ vpc.vpcid }}"
    > "{{ prefix }}vpcinfo
- name: write subnets id to {{ prefix }}vpcinfo file
  shell: echo "{{ item.resource_tags.Name }}:" "{{ item.id }}"
    >> "{{ prefix }}vpcinfo
  with_items: vpc.subnets ~~
```

Run the playbook:

```
$ ansible-playbook -i hosts vpc_create.yml
```

The playbook will create a VPC with resource tags Name=staging_vpc and 2 subnets with resource tags Name=staging_subnet_public and Name=staging_subnet_private. You could easily create a duplicate VPC with its subnets, route tables, and so on simply by changing the prefix variable and run the playbook again.

You should see the `staging_vpc` created and listed on the VPC console. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/> and select **Your VPCs** in the navigation pane.

The screenshot shows the AWS VPC Dashboard. On the left, there's a sidebar with options: VPC Dashboard, Filter by VPC (set to None), Virtual Private Cloud, Your VPCs (selected), Subnets, Route Tables, Internet Gateways, and DHCP Options Sets. The main area has a search bar 'Search VPCs...' and a table with columns: Name, VPC ID, State, and VPC CIDR. The table contains three rows:

Name	VPC ID	State	VPC CIDR
vpc-[REDACTED]	vpc-[REDACTED]	available	172.31.0.0/16
staging_vpc	vpc-[REDACTED]	available	10.0.0.0/16
test-vpc-01	vpc-[REDACTED]	available	10.0.0.0/16

You should also see new staging subnets on the **Subnets** list:

The screenshot shows the AWS VPC Dashboard. On the left, there's a sidebar with options: VPC Dashboard, Filter by VPC (set to None), Virtual Private Cloud, Your VPCs, Subnets (selected), Route Tables, Internet Gateways, and DHCP Options Sets. The main area has a search bar 'Search Subnets...' and a table with columns: Name, Subnet ID, State, and VPC. The table contains six rows:

Name	Subnet ID	State	VPC
subnet-[REDACTED]	subnet-[REDACTED]	available	vpc-[REDACTED]
subnet-[REDACTED]	subnet-[REDACTED]	available	vpc-[REDACTED]
staging_subnet_private	subnet-[REDACTED]	available	vpc-[REDACTED]
staging_subnet_public	subnet-[REDACTED]	available	vpc-[REDACTED]
test-subnet-private	subnet-[REDACTED]	available	vpc-[REDACTED]
test-subnet-public	subnet-[REDACTED]	available	vpc-[REDACTED]

You can see from the preceding playbook that I registered the output of `ec2_vpc` module then wrote the VPC id and subnets id to a file called `staging_vpc_info`.

The content of `staging_vpc_info` file should look like this:

```
staging_vpc: vpc-xxxxxxxxxx
staging_subnet_public: subnet-xxxxxxxxxx
staging_subnet_private: subnet-xxxxxxxxxx
```

We can use the `staging_vpc_info` file as a variables file for another playbook. For example, the following is a playbook to delete the VPC we have created with the preceding playbook.

```
$ vi vpc_delete.yml
---
- hosts: localhost
  connection: local
  gather_facts: no
  vars:
    region: ap-southeast-2
  vars_files:
    - staging_vpc_info
  tasks:
    - name: delete vpc
      ec2_vpc:
        region: "{{ region }}"
        state: absent
        # get the vpc_id from staging_vpc_info file
        vpc_id: "{{ staging_vpc }}"
        wait: yes
```

You could run the `vpc_delete.yml` playbook to delete the `staging_vpc` you have created with the `vpc_create.yml` playbook. Simply re-run the `vpc_create.yml` playbook to create new `staging_vpc` after the deletion.



When you delete a VPC, Amazon deletes all its components, such as subnets, security groups, network ACLs, route tables, Internet gateways, VPC peering connections, and DHCP options. If you have instances launched in the VPC, you have to terminate all instances in the VPC first before deleting the VPC.

VPC security groups

We have learned about security groups provisioning in [Chapter 3, EC2 Provisioning and Configuration Management with Ansible Python](#). A *security group* acts as a virtual firewall for your instance to control inbound and outbound traffic. When you launch an instance in a VPC, you can assign the instance to up to five security groups. Security groups act at the instance level, not the subnet level. Therefore, each instance in a subnet in your VPC could be assigned to a different set of security groups. If you don't specify a particular group at launch time, the instance is automatically assigned to the default security group for the VPC (For more information regarding it please visit http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_SecurityGroups.html).

The following are the basic characteristics of security groups for your VPC:

- You can create up to 100 security groups per VPC. You can add up to 50 rules to each security group. If you need to apply more than 50 rules to an instance, you can associate up to 5 security groups with each network interface.
- You can specify allow rules, but not deny rules.
- You can specify separate rules for inbound and outbound traffic.
- By default, no inbound traffic is allowed until you add inbound rules to the security group.
- By default, all outbound traffic is allowed until you add outbound rules to the group (and then, you specify the outbound traffic that's allowed).
- Responses to allowed inbound traffic are allowed to flow outbound regardless of outbound rules, and vice versa (security groups are therefore stateful).
- Instances associated with a security group can't talk to each other unless you add rules allowing it (exception: the default security group has these rules by default).
- After you launch an instance, you can change which security groups the instance is associated with.

To create, modify, or delete security groups in a VPC, we'll use the `ec2_group` module like the one we used in [Chapter 3, EC2 Provisioning and Configuration Management with Ansible Python](#). To see the options for this module, please refer to the table in *Security Groups* section of that chapter. We need to add the `vpc_id` option to specify in which VPC we want to create/modify/delete the security group.

We will create security groups for web servers in public subnet, database servers in private subnet, and NAT instance in public subnet:

- Webservers security group will allow the web servers to receive Internet traffic (TCP port 80 and 443), SSH (TCP port 22) from your computer's or network's public IP address, and allow MySQL database access (TCP port 3306) to database servers group
- Database security group will allow the web servers group to access MySQL database, and allow outbound HTTP and HTTPS access to the Internet
- NAT security group will allow the NAT instance to receive inbound HTTP and HTTPS traffic from private subnet, allow SSH from your computer's or network's public IP address, and allow outbound HTTP and HTTPS access to the Internet

First, we will create a playbook to provision the security groups with empty rules. This playbook will be useful to remove dependencies from the security groups. Later on, if you want to delete the security groups run this playbook first to empty the rules, so the deletion

won't produce dependency error.

```
$ cd /home/yan/ansible4aws$ vi sg_empty.yml

---
- hosts: localhost
  connection: local
  gather_facts: no
  vars_files:
    - staging_vpc_info
  vars:
    #your region
    region: ap-southeast-2      #prefix for naming
    prefix: staging
    vpc_id: "{{ staging_vpc }}"
  tasks:
    - name: create empty security group for webservers
      ec2_group:
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        name: "{{ prefix }}sgweb"
        description: security group for webservers
    - name: create empty security group for databases
      ec2_group:
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        name: "{{ prefix }}sgdatabase"
        description: security group for databases
    - name: create empty security group for nat
      ec2_group:
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        name: "{{ prefix }}sgnat"
        description: security group for nat
```

Run the playbook:

```
$ ansible-playbook -i hosts sg_empty.yml
```

The following security groups should be created in your VPC with empty rules:
staging_sg_web, staging_sg_database, and staging_sg_nat. You can see the
security groups on your VPC console <https://console.aws.amazon.com/vpc/>, select
Security Groups in the navigation pane.

We will modify the rules using another playbook:

```
$ vi sg_modify.yml
```

```
---- - hosts: localhost
  connection: local
  gather_facts: no
  vars_files:
    - staging_vpc_info
  vars:
    #your region
    region: ap-southeast-2
    #your ip address
    allowed_ip: 123.xxx.xxx.xxx/32
    #prefix for naming
    prefix: staging
    vpc_id: "{{ staging_vpc }}"
    private_subnet: 10.0.1.0/24
  tasks:
    - name: modify sg_web rules
      ec2_group:
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        #your security group name
        name: "{{ prefix }}sgweb"
        description: security group for webservers
        rules:
          # allow ssh access from your ip address
          - proto: tcp
            from_port: 22
            to_port: 22
            cidr_ip: "{{ allowed_ip }}"
          # allow http access from anywhere
          - proto: tcp
            from_port: 80
            to_port: 80
            cidr_ip: 0.0.0.0/0
          # allow https access from anywhere
          - proto: tcp
            from_port: 443
            to_port: 443
            cidr_ip: 0.0.0.0/0
        rules_egress:
          - proto: tcp
            from_port: 3306
            to_port: 3306
            group_name: "{{ prefix }}sgdatabase"
    - name: modify sg_database rules
      ec2_group:
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        name: "{{ prefix }}sgdatabase"
        description: security group for databases
        rules:
```

```
- proto: tcp
  from_port: 3306
  to_port: 3306
  group_name: "{{ prefix }}sgweb"
rules_egress:
- proto: tcp
  from_port: 80
  to_port: 80
  cidr_ip: 0.0.0.0/0
- proto: tcp
  from_port: 443
  to_port: 443
  cidr_ip: 0.0.0.0/0
- name: modify sg_nat rules
ec2_group:
  region: "{{ region }}"
  vpc_id: "{{ vpc_id }}"
  name: "{{ prefix }}sgnat"
  description: security group for nat
  rules:
    # allow ssh access from your ip address
    - proto: tcp
      from_port: 22
      to_port: 22
      cidr_ip: "{{ allowed_ip }}"
    # allow http access from private subnet
    - proto: tcp
      from_port: 80
      to_port: 80
      cidr_ip: "{{ private_subnet }}"
    # allow https access from private subnet
    - proto: tcp
      from_port: 443
      to_port: 443
      cidr_ip: "{{ private_subnet }}"
  rules_egress:
    - proto: tcp
      from_port: 80
      to_port: 80
      cidr_ip: 0.0.0.0/0
    - proto: tcp
      from_port: 443
      to_port: 443
      cidr_ip: 0.0.0.0/0
```

Run the playbook and you can see the rules changed.

Now from the VPC console, try to delete `staging_sg_web`. Right click on the security group and select **Delete Security Group**, click **Yes, Delete**. It will tell you that you could not delete the security group because it has a dependent object, which is `staging_sg_database` in its outbound rules.

You could run the `sg_empty.yml` playbook to remove all rules from the security groups, then you could delete the security group without dependency issue.

You can use the following playbook to delete the security groups:

```
$ vi sg_delete.yml

---
- hosts: localhost
  connection: local
  gather_facts: no
  vars_files:
    - staging_vpc_info
  vars:
    #your region      region: ap-southeast-2
    #prefix for naming
    prefix: staging
    vpc_id: "{{ staging_vpc }}"
  tasks:
    - name: delete {{ prefix }}sgweb
      ec2_group:
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        name: "{{ prefix }}sgweb"
        description: security group for webservers
        state: absent
    - name: delete {{ prefix }}sgdatabase
      ec2_group:
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        name: "{{ prefix }}sgdatabase"
        description: security group for databases
        state: absent
    - name: delete {{ prefix }}sgnat
      ec2_group:
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        name: "{{ prefix }}sgnat"
        description: security group for nat
        state: absent
```

If you run `sg_delete.yml` playbook without deleting the security groups rules first, it will produce dependency error. You have to run `sg_empty.yml` first before deleting the security groups.

EC2-VPC provisioning

In Chapter 3, *EC2 Provisioning and Configuration Management with Ansible Python*, we used Ansible to launch EC2 instances without creating and specifying a non-default VPC, therefore the instances launched in the default VPC. In this chapter we have created a non-default VPC, subnets, and VPC security groups. To launch an instance in a particular subnet in your VPC using the Ansible `ec2` module, you need to specify the subnet id using the `vpc_subnet_id` option.

The following playbook will launch an EC2 instance for our web server in the public subnet of our VPC:

```
$ vi ec2_vpc_web_create.yml

---
- hosts: localhost
  connection: local
  gather_facts: no
  vars_files:
    - staging_vpc_info
  vars:
    region: ap-southeast-2
    key: yan-key-pair-apsydney
    instance_type: t2.micro
    image: ami-d9fe9be3
    prefix: staging
  tasks:
    - name: web instance provisioning
      ec2:
        region: "{{ region }}"
        key_name: "{{ key }}"
        instance_type: "{{ instance_type }}"
        image: "{{ image }}"
        wait: yes
        group: "{{ prefix }}sgweb"
        instance_tags:
          Name: "{{ prefix }}web"
          class: web
          environment: staging
        id: weblaunch_01
```

```
vpc_subnet_id: "{{ staging_subnet_public }}"
register: ec2
- name: associate new EIP for the instance
  ec2_eip:
    region: "{{ region }}"
    instance_id: "{{ item.id }}"
  with_items: ec2.instances
```

And the following playbook will launch an EC2 instance for our database server in the private subnet of our VPC, without assigning a public IP address:

```
$ vi ec2_vpc_db_create.yml

---
- hosts: localhost
  connection: local
  gather_facts: no
  vars_files:
    - staging_vpc_info
  vars:
    region: ap-southeast-2
    key: yan-key-pair-apsydne
    instance_type: t2.micro
    image: ami-d9fe9be3
    prefix: staging
  tasks:
    - name: database instance provisioning
      ec2:
        region: "{{ region }}"
        key_name: "{{ key }}"
        instance_type: "{{ instance_type }}"
        image: "{{ image }}"
        wait: yes
        group: "{{ prefix }}sgdatabase"
        instance_tags:
          Name: "{{ prefix }}database"
          class: database
          environment: staging
        id: dblaunch_01
        vpc_subnet_id: "{{ staging_subnet_private }}"
        assign_public_ip: no
```

NAT instance

Instances that you launch into a private subnet in a VPC can't communicate with the Internet. You can optionally use a NAT instance in a public subnet in your VPC to enable instances in the private subnet to initiate outbound traffic to the Internet, but prevent the instances from receiving inbound traffic initiated by someone on the Internet. For more information please visit http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_NAT_Instance.html.

Amazon provides Amazon Linux AMIs that are configured to run as NAT instances. These AMIs include the string `amzn-ami-vpc-nat` in their names, so you can search for them in the Amazon EC2 console.

To get the NAT AMI ID:

1. Open the Amazon EC2 console <https://console.aws.amazon.com/ec2>.
2. On the dashboard, click the **Launch Instance** button.
3. On the **Choose an Amazon Machine Image (AMI)** page, select the Community AMIs category, and search for `amzn-ami-vpc-nat`. In the results list, each AMI's name includes the version to enable you to select the most recent AMI, for example, `2013.09`.
4. Take a note of the AMI ID.

Step 1: Choose an Amazon Machine Image (AMI)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Quick Start	My AMIs	AWS Marketplace	Community AMIs	Operating system
				<input type="checkbox"/> Amazon Linux <input type="checkbox"/> Cent OS <input type="checkbox"/> Debian
				
				<input type="checkbox"/> Amazon Linux <input type="checkbox"/> Cent OS <input type="checkbox"/> Debian

Search: amzn-ami-vpc-nat

1 to 2 of 2 AMIs

 amzn-ami-vpc-nat-pv-2013.03.1.x86_64-ebs - ami-0154c73b	Select
Amazon Linux AMI VPC NAT x86_64 PV	64-bit
Root device type: ebs Virtualization type: paravirtual	
 amzn-ami-vpc-nat-pv-2013.09.0.x86_64-ebs - ami-3bae3201	Select
Amazon Linux AMI VPC NAT x86_64 PV	64-bit
Root device type: ebs Virtualization type: paravirtual	

Cancel and Exit

NAT AMI

This AMI is using paravirtual virtualization so it won't work with t2.micro instance type. We will use the t1.micro instance type.

The following playbook will launch a NAT instance in the public subnet of our VPC and associate an Elastic IP address to the instance.

```
$ vi nat_launch.yml

---

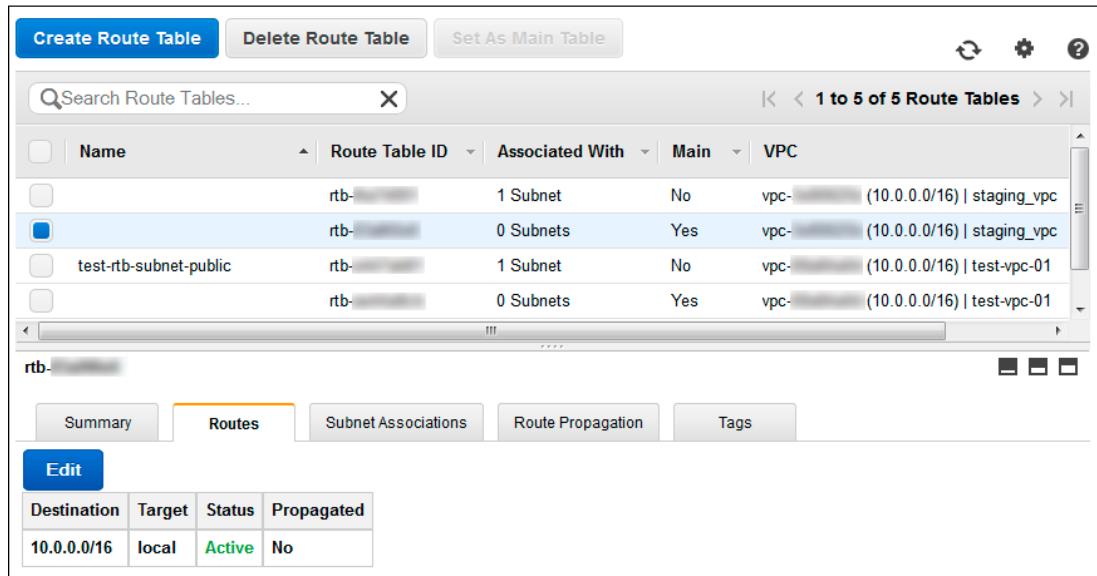
- hosts: localhost
  connection: local
  gather_facts: no
  vars_files:
    - staging_vpc_info
  vars:
    region: ap-southeast-2
    key: yan-key-pair-apsydne
    instance_type: t1.micro
    image: ami-3bae3201
    prefix: staging
  tasks:
    - name: NAT instance provisioning
      ec2:
        region: "{{ region }}"
        key_name: "{{ key }}"
        instance_type: "{{ instance_type }}"
        image: "{{ image }}"
        wait: yes
        group: "{{ prefix }}sgnat"
        instance_tags:
          Name: "{{ prefix }}nat"
          class: nat
          environment: staging
        id: natlaunch_01
        vpc_subnet_id: "{{ staging_subnet_public }}"
        source_dest_check: no
        wait: yes
        register: ec2
    - name: associate new EIP for the instance
      tags: eip
      ec2_eip:
        region: "{{ region }}"
        instance_id: "{{ item.id }}"
      with_items: ec2.instances
      when: item.id is defined
```

Run the playbook and check your AWS EC2 console. A new `staging_nat` instance should be created in `staging_subnet_public` subnet and associated with an EIP address.

Each EC2 instance performs source/destination checks by default. This means that the instance must be the source or destination of any traffic it sends or receives. However, a NAT instance must be able to send and receive traffic when the source or destination is not itself. Therefore, you must disable source/destination checks on the NAT instance. To do this, in the playbook we set the `ec2` module's option `source_dest_check: no`.

To allow instances in private subnet to connect to the Internet via the NAT instance, we must update the Main route tables. We need to do this from the AWS VPC console:

1. In the VPC console navigation pane select **Route tables**.
2. Select the Main route table of your `staging_vpc` VPC and select the **Routes** tab.



3. Click **Edit**.

4. Enter 0.0.0.0/0 CIDR block in the **Destination** field and select the staging_nat instance id from the **Target** list.

The screenshot shows the 'Routes' tab of a VPC route table configuration. There are two routes listed:

Destination	Target	Status	Propagated	Remove
10.0.0.0/16	local	Active	No	
0.0.0.0/0	i- [REDACTED]		No	X

Below the table is a button labeled 'Add another route'.

5. Click **Save**.

Now we have completed the VPC infrastructure provisioning using Ansible. At the time of writing, there is not yet a module for *Network ACLs* provisioning. If you want to add some Network ACLs for your VPC subnet, you could do it from the VPC console: select **Network ACLs** in the navigation pane and then click **Create Network ACLs**. You can find more information about ACLs here:

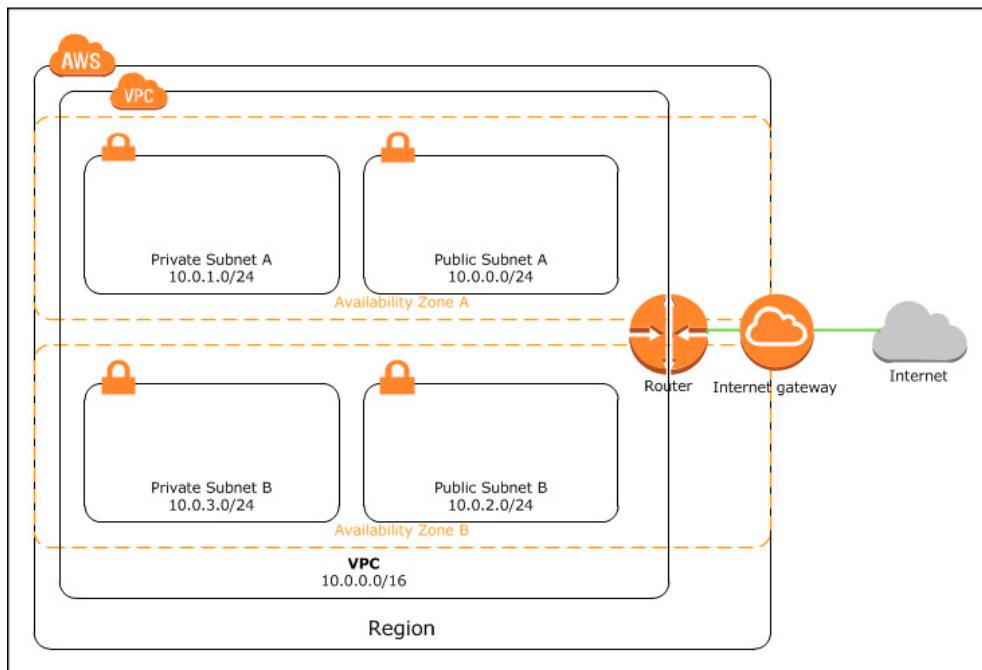
http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_ACLs.html.

If you have finished with the examples, you can terminate the EC2 instances to avoid any cost.

Multi-AZ deployment

You can span your Amazon VPC across multiple subnets in multiple Availability Zones inside a region. This will create a high availability system, adding redundancy to the system so that failure of a component does not mean failure of the entire system.

The following diagram shows a Multi-AZ version of our public and private subnets scenario. We'll add a public subnet and a private subnet in another availability zone within our region.



Multi-AZ VPC

We will use Ansible to provision our Multi-AZ VPC. First, we need to delete the staging_vpc VPC. You can use `vpc_delete.yml` to delete the VPC or you can delete the VPC from your AWS VPC console. Make sure you have terminated all EC2 instances in the VPC before deleting the VPC.

The following playbook will create a VPC with Multi-AZ subnets.

```
$ cd /home/yan/ansible4aws$ vi vpc_create_multi_az.yml
```

```
---
- hosts: localhost
  connection: local
  gather_facts: no
  vars:
    region: ap-southeast-2
    # prefix for naming
    prefix: staging
    # availability zones
    az0: ap-southeast-2a
    az1: ap-southeast-2b
```

```
tasks:
  - name: create vpc with multi-az subnets
    ec2_vpc:
      region: "{{ region }}"
      cidr_block: 10.0.0.0/16
      resource_tags: '{"Name":"'{{ prefix }}vpc"}'
      subnets:
        - cidr: 10.0.0.0/24
          az: "{{ az0 }}"
          resourcetags: '{"Name":"'{{ prefix }}subnetpublic_0"}'
        - cidr: 10.0.1.0/24
          az: "{{ az0 }}"
          resource_tags: '{"Name":"'{{ prefix }}subnetprivate_0"}'
        - cidr: 10.0.2.0/24
          az: "{{ az1 }}"
          resource_tags: '{"Name":"'{{ prefix }}subnetpublic_1"}'
        - cidr: 10.0.3.0/24
          az: "{{ az1 }}"
          resource_tags: '{"Name":"'{{ prefix }}subnetprivate_1"}'
      internet_gateway: yes
      route_tables:
        - subnets:
            - 10.0.0.0/24
            - 10.0.2.0/24
          routes:
            - dest: 0.0.0.0/0
              gw: igw
      register: vpc
  - name: write vpc id to {{ prefix }}vpcinfo file
    shell: echo "{{ prefix }}vpc:" "{{ vpc.vpcid }}"
           > "{{ prefix }}vpcinfo
  - name: write subnets id to {{ prefix }}vpcinfo file
    shell: echo "{{ item.resource_tags.Name }}:" "{{ item.id }}"
           >> "{{ prefix }}vpcinfo
  with_items: vpc.subnets
```

After running the playbook, a new VPC will be created, with 2 subnets in ap-southeast-2a zone, named staging_subnet_private_0 and staging_subnet_public_0, and 2 subnets in ap-southeast-2b zone, named staging_subnet_private_1 and staging_subnet_public_1.

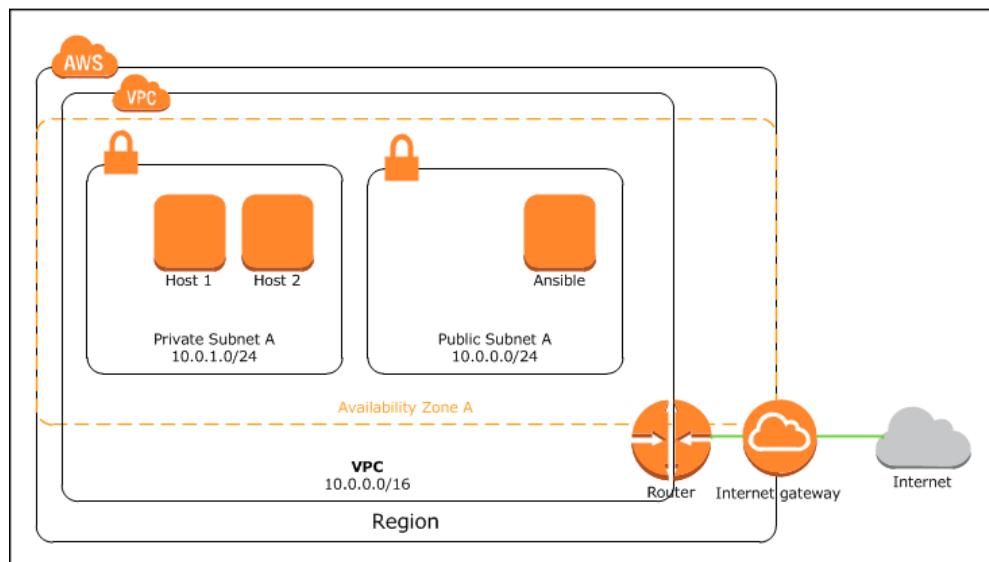
Run `sg_empty.yml` playbook, and then `sg_modify.yml` playbook to recreate our VPC security groups.

To achieve high availability, you can deploy your web application cluster in 2 (or more) availability zones (`staging_subnet_public_0` and `staging_subnet_public_1`), and distribute the load using Amazon **Elastic Load Balancing (ELB)**. For the database tier, you can use Amazon **Relational Database Service (RDS)**, deployed in 2 (or more) availability zones (`staging_subnet_private_0` and `staging_subnet_private_1`).

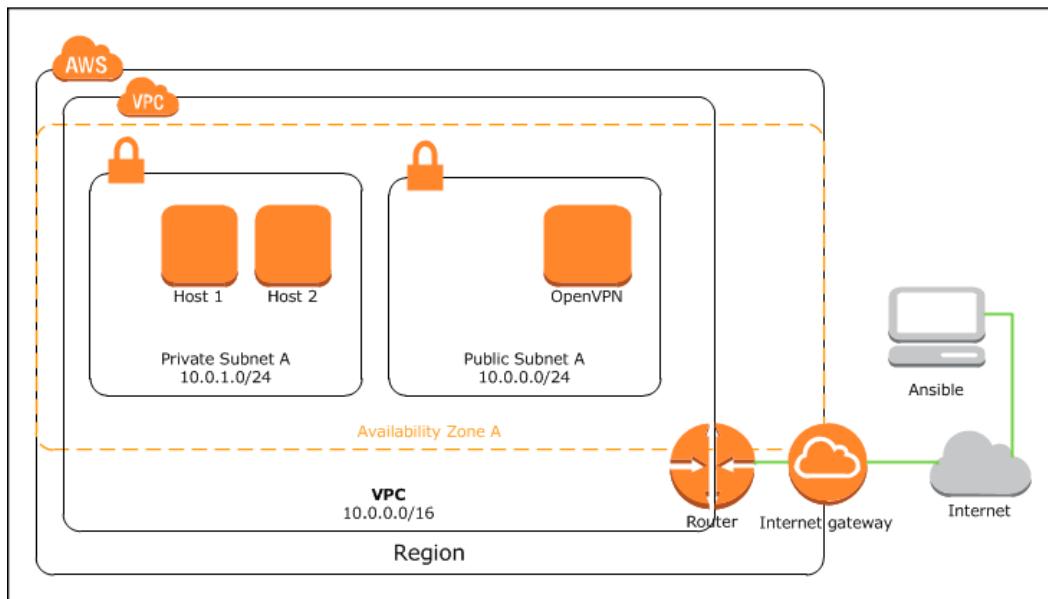
Ansible in VPC

Instances in private subnet of a VPC cannot directly receive inbound traffic from the internet. Therefore you can't use Ansible from the internet to manage the private server configuration. To use Ansible to manage configuration of servers in the private subnet of a VPC, we have 2 options:

- Install Ansible in an instance in public subnet of the VPC and allow SSH connection from the Ansible machine to the hosts to be managed in private subnet. This Ansible machine can also be used as a *jump box* to allow SSH access from the internet to hosts in private subnet (SSH to the Ansible machine first and then use the Ansible machine to SSH to hosts in private subnet).



- Create a **Virtual Private Network (VPN)** connection between Ansible machine (over the internet) and the private subnet. We can launch an OpenVPN server (available in AWS marketplace) instance in the public subnet which will allow Ansible machine to log in using OpenVPN client and connect via SSH to hosts in the private subnet.



We can use our current Ansible machine to launch a `jump_box` instance in the public subnet and install Ansible on the instance. First, we need to create a new security group for this instance.

The following playbook will create a new security group for the Ansible or `jump_box` instance:

```
$ cd /home/yan/ansible4aws
$ vi sg_jumpbox.yml

---
- hosts: localhost
  connection: local
  gather_facts: no
  vars_files:
    - staging_vpc_info
```

```
vars:
  #your region
  region: ap-southeast-2
  #your ip address
  allowed_ip: 123.xxx.xxx.xxx/32
  #prefix for naming
  prefix: staging
  vpc_id: "{{ staging_vpc }}"

tasks:
  - name: create security group for jump box instance
    ec2_group:
      region: "{{ region }}"
      vpc_id: "{{ vpc_id }}"
      #your security group name
      name: "{{ prefix }}sgjumpbox"
      description: security group for jump box
      rules:
        # allow ssh access from your ip address
        - proto: tcp
          from_port: 22
          to_port: 22
          cidr_ip: "{{ allowed_ip }}"
      rules_egress:
        - proto: all
          cidr_ip: 0.0.0.0/0
```

```
$ ansible-playbook -i hosts sg_jumpbox.yml
```

The following playbook will launch our jump box instance in public subnet A:

```
$ vi ec2_vpc_jumpbox.yml

---

- hosts: localhost
  connection: local
  gather_facts: no
  vars_files:
    - staging_vpc_info
  vars:
    region: ap-southeast-2
    key: yan-key-pair-apsydne
    instance_type: t2.micro
    image: ami-d9fe9be3
    prefix: staging
    vpc_subnet_id: "{{ staging_subnet_public_0 }}"
  tasks:
    - name: jump box instance provisioning
      ec2:
```

```
region: "{{ region }}"
key_name: "{{ key }}"
instance_type: "{{ instance_type }}"
image: "{{ image }}"
wait: yes
group: "{{ prefix }}sgjumpbox"
instance_tags:
  Name: "{{ prefix }}jumpbox"
  class: jumpbox
  environment: "{{ prefix }}"
id: jumpboxlaunch_01
vpc_subnet_id: "{{ vpc_subnet_id }}"
register: ec2
- name: associate new EIP for the instance
  ec2_eip:
    region: "{{ region }}"
    instance_id: "{{ item.id }}"
  with_items: ec2.instances
```

Ping the instance, make sure Ansible can connect via SSH to the host:

```
$ ansible -i ec2.py tag_class_jumpbox -m ping
```



You might want to disable host key checking in SSH configuration so SSH will automatically add new host keys to the user known hosts files without asking (the default is *ask*). To disable host key checking, set `StrictHostKeyChecking no` in your `/etc/ssh/ssh_config` file.

Create roles to install Ansible:

```
# mkdir roles/ansible # mkdir roles/ansible/tasks $ vi
roles/ansible/tasks/main.yml

---
- name: upgrade all packages
  yum: name=* state=latest
- name: install the 'Development tools' package group
  yum: name="@Development tools" state=present
- name: install required packages
  yum: name={{ item }} state=present
  with_items:
    - epel-release.noarch
    - python-pip
    - python-devel
- name: install setuptools
  pip: name=setuptools extra_args='--upgrade'
- name: install ansible
  pip: name=ansible
```

And the playbook to install Ansible in the jump box instance:

```
$ vi install_ansible.yml

---
- hosts: tag_class_jumpbox
  become: yes
  roles:
    - ansible

$ ansible-playbook -i ec2.py install_ansible.yml
```

To allow SSH access from this new Ansible machine, do not forget to modify the security group of hosts you want to manage. For example, if you want to install your own MySQL database server in the private subnet and manage the configuration using Ansible, you can modify the rules in `sg_modify.yml`:

```
- name: modify sg_database rules
  ec2_group:
    region: "{{ region }}"
    vpc_id: "{{ vpc_id }}"
    name: "{{ prefix }}sgdatabase"
    description: security group for databases
    rules:
      # allow ssh from the jump box
      - proto: tcp
        from_port: 22
        to_port: 22
        group_name: "{{ prefix }}sgjumpbox"
      # allow mysql access from web servers
      - proto: tcp
        from_port: 3306
        to_port: 3306
        group_name: "{{ prefix }}sgweb"
    rules_egress:
      - proto: tcp
        from_port: 80
        to_port: 80
        cidr_ip: 0.0.0.0/0
      - proto: tcp
        from_port: 443
        to_port: 443
        cidr_ip: 0.0.0.0/0
```

And run the `sg_modify.yml` playbook to modify security group rules.

OpenVPN server

This section will show you how to launch an OpenVPN EC2 instance in the public subnet using Ansible, and configure the server from its web UI.

OpenVPN Access Server is a full featured SSL VPN software solution that integrates OpenVPN server capabilities, enterprise management capabilities, simplified OpenVPN Connect UI, and OpenVPN Client software packages that accommodate Windows, MAC, and Linux OS environments. OpenVPN Access Server supports a wide range of configurations, including secure and granular remote access to internal network and/or private cloud network resources and applications with fine-grained access control. For more information, visit <http://openvpn.net/index.php/access-server/overview.html>.

To launch an OpenVPN instance, first we need to know the AMI ID of the OpenVPN Access Server AMI for our region.

To get the AMI ID:

1. Go to your EC2 dashboard, select your region, and then click **Launch Instance** button.
2. On the left hand navigation bar, select **Community AMIs**.
3. When the AMI selection dialog appears, type **OpenVPN** in the search box.
4. Locate the latest version of OpenVPN Access Server AMI provided by openvpn.net and note the AMI ID.



5. Select **Cancel and Exit**.

Create the security group for our OpenVPN server:

```
$ vi sg_openvpn.yml  
---  
- hosts: localhost  
  connection: local
```

```
gather_facts: no
vars_files:
  - staging_vpc_info
vars:
  #your region
  region: ap-southeast-2
  #your ip address
  allowed_ip: 123.xxx.xxx.xxx/32
  #prefix for naming
  prefix: staging
  vpc_id: "{{ staging_vpc }}"
tasks:
  - name: create security group for openvpn instance
    ec2_group:
      region: "{{ region }}"
      vpc_id: "{{ vpc_id }}"
      #your security group name
      name: "{{ prefix }}sgopenvpn"
      description: security group for openvpn
      rules:
        - proto: tcp
          from_port: 22
          to_port: 22
          cidr_ip: "{{ allowed_ip }}"
        - proto: tcp
          from_port: 443
          to_port: 443
          cidr_ip: 0.0.0.0/0
        - proto: tcp
          from_port: 943
          to_port: 943
          cidr_ip: 0.0.0.0/0
        - proto: udp
          from_port: 1194
          to_port: 1194
          cidr_ip: 0.0.0.0/0
      rules_egress:
        - proto: all
          cidr_ip: 0.0.0.0/0 ~~
```

Run the playbook:

```
$ ansible-playbook -i hosts sg_openvpn.yml
```

The following playbook will launch our OpenVPN server instance in public subnet A:

```
$ vi ec2_vpc_openvpn.yml

---
- hosts: localhost
  connection: local
  gather_facts: no
  vars_files:
    - staging_vpc_info
  vars:
    region: ap-southeast-2
    key: yan-key-pair-apsydney
    instance_type: t2.micro
    image: ami-a17f199b
    prefix: staging
    vpc_subnet_id: "{{ staging_subnet_public_0 }}"
  tasks:
    - name: openvpn server instance provisioning
      ec2:
        region: "{{ region }}"
        key_name: "{{ key }}"
        instance_type: "{{ instance_type }}"
        image: "{{ image }}"
        source_dest_check: no
        wait: yes
        group: "{{ prefix }}sgopenvpn"
        instance_tags:
          Name: "{{ prefix }}openvpn"
          class: openvpn
          environment: "{{ prefix }}"
          id: openvpnlaunch_01
        vpc_subnet_id: "{{ vpc_subnet_id }}"
      register: ec2
    - name: associate new EIP for the instance
      ec2_eip:
        region: "{{ region }}"
        instance_id: "{{ item.id }}"
      with_items: ec2.instances
```

Configure OpenVPN server

To configure, SSH to the OpenVPN Access Server as `openvpnas` user:

```
# ssh -i ~/.ssh/yan-key-pair-apsydney.pem openvpnas@openvpn-ipaddress
```

The OpenVPN Access Server Setup Wizard runs automatically upon your initial login to the appliance. If you would like to run this wizard again in the future, issue the become `ovpn-init --ec2` command in the terminal.

```
Please enter 'yes' to indicate your agreement [no]: yes
Will this be the primary Access Server node? (enter 'no' to configure
as a backup or standby node)
> Press ENTER for default [yes]:
Please specify the network interface and IP address to be used by the
Admin Web UI:
(1)all interfaces: 0.0.0.0
(2)eth0: 10.0.0.40
Please enter the option number from the list above (1-2).
> Press Enter for default [2]: 1
Please specify the port number for the Admin Web UI.
> Press ENTER for default [943]:
Please specify the TCP port number for the OpenVPN Daemon
> Press ENTER for default [443]:
Should client traffic be routed by default through the VPN?
> Press ENTER for default [no]:
Should client DNS traffic be routed by default through the VPN?
> Press ENTER for default [no]:
Use local authentication via internal DB?
> Press ENTER for default [yes]:
Private subnets detected: ['10.0.0.0/16']
Should private subnets be accessible to clients by default?
> Press ENTER for EC2 default [yes]:
To initially login to the Admin Web UI, you must use a username and
password that successfully authenticates you with the host UNIX system (you
can later modify the settings so that RADIUS or LDAP is used for
authentication instead).
You can login to the Admin Web UI as "openvpn" or specify a different
user account to use for this purpose.
Do you wish to login to the Admin UI as "openvpn"?
> Press ENTER for default [yes]: no
>Specify the username for an existing user or for the new user account:
openvpn-admin
>Type the password for the 'openvpn-admin' account:
>Confirm the password for the 'openvpn-admin' account:
>Please specify your OpenVPN-AS license key (or leave blank to specify
later):
Initializing OpenVPN...
```

After you complete the setup wizard, you can access the Admin Web UI area to configure other aspects of your VPN:

1. Go to <https://openvpn-ipaddress/admin>.
2. Go to **VPN Settings** menu.
3. Configure subnets for the clients. On the Dynamic IP Address network allocate address for VPN clients, for example 10.1.0.0/23.
4. Static IP Address Network: (leave empty)
5. Group Default IP Address Network: (leave empty)
6. Click **Save settings**.
7. Click **Update Running Server**.

To add user:

1. Go to **User Permissions** menu.
2. Add a user name.
3. Click **show** and set password.
4. Click **Update Running Server**.

Connect client

The Connect Client can be accessed via a preferred web browser by entering the following address into the address bar: <https://openvpn-ipaddress>.

Users have the option to either Connect to the VPN or Login to the Connect Client. When connecting, the user will be connected to the VPN directly through their web browser. When the user decides to login to the Connect Client they can download their user configuration files (`client.ovpn`) and use them to connect to the VPN with other OpenVPN Clients.

For more information on OpenVPN Access Server go to <https://openvpn.net/index.php/access-server/docs.html>.

Getting VPC and subnet ID

One reader told me that it's horrible to store VPC and subnets ID in a file. Too bad Ansible doesn't have a module yet to get the VPC or subnets ID based on particular filter. This following Python script can be added as Ansible module and called from playbook, to get the VPC or subnet ID based on resource tags. Of course when you created the VPC or subnets you have to give a specific tag for each resource to make this module works. This will also give you an example on how to add your own Ansible module. The output of this module is in JSON format { vpc_ids: [list of vpc ids], subnet_ids: [list of subnet ids] }.

The original code can be found here: https://github.com/edx/configuration/blob/master/playbooks/library/vpc_lookup. I modified some parts of the code to make it work.

Put the following script in `library/` directory, relative to your playbook.

```
$ cd /home/yan/ansible4aws
# mkdir library
$ vi library/vpc_lookup

#!/usr/bin/python

#author: John Jarvis

import sys

AWS_REGIONS = ['ap-northeast-1',
               'ap-southeast-1',
               'ap-southeast-2',
               'eu-west-1',
               'sa-east-1',
               'us-east-1',
               'us-west-1',
               'us-west-2']

try:
    from boto.vpc import VPCCConnection
    from boto.vpc import connect_to_region
except ImportError:
    print "failed=True msg='boto required for this module'"
    sys.exit(1)

def main():

    module=AnsibleModule(
        argument_spec=dict(
            region=dict(choices=AWS_REGIONS),
```

```
aws_secret_key=dict(aliases=['ec2_secret_key', 'secret_key'],
                    no_log=True),
aws_access_key=dict(aliases=['ec2_access_key', 'access_key']),
tags=dict(default=None, type='dict'),
)
)

tags = module.params.get('tags')
aws_secret_key = module.params.get('aws_secret_key')
aws_access_key = module.params.get('aws_access_key')
region = module.params.get('region')

# If we have a region specified, connect to its endpoint.
if region:
    try:
        vpc = connect_to_region(region, aws_access_key_id=aws_access_key,
                               aws_secret_access_key=aws_secret_key)
    except boto.exception.NoAuthHandlerFound, e:
        module.fail_json(msg=str(e))
else:
    module.fail_json(msg="region must be specified")

subnet_ids = []
for tag, value in tags.iteritems():
    for subnet in vpc.get_all_subnets(filters={"tag:" + tag: value}):
        subnet_ids.append(subnet.id)

vpc_ids = []
for tag, value in tags.iteritems():
    for vpc in vpc.get_all_vpcs(filters={"tag:" + tag: value}):
        vpc_ids.append(vpc.id)

module.exit_json(changed=False, vpc_ids=vpc_ids, subnet_ids=subnet_ids)

#this is magic, see lib/ansible/module_common.py
#<<INCLUDE_ANSIBLE_MODULE_COMMON>>
main()

# chmod 755 library/vpc_lookup
```

The following playbook will show you how to use this additional module. This example playbook will get the ID of VPC with resource tags Name=test-vpc (if exists) and delete the VPC.

```
$ vi vpc_delete.yml

- hosts: localhost
  connection: local
  gather_facts: no
  vars:
    region: ap-southeast-2
  tasks:
    - name: get vpc id
      vpc_lookup:
        region: "{{ region }}"
        tags:
          Name: test-vpc
      register: vpc

    - name: delete vpc
      ec2_vpc:
        region: "{{ region }}"
        state: absent
        vpc_id: "{{ item }}"
        wait: yes
      with_items: vpc.vpc_ids
```

You can use the same module to get subnet id based on resource tags, the JSON output used is subnet_ids.



You can also use Ansible extra modules called `ec2_vpc_net_facts` and `ec2_vpc_subnet_facts` (added in version 2.1) to gather VPC and subnet facts. For more information, go to http://docs.ansible.com/ansible/ec2_vpc_net_facts_module.html and http://docs.ansible.com/ansible/ec2_vpc_subnet_facts_module.html.

7

RDS Provisioning with Ansible

Introduction

Amazon **Relational Database Service** (Amazon RDS) is a web service that makes it easier to set up, operate, and scale a relational database in the cloud. It provides cost-efficient, resizable capacity for an industry-standard relational database and manages common database administration tasks (for more information visit

[Why would you want a managed relational database service? Because Amazon RDS takes over many of the difficult or tedious management tasks of a relational database.](http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide>Welcome.html).</p></div><div data-bbox=)

- When you buy a server, you get CPU, memory, storage, and IOPS, all bundled together. With Amazon RDS, these are split apart so that you can scale them independently. So, for example, if you need more CPU, less IOPS, or more storage, you can easily allocate them.
- Amazon RDS manages backups, software patching, automatic failure detection, and recovery.
- In order to deliver a managed service experience, Amazon RDS does not provide shell access to DB instances, and it restricts access to certain system procedures and tables that require advanced privileges.
- You can have automated backups performed when you need them, or create your own backup snapshot. These backups can be used to restore a database, and Amazon RDS's restore process works reliably and efficiently.
- You can get high availability with a primary instance and a synchronous secondary instance that you can failover to when problems occur. You can also use MySQL read replicas to increase read scaling.

- You can use the database products you are already familiar with: **MySQL**, **MariaDB**, **PostgreSQL**, **Oracle**, **Microsoft SQL Server**, and the new, **MySQL-compatible Amazon Aurora DB engine**.
- In addition to the security in your database package, you can help control who can access your RDS databases by using AWS IAM to define users and permissions. You can also help protect your databases by putting them in a virtual private cloud.

DB instances

The basic building block of Amazon RDS is the DB instance. A DB instance is an isolated database environment in the cloud. A DB instance can contain multiple user-created databases, and you can access it by using the same tools and applications that you use with a stand-alone database instance. You can create and modify a DB instance by using the Amazon RDS command line interface, the Amazon RDS API, or the AWS Management Console.

Each DB instance runs a DB engine. Amazon RDS currently supports the MariaDB, MySQL, PostgreSQL, Oracle, and Microsoft SQL Server DB engines. Each DB engine has its own supported features, and each version of a DB engine may include specific features. Additionally, each DB engine has a set of parameters in a DB parameter group that control the behavior of the databases that it manages.

The computation and memory capacity of a DB instance is determined by its DB instance class. You can select the DB instance that best meets your needs. If your needs change over time, you can change DB instances. For information about DB instance classes, see <http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.DBInstanceClass.html>. For pricing information on DB instance classes see <http://aws.amazon.com/rds/pricing>.

For each DB instance, you can select from 5 GB to 6 TB of associated storage capacity. Each DB instance class has minimum and maximum storage requirements for the DB instances that are created from it. It's important to have sufficient storage so that your databases have room to grow and that features for the DB engine have room to write content or log entries.

DB instance storage comes in two types, standard and provisioned IOPS. Standard storage is allocated on Amazon EBS volumes and connected to your DB instance. Provisioned IOPS uses optimized EBS volumes and an optimized configuration stack and provides additional, dedicated capacity for EBS I/O. These optimizations enable instances to fully utilize the IOPS provisioned on an EBS volume. For more information on Provisioned IOPS, see http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_PIOPS.html.

You can run a DB instance on a virtual private cloud using Amazon's **Virtual Private Cloud (VPC)** service. When you use a virtual private cloud, you have control over your virtual networking environment: you can select your own IP address range, create subnets, and configure routing and access control lists. The basic functionality of Amazon RDS is the same whether it is running in a VPC or not; Amazon RDS manages backups, software patching, automatic failure detection, and recovery. There is no additional cost to run your DB instance in a VPC.

Regions and availability zones

Amazon cloud computing resources are housed in highly available data center facilities in different areas of the world (for example, North America, Europe, or Asia). Each data center location is called a **region**.

Each region contains multiple distinct locations called **Availability Zones**, or **AZs**. Each Availability Zone is engineered to be isolated from failures in other Availability Zones, and to provide inexpensive, low-latency network connectivity to other Availability Zones in the same region. By launching instances in separate Availability Zones, you can protect your applications from the failure of a single location.

You can run your DB instance in several Availability Zones, an option called a **Multi-AZ deployment**. When you select this option, Amazon automatically provisions and maintains a synchronous standby replica of your DB instance in a different Availability Zone. The primary DB instance is synchronously replicated across Availability Zones to the standby replica to provide data redundancy, failover support, eliminate I/O freezes, and minimize latency spikes during system backups.

Security groups

A security group controls the access to a DB instance. It does so by allowing access to IP address ranges or Amazon EC2 instances that you specify.

Amazon RDS uses DB security groups, VPC security groups, and EC2 security groups. In simple terms, a DB security group controls access to a DB instance that is not in a VPC, a VPC security group controls access to a DB instance inside a VPC, and an Amazon EC2 security group controls access to an EC2 instance.

DB parameter groups

You manage the configuration of a DB engine by using a DB parameter group. A DB parameter group contains engine configuration values that can be applied to one or more DB instances of the same instance type. Amazon RDS applies a default DB parameter group if you don't specify a DB parameter group when you create a DB instance. The default group contains defaults for the specific database engine and instance class of the DB instance.

DB option groups

Some DB engines offer additional features that make it easier to manage data and databases, and to provide additional security for your database. Amazon RDS uses option groups to enable and configure these features. An option group can specify features, called **options**, that are available for a particular Amazon RDS DB instance. Options can have settings that specify how the option works. When you associate a DB instance with an option group, the specified options and option settings are enabled for that DB instance.

How you are charged for Amazon RDS

When you use Amazon RDS, you pay only for what you use, and there are no minimum or setup fees. You are billed according to the following criteria:

- **Instance class:** Pricing is based on the class (for example, micro, small, large, xlarge) of the DB instance consumed.
- **Running time:** You are billed by the instance-hour, which is equivalent to a single instance running for an hour. For example, both a single instance running for two hours and two instances running for one hour consume 2 instance-hours. If a DB instance runs for only part of an hour, you are charged for a full instance-hour.
- **Storage:** The storage capacity that you have provisioned to your DB instance is billed per GB per month. If you scale your provisioned storage capacity within the month, your bill will be pro-rated.
- **I/O requests per month:** Total number of storage I/O requests that you have made in a billing cycle.
- **Backup storage:** Backup storage is the storage that is associated with automated database backups and any active database snapshots that you have taken. Increasing your backup retention period or taking additional database snapshots increases the backup storage consumed by your database. Amazon RDS provides

backup storage up to 100% of your provisioned database storage at no additional charge. For example, if you have 10 GB-months of provisioned database storage, Amazon will provide up to 10 GB-months of backup storage at no additional charge. Most databases require less raw storage for a backup than for the primary dataset, so if you don't keep multiple backups, you will never pay for backup storage. Backup storage is free only for active DB instances.

- **Data transfer:** Internet data transfer in and out of your DB instance.

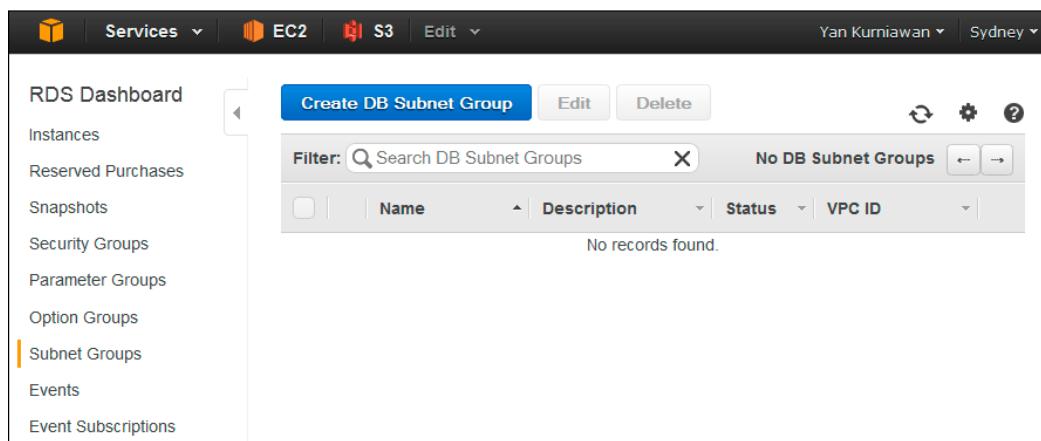
Getting started with RDS

In this section we'll create a Multi-AZ MySQL DB instance using the Amazon RDS console. We'll launch the DB instance in the private subnets of our VPC which we've created in Chapter 6, *VPC Provisioning with Ansible*.

To support Multi-AZ deployment, we'll need a Multi-AZ DB Subnet Group. When you create a DB instance in a VPC, you must select a DB subnet group. Amazon RDS then uses that DB subnet group and your preferred AZ to select a subnet and an IP address within that subnet. For Multi-AZ deployment, defining a subnet for two or more AZ in a region allows Amazon RDS to create a new standby in another AZ should the need arise.

To create a DB Subnet group:

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, click on **Subnet Groups**.



3. Click on **Create DB Subnet Group**.
4. Enter the desired **Name** and **Description** for the Subnet Group, select VPC (VPC ID of staging_vpc) and add the first AZ (ap-southeast-2a) and subnet ID (staging_subnet_private_0).

Create DB Subnet Group

To create a new Subnet Group give it a name, description, and select an existing VPC below. Once you select an existing VPC, you will be able to add subnets related to that VPC.

Name	dbsg1	i
Description	rds subnet group	i
VPC ID	vpc- [dropdown]	i

Add Subnet(s) to this Subnet Group. You may add subnets one at a time below or [add all the subnets](#) related to this VPC. You may make additions/edits after this group is created.

Availability Zone	Subnet ID	Availability Zone	Subnet ID	CIDR Block	Action
ap-southeast-2a	subnet-[dropdown]				None added
Add					

Cancel **Yes, Create**

5. Click on **Add** and select the second AZ (ap-southeast-2b) and subnet ID (staging_subnet_private_1). Click on **Add**.

Add Subnet(s) to this Subnet Group. You may add subnets one at a time below or [add all the subnets](#) related to this VPC. You may make additions/edits after this group is created.

Availability Zone	Subnet ID	Availability Zone	Subnet ID	CIDR Block	Action
ap-southeast-2b	subnet-[dropdown]	ap-southeast-2a	subnet-[dropdown]	10.0.1.0/24	Remove
Add					

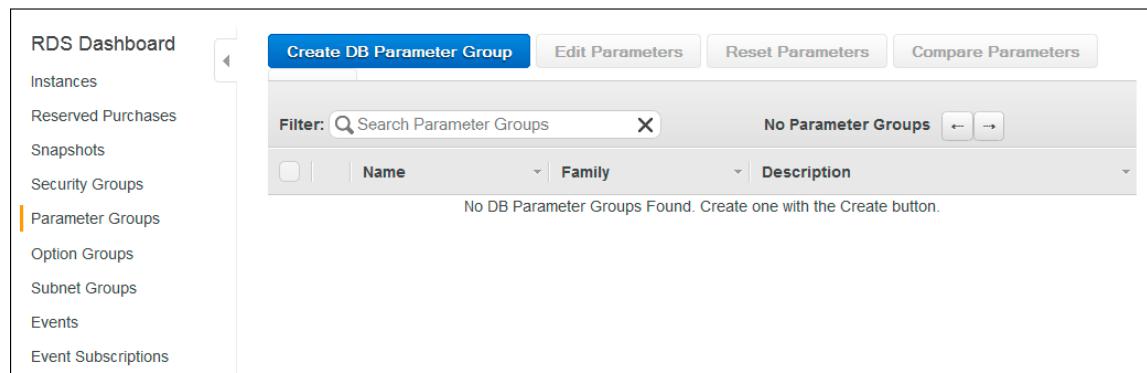
Cancel **Yes, Create**

6. Click on **Yes, Create**.

We'll also create a DB parameter group. You manage the DB engine configuration through the use of DB parameter groups. DB parameter groups act as a container for engine configuration values that are applied to one or more DB instances. A default DB parameter group is used if you create a DB instance without specifying a DB parameter group. This default group contains database engine defaults and Amazon RDS system defaults based on the engine, compute class, and allocated storage of the instance. Note that you cannot modify the parameter settings of a default DB parameter group, you must create your own parameter group to change the settings, and not all DB engine parameters are available for modification in a DB parameter group (for more information visit http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_WorkingWithParamGroups.html).

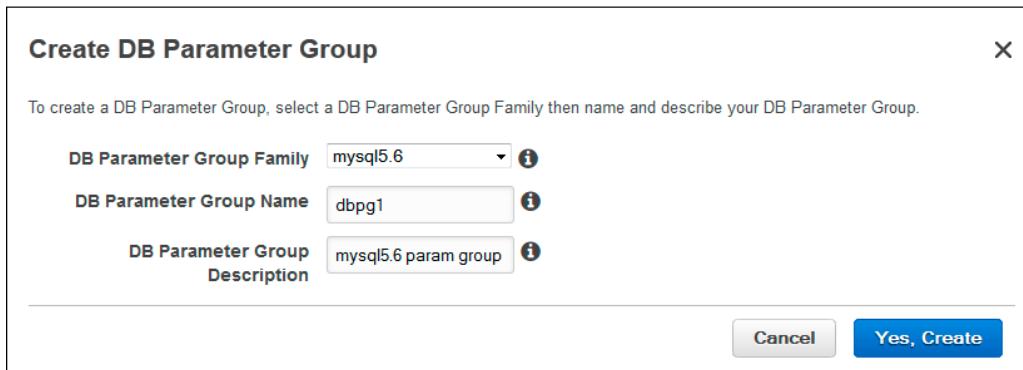
To create a DB parameter group:

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, click on **Parameter Groups**.



3. Click on **Create DB Parameter Group**.

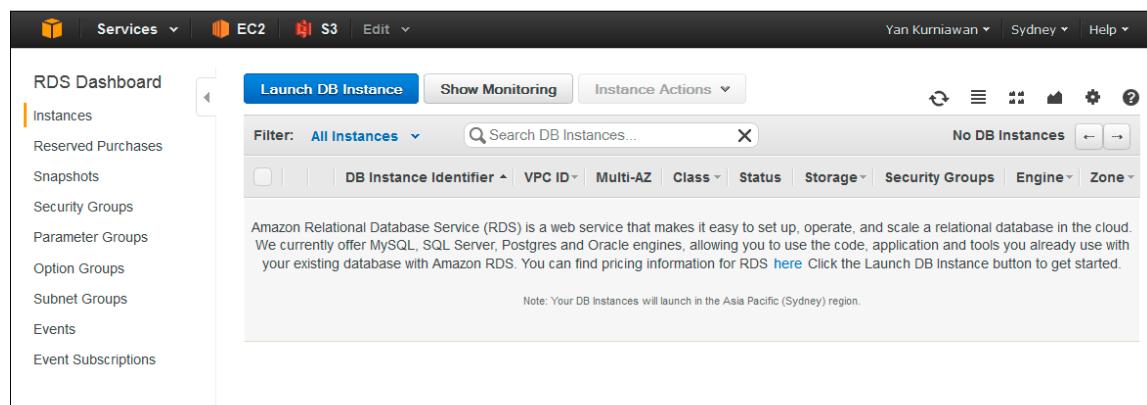
4. Enter the desired **DB Parameter Group Family**, **DB Parameter Group Name**, and **DB Parameter Group Description**.



5. Click on **Yes, Create**.
6. If you want to edit database parameters, select the parameter group and click on **Edit Parameters**. For more info about the parameters, please refer to MySQL documentation (<http://dev.mysql.com/doc/refman/5.6/en/mysqld-option-tables.html>). After you made changes, click on **Save Changes**.

To create a DB instance:

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, click on **Instances**.



3. Click on **Launch DB Instance**.
4. On step 1, **Select Engine** page, select **mysql**, and click on **Select**.
5. On step 2, **Production?** page, select on **Yes, use Multi-AZ Deployment and Provisioned IOPS Storage as defaults while creating this instance** option, then click on **Next**.

<p>Step 1: Select Engine</p> <p>Step 2: Production?</p> <p>Step 3: Specify DB Details</p> <p>Step 4: Configure Advanced Settings</p>	<p>Do you plan to use this database for production purposes?</p> <p>For databases used in production or pre-production we recommend:</p> <ul style="list-style-type: none">• Multi-AZ Deployment for high availability (99.95% monthly up time SLA)• Provisioned IOPS Storage for fast, consistent performance <p>Billing is based upon the RDS pricing table. An instance which uses these features is not eligible for the RDS Free Usage Tier.</p> <p><input checked="" type="radio"/> Yes, use Multi-AZ Deployment and Provisioned IOPS Storage as defaults while creating this instance</p> <p><input type="radio"/> No, this instance is intended for use outside of production or under the RDS Free Usage Tier</p> <hr/> <p>Cancel Previous Next</p>
---	--

6. On step 3, **Specify DB Details** page, enter the required information, configure **DB Instance Class**, **Allocated Storage**, and **IOPS**.

The screenshot shows the 'Specify DB Details' configuration page. It is divided into two main sections: 'Instance Specifications' and 'Settings'.
Instance Specifications:

- DB Engine: mysql
- License Model: general-public-license
- DB Engine Version: 5.6.17
- DB Instance Class: db.t1.micro – 1 vCPU, 0.613 GiE
- Multi-AZ Deployment: Yes

Allocated Storage: 200 GB
Use Provisioned IOPS: Yes
Provisioned IOPS: 1000

Settings:

- DB Instance Identifier*: mysql1
- Master Username*: dbadmin
- Master Password*: [REDACTED]
- Confirm Password*: [REDACTED]

A tooltip on the 'Confirm Password*' field says: 'Retype the value you specified for Master Password.'

Buttons at the bottom: Cancel, Previous, Next (highlighted in blue).

7. Click on **Next**.
8. On step 4, **Configure Advanced Settings** page, select the **VPC**, **Subnet Group**, **VPC Security Groups**, and **Parameter Group**. Enter **Database Name** if you want to create an initial MySQL database on the DB Instance.

Configure Advanced Settings

Network & Security

VPC: vpc- [dropdown]
DB Subnet Group: dbsg1 [dropdown]
Publicly Accessible: No [dropdown]
Availability Zone: No Preference [dropdown]
VPC Security Group(s): staging_sg_database (VPC) [selected]
staging_sg_nat (VPC)
staging_sg_web (VPC)
default (VPC)

Select the Virtual Private Cloud (VPC) that defines the virtual networking environment for this DB instance. [Learn More](#).

Database Options

Database Name: mysql1
Note: if no database name is specified then no initial MySQL database will be created on the DB Instance.

Database Port: 3306
Parameter Group: dbpg1 [dropdown]
Option Group: default.mysql-5.6 [dropdown]

Backup

Please note that automated backups are currently supported for InnoDB storage engine only. If you are using MyISAM, refer to detail [here](#).

Backup Retention Period: 1 days
Backup Window: No Preference [dropdown]

Maintenance

Auto Minor Version Upgrade: Yes [dropdown]
Maintenance Window: No Preference [dropdown]

9. Click on **Launch DB Instance**.

Your DB instance will be launched in your VPC. It will take a few minutes to create the DB instance.

If you have finished with this example, delete the DB instance from your RDS console to avoid any cost. You could also delete the subnet group and parameter group. In the next section we will use Ansible to create DB subnet group, parameter group, and launch RDS DB instance.

The rds_subnet_group module

In this section we will use Ansible `rds_subnet_group` module to create, modify, or delete RDS database subnet groups.

Options for this module are listed in the following table:

parameter	required	default	choices	comments
<code>aws_access_key</code>	no			AWS access key. If not set then the value of the <code>AWS_ACCESS_KEY</code> environment variable is used aliases: <code>ec2_access_key</code> , <code>access_key</code>
<code>aws_secret_key</code>	no			AWS secret key. If not set then the value of the <code>AWS_SECRET_KEY</code> environment variable is used aliases: <code>ec2_secret_key</code> , <code>secret_key</code>
<code>description</code>	no			Database subnet group description. Only set when a new group is added.
<code>ec2_url</code>	no			URL to use to connect to EC2 or your Eucalyptus cloud (by default the module will use EC2 endpoints). Must be specified if region is not used. If not set then the value of the <code>EC2_URL</code> environment variable, if any, is used
<code>name</code>	yes			Database subnet group identifier.
<code>profile</code> (added in 1.6)	no			Uses a boto profile. Only works with <code>boto >= 2.24.0</code>
<code>region</code>	no			The AWS region to use. If not specified then the value of the <code>AWS_REGION</code> or <code>EC2_REGION</code> environment variable, if any, is used. aliases: <code>aws_region</code> , <code>ec2_region</code> ⁶²
<code>security_token</code> (added in 1.6)	no			AWS STS security token. If not set then the value of the <code>AWS_SECURITY_TOKEN</code> or <code>EC2_SECURITY_TOKEN</code> environment variable is used. aliases: <code>access_token</code>
<code>state</code>	yes	<code>present</code>	<code>present</code> <code>absent</code>	Specifies whether the subnet should be present or absent.
<code>subnets</code>	no			List of subnet IDs that make up the database subnet group.
<code>validate_certs</code> (added in 1.5)	no	<code>yes</code>	<code>yes</code> <code>no</code>	When set to "no", SSL certificates will not be validated for boto versions $\geq 2.6.0$.

The following is a playbook to create a Multi-AZ subnet group in our VPC:

```
$ cd /home/yan/ansible4aws
$ vi dbsg_create.yml

---
- hosts: localhost
gather_facts: no
connection: local
vars:
  region: ap-southeast-2
vars_files:
  - staging_vpc_info
tasks:
  - name: create Multi-AZ DB subnet group
    rds_subnet_group:
      name: dbsg2
      state: present
      region: "{{ region }}"
      description: DB Subnet Group 2
      subnets:
        - "{{ staging_subnet_private_0 }}"
        - "{{ staging_subnet_private_1 }}"
```

This module is idempotent, using name option as identifier. You can modify the subnets list in the playbook and run the playbook again to modify the subnet group.

To delete the subnet group, we can use the following playbook:

```
$ vi dbsg_delete.yml

---
- hosts: localhost
gather_facts: no
connection: local
vars:
  region: ap-southeast-2
tasks:
  - name: delete DB subnet group
    rds_subnet_group:
      name: dbsg2
      state: absent
      region: "{{ region }}"
```

The rds_param_group module

To create, modify, and delete RDS parameter groups we'll use Ansible `rds_param_group` module. Options for this module are listed in the following table:

parameter	required	default	choices	comments
<code>aws_access_key</code>	no			AWS access key. If not set then the value of the <code>AWS_ACCESS_KEY</code> environment variable is used aliases: <code>ec2_access_key</code> , <code>access_key</code>
<code>aws_secret_key</code>	no			AWS secret key. If not set then the value of the <code>AWS_SECRET_KEY</code> environment variable is used aliases: <code>ec2_secret_key</code> , <code>secret_key</code>
<code>description</code>	no			Database parameter group description. Only set when a new group is added.
<code>ec2_url</code>	no			URL to use to connect to EC2 or your Eucalyptus cloud (by default the module will use EC2 endpoints). Must be specified if region is not used. If not set then the value of the <code>EC2_URL</code> environment variable, if any, is used
<code>engine</code>	no		see below	The type of database for this group. Required for state=present.
<code>immediate</code>	no			Whether to apply the changes immediately, or after the next reboot of any associated instances
<code>name</code>	yes			Database parameter group identifier.
<code>params</code>	no		see below	Map of parameter names and values.
<code>profile</code> (added in 1.6)	no			Uses a boto profile. Only works with <code>boto >= 2.24.0</code>
<code>region</code>	no			The AWS region to use. If not specified then the value of the <code>AWS_REGION</code> or <code>EC2_REGION</code> environment variable, if any, is used. aliases: <code>aws_region</code> , <code>ec2_region</code> ⁶³
<code>security_token</code> (added in 1.6)	no			AWS STS security token. If not set then the value of the <code>AWS_SECURITY_TOKEN</code> or <code>EC2_SECURITY_TOKEN</code> environment variable is used. aliases: <code>access_token</code>
<code>state</code>	yes	present	present	Specifies whether the group should be present

parameter	required	default	choices	comments
			absent	or absent.
validate_certs (added in 1.5)	no	yes	yes no	When set to "no", SSL certificates will not be validated for boto versions >= 2.6.0.

Valid options for engine: aurora5.6, mariadb10.0, mysql5.1, mysql5.5, mysql5.6, mysql5.7, oracle-ee-11.2, oracle-ee-12.1, oracle-se-11.2, oracle-se-12.1, oracle-se1-11.2, oracle-se1-12.1, postgres9.3, postgres9.4, postgres9.5, sqlserver-ee-10.5, sqlserver-ee-11.0, sqlserver-ex-10.5, sqlserver-ex-11.0, sqlserver-ex-12.0, sqlserver-se-10.5, sqlserver-se-11.0, sqlserver-se-12.0, sqlserver-web-10.5, sqlserver-web-11.0 and sqlserver-web-12.0.

For params option, the numeric values may be represented as K for Kilo (1024), M for Mega (10242), G for Giga (10243), or T for Tera (10244), and these values will be expanded into the appropriate number before being set in the parameter group.

The following example will create an RDS Parameter Group for MySQL 5.6. For this example, I will set the innodb_lock_timeout parameter to 3600 seconds. It's useful for long-running transactions.

```
$ cd /home/yan/ansible4aws
$ vi mysql_pg_create.yml
---
- hosts: localhost
gather_facts: no
connection: local
vars:
region: ap-southeast-2
tasks:
- name: create mysql parameter group
>rds_param_group:
name: mysqlpg1
state: present
region: "{{ region }}"
description: MySQL Parameter Group 1
engine: mysql5.6
params:
innodb_lock_timeout: 3600
```

Run the playbook to create mysqlpg1 Parameter Group.

To see which MySQL parameter you can change, go to your RDS console, select **Parameters Groups** in the navigation pane, select `mysqlpg1` and click on **Edit Parameters**.

Name	Value	Allowed Values	Is Modifiable
allow-suspicious-udfs			false
auto_increment_increment		1-65535	true
auto_increment_offset		1-65535	true
autocommit	<engine-default>		true
automatic_sp_privileges	<engine-default>		true
back_log		1-65535	true
basedir	/rdsdbbin/mysql		false
binlog_cache_size	32768	4096-18446744073709547520	true
binlog_checksum	<engine-default>		true
binlog_format	MIXED		true
binlog_max_flush_queue_time		0-100000	true
binlog_order_commits	<engine-default>		true
binlog_row_image			false
binlog_rows_query_log_events			false

MySQL parameters

For more information about MySQL 5.6 parameters, go to <http://dev.mysql.com/doc/refman/5.6/en/mysqld-option-tables.html>.

To modify the parameters, we can change the `params` dictionary in our playbook and run the playbook again. For example, I will modify `max_allowed_packet` parameter from the default 1 MB to 512 MB. Some databases make use of the MySQL BLOB type. In order to send large BLOBs over the network, this value needs to be set larger than the default. I will also modify `net_write_timeout` parameter, the default value is 60, increase the timeout to 300 seconds. This will allow for large or congested network transfers.

```
$ vi mysql_pg_create.yml
```

```
---
- hosts: localhost
```

```
gather_facts: no
connection: local
vars:
  region: ap-southeast-2
tasks:
  - name: create mysql parameter group
    rds_param_group:
      name: mysqlpg1
      state: present
      region: "{{ region }}"
      description: MySQL Parameter Group 1
      engine: mysql5.6
    params:
      innodb_lock_wait_timeout: 3600
      max_allowed_packet: 512M
      net_write_timeout: 300
```

The following playbook can be used to delete the parameter group:

```
$ vi mysql_pg_delete.yml

---
- hosts: localhost
  gather_facts: no
  connection: local
  vars:
    region: ap-southeast-2
  tasks:
    - name: delete mysql parameter group
      rds_param_group:
        name: mysqlpg1
        state: absent
        region: "{{ region }}"
```

The rds module

In this section we will use Ansible's `rds` module to create, delete, or modify rds instances. When creating an instance it can be either a new instance or a read-only replica of an existing instance.



Amazon RDS uses MySQL's built-in replication functionality to create a special type of DB instance called a read replica from a source DB instance. Updates made to the source DB instance are copied to the read replica. You can reduce the load on your source DB instance by routing read queries from your applications to the read replica. Read replicas allow you to elastically scale out beyond the capacity constraints of a single DB instance for read-heavy database workloads (for more information visit http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ReadReplicas.html).

Options for `rds` module are listed in the following table:

parameter	required	default	choices	comments
<code>apply_immediately</code>	no		yes no	Used only when <code>command=modify</code> . If enabled, the modifications will be applied as soon as possible rather than waiting for the next preferred maintenance window.
<code>aws_access_key</code>	no			AWS access key. If not set then the value of the <code>AWS_ACCESS_KEY</code> environment variable is used aliases: <code>ec2_access_key</code> , <code>access_key</code>
<code>aws_secret_key</code>	no			AWS secret key. If not set then the value of the <code>AWS_SECRET_KEY</code> environment variable is used aliases: <code>ec2_secret_key</code> , <code>secret_key</code>
<code>backup_retention</code>	no			Number of days backups are retained. Set to 0 to disable backups. Default is 1 day. Valid range: 0-35. Used only when <code>command=create</code> or <code>command=modify</code> .
<code>backup_window</code>	no			Backup window in format of hh24:mi-hh24:mi. If not specified then a random backup window is assigned. Used only when <code>command=create</code> or <code>command=modify</code> .
<code>character_set_name</code> (added in 1.9)	no			Associate the DB instance with a specified character set. Used with <code>command=create</code> .
<code>command</code>	yes		see below	Specifies the action to take. The 'reboot' option is available starting at version 2.0
<code>db_engine</code>	no		see	The type of database. Used only when

parameter	required	default	choices	comments
			below	command=create.
db_name	no			Name of a database to create within the instance. If not specified then no database is created. Used only when command=create.
engine_version	no			Version number of the database engine to use. Used only when command=create. If not specified then the current Amazon RDS default engine version is used.
force_failover (added in 2.0)	no	no		Used only when command=reboot. If enabled, the reboot is done using a MultiAZ failover.
instance_name	yes			Database instance identifier.
instance_type	no			The instance type of the database. Must be specified when command=create. Optional when command=replicate, command=modify or command=restore. If not specified then the replica inherits the same instance type as the source instance
iops	no			Specifies the number of IOPS for the instance. Used only when command=create or command=modify. Must be an integer greater than 1000.
license_model	no		see below	The license model for this DB instance. Used when command=create or command=restore.
maint_window	no			Maintenance window in format of ddd:hh24:mi-ddd:hh24:mi. (Example: Mon:22:00-Mon:23:15). If not specified then a random maintenance window is assigned. Used only when command=create or command=modify.
multi_zone	no		yes no	Specifies if this is a Multi-availability-zone deployment. Can not be used in conjunction with zone parameter. Used when command=create or command=modify.
new_instance_name (added in 1.5)	no			Name to rename an instance to. Used only when command=modify.
option_group	no			The name of the option group to use. If not specified then the default option group is used. Used only when command=create.
parameter_group	no			Name of the DB parameter group to associate with this instance. If omitted, the RDS default

parameter	required	default	choices	comments
				DBParameterGroup will be used. Used only when command=create or command=modify.
password	no			Password for the master database username. Used when command=create or command=modify.
port	no			Port number that the DB instance uses for connections. Used only when command=create or command=replicate.
				Prior to 2.0 it always defaults to null and the API would use 3306, it had to be set to other DB default values when not using MySQL.
				Starting at 2.0 it automatically defaults to what is expected for each DB engine.
publicly_accessible (added in 1.9)	no			explicitly set whether the resource should be publicly accessible or not. Used with command-create, command=replicate. Requires boto >= 2.26
region	no			The AWS region to use. If not specified then the value of the AWS_REGION or EC2_REGION environment variable, if any, is used. aliases: aws_region, ec2_region ⁶⁵
security_groups	no			Comma separated list of one or more security groups. Used only when command=create or command=modify.
size	no			Size in gigabytes of the initial storage for the DB instance. Used only when command=create or command=modify.
snapshot	no			Name of snapshot to take. When command=delete, if no snapshot name is provided, then no snapshot is taken. Used when command=delete or command=snapshot.
source_instance	no			Name of the database to replicate. Used only when command=replicate.
subnet	no			VPC subnet group. If specified then a VPC instance is created. Used only when command=create.
tags (added in 1.9)	no			tags dict to apply to a resource. Used with command create, replicate, or restore.
upgrade	no	yes no		Indicates that minor version upgrades should be applied automatically. Used only when

parameter	required	default	choices	comments
				command=create or command=replicate.
username	no			Master database username. Used only when command=create.
vpc_security_groups	no			Comma separated list of one or more VPC security group ids. Also requires subnet to be specified. Used only when command=create or command=modify.
wait	no	no	yes no	When command=create, replicate, modify or restore then wait for the database to enter the 'available' state. When command=delete wait for the database to be terminated.
wait_timeout	no	300		How long before wait gives up, in seconds.
zone	no			Availability zone in which to launch the instance. Used only when command=create, command=replicate or command=restore.

Valid options for command: create, replicate, delete, facts, modify, promote, snapshot, reboot and restore.

Valid options for db_engine: mariadb, aurora, MySQL, oracle-se1, oracle-se, oracle-ee, sqlserver-ee, sqlserver-se, sqlserver-ex, sqlserver-web, and postgres.

Valid options for license_model: license-included, bring-your-own-license, and general-public-license.

Port default value: 3306 for mysql, 1521 for Oracle, 1433 for SQL Server, 5432 for PostgreSQL.

The following playbook will create a Multi-AZ MySQL RDS instance, using the subnet group and parameter group we have created in previous section. We will use the `staging_sg_database` security group for this instance. Open the VPC console and note the security group ID of `staging_sg_database` security group.

```
$ vi mysql_rds_create.yml

---
- hosts: localhost
  gather_facts: no
  connection: local
  vars:
    region: ap-southeast-2
    size: 100
    instance_type: db.m1.small
    db_engine: MySQL
    engine_version: 5.6.17
    subnet: dbsg2
    parameter_group: dbpg1
    # staging_sg_database security group ID
    security_groups: sg-xxxxxxx
    iops: 1000
    db_name: mydb
    username: dbadmin
    password: mypassword
  tasks:
    - name: create mysql RDS instance
      rds:
        command: create
        instance_name: staging-mysql-1
        region: "{{ region }}"
        size: "{{ size }}"
        instance_type: "{{ instance_type }}"
        db_engine: "{{ db_engine }}"
        engine_version: "{{ engine_version }}"
        subnet: "{{ subnet }}"
        parameter_group: "{{ parameter_group }}"
        multi_zone: yes
        db_name: "{{ db_name }}"
        username: "{{ username }}"
        password: "{{ password }}"
        vpc_security_groups: "{{ security_groups }}"
        iops: "{{ iops }}"
```

You can use this following playbook to delete the RDS instance:

```
$ vi mysql_rds_delete.yml

---
- hosts: localhost
  gather_facts: no
  connection: local
  vars:
    region: ap-southeast-2
  tasks:
    - name: delete mysql RDS instance
      rds:
        command: delete
        region: "{{ region }}"
        instance_name: staging-mysql-1
```

8

S3 Management with Ansible

Amazon S3 provides highly durable and available cloud storage for a variety of content, ranging from web applications to media files. It allows you to offload your entire storage infrastructure onto the cloud, where you can take advantage of Amazon S3's scalability and pay as you go pricing to handle your growing object storage needs. Amazon S3 can also be used as a backup storage. Amazon S3 offers highly durable, scalable, and secure cloud storage for backing up and archiving your critical data. For more information, visit <http://aws.amazon.com/s3>.

Bucket

Every object stored in Amazon S3 is contained in a bucket. Buckets partition the namespace of objects stored in Amazon S3 at the top level. Within a bucket, you can use any names for your objects, but bucket names must be unique across all of Amazon S3.

Buckets are similar to Internet domain names. Just as Amazon is the only owner of the domain name <http://aws.amazon.com>, only one person or organization can own a bucket within Amazon S3. After you create a uniquely named bucket in Amazon S3, you can organize and name the objects within the bucket in any way you like, and the bucket will remain yours for as long as you want and as long as you have the Amazon S3 account.

The similarities between buckets and domain names is not a coincidence-there is a direct mapping between Amazon S3 buckets and subdomains of <http://s3.amazonaws.com>. Objects stored in Amazon S3 are addressable by the REST API under the domain <http://bucketname.s3.amazonaws.com>. For example, if the object `homepage.html` is stored in the Amazon S3 bucket `mybucket`, its address would be <http://mybucket.s3.amazonaws.com/homepage.html>. For more documentation <http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingBucket.html>.

A bucket is owned by the AWS account that created it. Each AWS account can own up to 100 buckets at a time. Bucket ownership is not transferable; however, if a bucket is empty, you can delete it. After a bucket is deleted, the name becomes available to reuse, but the name might not be available for you to reuse for various reasons. For example, some other account could create a bucket with that name. Note, too, that it might take some time before the name can be reused. So if you want to use the same bucket name, don't delete the bucket. For more documentation, visit

<http://docs.aws.amazon.com/AmazonS3/latest/dev/BucketRestrictions.html>.

There is no limit to the number of objects that can be stored in a bucket and no difference in performance whether you use many buckets or just a few. You can store all of your objects in a single bucket, or you can organize them across several buckets.

You cannot create a bucket within another bucket.

It is recommended that all bucket names comply with DNS naming conventions:

- Bucket names must be at least 3 and no more than 63 characters long.
- Bucket names must be a series of one or more labels. Adjacent labels are separated by a single period (.). Bucket names can contain lowercase letters, numbers, and hyphens. Each label must start and end with a lowercase letter or a number.
- Bucket names must not be formatted as an IP address (for example, 192.168.5.4).

By default, all Amazon S3 resources (buckets, objects, and related subresources) are private: only the resource owner, an AWS account that created it, can access the resource. The resource owner can optionally grant access permissions to others by writing an access policy.

Amazon S3 offers access policy options broadly categorized as resource-based policies and user policies. Access policies you attach to your resources (buckets and objects) are referred to as resource-based policies. For example, bucket policies and access control lists (ACLs) are resource-based policies. You can also attach access policies to users in your account. These are called user policies. You may choose to use resource-based policies, user policies, or some combination of these to manage permissions to your Amazon S3 resources.

For more information on managing S3 access permission go to <http://docs.aws.amazon.com/AmazonS3/latest/dev/s3-access-control.html>.

Objects

Amazon S3 is a simple key, value store designed to store as many objects as you want. You store these objects in one or more buckets. An object consists of the following (for more documentation, visit

<http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingObjects.html>):

- **Key:** The name that you assign to an object. You use the object key to retrieve the object.
- **Version ID:** Within a bucket, a key and version ID uniquely identify an object. The version ID is a string that Amazon S3 generates when you add an object to a bucket.
- **Value:** The content that you are storing. An object value can be any sequence of bytes. Objects can range from zero to 5 TB in size.
- **Metadata:** A set of name-value pairs with which you can store information regarding the object. You can assign metadata, referred to as user-defined metadata, to your objects in Amazon S3. Amazon S3 also assigns system-metadata to these objects, which it uses for managing objects.
- **Subresources:** Amazon S3 uses the subresource mechanism to store object-specific additional information. Because subresources are subordinates to objects, they are always associated with some other entity such as an object or a bucket.
- **Access Control Information:** You can controls access to the objects you store in Amazon S3. Amazon S3 supports both the resource-based access control, such as an Access Control List (ACL) and bucket policies, and user-based access control.

Although you can use any UTF-8 characters in an object key name, the following key naming best practices help ensure maximum compatibility with other applications. The following character sets are generally safe for use in key names:

- **Alphanumeric characters:** [0-9, a-z, A-Z]
- **Special characters:** !, -, _, ., *, ', (, and)

The following are examples of valid object key names:

- 4my-organization
- my.great_photos-2014/jan/myvacation.jpg
- videos/2014/birthday/video1.wmv

Ansible's S3 module

Ansible comes with `s3` module which allows the user to dictate the presence of a given file in an S3 bucket. If or once the key (file) exists in the bucket, it returns a time-expired download URL.

The following are options for `s3` module (for more documentation, visit http://docs.ansible.com/s3_module.html):

parameter	required	default	choices	comments
aws_access_key	no			AWS access key. If not set then the value of the AWS_ACCESS_KEY environment variable is used aliases: ec2_access_key, access_key
aws_secret_key	no			AWS secret key. If not set then the value of the AWS_SECRET_KEY environment variable is used aliases: ec2_secret_key, secret_key
bucket	yes			Bucket name.
dest	no			The destination file path when downloading an object/key with a GET operation.
ec2_url	no			URL to use to connect to EC2 or your Eucalyptus cloud (by default the module will use EC2 endpoints). Must be specified if region is not used. If not set then the value of the EC2_URL environment variable, if any, is used
encrypt (added in 2.0)	no			When set for PUT mode, asks for server-side encryption
expiration	no	600		Time limit (in seconds) for the URL generated and returned by S3/Walrus when performing a mode=put or mode=geturl operation.
headers (added in 2.0)	no			Custom headers for PUT operation, as a dictionary of 'key=value' and 'key=value,key=value'.
marker (added in 2.0)	no			Specifies the key to start with when using list mode. Object keys are returned in alphabetical order, starting with key after the marker in order.
max_keys (added in 2.0)	no	1000		Max number of results to return in list mode, set this if you want to retrieve fewer than the default 1000 keys.
metadata	no			Metadata for PUT operation, as a dictionary of 'key=value' and 'key=value,key=value'.
mode	yes			Switches the module behaviour between put

parameter	required	default	choices	comments
				(upload), get (download), geturl (return download url), getstr (download object as string), list (list keys, Ansible 2.0+), create (bucket), delete (bucket), and delobj (delete object, Ansible 2.0+).
object	no			Keyname of the object inside the bucket. Can be used to create “virtual directories”.
overwrite	no	always		Force overwrite either locally on the filesystem or remotely with the object/key. Used with PUT and GET operations. Boolean or one of [always, never, different], true is equal to ‘always’ and false is equal to ‘never’, new in 2.0
permission (added in 2.0)	no	private		This option lets the user set the canned permissions on the object/bucket that are created. The permissions that can be set are ‘private’, ‘public-read’, ‘public-read-write’, ‘authenticated-read’. Multiple permissions can be specified as a list.
prefix (added in 2.0)	no			Limits the response to keys that begin with the specified prefix for list mode
profile (added in 1.6)	no			Uses a boto profile. Only works with boto >= 2.24.0
region (added in 1.8)	no			The AWS region to use. If not specified then the value of the AWS_REGION or EC2_REGION environment variable are checked, followed by the aws_region and ec2_region settings in the Boto config file. If none of those are set the region defaults to the S3 Location: US Standard. aliases: aws_region, ec2_region
retries (added in 2.0)	no			On recoverable failure, how many times to retry before actually failing.
s3_url	no			S3 URL endpoint for usage with Eucalyptus, fakes3, etc. Otherwise assumes AWS. aliases: S3_URL
security_token (added in 1.6)	no			AWS STS security token. If not set then the value of the AWS_SECURITY_TOKEN or EC2_SECURITY_TOKEN environment variable is used. aliases: access_token
src	no			The source file path when performing a PUT

parameter	required	default	choices	comments
				operation.
validate_certs (added in 1.5)	no	yes	yes no	When set to "no", SSL certificates will not be validated for boto versions >= 2.6.0.
version (added in 2.0)	no			Version ID of the object inside the bucket. Can be used to get a specific version of a file if versioning is enabled in the target bucket.

Creating S3 bucket

This following playbook will create an empty bucket called yan001:

```
$ cd /home/yan/ansible4aws$ vi s3_create_bucket.yml

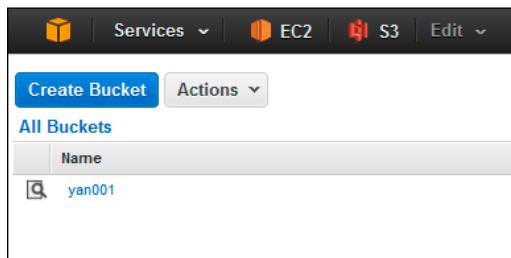
---
- hosts: localhost
  gather_facts: no
  connection: local
  vars:
    bucketname: yan001
  tasks:
    - name: create an S3 bucket
      s3: bucket: "{{ bucketname }}"
      mode: create
```



The S3 bucket namespace is shared by all users of the system. You have to choose a unique name for the bucket. Creating a bucket using a name that is already used by other user will produce an error:

BucketAlreadyExists: The requested bucket name is not available. The bucket namespace is shared by all users of the system. Please select a different name and try again.

After successful playbook execution you can see the new bucket in the S3 console: <https://console.aws.amazon.com/s3>.



The S3 console

Virtual directory

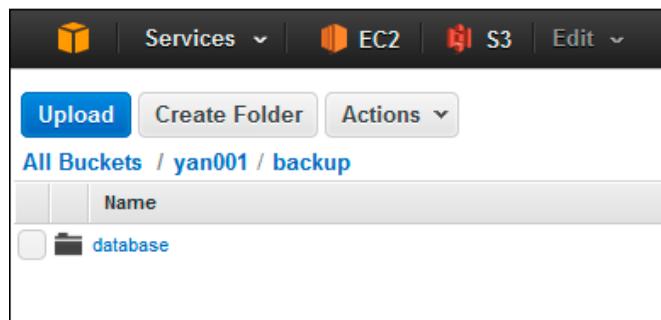
The Amazon S3 data model is a flat structure: you create a bucket, and the bucket stores objects. There is no hierarchy of subbuckets or subfolders; however, you can infer logical hierarchy using keyname prefixes and delimiters as the Amazon S3 console does. The Amazon S3 console supports a concept of folders. For more information, <http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingMetadata.html>

A playbook to create virtual directory:

```
$ vi s3_create_dir.yml

---
- hosts: localhost
  gather_facts: no
  connection: local
  vars:
    bucketname: yan001
  tasks:
    - name: create a virtual directory
      s3:
        bucket: "{{ bucketname }}"
        object: /backup/database/
        mode: create
```

This is actually an object key called `/backup/database/` but in S3 console it will be displayed as a directory and sub directory.



Virtual directory

Uploading File

Let's create a text file and upload it to S3:

```
$ vi test.txt
```

Write some text:

```
Hello World
```

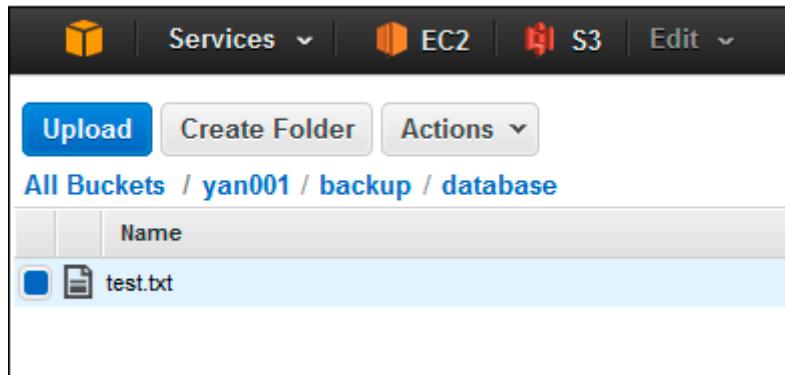
Save the file and upload it using the following playbook:

```
$ vi s3_upload_file.yml
```

```
---
```

```
- hosts: localhost
  gather_facts: no
  connection: local
  vars:
    bucketname: yan001
  tasks:
    - name: upload file
      s3:
        bucket: "{{ bucketname }}"
        object: /backup/database/test.txt
        src: test.txt
        mode: put
```

The preceding playbook will upload `test.txt` to `yan001` bucket and in S3 console it will be displayed as a file inside `/backup/database/` directory.



You can right click on the file and select **Open** to see the content of the file in a new window.

Let's try to change the content of our text file and upload it again:

```
$ vi test.txt
```

Add some text:

```
Hello World. This is a test.  
$ ansible-playbook -i hosts s3_upload_file.yml
```

This will overwrite the previous `test.txt` because the default option for `s3` module is `overwrite=True`. If you open the file in S3 console, it will display `Hello World. This is a test.`

To change this behavior and avoid overwriting object in S3 we need to add option `overwrite=false` in the playbook.

```
$ vi s3_upload_file.yml
```

```
---  
- hosts: localhost  
  gather_facts: no  
  connection: local  
  vars:
```

```
bucketname: yan001
tasks:
- name: upload file
  s3:
    bucket: "{{ bucketname }}"
    object: /backup/database/test.txt
    src: test.txt
    overwrite: false
    mode: put
```

The preceding playbook will produce an error message if we try to put object with different content but using the same key/name as object that already exist in our S3 bucket.

Sharing a file

Using the S3 module, you can generate a download URL to share an object to anyone. You can also set the expiration time for the URL generated.

The following playbook will generate a URL for our `test.txt` file and set the expiration time to 3600 seconds (1 hour):

```
$ vi s3_share_file.yml

---
- hosts: localhost
  gather_facts: no
  connection: local
  vars:
    bucketname: yan001
  tasks:
    - name: share file
      s3:
        bucket: "{{ bucketname }}"
        object: /backup/database/test.txt
        expiration: 3600
        mode: geturl
```

Execute the playbook using `-v` option to see the generated URL:

```
$ ansible-playbook -i hosts -v s3_share_file.yml
```

You can use the URL generated to open/download the file from a web browser. The URL will expire in 1 hour.

Downloading file

To download a file from S3 bucket to a destination in Ansible machine we can use the following playbook.

```
$ vi s3_download_file.yml

---
- hosts: localhost
  gather_facts: no
  connection: local
  become: yes
  vars:
    bucketname: yan001
  tasks:
    - name: download file
      s3:
        bucket: "{{ bucketname }}"
        object: /backup/database/test.txt
        dest: test1.txt
        mode: get
```

The default option is `overwrite=True`. To avoid overwriting file with same name in the destination path, set the option `overwrite` to `false` or `no`.

Deleting S3 bucket

To delete an S3 bucket and all of its contents we can use the following playbook:

```
$ vi s3_delete_bucket.yml

---
- hosts: localhost
  gather_facts: no
  connection: local
  vars:
    bucketname: yan001
  tasks:
    - name: delete an S3 bucket and all of its contents
      s3:
        bucket: "{{ bucketname }}"
        mode: delete
```

Backup storage

This section will give you an example of how to use S3 as a backup storage. We will create MySQL database backup from the example WordPress site we have created in Chapter 4, Project 1: A WordPress Site, and store the backup in an S3 bucket.

The following playbook will backup our WordPress MySQL database, archive it, and store the file in an S3 bucket:

```
$ cd /home/yan/ansible4aws/wordpress $ vi backup.yml

---
- name: backup database and store in S3
  hosts: tag_class_wordpress
  gather_facts: no
  vars:
    bucketname: yan_wordpress
  tasks:
    - name: get date
      shell: date +%Y%m%d
      register: date
    - name: create mysql backup
      shell: mysqldump -u {{ wp_db_user }} -p{{ wp_db_password }} {{ wp_db_name }} > /tmp/{{ date.stdout }}backup.sql
    - name: archive backup
      shell: tar -czf {{ date.stdout }}backup.tar.gz {{ date.stdout }}backup.sql && rm -f {{ date.stdout }}backup.sql
      chdir=/tmp
    - name: create s3 bucket
      local_action:
        module: s3
        bucket: "{{ bucketname }}"
        mode: create
    - name: upload backup archive
      local_action:
        module: s3
        bucket: "{{ bucketname }}"
        object: /backup/database/{{ date.stdout }}backup.tar.gz
        src: /tmp/{{ date.stdout }}backup.tar.gz
        mode: put
```

Run the playbook:

```
$ ansible-playbook -i ec2.py backup.yml
```



To perform backup regularly, you could create a daily (or weekly) cron job on the Ansible machine to run the preceding ansible-playbook command from /home/yan/ansible4aws/wordpress/ directory.

To restore the backup, use the following playbook:

```
$ vi restore.yml

---

- name: download backup from S3 and restore
  hosts: tag_class_wordpress
  gather_facts: no
  vars:
    bucketname: yan_wordpress
    date: 20140920
  tasks:
    - name: download backup archive
      local_action:
        module: s3
        bucket: "{{ bucketname }}"
        object: /backup/database/{{ date }}_backup.tar.gz
        dest: /tmp/{{ date }}_backup.tar.gz
        mode: get
    - name: extract archive and restore mysql backup
      shell: tar -xzf {{ date }}_backup.tar.gz &&
            mysql -u {{ wp_db_user }} -p{{ wp_db_password }}
            {{ wp_db_name }} < {{ date }}_backup.sql
            chdir=/tmp
```

To run the playbook:

```
$ ansible-playbook -i ec2.py restore.yml
```

Using s3cmd with Ansible

Sometimes using Ansible S3 module alone is not enough. For example to delete an object in S3 bucket, I could not find a way to do that using S3 module. May be it will have this feature in the future version. One way to deal with this problem is to use S3 command line tool which can be executed from Ansible using command module. An example of this tool is s3cmd. For more information, visit <http://s3tools.org/s3cmd>.

s3cmd is a free command line tool and client for uploading, retrieving and managing data in Amazon S3 and other cloud storage service providers that use the S3 protocol, such as Google Cloud Storage or DreamHost DreamObjects.

To install s3cmd:

```
$ sudo yum --enablerepo epel-testing install s3cmd
```

To configure s3cmd:

```
$ s3cmd --configure
```

Enter your AWS credentials and just press *Enter* for other prompts. For more information on s3cmd usage visit <http://s3tools.org/usage>.

The following playbook shows you example of how to use s3cmd with Ansible:

```
$ cd /home/yan/ansible4aws/wordpress
$ vi delete_backup.yml

---
- name: delete object from S3 bucket
  hosts: localhost
  gather_facts: no
  vars:
    bucketname: yan_wordpress
    date: 20141031
  tasks:
    - name: delete backup file
      command: s3cmd del s3://{{ bucketname }}/backup/database/{{ date
}}_backup.tar.gz
      ignore_errors: yes
```

Run the playbook:

```
$ ansible-playbook -i hosts delete_backup.yml
```

It will delete file /backup/database/20141031_backup.tar.gz from yan_wordpress bucket.

9

Using AWS CLI with Ansible

The **AWS Command Line Interface (CLI)** is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts. For more information, visit <http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html>.

The AWS CLI can be used with the following services (with the link to the reference guide):

- **acm:** <http://docs.aws.amazon.com/cli/latest/reference/acm/index.html>
- **apigateway:** <http://docs.aws.amazon.com/cli/latest/reference/apigateway/index.html>
- **application-autoscaling:** <http://docs.aws.amazon.com/cli/latest/reference/application-autoscaling/index.html>
- **autoscaling:** <http://docs.aws.amazon.com/cli/latest/reference/autoscaling/index.html>
- **cloudformation:** <http://docs.aws.amazon.com/cli/latest/reference/cloudformation/index.html>
- **cloudfront:** <http://docs.aws.amazon.com/cli/latest/reference/cloudfront/index.html>
- **cloudhsm:** <http://docs.aws.amazon.com/cli/latest/reference/cloudhsm/index.html>
- **cloudsearch:** <http://docs.aws.amazon.com/cli/latest/reference/cloudsearch/index.html>
- **cloudsearchdomain:** <http://docs.aws.amazon.com/cli/latest/reference/cloudsearchdomain/index.html>
- **cloudtrail:** <http://docs.aws.amazon.com/cli/latest/reference/cloudtrail/index.html>
- **cloudwatch:** <http://docs.aws.amazon.com/cli/latest/reference/cloudwatch/index.html>

- **codecommit:** <http://docs.aws.amazon.com/cli/latest/reference/codecommit/index.html>
- **codepipeline:** <http://docs.aws.amazon.com/cli/latest/reference/codepipeline/index.html>
- **cognito-identity:** <http://docs.aws.amazon.com/cli/latest/reference/cognito-identity/index.html>
- **cognito-idp:** <http://docs.aws.amazon.com/cli/latest/reference/cognito-idp/index.html>
- **cognito-sync:** <http://docs.aws.amazon.com/cli/latest/reference/cognito-sync/index.html>
- **configservice:** <http://docs.aws.amazon.com/cli/latest/reference/configservice/index.html>
- **configure:** <http://docs.aws.amazon.com/cli/latest/reference/configure/index.html>
- **datapipeline:** <http://docs.aws.amazon.com/cli/latest/reference/datapipeline/index.html>
- **deploy:** <http://docs.aws.amazon.com/cli/latest/reference/deploy/index.html>
- **devicefarm:** <http://docs.aws.amazon.com/cli/latest/reference/devicefarm/index.html>
- **directconnect:** <http://docs.aws.amazon.com/cli/latest/reference/directconnect/index.html>
- **discovery:** <http://docs.aws.amazon.com/cli/latest/reference/discovery/index.html>
- **dms:** <http://docs.aws.amazon.com/cli/latest/reference/dms/index.html>
- **ds:** <http://docs.aws.amazon.com/cli/latest/reference/ds/index.html>
- **dynamodb:** <http://docs.aws.amazon.com/cli/latest/reference/dynamodb/index.html>
- **dynamodbstreams:** <http://docs.aws.amazon.com/cli/latest/reference/dynamodbstreams/index.html>
- **ec2:** <http://docs.aws.amazon.com/cli/latest/reference/ec2/index.html>
- **ecr:** <http://docs.aws.amazon.com/cli/latest/reference/ecr/index.html>
- **ecs:** <http://docs.aws.amazon.com/cli/latest/reference/ecs/index.html>
- **efs:** <http://docs.aws.amazon.com/cli/latest/reference/efs/index.html>
- **elasticache:** <http://docs.aws.amazon.com/cli/latest/reference/elasticache/index.html>
- **elasticbeanstalk:** <http://docs.aws.amazon.com/cli/latest/reference/elasticbeanstalk/index.html>
- **elastictranscoder:** <http://docs.aws.amazon.com/cli/latest/reference/elastictranscoder/index.html>

- **ictranscoder:** <http://docs.aws.amazon.com/cli/latest/reference/ictranscoder/index.html>
- **elb:** <http://docs.aws.amazon.com/cli/latest/reference/elb/index.html>
- **emr:** <http://docs.aws.amazon.com/cli/latest/reference/emr/index.html>
- **es:** <http://docs.aws.amazon.com/cli/latest/reference/es/index.html>
- **events:** <http://docs.aws.amazon.com/cli/latest/reference/events/index.html>
- **firehose:** <http://docs.aws.amazon.com/cli/latest/reference/firehose/index.html>
- **gamelift:** <http://docs.aws.amazon.com/cli/latest/reference/gamelift/index.html>
- **glacier:** <http://docs.aws.amazon.com/cli/latest/reference/glacier/index.html>
- **iam:** <http://docs.aws.amazon.com/cli/latest/reference/iam/index.html>
- **importexport:** <http://docs.aws.amazon.com/cli/latest/reference/importexport/index.html>
- **inspector:** <http://docs.aws.amazon.com/cli/latest/reference/inspector/index.html>
- **iot:** <http://docs.aws.amazon.com/cli/latest/reference/iot/index.html>
- **iot-data:** <http://docs.aws.amazon.com/cli/latest/reference/iot-data/index.html>
- **kinesis:** <http://docs.aws.amazon.com/cli/latest/reference/kinesis/index.html>
- **kms:** <http://docs.aws.amazon.com/cli/latest/reference/kms/index.html>
- **lambda:** <http://docs.aws.amazon.com/cli/latest/reference/lambda/index.html>
- **logs:** <http://docs.aws.amazon.com/cli/latest/reference/logs/index.html>
- **machinelearning:** <http://docs.aws.amazon.com/cli/latest/reference/machinelearning/index.html>
- **marketplacecommerceanalytics:** <http://docs.aws.amazon.com/cli/latest/reference/marketplacecommerceanalytics/index.html>
- **meteringmarketplace:** <http://docs.aws.amazon.com/cli/latest/reference/meteringmarketplace/index.html>
- **opsworks:** <http://docs.aws.amazon.com/cli/latest/reference/opsworks/index.html>
- **rds:** <http://docs.aws.amazon.com/cli/latest/reference/rds/index.html>
- **redshift:** <http://docs.aws.amazon.com/cli/latest/reference/redshift/index.html>
- **route53:** <http://docs.aws.amazon.com/cli/latest/reference/route53/index.html>

- **route53domains:** <http://docs.aws.amazon.com/cli/latest/reference/route53domains/index.html>
- **s3:** <http://docs.aws.amazon.com/cli/latest/reference/s3/index.html>
- **s3api:** <http://docs.aws.amazon.com/cli/latest/reference/s3api/index.html>
- **sdb:** <http://docs.aws.amazon.com/cli/latest/reference/sdb/index.html>
- **servicecatalog:** <http://docs.aws.amazon.com/cli/latest/reference/servicecatalog/index.html>
- **ses:** <http://docs.aws.amazon.com/cli/latest/reference/ses/index.html>
- **sns:** <http://docs.aws.amazon.com/cli/latest/reference/sns/index.html>
- **sqs:** <http://docs.aws.amazon.com/cli/latest/reference/sqs/index.html>
- **ssm:** <http://docs.aws.amazon.com/cli/latest/reference/ssm/index.html>
- **storagegateway:** <http://docs.aws.amazon.com/cli/latest/reference/storagegateway/index.html>
- **sts:** <http://docs.aws.amazon.com/cli/latest/reference/sts/index.html>
- **support:** <http://docs.aws.amazon.com/cli/latest/reference/support/index.html>
- **swf:** <http://docs.aws.amazon.com/cli/latest/reference/swf/index.html>
- **waf:** <http://docs.aws.amazon.com/cli/latest/reference/waf/index.html>
- **workspaces:** <http://docs.aws.amazon.com/cli/latest/reference/workspace/index.html>

You can see from this list that AWS CLI supports a lot more AWS services than Ansible does. By combining Ansible and AWS CLI you'll have the most powerful tools to control and manage multiple AWS services.

In this chapter I will show you some examples of using AWS CLI in our Ansible playbook. I won't cover all CLI commands and services, but after you see the examples you'll get the idea and you can learn other CLI commands from the reference guides and use them to suit your needs.

Installing the AWS CLI

You can install AWS CLI from pip:

```
$ sudo pip install awscli
```

To upgrade an existing awscli installation:

```
$ sudo pip install --upgrade awscli
```

Check AWS CLI installation:

```
$ aws --version
```

Configuring the AWS CLI

Lets see how to configure different components of AWS CLI:

Configuration settings and precedence

To connect to any of the supported services with the AWS CLI, you must provide your AWS credentials. The AWS CLI uses a *provider chain* to look for AWS credentials in a number of different places, including system or user environment variables and local AWS configuration files. For more documentation, please visit <http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html>.

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

- Environment Variables – `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
- The AWS credential profiles file – located at `~/.aws/credentials`. This file can contain multiple named profiles in addition to a default profile.
- The CLI configuration file – typically located at `~/.aws/config`. This file can contain a default profile, named profiles, and CLI specific configuration parameters for each.
- Instance profile credentials – these credentials can be used on EC2 instances with an assigned instance role, and are delivered through the Amazon EC2 metadata service.

Configuring credentials

Setting your credentials for use by the AWS CLI can be done in a number of ways, but here are the recommended approaches:

- Use the AWS CLI to set credentials with the following command:

```
$ aws configure
```

- Enter your access key and secret key when prompted. Pressing the *Enter* key without typing a value will keep any previously configured value or assign a default if no value currently exists.



Use `aws configure --profile PROFILE_NAME` to configure a named profile. The CLI will store credentials that are specified with `aws configure` in a local file, typically `~/.aws/config`. If a profile is configured in both this file and the AWS credentials file, the profile in the AWS credentials file will take precedence. For information about using a named profile when executing an AWS CLI command, see <http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html#using-profiles>.

- The default CLI config file location can be overridden by setting the `AWS_CONFIG_FILE` environmental variable to another local path. If this variable is set, `aws configure` will write to the specified file, and the CLI will attempt to read profiles from there instead of the default path. Regardless of the location of the config file, if a credentials file exists, it takes precedence when the CLI looks for credentials.
- Set credentials in the AWS credentials profile file on your local system, located at:

`~/.aws/credentials`.

- This file should contain lines in the following format:

```
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key
```

- Set the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables. To set these variables, use `export`:

```
export AWS_ACCESS_KEY_ID=your_access_key_id
export AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

- To use the CLI from an EC2 instance, create a role that has access to the resources needed and assign that role to the instance when it is launched. Credentials will be configured automatically and CLI commands will work without any additional setup.



The AWS CLI does not allow you to specify credentials on an AWS CLI command (aws s3 ..., and so forth). You must provide credentials using one of the preceding methods before running any AWS CLI commands.

Configuring the AWS Region

You must specify an AWS region when using the AWS CLI. For a list of services and available regions, see <http://docs.aws.amazon.com/general/latest/gr/rande.html>.

To specify the region, you have the following options:

- Configure the region setting using the `aws configure` command:

```
$ aws configure
```

- Here's an example where the user is changing the default region from `us-west-2` to `us-east-1`.

```
AWS Access Key ID [*****]:  
AWS Secret Access Key [*****]:  
Default region name [us-west-2]: us-east-1  
Default output format [None]:
```

- Specify the region in the `AWS_DEFAULT_REGION` environment variable.
- Use the `--region` option with an AWS CLI command. The following example lists the Amazon SQS queues for the `us-west-2` region.

```
$ aws sqs list-queues --region us-west-2
```

Using the AWS CLI

This section describes how to use AWS CLI from terminal.

Getting help with the AWS CLI

To get help when using the AWS CLI, you can simply add help to the end of a command. For example, the following command lists help for the general AWS CLI options and the available top level commands.

```
$ aws help
```

The following command lists the available subcommands for Amazon EC2.

```
$ aws ec2 help
```

The next example lists the detailed help for the EC2 `describe-instances` operation, including descriptions of its input parameters, filters, and output. Check the examples section of the help if you are not sure how to phrase a command.

```
$ aws ec2 describe-instances help
```

Synopsis:

```
aws [options] <command> <subcommand> [parameters]
```

Use *aws command help* for information on a specific command.

Options:

```
--debug (boolean)
```

Turn on debug logging.

```
--endpoint-url (string)
```

Override command's default URL with the given URL.

```
--no-verify-ssl (boolean)
```

By default, the AWS CLI uses SSL when communicating with AWS services. For each SSL connection, the AWS CLI will verify SSL certificates. This option overrides the default behavior of verifying SSL certificates.

```
--no-paginate (boolean)
```

Disable automatic pagination.

```
--output (string)
```

The formatting style for command output.

- json
- text
- table

--query (string)

A JMESPath query to use in filtering the response data.

--profile (string)

Use a specific profile from your credential file.

--region (string)

The region to use. Overrides config/env settings.

--version (string)

Display the version of this tool.

--color (string)

Turn on/off color output.

- on
- off
- auto

--no-sign-request (boolean)

Do not sign requests. Credentials will not be loaded if this argument is provided.

AWS CLI in Ansible playbook

You will learn how to get VPC Information based on name tag.

Getting VPC information based on name tag

In Chapter 6, *VPC Provisioning with Ansible*, we created Ansible module to get VPC ID based on its Name Tag. In this section we will use AWS CLI to achieve the same result.

The command we will use to get VPC information is `aws ec2 describe-vpcs`. You can go to the reference guide for `aws ec2` <http://docs.aws.amazon.com/cli/latest/reference/ec2/index.html> and see the available commands. The `describe-vpcs` command is perfect for our need to get the VPC ID based on its Name tag.

You can see the complete reference guide for this command here: <http://docs.aws.amazon.com/cli/latest/reference/ec2/describe-vpcs.html>. We'll use the `--filters` option to filter VPC using Name tag.

Try this command from terminal:

```
$ aws ec2 describe-vpcs --filters Name=tag:Name,Values=staging_vpc
```

(If you don't have a VPC with tags `Name=staging_vpc`, create one, or use any VPC you have, and change the `values` to match your VPC Name tag)

The command output:

```
{
  "Vpcs": [
    {
      "VpcId": "vpc-xxxxxxxx",
      "InstanceTenancy": "default",
      "Tags": [
        {
          "Value": "staging_vpc",
          "Key": "Name"
        }
      ],
      "State": "available",
      "DhcpOptionsId": "dopt-xxxxxxxx",
      "CidrBlock": "10.0.0.0/16",
      "IsDefault": false
    }
  ]
}
```

The output is in default JSON format and you can see that it contains the VPC ID we wanted. To get the VPC ID only, we can use --query option to filter the command output.

```
$ aws ec2 describe-vpcs --filters Name=tag:Name,Values=staging_vpc  
--query 'Vpcs[0].VpcId' --output text
```

The command output:

```
vpc-xxxxxxx
```

Perfect! Now we can use this command in our playbook to get the same result as the `vpc_lookup` module we used in Chapter 6, *VPC Provisioning with Ansible*.

The following playbook can be used to delete a VPC with specific `Name` tag (in this example `Name=test-vpc`):

```
$ vi vpc_delete1.yml
```

```
---  
- hosts: localhost  
  connection: local  
  gather_facts: no  
  vars:  
    region: ap-southeast-2  
    name: test-vpc  
  tasks:  
    - name: get vpc id  
      command: "aws ec2 describe-vpcs --filters Name=tag:Name, Values={{  
name }}"  
      --query 'Vpcs[0].VpcId' --output text"  
      register: vpcid  
    - name: delete vpc  
      ec2_vpc:  
        region: "{{ region }}"  
        state: absent  
        vpc_id: "{{ vpcid.stdout }}"  
        wait: yes
```

While writing Ansible module is fun, it takes time. Using this AWS CLI approach we can quickly add more AWS functions/services to our Ansible playbook.

Create a DHCP options set and associate it with a VPC

This important feature is also not supported yet in Ansible module. The *DHCP options set* provides configuration information to hosts in the VPC associated with it. Some configuration parameters we can set in DHCP options set are the domain name, domain name server, ntp servers, netbios name servers and the netbios node type.

When you create a VPC, Amazon automatically create a set of DHCP options and associate them with the VPC. This set includes two options: `domain-name-servers=AmazonProvidedDNS`, and `domain-name=*domain-name-for-your-region*`. **AmazonProvidedDNS** is an Amazon DNS server, and this option enables DNS for instances that need to communicate over the VPC's Internet gateway.

Let's say we want to use our own DNS server and configure domain name and domain name servers in DHCP options. We can use AWS CLI within our Ansible playbook to automate all tasks.

Example parameters:

- `domain-name: example.com`
- `domain-name-servers: 10.0.0.7, 10.0.0.8`

The AWS CLI command we will use to create DHCP options set is `create-dhcp-options`. You can read the reference guide here:

<http://docs.aws.amazon.com/cli/latest/reference/ec2/create-dhcp-options.html>.

We will also use `associate-dhcp-options` command and the reference guide is here:

<http://docs.aws.amazon.com/cli/latest/reference/ec2/associate-dhcp-options.html>.

In the following playbook we will create the DHCP options set and associate it with our VPC, which has a tag `Name=test-vpc`.

```
$ vi dhcp_options.yml

---
- hosts: localhost
  connection: local
  gather_facts: no
  vars:
    region: ap-southeast-2
    name: test-vpc
  tasks:
```

```
- name: create dhcp options set
  command: aws ec2 create-dhcp-options --dhcp-configuration
            "Key=domain-name,Values=example.com"
            "Key=domain-name-servers,Values=10.0.0.7,10.0.0.8"
            --query 'DhcpOptions.DhcpOptionsId' --output text
  register: dopt

- name: get vpc id
  command: "aws ec2 describe-vpcs --filters Name>tag:Name,Values={{ name }}"
            --query 'Vpcs[0].VpcId' --output text"
  register: vpcid

- name: associate vpc with dhcp options set
  command: aws ec2 associate-dhcp-options --dhcp-options-id {{ dopt.stdout }}
            --vpc-id {{ vpcid.stdout }}
```

This playbook is not idempotent yet. A new DHCP options set will be created and associated with `test-vpc` every time the playbook is executed.

Idempotent EC2 instance creation based on Name tag

The following example will show you how to use Name tag to maintain idempotency in EC2 instance creation. The idea is to use unique Name tag for each instance (may be the same name as its hostname) and check first if any instance with particular Name tag already exists before creating a new instance.

We will use `describe-instances` command to check if any instance with particular Name tag already exists. You can find the reference guide for this command here: <http://docs.aws.amazon.com/cli/latest/reference/ec2/describe-instances.html>.

```
$ vi ec2_check_name.yml

---
- hosts: localhost
  gather_facts: no
  connection: local
  vars:
    region: ap-southeast-2
    key: yan-key-pair-apsydne
    type: t2.micro
    image: ami-d9fe9be3
```

```
sg: sg_webserver_apsydney
name: test-01
tasks:
- name: check if instance with name tag exists
  command: aws ec2 describe-instances
            --filter Name=tag:Name,Values={{ name }}
            --query 'Reservations[0].Instances[0].InstanceId' --output
text
  register: instanceid

- name: create EC2 if not exists
  ec2:
    region: "{{ region }}"
    key_name: "{{ key }}"
    instance_type: "{{ type }}"
    image: "{{ image }}"
    group: "{{ sg }}"
    instance_tags:
      Name: "{{ name }}"
    wait: yes
  when: instanceid.stdout=="None"
```

The command:

```
aws ec2 describe-instances --filter Name=tag:Name,Values={{ name }}
--query 'Reservations[0].Instances[0].InstanceId' --output text
```

will return the Instance ID of the EC2 instance with tag Name=test-01 if it exists and return None if it doesn't exist. The EC2 creation play use this condition to avoid creation if any EC2 instance with tag Name=test-01 already exists.

Start or stop an instance with particular Name tag

We can use the same command describe-instances to achieve this purpose.

```
$ vi ec2_start.yml

---
- hosts: localhost
  gather_facts: no
  connection: local
  vars:
    region: ap-southeast-2
    name: test-01
  tasks:
    - name: get instance id
```

```
command: aws ec2 describe-instances
          --filter Name=tag:Name,Values={{ name }}
          --query 'Reservations[0].Instances[0].InstanceId' --output
text
      register: instanceid

- name: stop instance
  ec2:
    region: "{{ region }}"
    instance_ids: "{{ instanceid.stdout }}"
    state: stopped
    wait: yes
  when: instanceid.stdout!="None"

$ vi ec2_stop.yml

---

- hosts: localhost
  gather_facts: no
  connection: local
  vars:
    region: ap-southeast-2
    name: test-01
  tasks:
    - name: get instance id
      command: aws ec2 describe-instances
              --filter Name=tag:Name,Values={{ name }}
              --query 'Reservations[0].Instances[0].InstanceId' --output
text
      register: instanceid

    - name: stop instance
      ec2:
        region: "{{ region }}"
        instance_ids: "{{ instanceid.stdout }}"
        state: stopped
        wait: yes
      when: instanceid.stdout!="None"
```

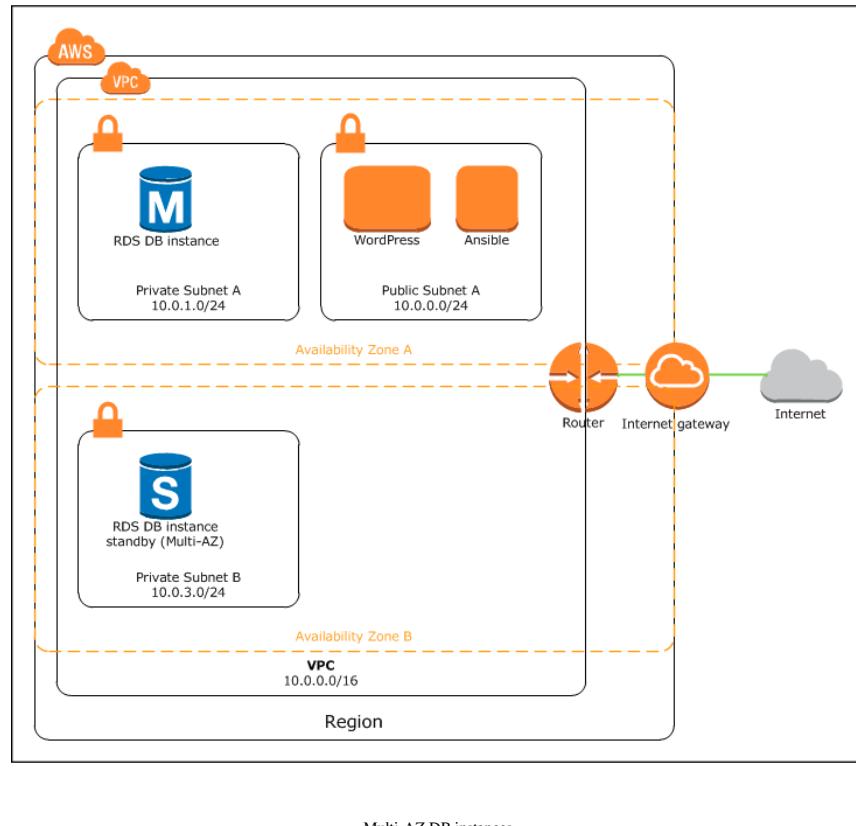
10

Project 2 - A Multi-Tier WordPress Site

In this example project, we will expand our WordPress site into a multi-tier architecture to provide high scalability and availability. With a multi-tier approach and using AWS features like Elastic Load Balancing and Auto Scaling, we can easily scale our WordPress site to handle high traffic.

The first step is to separate the database server from the web server. We will use multi-AZ RDS instance as our database server. One RDS instance will act as a standby secondary database server and synchronously replicated from the primary RDS database.

In case of primary instance failure, Amazon RDS automatically initiate a failover from the primary to the standby instance.



Multi-AZ DB instances

We'll use the Multi-AZ VPC we've created in [Chapter 6, VPC Provisioning with Ansible](#). Later on, we'll modify the VPC and security groups to add support for load balancing and Auto Scaling.

First, create a directory for this project:

```
$ cd /home/yan/ansible4aws  
$ mkdir wordpress_ha
```

Copy the external inventory script `ec2.py` and `ec2.ini` file to this project directory:

```
$ cp ec2* wordpress_ha/
```

Create group_vars/all:

```
$ mkdir group_vars  
$ vi group_vars/all  
  
---  
db_name: wordpress  
username: dbadmin  
password: mypassword
```

Multi-AZ RDS provisioning

Create subnet group, parameter group, and RDS instances. Please refer to Chapter 7, *RDS Provisioning with Ansible*.

```
$ cd wordpress_ha $ vi provisioning_rds.yml  
  
---  
- hosts: localhost  
  connection: local  
  gather_facts: no  
  vars:  
    region: ap-southeast-2  
    env: staging  
    size: 5  
    instance_type: db.t2.micro  
    db_engine: MySQL  
    engine_version: 5.6.23  
    subnet_group: dbsg_wordpress  
    param_group: wordpress  
    # staging_sg_database security group ID  
    security_groups: sg-xxxxxx  
  tasks:  
    - name: "get {{ env }}subnetprivate_0 subnet id"  
      command: "aws ec2 describe-subnets --filters Name=tag:Name,Values={{ env }}subnetprivate_0  
                --region {{ region }} --query 'Subnets[0].SubnetId' --output  
                text"  
      register: subnet0  
  
    - debug: var=subnet0.stdout  
  
    - name: "get {{ env }}_subnet_private_1 subnet id"  
      command: "aws ec2 describe-subnets --filters Name=tag:Name,Values={{ env }}_subnet_private_1  
                --region {{ region }} --query 'Subnets[0].SubnetId' --
```

```
output text"
register: subnet1

- debug: var=subnet1.stdout

- name: create Multi-AZ DB subnet group
rds_subnet_group:
  name: "{{ subnet_group }}"
  state: present
  region: "{{ region }}"
  description: DB Subnet Group for WordPress HA
  subnets:
    - "{{ subnet0.stdout }}"
    - "{{ subnet1.stdout }}"

- name: create mysql parameter group
rds_param_group:
  name: "{{ param_group }}"
  state: present
  region: "{{ region }}"
  description: MySQL Parameter Group for WordPress HA
  engine: mysql5.6
  params:
    innodb_lock_wait_timeout: 3600
    max_allowed_packet: 512M
    net_write_timeout: 300

- name: create mysql RDS instance
rds:
  command: create
  instance_name: "{{ env }}-wordpress-rds"
  region: "{{ region }}"
  size: "{{ size }}"
  instance_type: "{{ instance_type }}"
  db_engine: "{{ db_engine }}"
  engine_version: "{{ engine_version }}"
  subnet: "{{ subnet_group }}"
  parameter_group: "{{ param_group }}"
  multi_zone: yes
  db_name: "{{ db_name }}"
  username: "{{ username }}"
  password: "{{ password }}"
  vpc_security_groups: "{{ security_groups }}"
```

Execute the playbook:

```
# ansible-playbook -i hosts provisioning_rds.yml
```

After successful playbook run, open your RDS console, and note the RDS endpoint address.

Configuration Details		Security and Network	
Engine	MySQL 5.6.22	Availability Zone	ap-southeast-2b
License Model	General Public License	VPC	staging_vpc (vpc-XXXXXX)
Created Time	February 23, 2015 at 11:29:18 PM UTC+11	Subnet Group	dbsg_wordpress (Complete)
DB Name	wordpress	Subnets	subnet-XXXXXX subnet-XXXXXX
Username	dbadmin	Security Groups	staging_sg_database (sg-XXXXXX) (active)
Option Group	default:mysql-5-6 (in-sync)	Publicly Accessible	No
DB Parameter Group	wordpress (pending-reboot)	Port	3306

RDS instance

Update the `group_vars/all` file, add `dbhost`:

```
---  
db_name: wordpress  
username: dbadmin  
password: mypassword  
dbhost: staging-wordpress-rds.ap-southeast-2.rds.amazonaws.com
```

Master WordPress instance

Next, we'll create our master WordPress instance. All WordPress administration task will be done and tested on this instance. Then we will create an AMI from this master instance and we will use this AMI to launch WordPress instances in an *Auto Scaling* group.

Provision EC2 instance for master WordPress:

```
$ cd /home/yan/ansible4aws/wordpress_ha  
$ vi provisioning_wp.yml  
  
---  
- hosts: localhost  
  connection: local
```

```
gather_facts: no
vars:
    #your region
    region: ap-southeast-2
    keyname: wordpress-apsydney
    instance_type: t2.micro
    env: staging
    image: ami-d9fe9be3
    ins_name: wordpress_master
tasks:
    - name: "get {{ env }}subnetpublic_0 subnet id"
        command: "aws ec2 describe-subnets
                    --region {{ region }}
                    --filters Name=tag:Name,Values={{ env }}subnetpublic_0
                    --query 'Subnets[0].SubnetId' --output text"
        register: subnet0

    - name: launch ec2 instance
        ec2:
            region: "{{ region }}"
            key_name: "{{ keyname }}"
            instance_type: "{{ instance_type }}"
            image: "{{ image }}"
            wait: yes
            group: "{{ env }}sgweb"
            id: wordpress_ha_8
            instance_tags:
                Name: "{{ ins_name }}"
                class: wordpress_ha
                vpc_subnet_id: "{{ subnet0.stdout }}"
            register: ec2
            when: subnet0.stdout!="None"

    - name: check EIP association
        command: "aws ec2 describe-instances
                    --region {{ region }}
                    --filters Name=tag:Name,Values={{ ins_name }}
                    --query
'Reservations[0].Instances[0].NetworkInterfaces[0].Association'
                    --output text"
        register: eip

    - name: associate new EIP for the instance
        ec2_eip:
            region: "{{ region }}"
            instance_id: "{{ item.id }}"
        with_items: ec2.instances
        when: item.id is defined and eip.stdout=="None"
```

The roles:

```
$ cd wordpress_ha/roles
$ mkdir web wordpress
$ mkdir web/tasks
$ mkdir wordpress/tasks wordpress/templates

$ vi web/tasks/main.yml

---
- name: install apache, php, and php-mysql
  yum: name={{ item }} state=present
  with_items:
    - httpd
    - php
    - php-mysql
- name: start and enable httpd
  service: name=httpd state=started enabled=yes
```

Copy the WordPress configuration file from [Chapter 4, Project 1: A WordPress Site](#), and change the database setting to point to your RDS instance and to use the correct database and user information.

```
$ cd /home/yan/ansible4aws/wordpress
$ cp roles/wordpress/templates/wp-config.php
..../wordpress_ha/roles/wordpress/templates/
$ cd ../wordpress_ha
$ vi roles/wordpress/templates/wp-config.php

<?php
/**
 * The base configurations of the WordPress.
 *
 * This file has the following configurations: MySQL settings, Table
Prefix,
 * Secret Keys, WordPress Language, and ABSPATH. You can find more
information
 * by visiting {@link http://codex.wordpress.org/Editing_wp-config.php
Editing
 * wp-config.php} Codex page. You can get the MySQL settings from your web
host.
 *
 * This file is used by the wp-config.php creation script during the
 * installation. You don't have to use the web site, you can just copy this
file
 * to "wp-config.php" and fill in the values.
 *
 * @package WordPress
```

```
*/  
  
// ** MySQL settings - You can get this info from your web host ** //  
/** The name of the database for WordPress */  
define('DB_NAME', '{{ db_name }}');  
  
/** MySQL database username */  
define('DB_USER', '{{ username }}');  
  
/** MySQL database password */  
define('DB_PASSWORD', '{{ password }}');  
  
/** MySQL hostname */  
define('DB_HOST', '{{ dbhost }}');
```

And the roles for WordPress:

```
$ vi roles/wordpress/tasks/main.yml  
  
----  
- name: download wordpress  
  get_url: url=http://wordpress.org/wordpress-{{ wp_version }}.tar.gz  
    dest=~/wordpress-{{ wp_version }}.tar.gz  
  
- name: extract wordpress archive  
  command: chdir=~/bin/tar xvf wordpress-{{ wp_version }}.tar.gz  
    creates=~/wordpress  
  
- name: copy wordpress to apache root directory  
  shell: cp -r ~/wordpress/* /var/www/html  
  
- name: fetch random salts for wordpress config  
  local_action: command curl https://api.wordpress.org/secret-key/1.1/salt/  
    register: "wp_salt"  
  
- name: copy wordpress config file  
  template: src=wp-config.php dest=/var/www/html/  
  
- name: change ownership of wordpress installation  
  file: path=/var/www/html/ owner=apache group=apache state=directory  
  recurse=yes
```

Playbook for WordPress installation:

```
$ cd /home/yan/ansible4aws/wordpress_ha  
$ vi site.yml
```

```
----
```

```
- name: install apache, php, wordpress
  hosts: tag_class_wordpress_ha
  sudo: yes

  roles:
    - web
    - wordpress
```

The project's directory structure should look like this:

```
/home/
/yan/
ansible4aws/
wordpress_ha/
  ec2.ini
  ec2.py
  provisioning_rds.yml
  provisioning_wp.yml
  site.yml
  group_vars/
    all
  roles/
    web/
      tasks/
        main.yml
    wordpress/
      tasks/
        main.yml
      templates/
        wp-config.php
```

Run our `site.yml` playbook:

```
# ansible-playbook -i ec2.py site.yml
```

By the end of this playbook run, you will have the master WordPress site up and running. Start your web browser and type the public IP address of your master WordPress instance. You'll land on WordPress `wp-admin/install.php` page where you can set your admin username and password.

Next steps

In this chapter we have architected the Database Tier with high availability using Multi-AZ RDS. To improve the database performance, you can add RDS *Read Replicas*.

Amazon RDS Read Replicas provide enhanced performance and durability for Database Instances. This replication feature makes it easy to elastically scale out beyond the capacity constraints of a single DB Instance for read-heavy database workloads. You can create one or more replicas of a given source DB Instance and serve high volume application read traffic from multiple copies of your data, thereby increasing aggregate read throughput. Read replicas can also be promoted, so that they become standalone DB Instances. For more information, visit <http://aws.amazon.com/rds/details/read-replicas/>.

Amazon VPC Security Groups provide security for our Database Tier. Security groups we used in this chapter only allow access from instances with web security groups (`staging_sg_web`) attached. In the next step we need to add inbound rule to allow access from auto-scaled instances.

In the next chapters we will continue scaling our site. We will add Elastic Load Balancer and Auto Scaling feature so our site will be able to scale out and scale in dynamically depending upon traffic.

11

Amazon Machine Images - AMI

In Chapter 10, *Project 2 – A Multi-Tier WordPress Site*, we have created our master WordPress instance which we will use as a reference for our autoscaled instances. After the master WordPress has been configured properly, we need to create **Amazon Machine Image (AMI)** from the instance. The AMI will be used as base image to launch autoscaled instances.

An AMI provides the information required to launch an instance, which is a virtual server in the cloud. You specify an AMI when you launch an instance, and you can launch as many instances from the AMI as you need. You can also launch instances from as many different AMIs as you need (for more information visit http://docs.aws.amazon.com/AWS_EC2/latest/UserGuide/AMIs.html).

An AMI includes the following:

- A template for the root volume for the instance (for example, an operating system, an application server, and applications)
- Launch permissions that control which AWS accounts can use the AMI to launch instances
- A block device mapping that specifies the volumes to attach to the instance when it's launched

The ec2_ami module

We will use Ansible `ec2_ami` module to create our AMI. Options for this module are listed in the following table (for more information on `ec2_ami` module visit http://docs.ansible.com/ec2_module.html):

parameter	required	default	choices	comments
<code>aws_access_key</code>	no			AWS access key. If not set then the value of the <code>AWS_ACCESS_KEY</code> environment variable is used
<code>aws_secret_key</code>	no			AWS secret key. If not set then the value of the <code>AWS_SECRET_KEY</code> environment variable is used
<code>delete_snapshot</code>	no			Whether or not to delete an AMI while deregistering it
<code>description</code>	no			An optional human-readable string describing the contents and purpose of the AMI.

parameter	required	default	choices	comments
ec2_url	no			URL to use to connect to EC2 or your Eucalyptus cloud (by default the module will use EC2 endpoints). Must be specified if region is not used. If not set then the value of the EC2_URL environment variable, if any, is used
image_id	no			Image ID to be deregistered.
instance_id	no			Instance id of the image to create
name	no			The name of the new image to create
no_reboot	no		yes no	An optional flag indicating that the bundling process should not attempt to shutdown the instance before bundling. If this flag is True, the responsibility of maintaining file system integrity is left to the owner of the instance. The default choice is "no".
profile	no			Uses a boto profile. Only works with boto >= 2.24.0 (added in Ansible 1.6)
region	no			The AWS region to use. Must be specified if ec2_url is not used. If not specified then the value of the EC2_REGION environment variable, if any, is used.
security_token	no			Security token to authenticate against AWS (added in Ansible 1.6)
state	no	present		Create or deregister/delete image
validate_certs	no	yes no	yes no	When set to "no", SSL certificates will not be validated for boto versions >= 2.6.0. (added in Ansible 1.5)
wait	no	no	yes no	Wait for the AMI to be in state 'available' before returning
wait_timeout	no	300		How long before wait gives up, in seconds

Creating custom AMI

The following playbook will create AMI from the master WordPress instance we've created in Chapter 10, *Project 2 – A Multi-Tier WordPress Site*:

```
# cd /etc/ansible
# vi ami_create.yml

---
- hosts: localhost
  connection: local
  gather_facts: no
  vars:
    region: ap-southeast-2
    ins_name: wordpress_master
    ami_name: wordpress
  tasks:
    - name: get instance id
      command: "aws ec2 describe-instances
                  --filters Name>tag:Name,Values={{ ins_name }}
                  --query 'Reservations[0].Instances[0].InstanceId' --
      output text"
      register: instanceid
    - name: create ami
      ec2_ami:
        instance_id: "{{ instanceid.stdout }}"
        region: "{{ region }}"
        wait: yes
        name: "{{ ami_name }}"
        register: ami
        when: instanceid.stdout!="None"
    - debug: var=ami
```

Run the playbook:

```
# ansible-playbook ami_create.yml
```

After you run the playbook, an AMI will be created and the debug task will display the AMI ID. You can also view the AMI information on your AWS console.

Go to your EC2 dashboard and select **AMIs**.

The screenshot shows the AWS EC2 Dashboard. On the left, there's a sidebar with links like EC2 Dashboard, Events, Tags, Reports, Limits, Instances, Spot Requests, Reserved Instances, Images, and AMIs. The AMIs link is highlighted with an orange bar. The main area has a 'Launch' button and an 'Actions' dropdown. A search bar says 'Owned by me' and 'Filter by tags and attributes or search'. Below it is a table with columns: Name, AMI Name, and AMI ID. One row shows 'wordpress' in the Name column and 'ami-' followed by a redacted ID in the AMI ID column.

AMI list

Deleting custom AMI

If you decide that you no longer need an AMI, you can deregister it. After you deregister an AMI, you can't use it to launch new instances. However, it doesn't affect any instances that you already launched from the AMI. It also doesn't affect the snapshot that was created when you created the AMI. You'll continue to accrue charges for the snapshot until you delete it.

Example playbook to deregister custom AMI and delete the snapshot:

```
# cd /etc/ansible
# vi ami_delete.yml

---
- hosts: localhost
  connection: local
  gather_facts: no
  vars:
    region: ap-southeast-2
    ami_name: wordpress
  tasks:
    - name: get ami id
      command: "aws ec2 describe-images
                  --filters Name=name,Values={{ ami_name }}
                  --query 'Images[0].ImageId' --output text"
```

```
register: imageid
- name: delete ami
  ec2_ami:
    region: "{{ region }}"
    image_id: "{{ imageid.stdout }}"
    delete_snapshot: yes
    state: absent
when: imageid.stdout!="None"
```

12

Auto Scaling and Elastic Load Balancing – ELB

In this chapter we will continue scaling our WordPress site from [Chapter 10, Project 2 – A Multi-Tier WordPress Site](#). We will add *Elastic Load Balancing*, add more private subnets in both Availability Zones for auto-scaled instances, modify VPC security groups, and configure Auto Scaling.

What is Auto Scaling?

Auto Scaling is an AWS service that allows you to increase or decrease the number of EC2 instances within your application's architecture. *Auto Scaling group* is collection of EC2 instances with one or more scaling policies. These policies define when Auto Scaling launches or terminates EC2 instances within the group. You can specify the minimum and maximum number of instances in each Auto Scaling group (for more information visit <http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/WhatIsAutoScaling.html>).

You can create as many Auto Scaling groups as you need. For example, if an application consists of a web tier and an application tier, you can create two Auto Scaling groups, one for each tier.

Adding Auto Scaling to your network architecture is one way to maximize the benefits of the AWS cloud. With Auto Scaling, you can make your applications:

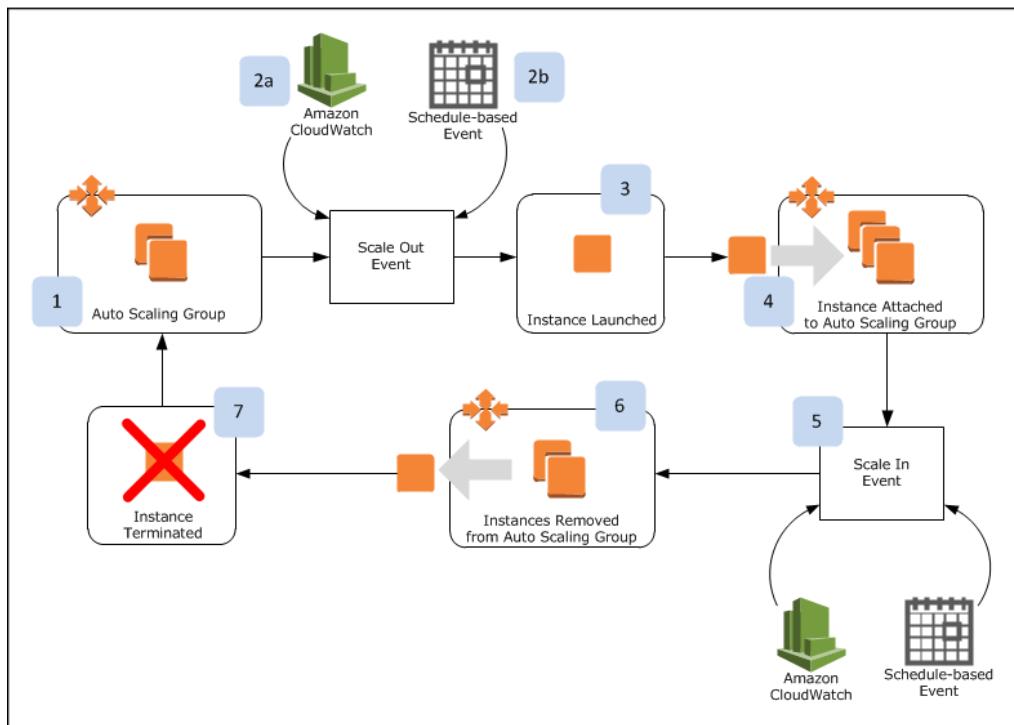
- More fault tolerant. Auto Scaling can detect when an instance is unhealthy, terminate it, and launch a new instance to replace it.
- More highly available. You can configure Auto Scaling to use multiple subnets or Availability Zones. If one subnet or Availability Zone becomes unavailable, Auto Scaling can launch instances in another one to compensate.
- Increase and decrease in capacity only when needed. Unlike on-premises solutions, with Auto Scaling you can have your network scale dynamically. You also don't pay for Auto Scaling. Instead, you pay only for the EC2 instances launched, and only for as long as you use them.

Key components of Auto Scaling:

- **Groups:** When you use Auto Scaling, your EC2 instances are grouped into Auto Scaling groups for the purposes of instance scaling and management. You create Auto Scaling groups by defining the minimum, maximum, and, optionally, the desired number of running EC2 instances the group must have at any point in time.
- **Launch Configurations:** Your Auto Scaling group uses a launch configuration to launch EC2 instances. You create the launch configuration by providing information about the image you want Auto Scaling to use to launch EC2 instances. The information can be the image ID, instance type, key pairs, security groups, and block device mapping.
- **Scaling Plans:** In addition to creating a launch configuration and an Auto Scaling group, you must also create a scaling plan for your Auto Scaling group. A scaling plan tells Auto Scaling when and how to scale. You can create a scaling plan based on the occurrence of specified conditions (dynamic scaling) or you can create a plan based on a specific schedule.

How Auto Scaling works?

Here's how Auto Scaling works in a typical AWS environment (for more information visit <http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/how-it-works.html>):



Autoscaling lifecycle

- We'll start with an Auto Scaling group set to have a desired capacity of two instances.
- A scale out event occurs. This is an event that instructs auto scaling to launch an additional instance. A scale out event could be something like an CloudWatch alarm (item 2a in the diagram), or a schedule-based scaling policy (item 2b in the diagram) that launches instances at a specific day and time.
- Auto Scaling launches and configures the instance.
- When the instance is fully configured, Auto Scaling attaches it to the Auto Scaling group.
- Eventually, a corresponding scale in event occurs. A scale in event is like a scale out event, except that these types of events instruct Auto Scaling to terminate one or more instances.
- Auto Scaling removes the instances from the group and marks it for termination.
- The instance terminates.

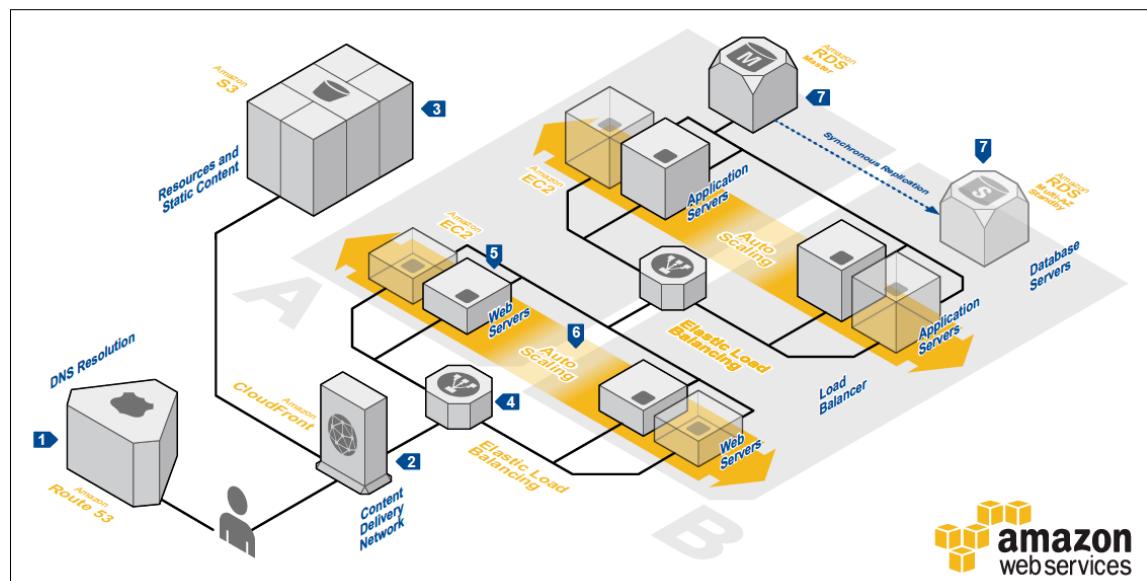
Availability zones and regions

Auto Scaling lets you take advantage of the safety and reliability of geographic redundancy by spanning Auto Scaling groups across multiple Availability Zones within a region. When one Availability Zone becomes unhealthy or unavailable, Auto Scaling launches new instances in an unaffected Availability Zone. When the unhealthy Availability Zone returns to a healthy state, Auto Scaling automatically redistributes the application instances evenly across all of the designated Availability Zones.

An Auto Scaling group can contain EC2 instances that come from one or more EC2 Availability Zones within the same region. However, Auto Scaling group cannot span multiple regions.

The architecture

To get an idea of how to deploy a highly available and scalable web hosting, you can take a look at the **AWS Web Application Hosting** reference architecture here: http://media.amazonaws.com/architecturecenter/AWS_ac_ra_web_01.pdf.



Reference architecture

System Overview

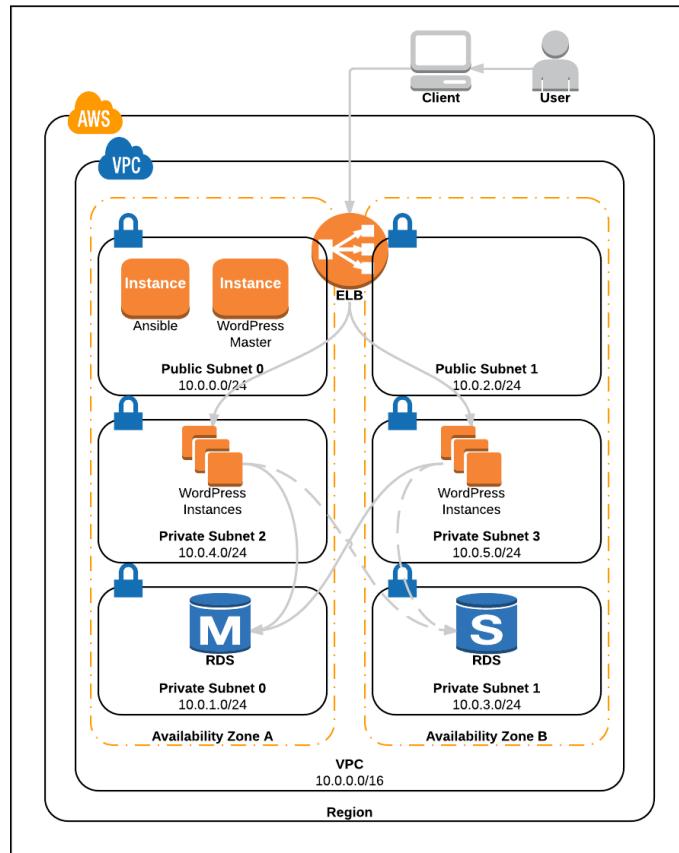
- The user's DNS requests are served by **Amazon Route 53**, a highly available Domain Name System (DNS) service. Network traffic is routed to infrastructure running in Amazon Web Services.
- Static, streaming, and dynamic content is delivered by **Amazon CloudFront**, a global network of edge locations. Requests are automatically routed to the nearest edge location, so content is delivered with the best possible performance.
- Resources and static content used by the web application are stored on **Amazon Simple Storage Service (S3)**, a highly durable storage infrastructure designed for mission-critical and primary data storage.
- HTTP requests are first handled by **Elastic Load Balancing**, which automatically distributes incoming application traffic among multiple **Amazon Elastic Compute Cloud (EC2)** instances across Availability Zones (AZs). It enables even greater fault tolerance in your applications, seamlessly providing the amount of load balancing capacity needed in response to incoming application traffic.
- Web servers and application servers are deployed on Amazon EC2 instances. Most organizations will select an **Amazon Machine Image (AMI)** and then customize it to their needs. This custom AMI will then become the starting point for future web development.
- Web servers and application servers are deployed in an **Auto Scaling** group. Auto scaling automatically adjusts your capacity up or down according to conditions you define. With Auto Scaling you can ensure that the number of **Amazon EC2** instances you're using increases seamlessly during demand spikes to maintain performance and decreases automatically during demand to minimize costs.
- To provide high availability, the relational database that contains application's data is hosted redundantly on a multi-AZ (multiple Availability Zones-zones A and B here) deployment of **Amazon Relational Database Service (amazon RDS)**.

You can find other AWS architecture references in the **AWS Architecture Center** page: <https://aws.amazon.com/architecture/>.



The AWS Architecture Center is designed to provide you with the necessary guidance and application architecture best practices to build highly scalable and reliable applications in the AWS cloud. These resources will help you understand the AWS platform, its services and features, and will provide architectural guidance for design and implementation of systems that run on the AWS infrastructure.

We'll follow the reference architecture from AWS and expand our WordPress hosting to be highly available and scalable. The following diagram shows you the architecture we'd like to build:



High availability WordPress site

Ansible will be used to manage a WordPress Master instance configuration. Then we can configure and update the plugins, themes, and so on, in this Master instance. Later we can create a custom AMI from this instance and the AMI will be used in the launch configuration of WordPress Auto Scaling group.

VPC update

Let's create the VPC provisioning playbook. This is similar to the multi-AZ VPC playbook in Chapter 6, *VPC Provisioning with Ansible*, with two additional private subnets:

```
# cd /etc/ansible/wordpress_ha
# vi provisioning_vpc.yml

---
- hosts: localhost
  connection: local
  gather_facts: no
  vars:
    region: ap-southeast-2
    env: staging
    az0: ap-southeast-2a
    az1: ap-southeast-2b
  tasks:
    - name: create vpc with multi-az subnets
      ec2_vpc:
        region: "{{ region }}"
        cidr_block: 10.0.0.0/16
        resource_tags: '{"Name":"'{{ env }}vpc"}'
        subnets:
          - cidr: 10.0.0.0/24
            az: "{{ az0 }}"
            resourcetags: '{"Name":"'{{ env }}subnetpublic_0"}'
          - cidr: 10.0.1.0/24
            az: "{{ az0 }}"
            resource_tags: '{"Name":"'{{ env }}subnetprivate_0"}'
          - cidr: 10.0.2.0/24
            az: "{{ az1 }}"
            resource_tags: '{"Name":"'{{ env }}subnetpublic_1"}'
          - cidr: 10.0.3.0/24
            az: "{{ az1 }}"
            resource_tags: '{"Name":"'{{ env }}subnetprivate_1"}'
          - cidr: 10.0.4.0/24
            az: "{{ az0 }}"
            resource_tags: '{"Name":"'{{ env }}subnetprivate_2"}'
          - cidr: 10.0.5.0/24
            az: "{{ az1 }}"
            resource_tags: '{"Name":"'{{ env }}subnetprivate_3"}'
        internet_gateway: yes
        route_tables:
          - subnets:
              - 10.0.0.0/24
              - 10.0.2.0/24
```

```
routes:
  - dest: 0.0.0.0/0
    gw: igw
```

If you still have the multi-AZ VPC from Chapter 6, *VPC Provisioning with Ansible*, running this playbook will only add two private subnets (`staging_subnet_private_2` and `staging_subnet_private_3`) within the VPC.

Security groups update

We need to create a new security group for WordPress tier in the private subnets, also a new security group for Elastic Load Balancer.

```
# cd /etc/ansible/wordpress_ha
# vi provisioning_sg.yml` {lang="text"}
```

```
---
- hosts: localhost
  connection: local
  gather_facts: no
  vars:
    region: ap-southeast-2
    allowed_ip: xx.xx.xx.xx/32
    vpc_cidr: 10.0.0.0/16
    env: staging
  tasks:
    - name: get vpc id
      command: "aws ec2 describe-vpcs --filters Name=tag:Name,Values={{ env }}_vpc
                --query 'Vpcs[0].VpcId' --output text"
      register: vpcid
    - name: create sg_web rules
      ec2_group:
        region: "{{ region }}"
        vpc_id: "{{ vpcid.stdout }}"
        name: "{{ env }}_sg_web"
        description: security group for public web
        rules:
          # allow ssh access from ansible group
          - proto: tcp
            from_port: 22
            to_port: 22
            group_name: "{{ env }}_sg_ansible"
            group_desc: security group for ansible
          # allow http access from anywhere
```

```
- proto: tcp
  from_port: 80
  to_port: 80
  cidr_ip: 0.0.0.0/0
# allow https access from anywhere
- proto: tcp
  from_port: 443
  to_port: 443
  cidr_ip: 0.0.0.0/0
- name: create sg_wordpress rules
  ec2_group:
    region: "{{ region }}"
    vpc_id: "{{ vpcid.stdout }}"
    name: "{{ env }}_sg_wordpress"
    description: security group for wordpress servers
    rules:
      # allow ssh access from ansible group
      - proto: tcp
        from_port: 22
        to_port: 22
        group_name: "{{ env }}_sg_ansible"
        group_desc: security group for ansible
      # allow http access from vpc cidr
      - proto: tcp
        from_port: 80
        to_port: 80
        group_name: "{{ env }}_sg_wordpress_lb"
        group_desc: security group for wordpress load balancer
      # allow https access from vpc cidr
      - proto: tcp
        from_port: 443
        to_port: 443
        group_name: "{{ env }}_sg_wordpress_lb"
        group_desc: security group for wordpress load balancer
- name: create sg_database rules
  ec2_group:
    region: "{{ region }}"
    vpc_id: "{{ vpcid.stdout }}"
    name: "{{ env }}sgdatabase"
    description: security group for database
    rules:
      - proto: tcp
        from_port: 3306
        to_port: 3306
        group_name: "{{ env }}_sg_web"
      - proto: tcp
        from_port: 3306
        to_port: 3306
```

```
group_name: "{{ env }}_sg_wordpress"
- name: create sg_ansible rules
  ec2_group:
    region: "{{ region }}"
    vpc_id: "{{ vpcid.stdout }}"
    name: "{{ env }}_sg_ansible"
    description: security group for ansible
    rules:
      - proto: tcp
        from_port: 22
        to_port: 22
        cidr_ip: "{{ allowed_ip }}"
- name: create sg_wordpress_lb
  ec2_group:
    region: "{{ region }}"
    vpc_id: "{{ vpcid.stdout }}"
    name: "{{ env }}_sg_wordpress_lb"
    description: security group for wordpress load balancer
    rules:
      # allow http access from anywhere
      - proto: tcp
        from_port: 80
        to_port: 80
        cidr_ip: 0.0.0.0/0
      # allow https access from anywhere
      - proto: tcp
        from_port: 443
        to_port: 443
        cidr_ip: 0.0.0.0/0
```

Getting started with Auto Scaling

To help you get a better understanding on how Auto Scaling works, let's try to configure Auto Scaling from AWS console (for more information visit <http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/GettingStartedTutorial.html>).

Step 1 – Create a launch configuration

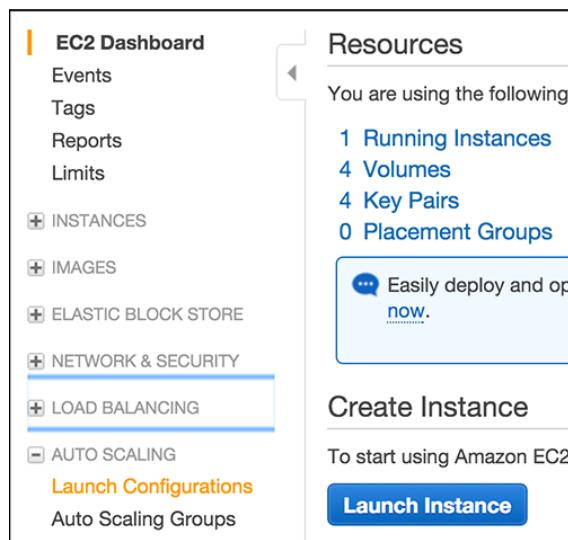
A launch configuration is a template that an Auto Scaling group uses to launch EC2 instances. When you create a launch configuration, you specify information for the instances such as the ID of the AMI, the instance type, a key pair, one or more security groups, and a block device mapping.

When you create an Auto Scaling group, you must specify a launch configuration. You can specify your launch configuration with multiple Auto Scaling groups. However, you can only specify one launch configuration for an Auto Scaling group at a time, and you can't modify a launch configuration after you've created it. Therefore, if you want to change the launch configuration for your Auto Scaling group, you must create a new launch configuration and then update your Auto Scaling group with the new launch configuration. When you change the launch configuration for your Auto Scaling group, any new instances are launched using the new configuration parameters, but existing instances are not affected (for more information visit

<http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/LaunchConfiguration.html>).

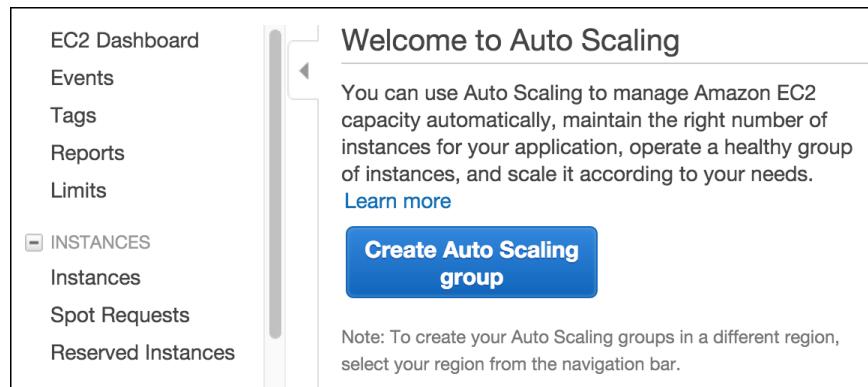
To create a launch configuration:

1. Open the Amazon EC2 console.
2. In the navigation pane, under **Auto Scaling**, click on **Launch Configurations**.

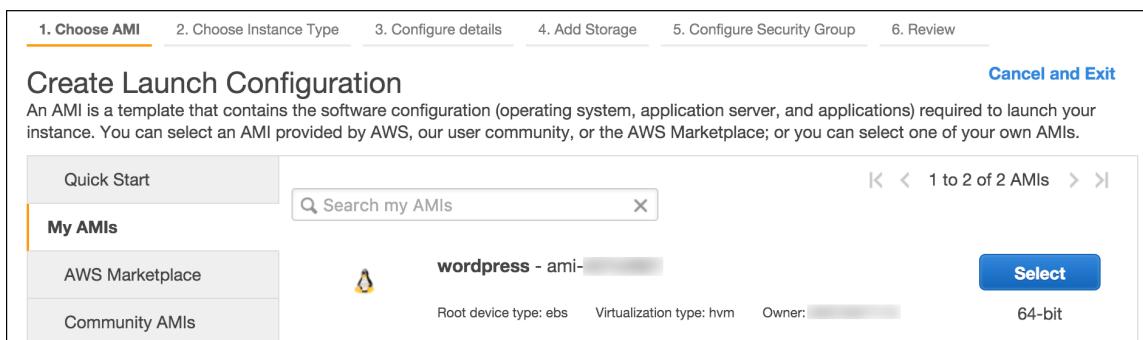


3. Select a region. The Auto Scaling resources that you create are tied to the region you specify and are not replicated across regions.

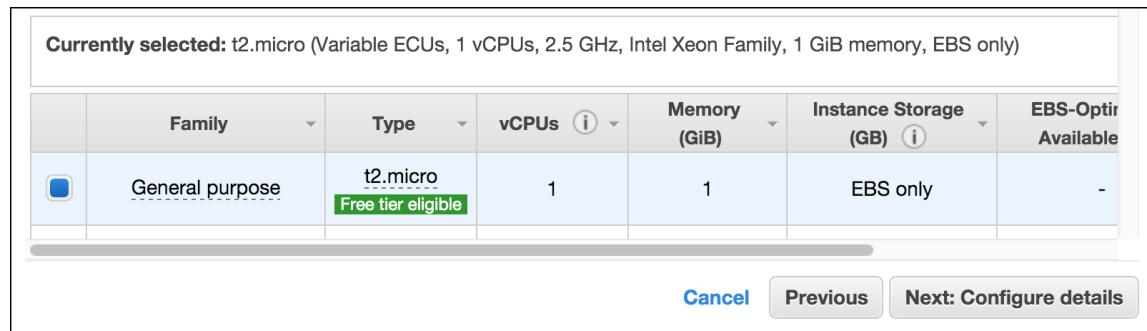
4. On the **Welcome to Auto Scaling** page, click on **Create Auto Scaling group**.



5. On the **Create Auto Scaling Group** page, click on **Create launch configuration**.
6. The **Choose AMI** page displays a list of basic configurations, called **Amazon Machine Images (AMIs)**, that serve as templates for your instance. Select **My AMIs** on the left pane, and select the AMI that you have created in Chapter 11, *Amazon Machine Images – AMI*.

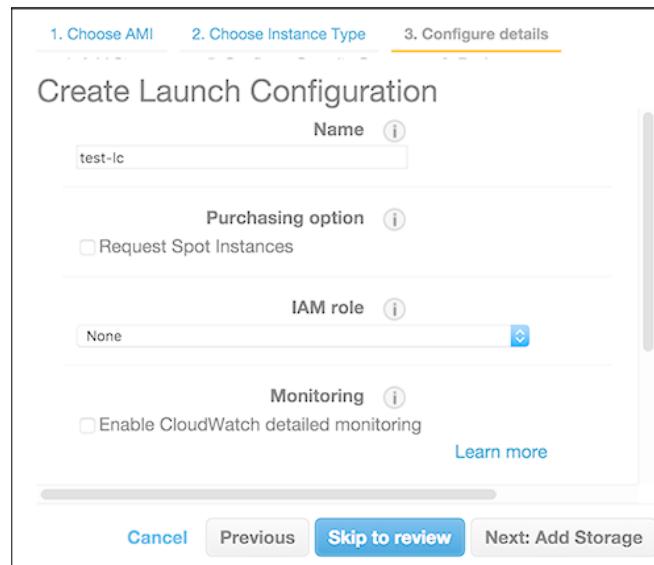


7. On the **Choose Instance Type** page, select a hardware configuration for your instance. Let's use the `t2.micro` instance that is selected by default. Click on **Next: Configure details**.

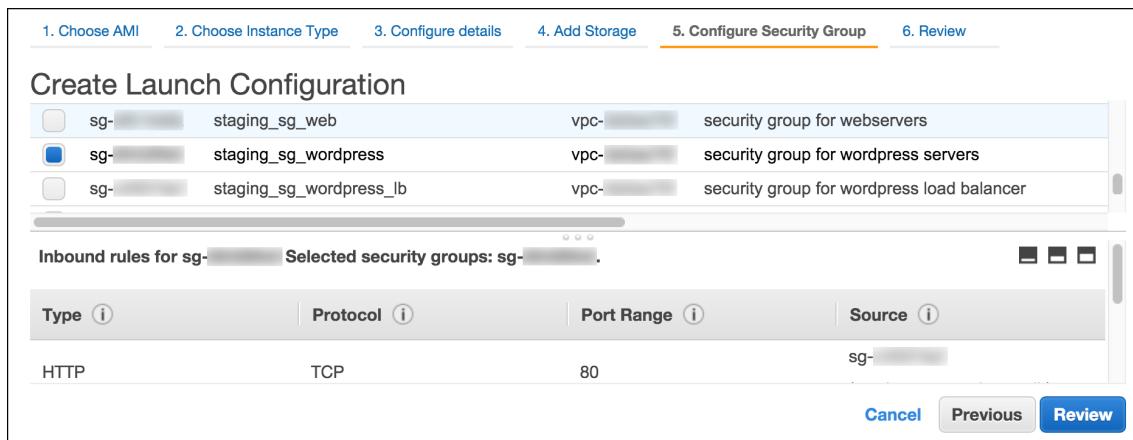


8. On the **Configure Details** page, do the following:

- In the **Name** field, enter a name of your launch configuration (for example, **test-lc**).
- Under **Advanced Details**, select an IP address type. Because we will launch instances in private subnets, select **Do not assign a public IP address to any instances**.
- Click **Skip to review**.



9. On the **Review** page, click **Edit security groups**, select **Select an existing security group**, choose the `staging_sg_wordpress` security group and then click on **Review**.



10. On the **Review** page, click on **Create launch configuration**.
11. In the **Select an existing key pair or create a new key pair** dialog box, select one of the listed options. Note that you won't connect to your instance as part of this tutorial. Therefore, you can select Proceed without a key pair unless you intend to connect to your instance.
12. Click on **Create launch configuration** to create your launch configuration.

Step 2 – Create an Auto Scaling group

An Auto Scaling group contains a collection of EC2 instances that share similar characteristics and are treated as a logical grouping for the purposes of instance scaling and management. For example, if a single application operates across multiple instances, you might want to increase the number of instances in that group to improve the performance of the application, or decrease the number of instances to reduce costs when demand is low. You can use the Auto Scaling group to scale the number of instances automatically based on criteria that you specify, or maintain a fixed number of instances even if an instance becomes unhealthy.

This automatic scaling and maintaining the number of instances in an Auto Scaling group is the core value of the Auto Scaling service (for more information visit <http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/AutoScalingGroup.html>).

To create an Auto Scaling group:

1. On the **Configure Auto Scaling group details** page, do the following:
 - In **Group name**, enter a name for your Auto Scaling group (for example, **test-asg**)
 - Leave **Group size** set to the default value of 1 instance for this tutorial.
 - Select the **staging_vpc** VPC in **Network** and select **staging_subnet_private_2** subnet and **staging_subnet_private_3** subnet from **Subnet**.
 - Click **Next: Configure scaling policies**.

The screenshot shows the 'Create Auto Scaling Group' wizard on step 1. The tabs at the top are 1. Configure Auto Scaling group details (selected), 2. Configure scaling policies, 3. Configure Notifications, 4. Configure Tags, and 5. Review. On the right, there is a 'Cancel and Exit' button. The main form fields include:

- Launch Configuration:** test-lc
- Group name:** test-asg
- Group size:** Start with 1 instances
- Network:** vpc- (10.0.0.0/16) | staging_vpc
- Subnet:** A dropdown menu showing two selected subnets:
 - subnet- (10.0.4.0/24) | staging_subnet_private_2 | ap-southeast-2a
 - subnet- (10.0.5.0/24) | staging_subnet_private_3 | ap-southeast-2bA 'Create new subnet' button is also present.

At the bottom, a note says "No instances in this Auto Scaling group will be assigned a public IP address." There are 'Cancel' and 'Next: Configure scaling policies' buttons.

2. In the **Configure scaling policies** page, select **Keep this group at its initial size** for this tutorial and click on **Review**.
3. On the **Review** page, click on **Create Auto Scaling group**.
4. On the **Auto Scaling group creation status** page, click on **Close**.

Step 3 – Verify your Auto Scaling group

Now that you have created your Auto Scaling group, you are ready to verify that the group has launched your EC2 instance.

To verify that your Auto Scaling group has launched your EC2 instance:

1. On the **Auto Scaling Groups** page, select the Auto Scaling group that you just created.
2. The **Details** tab provides information about the Auto Scaling group.

The screenshot shows the AWS Auto Scaling Groups page. At the top, there is a filter bar with a search input 'Filter: Filter Auto Scaling groups...'. Below the filter is a table header with columns: Name, Launch Configuration, Instances, Desired, Min, Max, Availability Zones, and Default Cooldown. A row for 'test-asg' is selected, showing 'test-lc' under Launch Configuration, '1' under Instances, Desired, Min, and Max, and 'ap-southeast-2b, ap-southeast-2a' under Availability Zones. The Default Cooldown is set to 300. Below the table, the 'Auto Scaling Group: test-asg' section is expanded. It contains tabs for Details, Activity History, Scaling Policies, Instances, Notifications, and Tags. The 'Details' tab is selected. The 'Edit' button is visible at the top right of this section. The 'Launch Configuration' section shows 'test-lc'. The 'Load Balancers' section shows 'Desired: 1', 'Min: 1', and 'Max: 1'. The 'Health Check Type' is 'EC2'. The 'Health Check Grace Period' is '300'. The 'Termination Policies' is 'Default'. The 'Creation Time' is 'Sun Nov 08 10:43:34 GMT+1100 2015'. To the right of these settings, there are sections for 'Availability Zone(s)', 'Subnet(s)', 'Default Cooldown', 'Placement Group', 'Suspended Processes', and 'Enabled Metrics'. The 'Availability Zone(s)' section lists 'ap-southeast-2b, ap-southeast-2a'. The 'Subnet(s)' section lists 'subnet-', 'subnet-', and 'subnet-'. The 'Default Cooldown' is '300'. The 'Placement Group' section is empty. The 'Suspended Processes' and 'Enabled Metrics' sections also contain no data.

3. Select the **Scaling History** tab. The **Status** column contains the current status of your instance. When your instance is launching, the status column shows **In progress**. The status changes to **Successful** after the instance is launched. You can also click the refresh button to see the current status of your instance.

Auto Scaling Group: test-asg

Details Activity History Scaling Policies Instances Notifications Tags

Filter: Any Status ▾ 1 to 1 of 1 History Items

Status	Description	Start Time	End Time
Successful	Launching a new EC2 instance: i-690556b7	2015 November 8 10:43:39 UTC+11	2015 November 8 10:44:11 UTC+11

Description: Launching a new EC2 instance: i-690556b7

Cause: At 2015-11-07T23:43:34Z a user request created an AutoScalingGroup changing the desired capacity from 0 to 1. At 2015-11-07T23:43:38Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 0 to 1.

4. Select the **Instances** tab. The **Lifecycle** column contains the state of your newly launched instance. You can see that your Auto Scaling group has launched your EC2 instance, and it is in the `InService` lifecycle state. The **Health Status** column shows the result of the EC2 instance health check on your instance.

Auto Scaling Group: test-asg

Details Activity History Scaling Policies Instances Notifications Tags

Actions ▾

Filter: Any Health Status ▾ Any Lifecycle State ▾ 1 to 1 of 1 Instances

Instance ID	Lifecycle	Launch Configuration Name	Availability Zone	Health Status
i-690556b7	InService	test-lc	ap-southeast-2b	Healthy

5. If you want, you can try the following experiment to learn more about Auto Scaling. The minimum size for your Auto Scaling group is 1 instance. Therefore, if you terminate the running instance, Auto Scaling must launch a new instance to replace it.

- On the **Instances** tab, click the ID of the instance. This takes you to the **Instances** page and selects the instance.
- Click on **Actions**, select **Instance State**, and then click on **Terminate**. When prompted for confirmation, click on **Yes, Terminate**.
- In the navigation pane, select **Auto Scaling Groups** and then select the **Scaling History** tab. The default cooldown for the Auto Scaling group is 300 seconds (5 minutes), so it takes about 5 minutes until you see the scaling activity. When the scaling activity starts, you'll see an entry for the termination of the first instance and an entry for the launch of a new instance. The **Instances** tab shows the new instance only.
- In the navigation pane, select **Instances**. This page shows both the terminated instance and the running instance.

The screenshot shows the AWS Auto Scaling Activity History page. The top navigation bar includes tabs for Details, Activity History (which is selected and highlighted in orange), Scaling Policies, Instances, Notifications, and Tags. Below the navigation bar is a search bar labeled 'Filter scaling history...' with a magnifying glass icon. To the right of the search bar are navigation icons for back, forward, and search. A status bar at the top right indicates '1 to 3 of 3 History Items'.

Status	Description	Start Time	End Time
Not yet in service	Launching a new EC2 instance: i-e6b02339	2015 November 8 12:07:15 UTC+11	
Description: Launching a new EC2 instance: i-e6b02339 Cause: At 2015-11-08T01:07:14Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 0 to 1.			
Successful	Terminating EC2 instance: i-690556b7	2015 November 8 12:06:44 UTC+11	2015 November 8 12:06:46 UTC+11
Successful	Launching a new EC2 instance: i-690556b7	2015 November 8 10:43:39 UTC+11	2015 November 8 10:44:11 UTC+11

Scaling the size of your Auto Scaling group

Scaling is the ability to increase or decrease the compute capacity of your application. Scaling starts with an event, or scaling action, which instructs Auto Scaling to either launch or terminate EC2 instances.

Auto Scaling provides a number of ways to adjust scaling to best meet the needs of your applications. As a result, it's important that you have a good understanding of your application. You should keep the following considerations in mind (for more information visit

http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/scaling_plan.html):

- What role do you want Auto Scaling to play in your application's architecture? It's common to think about Auto Scaling as a way to increase and decrease capacity, but Auto Scaling is also useful for when you want to maintain a steady number of servers.
- What cost constraints are important to you? Because Auto Scaling uses EC2 instances, you only pay for the resources you use. Knowing your cost constraints can help you decide when to scale your applications, and by how much.
- What metrics are important to your application? CloudWatch supports a number of different metrics that you can use with your Auto Scaling group. You can view available EC2 metrics here: <http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/ec2-metricscollected.html>. For example, you can use CPUUtilization metric in your scaling policy to increase/decrease Auto Scaling group size based on average CPU usage of the EC2 instances.

Some scaling implementations you can use, based on the needs of your applications:

- **Maintaining a fixed number of EC2 instances in your Auto Scaling group** (for more information visit <http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/as-maintain-instance-levels.html>): After you have created your launch configuration and Auto Scaling group, the Auto Scaling group starts by launching the minimum number of EC2 instances (or the desired capacity, if specified). If there are no other scaling conditions attached to the Auto Scaling group, the Auto Scaling group maintains this number of running instances at all times.
- To maintain the same number of instances, Auto Scaling performs a periodic health check on running instances within an Auto Scaling group. When it finds that an instance is unhealthy, it terminates that instance and launches a new one.
- All instances in your Auto Scaling group start in the healthy state. Instances are assumed to be healthy unless Auto Scaling receives notification that they are unhealthy. This notification can come from one or more of the following sources: Amazon EC2, Elastic Load Balancing, or your customized health check.

- **Manual Scaling** (for more information visit <http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/as-manual-scaling.html>): At any time, you can manually change the size of an existing Auto Scaling group. Auto Scaling manages the process of launching or terminating instances to maintain the updated group size.
- **Scheduled Scaling** (for more information visit <http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/schedule-time.html>): Scaling based on a schedule allows you to scale your application in response to predictable load changes. For example, every week the traffic to your web application starts to increase on Wednesday, remains high on Thursday, and starts to decrease on Friday. You can plan your scaling activities based on the predictable traffic patterns of your web application.
- To configure your Auto Scaling group to scale based on a schedule, you need to create scheduled actions. A scheduled action tells Auto Scaling to perform a scaling action at certain time in future. To create a scheduled scaling action, you specify the start time at which you want the scaling action to take effect, and you specify the new minimum, maximum, and desired size you want for that group at that time. At the specified time, Auto Scaling updates the group to set the new values for minimum, maximum, and desired sizes, as specified by your scaling action.
- **Dynamic Scaling** (for more information visit <http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/as-scale-based-on-demand.html>): Scale dynamically in response to changes in the demand for your application. You must specify when and how to scale.
- When you use Auto Scaling to scale dynamically, you must define how you want to scale in response to changing demand. For example, say you have a web application that currently runs on two instances. You want to launch two additional instances when the load on the current instances rises to 70 percent, and then you want to terminate those additional instances when the load falls to 40 percent. You can configure your Auto Scaling group to scale automatically based on these conditions. We will try this later in this chapter.

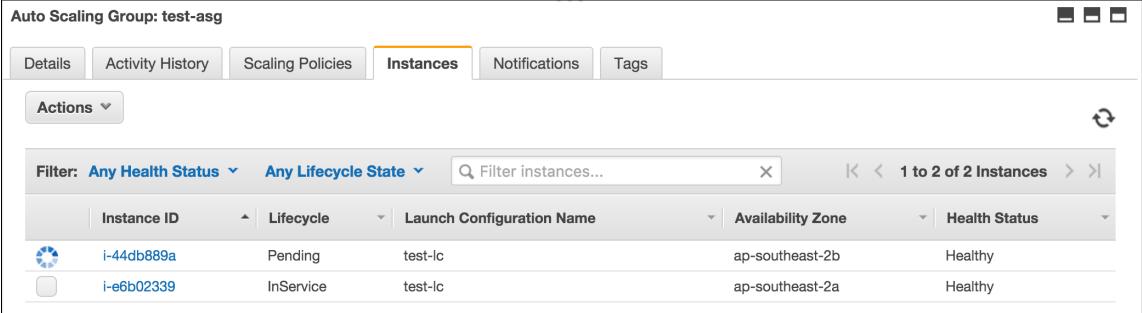
Scaling manually using the console

To change the size of your Auto Scaling group manually (for more information visit <http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/as-manual-scaling.html>):

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation pane, under **Auto Scaling**, click **Auto Scaling Groups**.
3. On the **Auto Scaling Groups** page, select your Auto Scaling group from the list.
4. The bottom pane displays the details of your Auto Scaling group. Select the **Details** tab and then click on **Edit**.
5. In **Desired**, increase the desired capacity by one. For example, if the current value is 1, enter 2. The desired capacity must be less than or equal to the maximum size of the group. Increase **Max** to 5. When you are finished, click on **Save**.

The screenshot shows the AWS Auto Scaling Groups Details page. A modal dialog is open for the 'test-asg' group. In the 'Launch Configuration' dropdown, 'test-lc' is selected. The 'Desired' field is set to 2, and the 'Max' field is set to 5. The 'Min' field is set to 1. Under 'Availability Zone(s)', the zones 'ap-southeast-2b, ap-southeast-2a' are listed, each associated with a subnet: 'subnet- (172.31.0.0/20)' and 'subnet- (172.31.16.0/20)'. The 'Default Cooldown' is set to 300. The 'Health Check Type' is set to 'EC2'. The 'Details' tab is selected in the navigation bar at the top of the dialog.

6. To verify that the size of your Auto Scaling group has changed, go to **Instances** tab. You can see that your Auto Scaling group has launched one new instance.



Instance ID	Lifecycle	Launch Configuration Name	Availability Zone	Health Status
i-44db889a	Pending	test-lc	ap-southeast-2b	Healthy
i-e6b02339	InService	test-lc	ap-southeast-2a	Healthy



If you're planning to make an ASG idle, you can set **Min**, **Desired**, and **Max** to 0 to terminate all instances within the ASG. You can increase these values later to launch new instance(s).

Load Balance your Auto Scaling group

When you use Auto Scaling, you can automatically increase the number of EC2 instances you're using when the user demand goes up, and you can decrease the number of EC2 instances when demand goes down. As Auto Scaling dynamically adds and removes EC2 instances, you need to ensure that the traffic coming to your application is distributed across all of your running EC2 instances. AWS provides the Elastic Load Balancing service to distribute the incoming traffic (called the **load**) automatically among all the EC2 instances that you are running. Elastic Load Balancing manages incoming requests by optimally routing traffic so that no one instance is overwhelmed. Using Elastic Load Balancing with your auto-scaled application makes it easy to route traffic across a dynamically changing fleet of EC2 instances (for more information visit http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/US_SetUpASLBApp.html).

What is Elastic Load Balancing?

Elastic Load Balancing automatically distributes incoming traffic across multiple EC2 instances. You create a load balancer and register instances with the load balancer in one or more Availability Zones. The load balancer serves as a single point of contact for clients. This enables you to increase the availability of your application. You can add and remove EC2 instances from your load balancer as your needs change, without disrupting the overall flow of information. If an EC2 instance fails, Elastic Load Balancing automatically reroutes the traffic to the remaining running EC2 instances. If a failed EC2 instance is restored, Elastic Load Balancing restores the traffic to that instance. Elastic Load Balancing can also serve as the first line of defense against attacks on your network. You can offload the work of encryption and decryption to your load balancer so that your EC2 instances can focus on their main work (for more information visit <http://docs.aws.amazon.com/ElasticLoadBalancing/latest/DeveloperGuide/elastic-load-balancing.html>).

If you enable Auto Scaling with Elastic Load Balancing, instances that are launched by Auto Scaling are automatically registered with the load balancer, and instances that are terminated by Auto Scaling are automatically de-registered from the load balancer.

Features of Elastic Load Balancing

Elastic Load Balancing provides the following features:

- You can use the operating systems and instance types supported by Amazon EC2. You can configure your EC2 instances to accept traffic only from your load balancer.
- You can configure the load balancer to accept traffic using the following protocols: HTTP, HTTPS (secure HTTP), TCP, and SSL (secure TCP).
- You can configure your load balancer to distribute requests to EC2 instances in multiple Availability Zones, minimizing the risk of overloading one single instance. If an entire Availability Zone goes offline, the load balancer routes traffic to instances in other Availability Zones.
- There is no limit on the number of connections that your load balancer can attempt to make with your EC2 instances. The number of connections scales with the number of concurrent requests that the load balancer receives.
- You can configure the health checks that Elastic Load Balancing uses to monitor the health of the EC2 instances registered with the load balancer so that it can send requests only to the healthy instances.
- You can use end-to-end traffic encryption on those networks that use secure (HTTPS/SSL) connections.

- [EC2-VPC] You can create an Internet-facing load balancer, which takes requests from clients over the Internet and routes them to your EC2 instances, or an internal-facing load balancer, which takes requests from clients in your VPC and routes them to EC2 instances in your private subnets. Load balancers in EC2-Classic are always Internet-facing.
- [EC2-Classic] Load balancers for EC2-Classic support both IPv4 and IPv6 addresses. Load balancers for a VPC do not support IPv6 addresses.
- You can monitor your load balancer using CloudWatch metrics, access logs, and AWS CloudTrail.
- You can associate your Internet-facing load balancer with your domain name. Because the load balancer receives all requests from clients, you don't need to create and manage public domain names for the EC2 instances to which the load balancer routes traffic. You can point the instance's domain records at the load balancer instead and scale as needed (either adding or removing capacity) without having to update the records with each scaling activity.

Getting started with ELB

In this tutorial we will add an Elastic Load Balancer and attach it to our WordPress Auto Scaling Group from AWS console (for more information visit <http://docs.aws.amazon.com/ElasticLoadBalancing/latest/DeveloperGuide/elb-getting-started.html>).

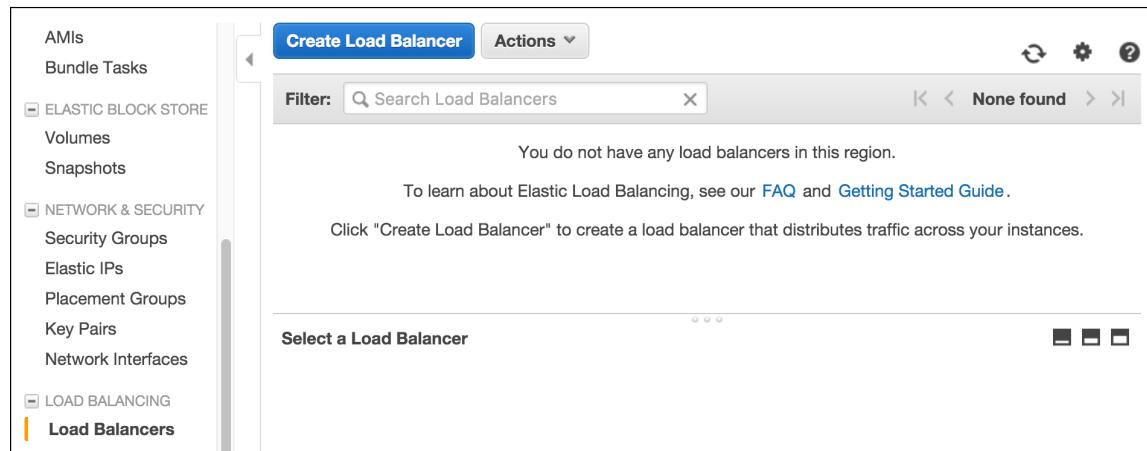
Step 1 – Define load balancer

First, provide some basic configuration information for your load balancer, such as a name, a network, and a listener.

A *listener* is a process that checks for connection requests. It is configured with a protocol and a port for front-end (client to load balancer) connections and a protocol, and protocol and a port for back-end (load balancer to back-end instance) connections. In this tutorial, you configure a listener that accepts HTTP requests on port 80 and sends them to the back-end instances on port 80 using HTTP.

To define your load balancer:

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. From the navigation bar, select a region for your load balancers. Be sure to select the same region that you selected for your EC2 instances.
3. In the navigation pane, under LOAD BALANCING, click Load Balancers.
4. Click on **Create Load Balancer**.



5. In **Load Balancer name**, enter a name for your load balancer. The name of your load balancer must be unique within your set of load balancers for the region, can have a maximum of 32 characters, and can contain only alphanumeric characters and hyphens.
6. From **Create LB inside**, select staging_vpc VPC.

7. Leave the default listener configuration.

Step 1: Define Load Balancer

Basic Configuration

This wizard will walk you through setting up a new load balancer. Begin by giving your new load balancer a unique name so that you can identify it from other load balancers you might create. You will also need to configure ports and protocols for your load balancer. Traffic from your clients can be routed from any load balancer port to any port on your EC2 instances. By default, we've configured your load balancer with a standard web server on port 80.

Load Balancer name:

Create LB Inside:

Create an internal load balancer: [\(what's this?\)](#)

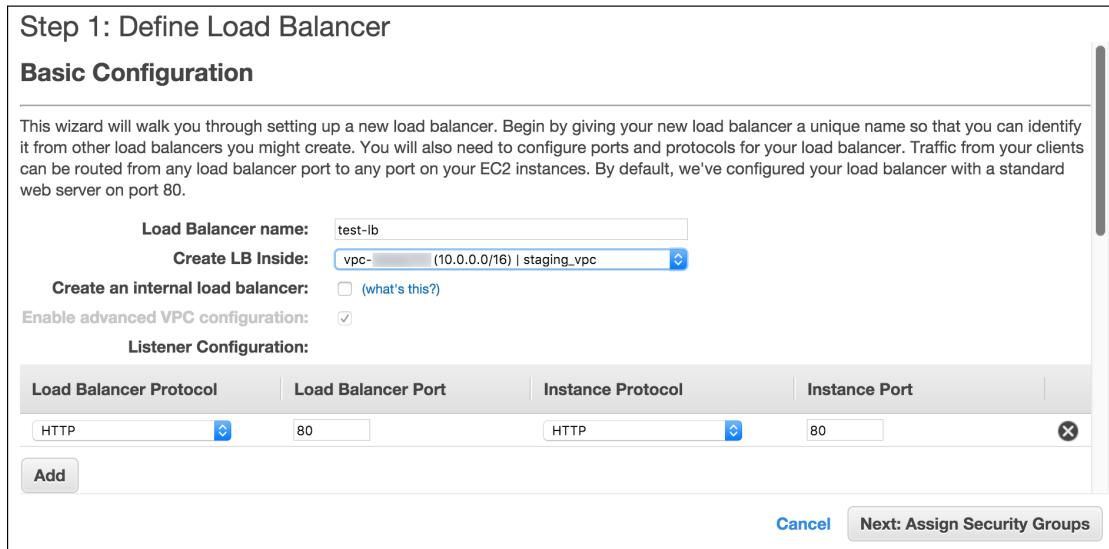
Enable advanced VPC configuration:

Listener Configuration:

Load Balancer Protocol	Load Balancer Port	Instance Protocol	Instance Port	
HTTP	80	HTTP	80	<input type="button" value="X"/>

Add

Cancel **Next: Assign Security Groups**



8. Under **Select Subnets**, select `staging_subnet_public_0` and `staging_subnet_public_1`. The available subnets for the VPC for your load balancer are displayed under **Available Subnets**. Click the icon in the **Actions** column for each subnet to attach. These subnets are moved under **Selected Subnets**.

You will need to select a Subnet for each Availability Zone where you wish traffic to be routed by your load balancer. If you have instances in only one Availability Zone, please select at least two Subnets in different Availability Zones to provide higher availability for your load balancer.

VPC vpc- (10.0.0.0/16) | staging_vpc

Available Subnets

Actions	Availability Zone	Subnet ID	Subnet CIDR	Name
	ap-southeast-2a	subnet-[REDACTED]	10.0.4.0/24	staging_subnet_private_2
	ap-southeast-2a	subnet-[REDACTED]	10.0.1.0/24	staging_subnet_private_0
	ap-southeast-2b	subnet-[REDACTED]	10.0.3.0/24	staging_subnet_private_1
	ap-southeast-2b	subnet-[REDACTED]	10.0.5.0/24	staging_subnet_private_3

Selected Subnets

Actions	Availability Zone	Subnet ID	Subnet CIDR	Name
	ap-southeast-2a	subnet-[REDACTED]	10.0.0.0/24	staging_subnet_public_0
	ap-southeast-2b	subnet-[REDACTED]	10.0.2.0/24	staging_subnet_public_1

[Cancel](#) [Next: Assign Security Groups](#)

9. Click on **Next: Assign Security Groups**.

Step 2 – Assign security groups to your load balancer

To assign security group to your load balancer:

1. On the **Assign Security Groups** page, select **Select an existing security group**.
2. Select `staging_sg_wordpress_lb` Security Group from the list.
3. Click **Next: Configure Security Settings**.

[1. Define Load Balancer](#) [2. Assign Security Groups](#) [3. Configure Security Settings](#) [4. Configure Health Check](#) [5. Add EC2 Instances](#) [6. Add Tags](#)

Step 2: Assign Security Groups

Security Group	Name	Description	Action
<input type="checkbox"/> sg-[REDACTED]	staging_sg_web	security group for webservers	Copy to new
<input type="checkbox"/> sg-[REDACTED]	staging_sg_wordpress	security group for wordpress servers	Copy to new
<input checked="" type="checkbox"/> sg-[REDACTED]	staging_sg_wordpress_lb	security group for wordpress load balancer	Copy to new

[Cancel](#) [Previous](#) [Next: Configure Security Settings](#)

Step 3 – Configure security settings

For this tutorial, you can click **Next: Configure Health Check** to continue to the next step. For more information about creating a HTTPS load balancer and using additional security features, see

<http://docs.aws.amazon.com/ElasticLoadBalancing/latest/DeveloperGuide/elb-https-load-balancers.html>.

Step 4 – Configure health checks for your EC2 instances

Elastic Load Balancing automatically checks the health of the EC2 instances for your load balancer. If Elastic Load Balancing finds an unhealthy instance, it stops sending traffic to the instance and reroutes traffic to healthy instances. In this step, you customize the health checks for your load balancer.

To configure health checks for your instances:

1. On the **Configure Health Check** page, do the following:
 - Leave **Ping Protocol** set to its default value, **HTTP**.
 - Leave **Ping Port** set to its default value, **80**.
 - In the **Ping Path** field, replace the default value with **/index.php**. This tells Elastic Load Balancing to send health check queries to the landing page of your WordPress server.
 - Leave the other fields set to their default values.

1. Define Load Balancer 2. Assign Security Groups 3. Configure Security Settings **4. Configure Health Check** 5. Add EC2 Instances 6. Add Tags

Step 4: Configure Health Check

Your load balancer will automatically perform health checks on your EC2 instances and only route traffic to instances that pass the health check. If an instance fails the health check, it is automatically removed from the load balancer. Customize the health check to meet your specific needs.

Ping Protocol: HTTP
Ping Port: 80
Ping Path: /index.php

Advanced Details

Response Timeout	5	seconds
Health Check Interval	30	seconds
Unhealthy Threshold	2	
Healthy Threshold	10	

Cancel **Previous** **Next: Add EC2 Instances**

2. Click **Next: Add EC2 Instances**.

Step 5 – Register EC2 instances with your load balancer

We will associate the ELB with our Auto Scaling Group later, so we can skip adding instance in this step.

Step 6 – Tag your load balancer (Optional)

You can tag your load balancer, or continue to the next step. Note that you can tag your load balancer later on.

To add tags to your load balancer:

1. On the **Add Tags** page, specify a key and a value for the tag.
2. To add another tag, click **Create Tag** and specify a key and a value for the tag.
3. After you are finished adding tags, click **Review and Create**.

Step 7 – Create and verify your load balancer

Before you create the load balancer, review the settings that you selected. After creating the load balancer, we will attach it to our ASG and you can verify that it's sending traffic to your EC2 instances.

To finish creating your load balancer:

1. On the **Review** page, check your settings. If you need to make changes, click the on corresponding link to edit the settings.

The screenshot shows the 'Step 7: Review' page of a wizard. At the top, there are tabs for each step: 1. Define Load Balancer, 2. Assign Security Groups, 3. Configure Security Settings, 4. Configure Health Check, 5. Add EC2 Instances, 6. Add Tags, and 7. Review. The 'Review' tab is highlighted with an orange underline. Below the tabs, the title 'Step 7: Review' is displayed, followed by the sub-instruction 'Please review the load balancer details before continuing'. The page is divided into two main sections: 'Define Load Balancer' and 'Configure Health Check'. Under 'Define Load Balancer', the configuration is summarized: Load Balancer name: test-lb, Scheme: internet-facing, and Port Configuration: 80 (HTTP) forwarding to 80 (HTTP). There is a link 'Edit load balancer definition' to the right. Under 'Configure Health Check', the settings are listed: Ping Target: HTTP:80/index.php, Timeout: 5 seconds, Interval: 30 seconds, Unhealthy Threshold: 2, and Healthy Threshold: 10. There is a link 'Edit health check' to the right. At the bottom right of the page are three buttons: 'Cancel', 'Previous', and a blue 'Create' button.

2. Click on **Create** to create your load balancer.
3. After you are notified that your load balancer was created, click on **Close**.
4. You can see your newly created Load Balancer in the EC2 console.

The screenshot shows the AWS Elastic Load Balancing console. At the top, there is a search bar labeled 'Search Load Balancers'. Below it is a table with columns: Load Balancer Name, DNS Name, Port Configuration, Availability Zones, and Instance Count. One row is visible for 'test-lb', which has a DNS name of 'test-lb-.ap-southea...', port configuration of '80 (HTTP) forwarding to 80 (...)', availability zones of 'ap-southeast-2b, ap-so...', and 0 instances.

Below the table, a section titled 'Load balancer: test-lb' is shown. It includes tabs for Description, Instances, Health Check, Monitoring, Security, Listeners, and Tags. The 'Description' tab is selected. It displays the DNS Name as 'test-lb-.ap-southeast-2.elb.amazonaws.com (A Record)'. A note below states: 'Note: Because the set of IP addresses associated with a LoadBalancer can change over time, you should never create an "A" record with any specific IP address. If you want to use a friendly DNS name for your load balancer instead of the name generated by the Elastic Load Balancing service, you should create a CNAME record for the LoadBalancer DNS name, or use Amazon Route 53 to create a hosted zone. For more information, see [Using Domain Names With Elastic Load Balancing](#).' It also shows the Scheme as 'internet-facing' and the Status as '0 of 0 instances in service'.

Attaching load balancer to ASG

Auto Scaling integrates with Elastic Load Balancing to enable you to attach one or more load balancers to an existing Auto Scaling group. After you attach the load balancer, it automatically registers the instances in the group and distributes incoming traffic across the instances.

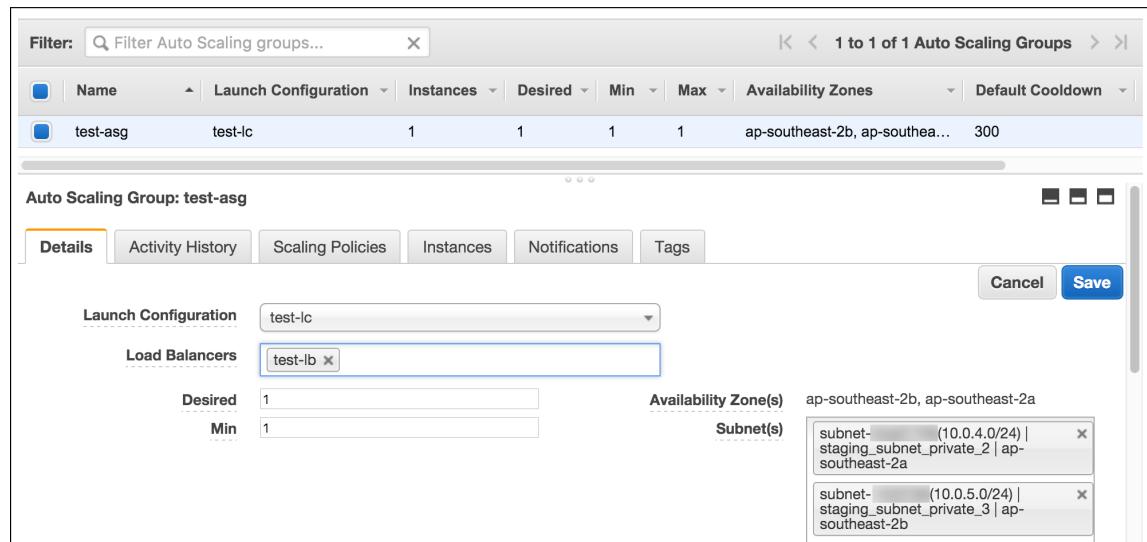
When you attach a load balancer, it enters the `Adding` state while registering the instances in the group. After all instances in the group are registered with the load balancer, it enters the `Added` state. After at least one registered instance passes the health check, it enters the `InService` state.

When you detach a load balancer, it enters the `Removing` state while deregistering the instances in the group. If connection draining is enabled, Elastic Load Balancing waits for in-flight requests to complete before deregistering the instances. Note that the instances remain running after they are deregistered (for more information visit <http://docs.aws.amazon.com/autoscaling/latest/DeveloperGuide/attach-load-balancer-asg.html>).

To attach a load balancer to a group:

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation pane, under **Auto Scaling**, click on **Auto Scaling Groups**.

3. Select your group.
4. In the bottom pane, on the **Details** tab, click on **Edit**.
5. In **Load Balancers**, select the load balancer.



6. Click on **Save**.

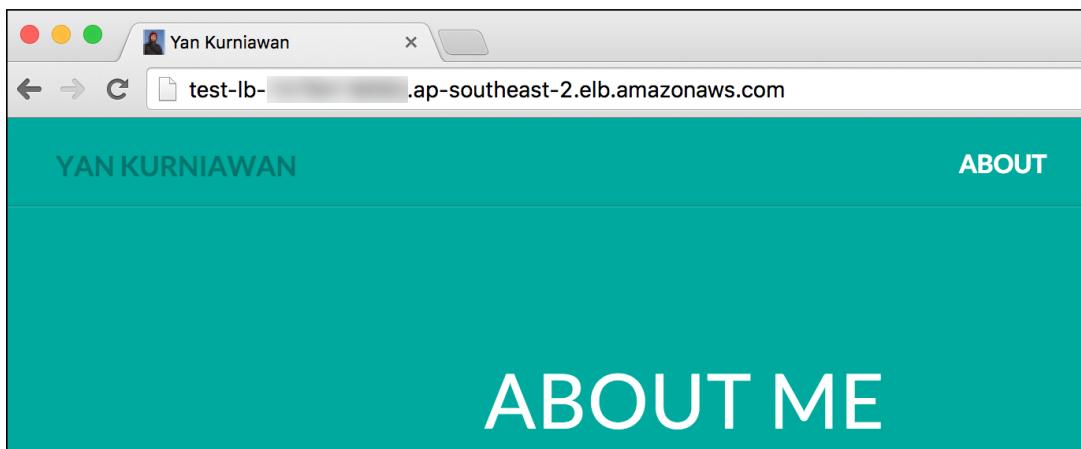
Verify your Load Balancer

To verify your load balancer:

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation pane, under **LOAD BALANCING**, click on **Load Balancers**.
3. Select your new load balancer.
4. In the bottom pane, on the **Description** tab, check the Status row. If it indicates that some of your instances are not in service, its probably because they are still in the registration process.

The screenshot shows the AWS Elastic Load Balancing console. At the top, there is a search bar labeled "Search Load Balancers" and a header with tabs: "Load Balancer Name", "DNS Name", "Port Configuration", "Availability Zones", and "Instance Count". Below the header, a table lists one load balancer: "test-lb" with DNS name "test-lb-...ap-southeast-2.elb.amazonaws.com", port configuration "80 (HTTP) forwarding to 80 (...)", availability zones "ap-southeast-2b, ap-so...", and instance count "1 Instance". Below the table, a section titled "Load balancer: test-lb" contains tabs: "Description" (selected), "Instances", "Health Check", "Monitoring", "Security", "Listeners", and "Tags". The "Description" tab shows the DNS Name "test-lb-...ap-southeast-2.elb.amazonaws.com (A Record)". A note below it states: "Note: Because the set of IP addresses associated with a LoadBalancer can change over time, you should never create an "A" record with any specific IP address. If you want to use a friendly DNS name for your load balancer instead of the name generated by the Elastic Load Balancing service, you should create a CNAME record for the LoadBalancer DNS name, or use Amazon Route 53 to create a hosted zone. For more information, see [Using Domain Names With Elastic Load Balancing](#)." The "Scheme" is listed as "internet-facing" and the "Status" is "1 of 1 instances in service".

5. After you've verified that at least one of your EC2 instances is `InService`, you can test your load balancer. Copy the string from the DNS Name field and paste it into the address field of an Internet-connected web browser (for example, `my-load-balancer-1234567890.us-west-2.elb.amazonaws.com`). If your load balancer is working, you'll see the default page of your HTTP server.



When you no longer need the load balancer, use the following procedure to detach it from your Auto Scaling group.

To detach a load balancer from a group:

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation pane, under **Auto Scaling**, click on **Auto Scaling Groups**.
3. Select your group.
4. In the bottom pane, on the **Details** tab, click on **Edit**.
5. In **Load Balancers**, remove the load balancer.
6. Click on **Save**.

Scheduled scaling

Scaling based on a schedule allows you to scale your application in response to predictable load changes. For example, every week the traffic to your web application starts to increase on Wednesday, remains high on Thursday, and starts to decrease on Friday. You can plan your scaling activities based on the predictable traffic patterns of your web application.

To configure your Auto Scaling group to scale based on a schedule, you need to create scheduled actions. A scheduled action tells Auto Scaling to perform a scaling action at certain time in future. To create a scheduled scaling action, you specify the start time at which you want the scaling action to take effect, and you specify the new minimum, maximum, and desired size you want for that group at that time. At the specified time, Auto Scaling updates the group to set the new values for minimum, maximum, and desired sizes, as specified by your scaling action.

You can create scheduled actions for scaling one time only or for scaling on a recurring schedule (for more information visit http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/schedule_time.html).

Considerations for scheduled actions

When you create a scheduled action, keep the following in mind:

- Auto Scaling guarantees the order of execution for scheduled actions within the same group, but not for scheduled actions across groups.

- A scheduled action generally executes within seconds. However, the action may be delayed for up to two minutes from the scheduled start time. Because Auto Scaling executes actions within an Auto Scaling group in the order they are specified, scheduled actions with scheduled start times close to each other may take longer to execute.
- You can create a maximum of 125 scheduled actions per month per Auto Scaling group. This allows scaling four times a day for a 31-day month for each Auto Scaling group.
- A scheduled action must have a unique time value. If you attempt to schedule an activity at a time when another existing activity is already scheduled, the call is rejected with an error message noting the conflict.
- Cooldown periods are not supported.

Create a scheduled action using the console

Complete the following procedure to create a scheduled action to scale your Auto Scaling group.

To create a scheduled action:

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation pane, under **Auto Scaling**, click on **Auto Scaling Groups**.
3. Select your Auto Scaling group from the list.
4. The bottom pane displays the details of your Auto Scaling group. Select the **Scheduled Actions** tab and then click on **Create Scheduled Action**.
5. In the **Create Scheduled Action** dialog box, do the following:
 - Specify the size of the group using at least one of **Min**, **Max**, and **Desired Capacity**.
 - Select an option from **Recurrence**. If you select **Once**, Auto Scaling performs the action at the specified time. If you select **Cron**, enter a CRON expression that specifies when Auto Scaling performs the action.
 - Specify the start and end time using **Start Time** and **End Time**.
 - Click on **Create**.

Update a scheduled action

If your requirements change, you can update a scheduled action.

To update a scheduled action:

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation pane, under **Auto Scaling**, click on **Auto Scaling Groups**.
3. Select your Auto Scaling group from the list.
4. The bottom pane displays the details of your Auto Scaling group. Select the **Scheduled Actions** tab, and then select the scheduled action.
5. Click on **Actions** and then select **Edit**.
6. In the **Edit Scheduled Action** dialog box, do the following:
 - Update the size of the group as needed using **Min**, **Max**, or **Desired Capacity**.
 - Update the specified recurrence as needed.
 - Update the start and end time as needed.
 - Click on **Save**.

Dynamic scaling

When you use Auto Scaling to scale dynamically, you must define how you want to scale in response to changing demand. For example, say you have a web application that currently runs on two instances. You want to launch two additional instances when the load on the current instances rises to 70 percent, and then you want to terminate those additional instances when the load falls to 40 percent. You can configure your Auto Scaling group to scale automatically based on these conditions.

An Auto Scaling group uses a combination of alarms and policies to determine when the conditions for scaling are met. An *alarm* is an object that watches over a single metric (for example, the average CPU utilization of the EC2 instances in your Auto Scaling group) over a specified time period. When the value of the metric breaches the threshold that you defined, for the number of time periods that you specified, the alarm performs one or more actions (such as sending messages to Auto Scaling). A *policy* is a set of instructions that tells Auto Scaling how to respond to alarm messages.

To set up dynamic scaling, you must create alarms and scaling policies and associate them with your Auto Scaling group. It's recommended that you create two policies for each scaling change that you want to perform: one policy to scale out and another policy to scale in. After the alarm sends a message to Auto Scaling, Auto Scaling executes the associated policy to scale your group in (by terminating instances) or out (by launching instances). The process is as follows (for more information visit <http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/as-scale-based-on-demand.html>):

- Amazon CloudWatch monitors the specified metrics for all the instances in the Auto Scaling group.
- As demand grows or shrinks, the change is reflected in the metrics.
- When the change in the metrics breaches the threshold of the CloudWatch alarm, the CloudWatch alarm performs an action. Depending on the breach, the action is a message sent to either the scale-in policy or the scale-out policy.
- After the Auto Scaling policy receives the message, Auto Scaling performs the scaling activity for the Auto Scaling group.
- This process continues until you delete either the scaling policies or the Auto Scaling group.

Scaling adjustment types

When a scaling policy is executed, it changes the current capacity of your Auto Scaling group using the scaling adjustment specified in the policy. A scaling adjustment can't change the capacity of the group above the maximum group size or below the minimum group size.

Auto Scaling supports the following adjustment types:

- **ChangeInCapacity:** Increase or decrease the current capacity of the group by the specified number of instances. A positive value increases the capacity and a negative adjustment value decreases the capacity.
Example: If the current capacity of the group is 3 instances and the adjustment is 5, then when this policy is performed, Auto Scaling adds 5 instances to the group for a total of 8 instances.

- **ExactCapacity:** Change the current capacity of the group to the specified number of instances. Note that you must specify a positive value with this adjustment type.

Example: If the current capacity of the group is 3 instances and the adjustment is 5, then when this policy is performed, Auto Scaling changes the capacity to 5 instances.

- **PercentChangeInCapacity:** Increment or decrement the current capacity of the group by the specified percentage. A positive value increases the capacity and a negative value decreases the capacity. If the resulting value is not an integer, Auto Scaling rounds it as follows:

- Values greater than 1 are rounded down. For example, 12.7 is rounded to 12.
- Values between 0 and 1 are rounded to 1. For example, .67 is rounded to 1.
- Values between 0 and -1 are rounded to -1. For example, -.58 is rounded to -1
- Values less than -1 are rounded up. For example, -6.67 is rounded to -6.

Example: If the current capacity is 10 instances and the adjustment is 10 percent, then when this policy is performed, Auto Scaling adds 1 instance to the group for a total of 11 instances.

Scaling policy types

When you create a scaling policy, you must specify its policy type. The policy type determines how the scaling action is performed. Auto Scaling supports the following policy types:

- **Simple scaling:** Increase or decrease the current capacity of the group based on a single scaling adjustment.
- **Step scaling:** Increase or decrease the current capacity of the group based on a set of scaling adjustments, known as step adjustments, that vary based on the size of the alarm breach.

Simple scaling policies

After a scaling activity is started, the policy must wait for the scaling activity or health check replacement to complete and the cooldown period to expire before it can respond to additional alarms. Cooldown periods help to prevent Auto Scaling from initiating additional scaling activities before the effects of previous activities are visible. You can use the default cooldown period associated with your Auto Scaling group, or you can override the default by specifying a cooldown period for your policy.

Note that Auto Scaling originally supported only this type of scaling policy. If you created your scaling policy before policy types were introduced, your policy is treated as a simple scaling policy.

Step scaling policies

After a scaling activity is started, the policy continues to respond to additional alarms, even while a scaling activity or health check replacement is in progress. Therefore, all alarms that are breached are evaluated by Auto Scaling as it receives the alarm messages. If you are creating a policy to scale out, you can specify the estimated warm-up time that it will take for a newly launched instance to be ready to contribute to the aggregated metrics.

Cooldown periods are not supported for step scaling policies. Therefore, you can't specify a cooldown period for these policies and the default cooldown period for the group doesn't apply. It's recommended that you use step scaling policies even if you have a single step adjustment, because Amazon continuously evaluates alarms and do not lock the group during scaling activities or health check replacements.

Step adjustments

When you create a step scaling policy, you add one or more step adjustments, which enables you to scale based on the size of the alarm breach. Each step adjustment specifies a lower bound for the metric value, an upper bound for the metric value, and the amount by which to scale, based on the scaling adjustment type.

There are a few rules for the step adjustments for your policy:

- The ranges of your step adjustments can't overlap or have a gap.
- At most one step adjustment can have a null lower bound (negative infinity). If one step adjustment has a negative lower bound, then there must be a step adjustment with a null lower bound.

- At most one step adjustment can have a null upper bound (positive infinity). If one step adjustment has a positive upper bound, then there must be a step adjustment with a null upper bound.
- The upper and lower bound can't be null in the same step adjustment.
- If the metric value is above the breach threshold, the lower bound is inclusive and the upper bound is exclusive. If the metric value is below the breach threshold, the lower bound is exclusive and the upper bound is inclusive.

If you are using the API or the CLI, you specify the upper and lower bounds relative to the value of the aggregated metric. If you are using the AWS Management Console, you specify the upper and lower bounds as absolute values.

Auto Scaling applies the aggregation type to the metric data points from all instances and compares the aggregated metric value against the upper and lower bounds defined by the step adjustments to determine which step adjustment to perform. For example, suppose that you have an alarm with a breach threshold of 50 and a scaling adjustment type of PercentChangeInCapacity. You also have scale-out and scale-in policies with the following step adjustments:

Scale-out policy:

Lower bound	Upper bound	Adjustment	Metric value
0	10	0	50 <= value < 60
10	20	10	60 <= value < 70
20	null	30	70 <= value < +infinity

Scale-in policy:

Lower bound	Upper bound	Adjustment	Metric value
-10	0	0	40 < value <= 50
-20	-10	-10	30 < value <= 40
null	-20	-30	-infinity < value <= 30

Your group has both a current capacity and a desired capacity of 10 instances. The group maintains its current and desired capacity while the aggregated metric value is greater than 40 and less than 60.

If the metric value gets to 60, Auto Scaling increases the desired capacity of the group by 1 instance, to 11 instances, based on the second step adjustment of the scale-out policy (add 10 percent of 10 instances). After the new instance is running and its specified warm-up time has expired, Auto Scaling increases the current capacity of the group to 11 instances. If the metric value rises to 70 even after this increase in capacity, Auto Scaling increases the desired capacity of the group by another 3 instances, to 14 instances, based on the third step adjustment of the scale-out policy (add 30 percent of 11 instances, 3.3 instances, rounded down to 3 instances).

If the metric value gets to 40, Auto Scaling decreases the desired capacity of the group by 1 instance, to 13 instances, based on the second step adjustment of the scale-in policy (remove 10 percent of 14 instances, 1.4 instances, rounded down to 1 instance). If the metric value falls to 30 even after this decrease in capacity, Auto Scaling decreases the desired capacity of the group by another 3 instances, to 10 instances, based on the third step adjustment of the scale-in policy (remove 30 percent of 13 instances, 3.9 instances, rounded down to 3 instances).

Instance warm-up

With step scaling policies, you can specify the number of seconds that it takes for a newly launched instance to warm up. Until its specified warm-up time has expired, an instance is not counted toward the aggregated metrics of the Auto Scaling group.

While scaling out, Auto Scaling does not consider instances that are warming up as part of the current capacity of the group. Therefore, multiple alarm breaches that fall in the range of the same step adjustment result in a single scaling activity. This ensures that Auto Scaling doesn't add more instances than you need. Using the example in the previous section, suppose that the metric gets to 60, and then it gets to 62 while the new instance is still warming up. The current capacity is still 10 instances, so Auto Scaling should add 1 instance (10 percent of 10 instances), but the desired capacity of the group is already 11 instances, so Auto Scaling does not increase the desired capacity further. However, if the metric gets to 70 while the new instance is still warming up, Auto Scaling should add 3 instances (30 percent of 10 instances), but the desired capacity of the group is already 11, so Auto Scaling adds only 2 instances, for a new desired capacity of 13 instances.

While scaling in, Auto Scaling considers instances that are terminating as part of the current capacity of the group. Therefore, Auto Scaling won't remove more instances from the Auto Scaling group than necessary.

Note that a scale-in activity can't start while a scale-out activity is in progress.

Scaling based on metrics

You can create a scaling policy that uses CloudWatch alarms to determine when your Auto Scaling group should scale out or scale in. Each CloudWatch alarm watches a single metric and sends messages to Auto Scaling when the metric breaches a threshold that you specify in your policy. You can use alarms to monitor any of the metrics that the services in AWS that you're using send to CloudWatch, or you can create and monitor your own custom metrics.

When you create a CloudWatch alarm, you can specify an Amazon SNS topic to send an email notification to when the alarm changes state (for more information visit http://docs.aws.amazon.com/autoscaling/latest/userguide/policy_creating.html).

Create an Auto Scaling group with scaling policies

Use the console to create an Auto Scaling group with two scaling policies: a scale out policy that increases the capacity of the group by 30 percent, and a scale in policy that decreases the capacity of the group to two instances.

To create an Auto Scaling group with scaling based on metrics:

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. On the navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**.
3. Choose **Create Auto Scaling group**.
4. On the **Create Auto Scaling Group** page, do one of the following:
 - Select **Create an Auto Scaling group from an existing launch configuration**, select an existing launch configuration, and then choose **Next Step**.
 - If you don't have a launch configuration that you'd like to use, choose **Create a new launch configuration** and follow the directions.

5. On the **Configure Auto Scaling group details** page, do the following:
 - For **Group name**, type a name for your Auto Scaling group.
 - For **Group size**, type the desired capacity for your Auto Scaling group.
 - If the launch configuration specifies instances that require a VPC, such as T2 instances, you must select a VPC from **Network**. Otherwise, if your AWS account supports EC2-Classic and the instances don't require a VPC, you can select either `Launch info EC2-Classic` or a VPC.
 - If you selected a VPC in the previous step, select one or more subnets from **Subnet**. If you selected EC2-Classic in the previous step, select one or more Availability Zones from **Availability Zone(s)**.
 - Choose **Next: Configure scaling policies**.
6. On the **Configure scaling policies** page, do the following:
 - Select **Use scaling policies to adjust the capacity of this group**.
 - Specify the minimum and maximum size for your Auto Scaling group using the row that begins with **Scale between**. For example, if your group is already at its maximum size, you need to specify a new maximum in order to scale out.
 - Specify your scale out policy under **Increase Group Size**. You can optionally specify a name for the policy, then choose **Add new alarm**.
 - On the **Create Alarm** page, choose **create topic**. For **Send a notification to**, type a name for the SNS topic. For **With these recipients**, type one or more email addresses to receive notification. If you want, you can replace the default name for your alarm with a custom name. Next, specify the metric and the criteria for the policy. For example, you can leave the default settings for **Whenever** (Average of CPU Utilization). For **Is**, choose `>=` and type 80 percent. For **For at least**, type 1 consecutive period of 5 Minutes. Choose **Create Alarm**.

Create Alarm

You can use CloudWatch alarms to be notified automatically whenever metric data reaches a level you define.

To edit an alarm, first choose whom to notify and then define when the notification should be sent.

Send a notification to: AddCapacityNotification [cancel](#)

With these recipients: notification@example.com

Whenever: Average of CPU Utilization
Is: >= 80 Percent

For at least: 1 consecutive period(s) of 5 Minutes

Name of alarm: AddCapacityAlarm

CPU Utilization Percent

Time	Utilization (%)
02:00	80
04:00	80
06:00	80

test-asg

[Cancel](#) [Create Alarm](#)

- For **Take the action**, choose Add, type 30 in the next field, and then choose percent of group. By default, the lower bound for this step adjustment is the alarm threshold and the upper bound is null (positive infinity). To add another step adjustment, choose **Add step**.
(Optional) Amazon recommends that you use the default to create both scaling policies with steps. If you need to create simple scaling policies, choose **Create a simple scaling policy**.

Scale between 1 and 5 instances. These will be the minimum and maximum size of your group.

Increase Group Size

Name: Increase Group Size

Execute policy when: AddCapacityAlarm [Edit](#) [Remove](#)
breaches the alarm threshold: CPUUtilization >= 80 for 300 seconds
for the metric dimensions AutoScalingGroupName = test-asg

Take the action: Add 30 percent of group when 80 <= CPUUtilization < +infinity

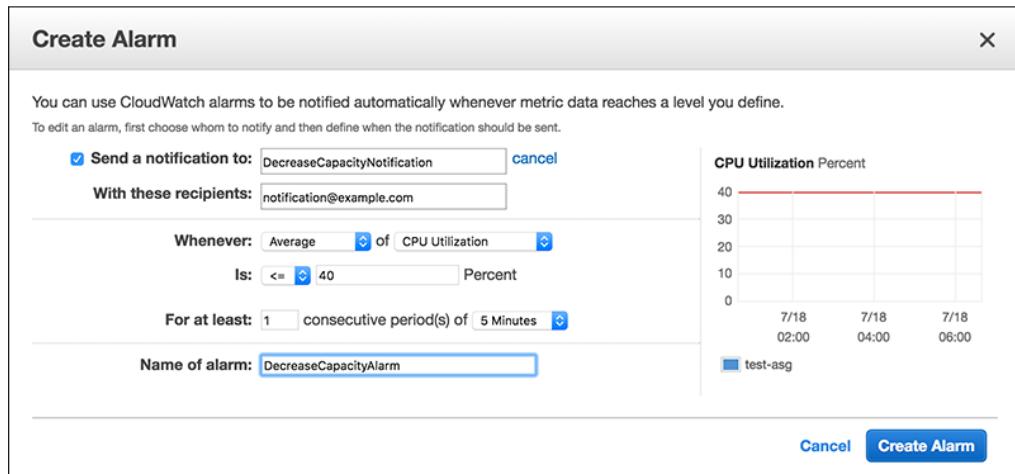
[Add step](#) [i](#)

Add instances in increments of at least 1 instance(s)

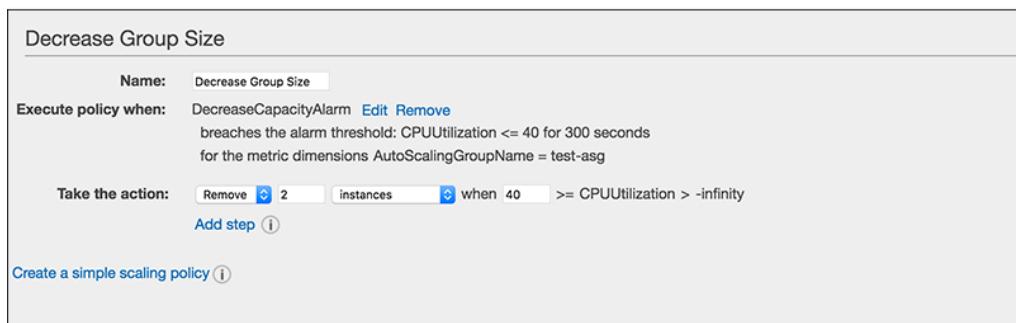
Instances need: 300 seconds to warm up after each step

[Create a simple scaling policy](#) [i](#)

- Specify your scale in policy under **Decrease Group Size**. You can optionally specify a name for the policy, then choose **Add new alarm**.
- On the **Create Alarm** page, you can select the same notification that you created for the scale out policy or create a new one for the scale in policy. If you want, you can replace the default name for your alarm with a custom name. Keep the default settings for **Whenever** (Average of CPU Utilization). For **Is**, choose \leq and type 40 percent. For **For at least**, type 1 consecutive period of 5 Minutes. Choose **Create Alarm**.



- For **Take the action**, choose **Remove**, type 2 in the next field, and then choose **instances**. By default, the upper bound for this step adjustment is the alarm threshold and the lower bound is null (negative infinity). To add another step adjustment, choose **Add step**. (Optional) Amazon recommends that you use the default to create both scaling policies with steps. If you need to create simple scaling policies, choose **Create a simple scaling policy**.



- Choose **Review**.
 - On the **Review** page, choose **Create Auto Scaling group**.
7. Use the following steps to verify the scaling policies for your Auto Scaling group.
- The **Auto Scaling Group creation status** page confirms that your Auto Scaling group was successfully created. Choose **View your Auto Scaling Groups**.
 - On the **Auto Scaling Groups** page, select the Auto Scaling group that you just created.
 - On the **Activity History** tab, the **Status** column shows whether your Auto Scaling group has successfully launched instances.
 - On the **Instances** tab, the **Lifecycle** column contains the state of your instances. It takes a short time for an instance to launch. After the instance starts, its lifecycle state changes to **InService**.
 - The **Health Status** column shows the result of the EC2 instance health check on your instance.
 - On the **Scaling Policies** tab, you can see the policies that you created for the Auto Scaling group.

Add a scaling policy to an Auto Scaling group

Use the console to add a scaling policy to an existing Auto Scaling group.

To update an Auto Scaling group with scaling based on metrics:

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. On the navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**.

3. Select the Auto Scaling group.
4. On the **Scaling Policies** tab, choose **Add policy**.
5. For **Name**, type a name for the policy, and then choose **Create new alarm**.
6. On the **Create Alarm** page, choose **create topic**. For **Send a notification to**, type a name for the SNS topic. For **With these recipients**, type one or more email addresses to receive notification. If you want, you can replace the default name for your alarm with a custom name. Next, specify the metric and the criteria for the alarm, using **Whenever, Is, and For at least**. Choose **Create Alarm**.
7. Specify the scaling activity for the policy using **Take the action**. By default, the lower bound for this step adjustment is the alarm threshold and the upper bound is null (positive infinity). To add another step adjustment, choose **Add step**.
 - (Optional) Amazon recommends that you use the default to create both scaling policies with steps. If you need to create simple scaling policies, choose **Create a simple scaling policy**.
8. Choose **Create**.

More about Auto Scaling

Here lets talk more about auto scaling.

Scaling based on amazon SQS

Amazon Simple Queue Service (Amazon SQS) is a scalable message queuing system that stores messages as they travel between various components of your application architecture. Amazon SQS enables web service applications to quickly and reliably queue messages that are generated by one component and consumed by another component. A queue is a temporary repository for messages that are awaiting processing.

Queues provide a convenient mechanism to determine the load on an application. You can use the length of the queue (number of messages available for retrieval from the queue) to determine the load. Because each message in the queue represents a request from a user, measuring the length of the queue is a fair approximation of the load on the application. CloudWatch integrates with Amazon SQS to collect, view, and analyze metrics from SQS queues. You can use the metrics sent by Amazon SQS to determine the length of the SQS queue at any point in time. For a list of all the metrics that Amazon SQS sends to CloudWatch, see Amazon SQS Metrics in the Amazon Simple Queue Service Developer Guide.

For more information about scaling based on Amazon SQS, see
<http://docs.aws.amazon.com/autoscaling/latest/userguide/as-using-sqs-queue.html>.

Auto Scaling cooldowns

The Auto Scaling cooldown period is a configurable setting for your Auto Scaling group that helps to ensure that Auto Scaling doesn't launch or terminate additional instances before the previous scaling activity takes effect. After the Auto Scaling group dynamically scales using a simple scaling policy, Auto Scaling waits for the cooldown period to complete before resuming scaling activities. When you manually scale your Auto Scaling group, the default is not to wait for the cooldown period, but you can override the default and honor the cooldown period. Note that if an instance becomes unhealthy, Auto Scaling does not wait for the cooldown period to complete before replacing the unhealthy instance.

For more information about Auto Scaling cool downs, see <http://docs.aws.amazon.com/autoscaling/latest/userguide/Cooldown.html>.

Auto Scaling instance termination

With each Auto Scaling group, you control when Auto Scaling adds instances (referred to as scaling out) or remove instances (referred to as scaling in) from your network architecture. You can scale the size of your group manually by attaching and detaching instances, or you can automate the process through the use of a scaling policy.

When you have Auto Scaling automatically scale in, you must decide which instances Auto Scaling should terminate first. You can configure this through the use of a termination policy.

You can also use instance protection to prevent Auto Scaling from selecting specific instances for termination when scaling in.

For more information about Auto Scaling instance termination, see
<http://docs.aws.amazon.com/autoscaling/latest/userguide/as-instance-termination.html>.

Auto Scaling lifecycle hooks

Auto Scaling lifecycle hooks enable you to perform custom actions as Auto Scaling launches or terminates instances. For example, you could install or configure software on newly launched instances, or download log files from an instance before it terminates.

For more information about Auto Scaling Lifecycle Hooks, see

<http://docs.aws.amazon.com/autoscaling/latest/userguide/lifecycle-hooks.html>.

Temporarily removing instances from your Auto Scaling group

Auto Scaling enables you to put an instance that is in the `InService` state into the `Standby` state, update or troubleshoot the instance, and then return the instance to service. Instances that are on standby are still part of the Auto Scaling group, but they do not actively handle application traffic.

You are billed for instances that are in a standby state.



For example, you can change the launch configuration for an Auto Scaling group at any time, and any subsequent instances that the Auto Scaling group launches use this configuration. However, the Auto Scaling group does not update the instances that are currently in service. You can either terminate these instances and let the Auto Scaling group replace them, or you can put the instances on standby, update the software, and then put the instances back in service.

For more information about removing instance temporarily from ASG, see

<http://docs.aws.amazon.com/autoscaling/latest/userguide/as-enter-exit-standby.html>.

Suspending and resuming Auto Scaling processes

Auto Scaling enables you to suspend and then resume one or more of the Auto Scaling processes in your Auto Scaling group. This can be very useful when you want to investigate a configuration problem or other issue with your web application and then make changes to your application, without triggering the Auto Scaling process.

Auto Scaling might suspend processes for Auto Scaling groups that repeatedly fail to launch instances. This is known as an administrative suspension, and most commonly applies to Auto Scaling groups that have been trying to launch instances for over 24 hours but have not succeeded in launching any instances. You can resume processes suspended for administrative reasons.

For more information about suspending and resuming Auto Scaling processes, see
<http://docs.aws.amazon.com/autoscaling/latest/userguide/as-suspend-resume-processes.html>.

Latest AWS Auto Scaling user guide

You can see the latest AWS Auto Scaling user guide here

<http://docs.aws.amazon.com/autoscaling/latest/userguide/WhatIsAutoScaling.html>.

13

ELB and Auto Scaling with Ansible

In this chapter we will use Ansible to create all of the remaining AWS resources needed to complete our Multi-Tier WordPress project from [Chapter 10, Project 2 – A Multi-Tier WordPress Site](#). We will create the public ELB, launch configuration, ASG, Auto Scaling policy, and Cloudwatch alarms. We will use these Ansible modules in this chapter:

- **ec2_elb_lb:** Creates or destroys Amazon ELB
(http://docs.ansible.com/ansible/ec2_elb_lb_module.html)
- **ec2_lc:** Create or delete AWS Auto Scaling Launch Configurations
(http://docs.ansible.com/ansible/ec2_lc_module.html)
- **ec2_asg:** Create or delete AWS **Auto Scaling Groups (ASG)**
(http://docs.ansible.com/ansible/ec2_asg_module.html)
- **ec2_scaling_policy:** Create or delete AWS scaling policies for ASG (http://docs.ansible.com/ansible/ec2_scaling_policy_module.html)
- **ec2_metric_alarm:** Create/update or delete AWS Cloudwatch metric alarms
(http://docs.ansible.com/ansible/ec2_metric_alarm_module.html)

The **ec2_elb_lb** module

Creates or destroys Amazon ELB. Options for this module are listed in the following table:

parameter	required	default	choices	comments
access_logs (added in 2.0)	no	None		An associative array of access logs configuration settings
aws_access_key	no			AWS access key. If not set then the value of the AWS_ACCESS_KEY_ID, AWS_ACCESS_KEY, or EC2_ACCESS_KEY environment variable is used. aliases: ec2_access_key, access_key
aws_secret_key	no			AWS secret key. If not set then the value of the AWS_SECRET_ACCESS_KEY, AWS_SECRET_KEY, or EC2_SECRET_KEY environment variable is used. aliases: ec2_secret_key, secret_key
connection_draining_timeout (added in 1.8)	no			Wait a specified timeout allowing connections to drain before terminating an instance
cross_az_load_balancing (added in 1.8)	no	no	yes no	Distribute load across all configured Availability Zones
ec2_url	no			URL to use to connect to EC2 or your Eucalyptus cloud (by default the module will use EC2 endpoints). Ignored for modules where region is required. Must be specified if region is not used. If not set then the value of the EC2_URL environment variable, if any, is used
health_check	no	None		An associative array of health check configuration settings
idle_timeout (added in 2.0)	no			ELB connections from clients and to servers are timed out after this amount of time
instance_ids (added in 2.1)	no			List of instance ids to attach to this ELB
listeners	no			List of ports/protocols for this ELB to listen on
name	yes			The name of the ELB
profile (added in 1.6)	no			uses a boto profile. Only works with boto >= 2.24.0
purge_instance_	no			Purge existing instance ids on ELB that are not

parameter	required	default	choices	comments
ids (added in 2.1)				found in instance_ids
purge_listeners	no	True		Purge existing listeners on ELB that are not found in listeners
purge_subnets (added in 1.7)	no			Purge existing subnet on ELB that are not found in subnets
purge_zones	no			Purge existing availability zones on ELB that are not found in zones
region	no			The AWS region to use. If not specified then the value of the AWS_REGION or EC2_REGION environment variable, if any, is used. aliases: aws_region, ec2_region ⁹⁷
scheme (added in 1.7)	no	internet-facing		The scheme to use when creating the ELB. For a private VPC-visible ELB use 'internal'.
security_group_ids (added in 1.6)	no	None		A list of security groups to apply to the elb
security_group_names (added in 2.0)	no	None		A list of security group names to apply to the elb
security_token (added in 1.6)	no			AWS STS security token. If not set then the value of the AWS_SECURITY_TOKEN or EC2_SECURITY_TOKEN environment variable is used. aliases: access_token
state	yes		present absent	Create or destroy the ELB
stickiness (added in 2.0)	no			An associative array of stickiness policy settings. Policy will be applied to all listeners.
subnets (added in 1.7)	no	None		A list of VPC subnets to use when creating ELB. Zones should be empty if using this.
tags (added in 2.1)	no			An associative array of tags. To delete all tags, supply an empty dict.
validate_certs	no	yes	yes no	When set to "no", SSL certificates will not be validated for boto versions >= 2.6.0.
wait (added in 2.1)	no		yes no	When specified, Ansible will check the status of the load balancer to ensure it has been success-

parameter	required	default	choices	comments
			no	fully removed from AWS.
wait_timeout <small>(added in 2.1)</small>	no	60		Used in conjunction with wait. Number of seconds to wait for the elb to be terminated. A maximum of 600 seconds (10 minutes) is allowed.
subnets <small>(added in 1.7)</small>	no	None		A list of VPC subnets to use when creating ELB. Zones should be empty if using this.

The ec2_lc module

Create or delete AWS Auto Scaling Launch Configurations. Options for this module are listed in the following table:

parameter	required	default	choices	comments
assign_public_ip (added in 1.8)	no			Used for Auto Scaling groups that launch instances into an Amazon Virtual Private Cloud. Specifies whether to assign a public IP address to each instance launched in a Amazon VPC.
aws_access_key	no			AWS access key. If not set then the value of the AWS_ACCESS_KEY_ID, AWS_ACCESS_KEY, or EC2_ACCESS_KEY environment variable is used. aliases: ec2_access_key, access_key
aws_secret_key	no			AWS secret key. If not set then the value of the AWS_SECRET_ACCESS_KEY, AWS_SECRET_KEY, or EC2_SECRET_KEY environment variable is used. aliases: ec2_secret_key, secret_key
classic_link_vpc_id (added in 2.0)	no			Id of ClassicLink enabled VPC
classic_link_vpc_security_groups (added in 2.0)	no			A list of security group id's with which to associate the ClassicLink VPC instances.
ebs_optimized (added in 1.8)	no			Specifies whether the instance is optimized for EBS I/O (true) or not (false).
ec2_url	no			URL to use to connect to EC2 or your Eucalyptus cloud (by default the module will use EC2 endpoints). Ignored for modules where region is required. Must be specified if region is not used. If not set then the value of the EC2_URL environment variable, if any, is used
image_id	no			The AMI unique identifier to be used for the group
instance_monitoring	no			whether instances in group are launched with detailed monitoring.
instance_profile_name (added in 1.8)	no			The name or the Amazon Resource Name (ARN) of the instance profile associated with the IAM role for the instances.

parameter	required	default	choices	comments
<code>instance_type</code>	yes			instance type to use for the instance
<code>kernel_id</code>	no			Kernel id for the EC2 instance
<code>key_name</code>	no			The SSH key name to be used for access to managed instances
<code>name</code>	yes			Unique name for configuration
<code>profile</code> (added in 1.6)	no			uses a boto profile. Only works with boto >= 2.24.0
<code>ramdisk_id</code> (added in 1.8)	no			A RAM disk id for the instances.
<code>region</code>	no			The AWS region to use. If not specified then the value of the <code>AWS_REGION</code> or <code>EC2_REGION</code> environment variable, if any, is used. aliases: <code>aws_region</code> , <code>ec2_region</code> ⁹⁹
<code>security_groups</code>	no			A list of security groups to apply to the instances. For VPC instances, specify security group IDs. For EC2-Classic, specify either security group names or IDs.
<code>security_token</code> (added in 1.6)	no			AWS STS security token. If not set then the value of the <code>AWS_SECURITY_TOKEN</code> or <code>EC2_SECURITY_TOKEN</code> environment variable is used. aliases: <code>access_token</code>
<code>spot_price</code>	no			The spot price you are bidding. Only applies for an ASG with spot instances.
<code>state</code>	yes		present absent	Create or delete the launch configuration
<code>user_data</code>	no			opaque blob of data which is made available to the ec2 instance
<code>validate_certs</code>	no	yes	yes no	When set to "no", SSL certificates will not be validated for boto versions >= 2.6.0.
<code>volumes</code>	no			a list of volume dicts, each containing device name and optionally ephemeral id or snapshot id. Size and type (and number of iops for io device type) must be specified for a new volume or a root volume, and may be passed for a snapshot volume. For any volume, a volume size less than 1

parameter	required	default	choices	comments
				will be interpreted as a request not to create the volume.

The ec2_asg module

Create or delete AWS Auto Scaling Groups (ASG). Options for this module:

parameter	required	default	choices	comments
availability_zones	no			List of availability zone names in which to create the group. Defaults to all the AZs in the region if vpc_zone_identifier is not set.
aws_access_key	no			AWS access key. If not set then the value of the AWS_ACCESS_KEY_ID, AWS_ACCESS_KEY, or EC2_ACCESS_KEY environment variable is used. aliases: ec2_access_key, access_key
aws_secret_key	no			AWS secret key. If not set then the value of the AWS_SECRET_ACCESS_KEY, AWS_SECRET_KEY, or EC2_SECRET_KEY environment variable is used. aliases: ec2_secret_key, secret_key
default_cooldown (added in 2.0)	no	300s		The number of seconds after a scaling activity completes before another can begin.
desired_capacity	no			Desired number of instances in group, if unspecified then the current group value will be used.
ec2_url	no			URL to use to connect to EC2 or your Eucalyptus cloud (by default the module will use EC2 endpoints). Ignored for modules where region is required. Must be specified if region is not used. If not set then the value of the EC2_URL environment variable, if any, is used
health_check_period (added in 1.7)	no	500s		Length of time in seconds after a new EC2 instance comes into service that Auto Scaling starts checking its health.
health_check_type (added in 1.7)	no	EC2 ELB		The service you want the health status from, Amazon EC2 or Elastic Load Balancer.
launch_config_name	yes			Name of the Launch configuration to use for the group. See the ec2_lc module for managing these
lc_check (added in 1.8)	no	True		Check to make sure instances that are being replaced with replace_instances do not already have the current launch_config.
load_balancers	no			List of ELB names to use for the group

parameter	required	default	choices	comments
max_size	no			Maximum number of instances in group, if unspecified then the current group value will be used.
min_size	no			Minimum number of instances in group, if unspecified then the current group value will be used.
name	yes			Unique name for group to be created or deleted
profile (added in 1.6)	no			uses a boto profile. Only works with boto >= 2.24.0
region	no			The AWS region to use. If not specified then the value of the AWS_REGION or EC2_REGION environment variable, if any, is used. aliases: aws_region, ec2_region ¹⁰¹
replace_all_instances (added in 1.8)	no			In a rolling fashion, replace all instances with an old launch configuration with one from the current launch configuration.
replace_batch_size (added in 1.8)	no	1		Number of instances you'd like to replace at a time. Used with replace_all_instances.
replace_instances (added in 1.8)	no	None		List of instance_ids belonging to the named ASG that you would like to terminate and be replaced with instances matching the current launch configuration.
security_token (added in 1.6)	no			AWS STS security token. If not set then the value of the AWS_SECURITY_TOKEN or EC2_SECURITY_TOKEN environment variable is used. aliases: access_token
state	yes		present absent	Create or destroy the ASG.
tags (added in 1.7)	no	None		A list of tags to add to the ASG. Optional key is propagate_at_launch, which defaults to true.
termination_policies (added in 2.0)	no	Default		An ordered list of criteria used for selecting instances to be removed from the ASG when reducing capacity. For 'Default', when used to create a new auto scaling group, the "Default" value is used. When used to change an existent auto scaling group, the current termination

parameter	required	default	choices	comments
				policies are maintained.
validate_certs	no	yes	yes no	When set to “no”, SSL certificates will not be validated for boto versions >= 2.6.0.
vpc_zone_identifier	no	None		List of VPC subnets to use
wait_for_instances (added in 1.9)	no	True		Wait for the ASG instances to be in a ready state before exiting. If instances are behind an ELB, it will wait until the ELB determines all instances have a lifecycle_state of “InService” and a health_status of “Healthy”.
wait_timeout (added in 1.8)	no	300		how long before wait instances to become viable when replaced. Used in conjunction with <code>instance_ids</code> option.

Valid options for `termination_policies`: `OldestInstance`, `NewestInstance`, `OldestLaunchConfiguration`, `ClosestToNextInstanceHour`, `Default`.

The ec2_scaling_policy module

Create or delete AWS scaling policies for ASG. Options for this module:

parameter	required	default	choices	comments
adjustment_type	no			The type of change in capacity of the ASG
asg_name	yes			Name of the associated autoscaling group
aws_access_key	no			AWS access key. If not set then the value of the AWS_ACCESS_KEY_ID, AWS_ACCESS_KEY, or EC2_ACCESS_KEY environment variable is used. aliases: ec2_access_key, access_key
aws_secret_key	no			AWS secret key. If not set then the value of the AWS_SECRET_ACCESS_KEY, AWS_SECRET_KEY, or EC2_SECRET_KEY environment variable is used. aliases: ec2_secret_key, secret_key
cooldown	no			The minimum period of time between which auto scaling actions can take place
ec2_url	no			URL to use to connect to EC2 or your Eucalyptus cloud (by default the module will use EC2 endpoints). Ignored for modules where region is required. Must be specified if region is not used. If not set then the value of the EC2_URL environment variable, if any, is used
min_adjustment_step	no			Minimum amount of adjustment when policy is triggered
name	yes			Unique name for the scaling policy
profile (added in 1.6)	no			uses a boto profile. Only works with boto >= 2.24.0
region	no			The AWS region to use. If not specified then the value of the AWS_REGION or EC2_REGION environment variable, if any, is used. aliases: aws_region, ec2_region ¹⁰³
scaling_adjustment	no			The amount by which the ASG is adjusted by the policy.
security_token (added in 1.6)	no			AWS STS security token. If not set then the value of the AWS_SECURITY_TOKEN or EC2_SECURITY_TOKEN environment variable is used.

parameter	required	default	choices	comments
				aliases: access_token
<code>state</code>	<code>yes</code>		<code>present</code> <code>absent</code>	Create or destroy the ELB
<code>validate_certs</code>	<code>no</code>	<code>yes</code>	<code>yes</code> <code>no</code>	When set to “no”, SSL certificates will not be validated for boto versions >= 2.6.0.

Valid options for `adjustment_type`: `ChangeInCapacity`, `ExactCapacity`, `PercentChangeInCapacity`.

The ec2_metric_alarm module

Create/update or delete AWS Cloudwatch metric alarms. Options for this module are listed in the following table:

parameter	required	default	choices	comments
alarm_actions	no			A list of the names action(s) taken when the alarm is in the 'alarm' status
aws_access_key	no			AWS access key. If not set then the value of the AWS_ACCESS_KEY_ID, AWS_ACCESS_KEY, or EC2_ACCESS_KEY environment variable is used. aliases: ec2_access_key, access_key
aws_secret_key	no			AWS secret key. If not set then the value of the AWS_SECRET_ACCESS_KEY, AWS_SECRET_KEY, or EC2_SECRET_KEY environment variable is used. aliases: ec2_secret_key, secret_key
comparison	no			Determines how the threshold value is compared
description	no			A longer description of the alarm
dimensions	no			Describes to what the alarm is applied
ec2_url	no			URL to use to connect to EC2 or your Eucalyptus cloud (by default the module will use EC2 endpoints). Ignored for modules where region is required. Must be specified if region is not used. If not set then the value of the EC2_URL environment variable, if any, is used
evaluation_periods	no			The number of times in which the metric is evaluated before final calculation
insufficient_data_actions	no			A list of the names of action(s) to take when the alarm is in the 'insufficient_data' status
metric	no			Name of the monitored metric (for example: CPUUtilization). Metric must already exist
name	yes			Unique name for the alarm
namespace	no			Name of the appropriate namespace ('AWS/EC2', 'System/Linux', etc.), which determines the category it will appear under in cloudwatch
ok_actions	no			A list of the names of action(s) to take when the alarm is in the 'ok' status
period	no			The time (in seconds) between metric evaluations

parameter	required	default	choices	comments
profile (added in 1.6)	no			uses a boto profile. Only works with boto >= 2.24.0
region	no			The AWS region to use. If not specified then the value of the AWS_REGION or EC2_REGION environment variable, if any, is used. aliases: aws_region, ec2_region ¹⁰⁵
security_token (added in 1.6)	no			AWS STS security token. If not set then the value of the AWS_SECURITY_TOKEN or EC2_SECURITY_TOKEN environment variable is used. aliases: access_token
state	yes		present absent	Create or destroy the ELB
statistic	no			Operation applied to the metric. Works in conjunction with period and evaluation_periods to determine the comparison value
threshold	no			Sets the min/max bound for triggering the alarm
unit	no			The threshold's unit of measurement
validate_certs	no	yes	yes no	When set to "no", SSL certificates will not be validated for boto versions >= 2.6.0.

The Playbook

Go to the `wordpress_ha` directory:

```
$ cd /home/yan/ansible4aws/wordpress_ha
```

Create the variables for staging environment:

```
$ mkdir vars
$ vi vars/staging.yml

---
region: ap-southeast-2
zones: ['ap-southeast-2a', 'ap-southeast-2b']
asg_subnet_ids: ['subnet-yyyyyy', 'subnet-yyyyyy'] #
staging_subnet_private_2 and staging_subnet_private_3
elb_subnets: ['subnet-aaaaaa', 'subnet-bbbbbbb'] # staging_subnet_public_0
```

```
and staging_subnet_public_1
asg_name: wordpress
image_id: ami-xxxxxx # ami created from the master wordpress instance
instance_type: t2.micro
security_groups: ['sg-aaaaaa'] # staging_sg_wordpress
elb_group_ids: ['sg-bbbbbbb'] # staging_sg_wordpress_lb
keypair: wordpress-apsydne
asg_min: 1
asg_max: 8
```

Next, we'll create the complete playbook to launch ELB, ASG, and configure Auto Scaling:

```
$ vi provisioning_asg.yml

---
- hosts: localhost
  connection: local
  gather_facts: no
  tasks:
    - include_vars: "{{ env }}.yml"

    - set_fact:
        timestamp: "{{ lookup('pipe', 'date +%g%m%d%H%M%S') }}"
    - name: Create public ELB
      ec2_elb_lb:
        region: "{{ region }}"
        name: "{{ asg_name }}-{{ env }}"
        state: present
        cross_az_load_balancing: yes
        security_group_ids: "{{ elb_group_ids }}"
        subnets: "{{ elb_subnets }}"
        listeners:
          - protocol: http
            load_balancer_port: 80
            instance_port: 80
        health_check:
          ping_protocol: http
          ping_port: 80
          ping_path: "/index.php"
          response_timeout: 2
          interval: 10
          unhealthy_threshold: 2
          healthy_threshold: 2
          connection_draining_timeout: 60
      register: elb
      - debug: var=elb

    - name: Create Launch Configuration
```

```
ec2_lc:
    region: "{{ region }}"
    name: "{{ asg_name }}-{{ env }}-{{ timestamp }}"
    image_id: "{{ image_id }}"
    key_name: "{{ keypair }}"
    instance_type: "{{ instance_type }}"
    security_groups: "{{ security_groups }}"
    instance_monitoring: yes
    register: lc
- debug: var=lc

- name: Configure Auto Scaling Group
  ec2_asg:
    region: "{{ region }}"
    name: "{{ asg_name }}-{{ env }}-{{ timestamp }}"
    vpc_zone_identifier: "{{ asg_subnet_ids }}"
    launch_config_name: "{{ lc.name }}"
    availability_zones: "{{ zones }}"
    health_check_type: EC2
    health_check_period: 300
    desired_capacity: "{{ asg_min }}"
    min_size: "{{ asg_min }}"
    max_size: "{{ asg_max }}"
    tags:
      - Name: "{{ asg_name }}-{{ env }}"
    load_balancers: "{{ elb.elb.name }}"
    state: present
  register: asg
- debug: var=asg

- name: Configure Scaling Policies
  ec2_scaling_policy:
    region: "{{ region }}"
    name: "{{ item.name }}"
    asg_name: "{{ asg_name }}-{{ env }}-{{ timestamp }}"
    state: present
    adjustment_type: "{{ item.adjustment_type }}"
    min_adjustment_step: "{{ item.min_adjustment_step }}"
    scaling_adjustment: "{{ item.scaling_adjustment }}"
    cooldown: "{{ item.cooldown }}"
  with_items:
    - name: "Increase Group Size"
      adjustment_type: "ChangeInCapacity"
      scaling_adjustment: +1
      min_adjustment_step: 1
      cooldown: 180
    - name: "Decrease Group Size"
      adjustment_type: "ChangeInCapacity"
```

```
scaling_adjustment: -1
min_adjustment_step: 1
cooldown: 300
register: scaling_policy
- debug: var=scaling_policy

- name: Define Metric Alarms configuration
  set_fact:
    metric_alarms:
      - name: "{{ asg.name }}-ScaleUp"
        comparison: ">="
        threshold: 70.0
        alarm_actions:
          - "{{ scaling_policy.results[0].arn }}"
      - name: "{{ asg.name }}-ScaleDown"
        comparison: "<="
        threshold: 30.0
        alarm_actions:
          - "{{ scaling_policy.results[1].arn }}"

- name: Configure Metric Alarms
  ec2_metric_alarm:
    region: "{{ region }}"
    name: "{{ item.name }}"
    state: present
    metric: "CPUUtilization"
    namespace: "AWS/EC2"
    statistic: "Average"
    comparison: "{{ item.comparison }}"
    threshold: "{{ item.threshold }}"
    period: 60
    evaluation_periods: 5
    unit: "Percent"
    dimensions:
      AutoScalingGroupName: "{{ asg.name }}"
      alarm_actions: "{{ item.alarm_actions }}"
    with_items: "{{ metric_alarms }}"
    when: "{{ asg.max_size }} > 1"
    register: alarms
- debug: var=alarms
```

Run the playbook with extra variable `env=staging`:

```
$ ansible-playbook -i hosts provisioning_asg.yml -e "env=staging"
```

After the playbook runs we will have our complete multi-tier WordPress site with Auto Scaling based on EC2 metric (CPU Utilization).

14

Identity and Access Management (IAM)

AWS Identity and Access Management (IAM) is a web service that enables Amazon Web Services (AWS) customers to manage users and user permissions in AWS. The service is targeted at organizations with multiple users or systems in the cloud that use AWS products such as Amazon EC2, Amazon SimpleDB, and the AWS Management Console. With IAM, you can centrally manage users, security credentials such as access keys, and permissions that control which AWS resources users can access. For more information, visit <https://aws.amazon.com/documentation/iam/>.

IAM gives you the following features (for more information about the features of IAM, visit <http://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html#intro-features>):

- **Shared access to your AWS account:** You can grant other people permission to administer and use resources in your AWS account without having to share your password or access key.
- **Granular permissions:** You can grant different permissions to different people for different resources. For example, you might allow some users complete access to Amazon EC2, Amazon S3, Amazon DynamoDB, Amazon Redshift, and other AWS services. For other users, you can allow read-only access to just some S3 buckets, or permission to administer just some EC2 instances, or to access your billing information but nothing else.
- **Secure access to AWS resources for applications that run on Amazon EC2:** You can use IAM features to securely give applications that run on EC2 instances the credentials that they need in order to access other AWS resources, like S3 buckets and RDS or DynamoDB databases.

- **Multi-factor authentication (MFA):** You can add two-factor authentication to your account and to individual users for extra security. With MFA you or your users must provide not only a password or access key to work with your account, but also a code from a specially configured device.
- **Identity federation:** You can allow users who already have passwords elsewhere—for example, in your corporate network or with an Internet identity provider—to get temporary access to your AWS account.
- **Identity information for assurance:** If you use AWS CloudTrail, you receive log records that include information about those who made requests for resources in your account. That information is based on IAM identities.
- **PCI DSS Compliance:** IAM supports the processing, storage, and transmission of credit card data by a merchant or service provider, and has been validated as being compliant with **Payment Card Industry (PCI) Data Security Standard (DSS)**.
- **Integrated with many AWS services:** For a list of AWS services that work with IAM, visit http://docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-services-that-work-with-iam.html.
- **Free to use:** AWS Identity and Access Management is a feature of your AWS account offered at no additional charge. You will be charged only for use of other AWS products by your IAM users.

Accessing IAM

You can work with AWS Identity and Access Management in any of the following ways:

- **AWS Management Console:** The console is a browser-based interface to manage IAM and AWS resources.
- **AWS Command Line Tools:** You can use the AWS command line tools to issue commands at your system's command line to perform IAM and AWS tasks; this can be faster and more convenient than using the console. The command line tools are also useful if you want to build scripts that perform AWS tasks. AWS provides two sets of command line tools: the **AWS Command Line Interface (AWS CLI)** and the **AWS Tools for Windows PowerShell**.
- **AWS SDKs:** AWS provides **Software Development Kits (SDKs)** that consist of libraries and sample code for various programming languages and platforms (Java, Python, Ruby, .NET, iOS, Android, and so on). The SDKs provide a convenient way to create programmatic access to IAM and AWS. For example, the SDKs take care of tasks such as cryptographically signing requests, managing errors, and retrying requests automatically.

- **IAM HTTPS API:** You can access IAM and AWS programmatically by using the IAM HTTPS API, which lets you issue HTTPS requests directly to the service. When you use the HTTPS API, you must include code to digitally sign requests using your credentials.

Getting started with IAM

When you create an AWS account, you create an account (or *root*) identity, which you use to sign in to AWS. You can sign in to the AWS Management Console using this root identity, the email address and password that you provided when creating the account. This combination of your email address and password is also called your **root account credentials**.

When you use your root account credentials, you have complete, unrestricted access to all resources in your AWS account, including access to your billing information and the ability to change your password. This level of access is necessary when you first set up your account. However, it's recommended that you don't use root account credentials for everyday access, and do not share your root account credentials with anyone, because doing so gives them unrestricted access to your account. It is not possible to restrict the permissions that are granted to the root account. For more information on IAM, visit http://docs.aws.amazon.com/IAM/latest/UserGuide/introduction_identity-management.html.

IAM users

The *identity* aspect of AWS IAM helps you with the question *Who is that user?*, often referred to as authentication. Instead of sharing your root account credentials with others, you can create individual IAM users within your account that correspond to users in your organization. IAM users are not separate accounts; they are users within your account. Each user can have its own password for access to the AWS Management Console. You can also create an individual access key for each user so that the user can make programmatic requests to work with resources in your account. An IAM user doesn't have to represent an actual person; you can create an IAM user in order to generate an access key for an application that runs in your corporate network and needs AWS access.

IAM policy

By default, users can't access anything in your account. You grant permissions to a user by creating a policy, which is a document that lists the actions that a user can perform and the resources that the actions can affect. The following example shows a policy.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "s3:*",  
            "Resource": "*"  
        }  
    ]  
}
```

This policy grants permission to perform all S3 actions (`s3:*`). When you attach the policy to a user, that user then has those S3 permissions. Typically, users in your account have different policies attached to them, policies that represent permissions that the users need in order to work in your AWS account.

Any actions or resources that are not explicitly allowed are denied by default. For example, if this is the only policy attached to a user, the user is not allowed to perform any actions in Amazon EC2, Amazon DynamoDB, or in any other AWS product, because permissions to work with those products are not included in the policy.

IAM group

You can organize IAM users into IAM groups and attach a policy to a group. In that case, individual users still have their own credentials, but all the users in a group have the permissions that are attached to the group. Users or groups can have multiple policies attached to them that grant different permissions. In that case, the users' permissions are calculated based on the combination of policies. But the basic principle still applies—if the user has not been granted an explicit permission for an action and a resource, the user does not have those permissions.

IAM best practices

To help secure your AWS resources, follow these recommendations for the AWS IAM service. For more information on IAM best practices, visit <http://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html>.

- **Lock away your AWS account (root) access keys:** You use an access key (an access key ID and secret access key) to make programmatic requests to AWS. However, do not use your AWS account (root) access key. The access key for your AWS account gives full access to all your resources for all AWS services, including your billing information. You cannot restrict the permissions associated with your AWS account access key.
- **Create individual IAM users:** Don't use your AWS root account credentials to access AWS, and don't give your credentials to anyone else. Instead, create individual users for anyone who needs access to your AWS account. Create an IAM user for yourself as well, give that user administrative privileges, and use that IAM user for all your work. For more information see http://docs.aws.amazon.com/IAM/latest/UserGuide/getting-started_create-admin-group.html.
- **Use groups to assign permissions to IAM users:** Instead of defining permissions for individual IAM users, it's usually more convenient to create groups that relate to job functions (administrators, developers, accounting, and so on), define the relevant permissions for each group, and then assign IAM users to those groups. All the users in an IAM group inherit the permissions assigned to the group. That way, you can make changes for everyone in a group in just one place. As people move around in your company, you can simply change what IAM group their IAM user belongs to.
- **Grant least privilege:** When you create IAM policies, follow the standard security advice of granting least privilege—that is, granting only the permissions required to perform a task. Determine what users need to do and then craft policies for them that let the users perform only those tasks. It's more secure to start with a minimum set of permissions and grant additional permissions as necessary, rather than starting with permissions that are too lenient and then trying to tighten them later.

- **Configure a strong password policy for your users:** If you allow users to change their own passwords, required that they create strong passwords and that they rotate their passwords periodically. On the **Account Settings** page of the IAM console, you can create a password policy for your account. You can use the password policy to define password requirements, such as minimum length, whether it requires non-alphabetic characters, how frequently it must be rotated, and so on. For more information, see http://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_passwords_account-policy.html.
- **Enable MFA for privileged users:** For extra security, enable multi-factor authentication (MFA) for privileged IAM users (users who are allowed access to sensitive resources or APIs). With MFA, users have a device that generates a unique authentication code (a **one-time password** or **OTP**) and users must provide both their normal credentials (like their user name and password) and the OTP. The MFA device can either be a special piece of hardware, or it can be a virtual device (for example, it can run in an app on a smartphone). For more information, see http://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa.html.
- **Use roles for applications that run on Amazon EC2 instances:** Applications that run on an Amazon EC2 instance need credentials in order to access other AWS services. To provide credentials to the application in a secure way, use IAM roles. A role is an entity that has its own set of permissions, but that isn't a user or group. Roles also don't have their own permanent set of credentials the way IAM users do. In the case of Amazon EC2, IAM dynamically provides temporary credentials to the EC2 instance, and these credentials are automatically rotated for you.
- **Delegate by using roles instead of by sharing credentials:** You might need to allow users from another AWS account to access resources in your AWS account. If so, don't share security credentials, such as access keys, between accounts. Instead, use IAM roles. You can define a role that specifies what permissions the IAM users in the other account are allowed, and from which AWS accounts the IAM users are allowed to assume the role.

- **Rotate credentials regularly:** Change your own passwords and access keys regularly, and make sure that all IAM users in your account do as well. That way, if a password or access key is compromised without your knowledge, you limit how long the credentials can be used to access your resources. You can apply a password policy to your account to require all your IAM users to rotate their passwords, and you can choose how often they must do so.
- **Remove unnecessary credentials:** Remove IAM user credentials (that is, passwords and access keys) that are not needed. For example, an IAM user that is used for an application does not need a password (passwords are necessary only to sign in to AWS websites). Similarly, if a user does not and will never use access keys, there's no reason for the user to have them.
- **Use policy conditions for extra security:** To the extent that it's practical, define the conditions under which your IAM policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from, or you can specify that a request is allowed only within a specified date range or time range. You can also set conditions that require the use of SSL or MFA (multi-factor authentication). For example, you can require that a user has authenticated with an MFA device in order to be allowed to terminate an Amazon EC2 instance.
- **Monitor activity in your AWS account:** You can use logging features in AWS to determine the actions users have taken in your account and the resources that were used. The log files show the time and date of actions, the source IP for an action, which actions failed due to inadequate permissions, and more.

The iam module

The `iam` module is added in Ansible v2. This module can be used to manage IAM users, groups, roles and keys. Options for this module are listed in the following table:

parameter	required	default	choices	comments
<code>access_key_ids</code>	no			A list of the keys that you want impacted by the <code>access_key_state</code> parameter.
<code>access_key_state</code>	no		create remove active inactive	When type is user, it creates, removes, deactivates or activates a user's access key(s). Note that actions apply only to keys specified.
<code>aws_access_key</code>	no			AWS access key. If not set then the value of the <code>AWS_ACCESS_KEY_ID</code> , <code>AWS_ACCESS_KEY</code> , or <code>EC2_ACCESS_KEY</code> environment variable is used. aliases: <code>ec2_access_key</code> , <code>access_key</code>
<code>aws_secret_key</code>	no			AWS secret key. If not set then the value of the <code>AWS_SECRET_ACCESS_KEY</code> , <code>AWS_SECRET_KEY</code> , or <code>EC2_SECRET_KEY</code> environment variable is used. aliases: <code>ec2_secret_key</code> , <code>secret_key</code>
<code>ec2_url</code>	no			URL to use to connect to EC2 or your Eucalyptus cloud (by default the module will use EC2 endpoints). Ignored for modules where region is required. Must be specified if region is not used. If not set then the value of the <code>EC2_URL</code> environment variable, if any, is used
<code>groups</code>	no			A list of groups the user should belong to. When update, will gracefully remove groups not listed
<code>iam_type</code>	yes		user group role	Type of IAM resource
<code>key_count</code>	no	1		When <code>access_key_state</code> is create it will ensure this quantity of keys are present. Defaults to 1
<code>name</code>	yes			Name of IAM resource to create or identify
<code>new_name</code>	no			When state is update, will replace name with <code>new_name</code> on IAM resource
<code>new_path</code>	no			When state is update, will replace the path with <code>new_path</code> on the IAM resource
<code>password</code>	no			When type is user and state is present, define the users login password. Also works with

parameter	required	default	choices	comments
path	no	/		update. Note that always returns changed.
profile	no			When creating or updating, specify the desired path of the resource. If state is present, it will replace the current path to match what is passed in when they do not match.
region	no			Uses a boto profile. Requires boto >= 2.24.0
security_token	no			The AWS region to use. If not specified then the value of the AWS_REGION or EC2_REGION environment variable, if any, is used. aliases: aws_region, ec2_region ¹¹⁰
state	yes		present absent update	AWS STS security token. If not set then the value of the AWS_SECURITY_TOKEN or EC2_SECURITY_TOKEN environment variable is used. aliases: access_token
update_password	no	always	always on_create	Whether to create, delete or update the IAM resource. Note, roles cannot be updated.
validate_certs	no	yes	yes no	always will update passwords if they differ. on_create will only set the password for newly created users.
				When set to "no", SSL certificates will not be validated for boto versions >= 2.6.0.

For more information visit http://docs.aws.amazon.com/general/latest/gr/rande.html#ec2_region.

The iam_policy module

The `iam_policy` module is used to manage IAM policies for users, groups, and roles. Options for this module are listed in the following table:

parameter	required	default	choices	comments
<code>aws_access_key</code>	no			AWS access key. If not set then the value of the <code>AWS_ACCESS_KEY_ID</code> , <code>AWS_ACCESS_KEY</code> , or <code>EC2_ACCESS_KEY</code> environment variable is used. aliases: <code>ec2_access_key</code> , <code>access_key</code>
<code>aws_secret_key</code>	no			AWS secret key. If not set then the value of the <code>AWS_SECRET_ACCESS_KEY</code> , <code>AWS_SECRET_KEY</code> , or <code>EC2_SECRET_KEY</code> environment variable is used. aliases: <code>ec2_secret_key</code> , <code>secret_key</code>
<code>ec2_url</code>	no			URL to use to connect to EC2 or your Eucalyptus cloud (by default the module will use EC2 endpoints). Ignored for modules where region is required. Must be specified if region is not used. If not set then the value of the <code>EC2_URL</code> environment variable, if any, is used
<code>iam_name</code>	yes			Name of IAM resource you wish to target for policy actions. In other words, the user name, group name or role name.
<code>iam_type</code>	yes		user group role	Type of IAM resource
<code>policy_document</code>	no			The path to the properly json formatted policy file (mutually exclusive with <code>policy_json</code>)
<code>policy_json</code>	no			A properly json formatted policy as string (mutually exclusive with <code>policy_document</code>)
<code>policy_name</code>	yes			The name label for the policy to create or remove.
<code>profile</code>	no			Uses a boto profile. Requires boto >= 2.24.0
<code>region</code>	no			The AWS region to use. If not specified then the value of the <code>AWS_REGION</code> or <code>EC2_REGION</code> environment variable, if any, is used. aliases: <code>aws_region</code> , <code>ec2_region</code>
<code>security_token</code>	no			AWS STS security token. If not set then the value of the <code>AWS_SECURITY_TOKEN</code> or <code>EC2_SECURITY_TOKEN</code> environment variable is

parameter	required	default	choices	comments
				used. aliases: access_token
skip_duplicates	no			By default the module looks for any policies that match the document you pass in, if there is a match it will not make a new policy object with the same rules. You can override this by specifying false which would allow for two policy objects with different names but same rules.
state	yes		present absent	Whether to create or delete the IAM policy.
validate_certs	no	yes	yes no	When set to "no", SSL certificates will not be validated for boto versions >= 2.6.0.

Example playbooks

This example playbook will create IAM group called `admin`:

```
$ cd /home/yan/ansible4aws
$ vi iam_group.yml

---
- hosts: localhost
  gather_facts: no
  connection: local
  tasks:
    - name: create IAM group admin
      iam:
        iam_type: group
        name: admin
        state: present
```

Next, I will create user yan and add the user to the admin group:

```
$ vi iam_user.yml

---
- hosts: localhost
  gather_facts: no
  connection: local
  tasks:
    - name: Create user yan
      iam_user:
        name: yan
        password: abc123
        path: /
        groups:
          - admin
```

```
- name: create IAM user yan
  iam:
    iam_type: user
    name: yan
    state: present
    groups: admin
```

To create an IAM policy for the IAM group we need to create the policy document first.

For the `admin` group I will create a policy document that will allow access to all AWS resources.

```
$ vi iam_policy_admin.json

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

The elements used in the policy document:

- **Version:** The `Version` element specifies the policy language version. If you include the `Version` element, it must appear before the `Statement` element. The only allowed values are these:
 - `2012-10-17`: This is the current version of the policy language, and you should use this version number for all policies.
 - `2008-10-17`: This was an earlier version of the policy language. You might see this version on existing policies. Do not use this version for any new policies or any existing policies that you are updating. If you do not include a `Version` element, the value defaults to `2008-10-17`. However, it is a good practice to always include a `Version` element and set it to `2012-10-17`.
- **Statement:** The `Statement` element is the main element for a policy. This element is required. It can include multiple elements. The `Statement` element contains an array of individual statements. Each individual statement is a JSON block enclosed in braces `{ }`. For example, `"Statement": [{...}, {...}, {...}]`.

- **Effect:** The `Effect` element is required and specifies whether the statement will result in an allow or an explicit deny. Valid values for `Effect` are `Allow` and `Deny`.
By default, access to resources is denied. To allow access to a resource, you must set the `Effect` element to `Allow`. To override an allow (for example, to override an allow that is otherwise in force), you set the `Effect` element to `Deny`. For more information, see http://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_evaluation-logic.html.
- **Action:** The `Action` element describes the specific action or actions that will be allowed or denied. The `Statements` element must include either an `Action` or `NotAction` element. Each AWS service has its own set of actions that describe tasks that you can perform with that service. You can see the list here http://docs.amazonaws.cn/en_us/IAM/latest/UserGuide/reference_policies_actions_conditions.html, to determine which actions can be used as permissions in an IAM policy.
- **Resource:** The `Resource` element specifies the object or objects that the statement covers. The `Statements` must include either a `Resource` or a `NotResource` element. You specify a resource using an **Amazon Resource Names (ARN)**.

For more information about IAM policy elements, see http://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements.html.

To create a policy called `Administrator` and assign it to the `admin` group:

```
$ vi iam_policy.yml

---
- hosts: localhost
  gather_facts: no
  connection: local
  tasks:
    - name: Assign a policy called Administrator to the admin group
      iam_policy:
        iam_type: group
        iam_name: admin
        policy_name: Administrator
        state: present
        policy_document: iam_policy_admin.json
```

Run the playbook to create the policy and assign it to the `admin` group.

Open the IAM console <https://console.aws.amazon.com/iam/> to see the user, group, and policy created.

The screenshot shows the AWS IAM User Details page for a user named 'yan'. The left sidebar has a 'Users' section selected. The main area displays the user's ARN, password status, groups, path, and creation time. Below this, there are tabs for Groups, Permissions, Security Credentials, and Access Advisor. The 'Groups' tab is active, showing that the user belongs to one group, 'admin'. A blue button labeled 'Add User to Groups' is visible. A table below the groups section shows the group name and an 'Actions' column with a 'Remove from Group' link.

Group	Actions
admin	Remove from Group

IAM user page

To see the group permissions, select **Groups** in the navigation pane, click the group name, and select the **Permissions** tab. Under **Inline Policies** section you can see the policy attached to the group.

The screenshot shows the AWS IAM Groups page. On the left, a sidebar lists navigation options: Dashboard, Search IAM, Details, Groups (which is selected and highlighted in orange), Users, Roles, Policies, Identity Providers, Account Settings, Credential Report, and Encryption Keys. The main content area has a header 'Summary' with a back arrow. It displays the Group ARN (arn:aws:iam::...:group/admin), Users (1), Path (/), and Creation Time (2016-08-13 22:59 UTC+1000). Below this, there are three tabs: Users (selected), Permissions (highlighted in orange), and Access Advisor. Under the Permissions tab, there are two sections: 'Managed Policies' (which is empty) and 'Inline Policies'. The 'Inline Policies' section contains a 'Create Group Policy' button and a table with one row for 'Administrator'. The table has columns for 'Policy Name' (Administrator) and 'Actions' (with links to Show Policy, Edit Policy, Remove Policy, and Simulate Policy).

IAM group page

Instance profile

Applications that run on an EC2 instance must include AWS credentials in their AWS API requests. You could have your developers store AWS credentials directly within the EC2 instance and allow applications in that instance to use those credentials. But developers would then have to manage the credentials and ensure that they securely pass the credentials to each instance and update each EC2 instance when it's time to rotate the credentials. That's a lot of additional work.

Instead, you can and should use an IAM role to manage temporary credentials for applications that run on an EC2 instance. When you use a role, you don't have to distribute long-term credentials to an EC2 instance. Instead, the role supplies temporary permissions that applications can use when they make calls to other AWS resources. When you launch an EC2 instance, you specify an IAM role to associate with the instance. Applications that run on the instance can then use the role-supplied temporary credentials to sign API requests.

Using roles to grant permissions to applications that run on EC2 instances requires a bit of extra configuration. An application running on an EC2 instance is abstracted from AWS by the virtualized operating system. Because of this extra separation, an additional step is needed to assign an AWS role and its associated permissions to an EC2 instance and make them available to its applications. This extra step is the creation of an *instance profile* that is attached to the instance. The *instance profile* contains the role and can provide the role's credentials to an application that runs on the instance. Those credentials can then be used in the application's API calls to access resources and to limit access to only those resources that the role specifies. Note that only one role can be assigned to an EC2 instance at a time, and all applications on the instance share the same role and permissions.

Following examples will show you how to create an instance profile with S3 read only permission.

The policy document:

```
$ vi iam_policy_s3_read.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3>List*"
            ],
            "Resource": "*"
        }
    ]
}
```

Creating IAM role:

```
$ vi iam_role.yml
---
- hosts: localhost
  gather_facts: no
  connection: local
  tasks:
    - name: create IAM role app1
      iam:
        iam_type: role
        name: app1
        state: present
```

Attaching policy to the role:

```
$ vi iam_policy_app1.yml

---
- hosts: localhost
  gather_facts: no
  connection: local
  tasks:
    - name: Assign a policy called S3ReadOnly to the app1 role
      iam_policy:
        iam_type: role
        iam_name: app1
        policy_name: S3ReadOnly
        state: present
        policy_document: iam_policy_s3_read.json
```

To launch an EC2 instance with this instance profile attached, run the `ec2` module with `instance_profile_name` option set to `app1`:

```
$ vi ec2_profile.yml
---
- hosts: localhost
  gather_facts: no
  connection: local
  vars:
    #your region
    region: ap-southeast-2
  tasks:
    - name: EC2 provisioning with instance profile
      ec2:
        region: "{{ region }}"
        key_name: yan-key-pair-apsydney
        instance_type: t2.micro
        image: ami-dc361ebf
        group: sg_webserver_apsydney
        instance_profile_name: app1
```

A

Installing Ansible from Source

You can install Ansible from latest development version to take advantage of new features before it is released as a pip or linux package.

To install Ansible from latest source:

```
$ cd ~  
$ git clone git://github.com/ansible/ansible.git  
$ cd ./ansible  
$ source ./hacking/env-setup  
$ ansible --version  
  
ansible 2.0.0 (devel 07d0d2720c) last updated 2015/11/04 11:16:55 (GMT  
+1100)  
  lib/ansible/modules/core: (detached HEAD 5b64993b22) last updated  
2015/11/04 11:17:14 (GMT +1100)  
  lib/ansible/modules/extras: (detached HEAD f281eb2b30) last updated  
2015/11/04 11:17:30 (GMT +1100)  
    config file =  
      configured module search path = None
```

To use specific release version, find the release tag available here:

<https://github.com/ansible/ansible/releases>, for example to try version v2.0.0-0.4.beta2:

```
$ cd ansible  
$ git checkout tags/v2.0.0-0.4.beta2  
$ source ./hacking/env-setup  
$ ansible --version  
  
ansible 2.0.0 (detached HEAD de54f26376) last updated 2015/11/04 11:34:30  
(GMT +1100)  
  lib/ansible/modules/core: (detached HEAD 5b64993b22) last updated  
2015/11/04 11:17:14 (GMT +1100)
```

```
lib/ansible/modules/extras: (detached HEAD f281eb2b30) last updated  
2015/11/04 11:17:30 (GMT +1100)  
config file =  
configured module search path = None
```

B

What's New in Ansible v2

Ansible has grown rapidly over the last 3 years. Some early design decisions have had additional features bolted on without concern for the overall system architecture. It was getting to the point where it was difficult to fix bugs and/or add new features without things breaking in unexpected ways. Difficulty of testing, especially in terms of python unit tests, did not help. For more help, visit <http://www.slideshare.net/jimi-c/v2-and-beyond>.

In version 2, the development team tries to refactor portions of Ansible's core executor engine, in order to add following capabilities or features, while maintaining backward compatibility with version 1 (for more information, please visit

<https://groups.google.com/forum/#!topic/ansible-devel/VznP3Ioppfc>):

- **Block support:** Try/Except type logic that allows blocks of code to specify other code that can recover from errors, without failing the playbook run.
- Block provides a method for catching errors during task execution, as well as an option to always execute some set of tasks regardless of whether an exception occurred or not. It also allows for easier grouping of related tasks.
- Nested block, you can have a block inside a block inside a block...
- **Extended inheritance of blocks/roles:** Values like `become*` and others are now settable on blocks and roles, which are then inherited by all tasks contained within.
- **Optional parallel asynchronicity:** There's a current architectural choice in ansible playbooks that makes things easier to read and study where each task must complete before going onto the next, and each set of hosts in a play must get the same number of task *objects*. This would enable some new things: `include + with_items` back in full, roles having `with_items`, and those items also including *inventory scoped* variables. This may also pave the way for running multiple plays at the same time, though this is not an immediate goal.

- **Make way for future Python3 compatibility on the control node:** Requiring Python 3 will not be an immediate requirement for the control system, but something that eventually must be supported as more distros may ship python3 default over time.
- **Improved error detection:** Line numbers on all errors throughout the application, show what task failed and where, not just the name of the task.
- **Execution strategy plugin:** Allows changes in the way tasks are executed:
 - **linear:** traditional Ansible, which waits for all hosts to complete a task before continuing
 - **free:** allows each host to process tasks as fast as possible, without waiting for other hosts
 - And anything else people can conceive, just write anew plugin for it!

Ansible 2.0 includes nearly 200 new modules spanning the public, private, and hybrid cloud that accommodates for containers, networking, Windows-empowering users to automate more things than ever before.

New module addition highlights include:

- A completely new set of modules for managing OpenStack, the leading open source cloud computing framework, developed in concert with the OpenStack community
- 30 new modules for improving and expanding the support for Amazon Web Services
- Greatly expanded support for configuring and managing VMware environments
- Expanded support for managing Microsoft Windows environments
- Substantial improvements to the Docker module and new Docker connection plugin
- Improved support for network automation

Index

A

Access Control Information 194
Access Control List (ACL) 125
 reference 153
access keys
 creating 64, 65
acm
 reference 207
Action element, IAM policy element
 reference 316
Amazon AWS Documentation
 reference 42
Amazon EBS
 about 100
 reference 100
 volume types 100
Amazon EC2 Dashboard
 reference 28, 34
Amazon EC2 instance type
 reference 38
Amazon EC2 User Guide
 reference 42
Amazon Machine Image (AMI)
 about 232
 reference 232
Amazon Machine Images (AMI) 34, 44
Amazon RDS Read Replicas
 about 231
 reference 231
Amazon Resource Names (ARN) 316
Amazon VPC console
 reference 141
Amazon VPC Security Groups 231
Amazon Web Services (AWS)
 about 7
 core modules 66

functionalities 8
reference 8
using 7
AmazonProvidedDNS 218
Ansible Module Index
 reference 52
Ansible Modules 43
Ansible v2
 new features 323
Ansible
 about 43
 basic commands 49
 in VPC 156, 157, 158, 159
 installing 44
 installing, from source 321
 reference 43
 references 60
 s3cmd, using with 205
apigateway
 reference 207
application-autoscaling
 reference 207
associate-dhcp-options command
 reference 218
Auto Scaling group
 creating 251
 load balancing 259
 size, scaling of 255
 verifying 253
Auto Scaling Groups (ASG) 288
Auto Scaling
 about 238, 247
 availability zones 241
 key components 239
 launch configuration, creating 247
 reference 238
 regions 241

working 239, 240
autoscaling
 reference 207
Availability Zones 125, 171
AWS access key 64
AWS account
 setting up 15, 16, 18, 19, 20, 22
AWS CLI, in Ansible playbook
 about 215
 DHCP options set, creating 218
 DHCP options, associating with VPC 218
 idempotent EC2 instance, creating based on
 Name tag 219
 instance, starting with particular Name tag 220
 instance, stopping with particular Name tag 220
 VPC information, obtaining based on named tag
 216, 217
AWS CLI
 about 207
 configuration settings 211
 configuring 211
 credentials, configuring 211, 212
 installing 210
 precedence 211
 regions, configuring 213
 services 207
 used, for getting help 214
 using 213
AWS Command Line Tools 305
AWS Documentation, on Launching an Instance
 reference 34
aws ec2
 reference 216
AWS Free Usage Tier page
 reference 23
AWS Global Infrastructure 13
AWS Management Console 305
 about 23
 monthly billing, example 24, 26, 27
 reference 27
AWS region
 reference 213
AWS SDKs 305
AWS services, from Foundation Services category
 EC2 (Elastic Compute Cloud) 9

Elastic Load Balancing (ELB) 12
Relational Database Service (RDS) 11

Route 53 13
Simple Storage Service (S3) 12
Virtual Private Cloud (VPC) 10

AWS services, with IAM

 reference 305

AWS videos

 reference 42

B

backup storage 204, 205

basic commands, Ansible 49

best practices, Identity Access Management (IAM)
 reference 308

billing criteria, Relational Database Service (RDS)

 backup storage 172

 data transfer 173

 I/O requests per month 172

 instance class 172

 running time 172

 storage 172

boto

 about 61

 installing 63

 reference 61

bucket

 about 192, 193

 DNS naming conventions 193

 reference 192

C

changelog file

 reference 66

Classless Inter-Domain Routing (CIDR)

 about 126

 reference 126

Cloud Computing

 reference 8

cloud modules

 about 66

 reference 66

cloudformation

 reference 207

cloudfront

reference 207

cloudhsm

- reference 207

cloudsearch

- reference 207

cloudsearchdomain

- reference 207

cloudtrail

- reference 207

cloudwatch

- reference 207

codecommit

- reference 208

codepipeline

- reference 208

cognito-identity

- reference 208

cognito-sync

- reference 208

configservice

- reference 208

configuration management 87, 88

configure

- reference 208

Connect Client

- about 165

core modules, AWS

- cloudformation 66
- ec2 66
- ec2_ami_find 66
- ec2_asg 66
- ec2_elb 66
- ec2_elb_lb 66
- ec2_facts 66
- ec2_group 66
- ec2_key 66
- ec2_lc 66
- ec2_metric_alarm 66
- ec2_scaling_policy 66
- ec2_snapshot 66
- ec2_tag 66
- ec2_vol 66
- ec2_vpc 66
- ec2_vpc_net 66
- elasticache 66

elasticache_subnet_group 66

iam 66

iam_cert 67

iam_policy 67

rds 67

rds_param_group 67

rds_subnet_group 67

reference 67

route53 67

s3 67

create-dhcp-options

- reference 218

custom AMI

- creating 235
- deleting 236

custom route tables

- creating 135
- editing 136, 137

D

Data Security Standard (DSS) 305

datapipeline

- reference 208

DB instance classes

- reference 170

default VPC

- about 124
- reference 124

deploy

- reference 208

describe-instances command

- reference 219

devicefarm

- reference 208

dictionary 52

directconnect

- reference 208

discovery

- reference 208

dms

- reference 208

DNS record management

- about 120, 122

Domain Name System (DNS) 117

Domain Name Systems (DNS) 13

ds
 reference 208
dynamic inventor 93, 95, 96, 97
dynamic inventory 92
dynamodb
 reference 208
dynamodbstreams
 reference 208

E

EC2 external external inventory
 about 92
 Availability Zone 92
 Global 92
 Instance ID 92
 Region 92
 Security Group 92
 Tags 93
EC2 instance
 connecting to 38, 39
 creating 27
 key pair, creating 28, 29, 30
 launching 34, 35, 36, 37
 security groups, creating 31, 33
 terminating 40, 41, 42
ec2 module
 reference 76
EC2 Pricing Scheme
 reference 23
EC2 provisioning 76, 80, 82, 83
EC2-VPC provisioning 148, 149
ec2
 reference 208
ec2_ami module
 about 233
 reference 233
ec2_asg module
 about 294
 reference 288
ec2_eip module
 reference 85
ec2_elb_lb module
 about 288
 reference 288
ec2_group module
 reference 72
ec2_key module
 reference 67
ec2_lc module
 about 291
 reference 288
ec2_metric_alarm module
 about 299
 reference 288
ec2_scaling_policy module
 about 297
 reference 288
ec2_vpc_net_facts
 reference 168
ec2_vpc_subnet_facts
 reference 168
ecr
 reference 208
ecs
 reference 208
Effect element, IAM policy document
 reference 316
Elastic Compute Cloud (Amazon EC2) 117
Elastic Compute Cloud (EC2)
 about 9
 reference 9
Elastic IP address (EIP)
 about 84
 reference 84
Elastic Load Balancing (ELB)
 about 12
 reference 12
Elastic Load Balancing
 about 260
elasticache
 reference 208
elasticbeanstalk
 reference 208
elastictranscoder
 reference 208
elb
 reference 209
elements, IAM policy document
 Action 316
 Effect 316

R
Resource 316
Statement 315
Version 315
emr
reference 209
es
reference 209
events
reference 209
example playbook 314
Extra Packages for Enterprise Linux (EPEL) 44

F

firehose
reference 209

G

gamelift
reference 209
General Purpose (SSD) volumes 100
glacier
reference 209
Global Reach
reference 14
group variables 48

H

handlers 56
hash 52
host variables
about 47
reference 48
hosted zone
about 117
creating, in Route 53 118
reference 117

I

IAM console
reference 317
IAM group 307
IAM HTTPS API 306
iam module
about 311

options 311
reference 312
IAM policy 307
IAM policy elements
reference 316
IAM role
creating 103, 105
IAM users 306
reference 65, 308
iam
reference 209
iam_policy module
about 313
options 313
Identity Access Management (IAM)
about 304
accessing 305
best practices 308
features 304
reference 304, 306
starting with 306
Identity and Access Management (IAM) 13, 65
ifs
reference 208
importexport
reference 209
include statements 58
inspector
reference 209
instance profile
about 103, 318
reference 103
Internet gateway
creating 131
inventory file
about 46
group variables 48
host variables 47
IOPS (Input/Output Operations Per Second)
about 101
reference 101
iot
reference 209

J

Jinja2 templating language
reference 91

K

key components, Auto Scaling
Groups 239
Launch Configurations 239
Scaling Plans 239
key pair
about 67
deleting 68, 70
kinesis
reference 209
kms
reference 209

L

Linux
used, for connecting to EC2 instance 40
logs
reference 209

M

Mac
used, for connecting to EC2 instance 40
machinelearning
reference 209
magnetic volumes 101
main route table 134
MariaDB 169
marketplacecommerceanalytics
reference 209
meteringmarketplace
reference 209
Microsoft SQL Server 170
MindTerm 30
multi-AZ deployment 153, 154, 171
multi-AZ RDS provisioning 224, 226
multi-factor authentication (MFA) 305
multi-tier WordPress site
about 222
master WordPress instance, creating 226, 228,
229, 230

multi-AZ RDS provisioning 224, 226
MySQL 169
MySQL-compatible Amazon Aurora DB engine
170

N

named profile, AWS CLI command
reference 212
NAT AMI ID
obtaining 150
NAT instance
about 150
launching 151
reference 150
Network Address Translation (NAT) 127
Network Time Protocol (NTP) 88

O

object
about 194
key 194
metadata 194
naming conventions 195
reference 194
subresources 194
value 194
version ID 194
one-time password (OTP) 309
online calculator, Amazon
reference 23
OpenSSH
reference 40
OpenVPN Access Server
reference 165
OpenVPN server
about 161, 163
configuring 163, 165
reference 161
opsworks
reference 209
options 172
Oracle 170

P

Payment Card Industry (PCI) 305
playbook
 about 50
 creating 56, 57, 300, 301, 303
 hosts section 52, 53
 tasks section 55
 variables section 54
PostgreSQL 170
Provisioned IOPS (SSD) 101
Provisioned IOPS
 reference 170
PuTTY
 reference 30
 used, for connecting to Linux instance from Windows 30, 31
Python boto 61

R

rds module
 about 185
 options 186
rds
 reference 209
rds_param_group module
 about 182
 options 182
rds_subnet_group module
 about 180
 options 180
Red Hat Enterprise Linux (RHEL) 44
redshift
 reference 209
region 171
Regional Product Services
 reference 14
Relational Database Service (RDS)
 about 11, 169
 availability zones 171
 billing criteria 172
 DB instances 170
 DB option groups 172
 DB parameter groups 172
 DB Subnet group, creating 173, 174

reference 11, 169
regions 171
security groups 171
starting 173
release tag, Ansible
 reference 321
roles 58
root account credentials 306
Route 53
 about 13, 117
 hosted zone, creating in 118
 reference 13, 117
route table
 main route table 134
route tables
 basics 134
 reference 134
route53
 reference 209
route53domains
 reference 210

S

S3 access permission
 reference 194
S3 bucket
 creating 198
 deleting 203
S3 module
 about 195
 file, downloading 203
 file, sharing 202
 file, uploading 200
 options 195
 reference 195
 virtual directory 199
S3 Pricing Scheme
 reference 23
s3
 reference 210
s3api
 reference 210
s3cmd
 reference 205, 206
 using, with Ansible 205

sdb
 reference 210
security groups update 245
security groups
 about 71
 creating 74
 deleting 73
servicecatalog
 reference 210
ses
 reference 210
Simple Storage Service (S3)
 about 12, 117, 129
 reference 12, 192
size
 scaling, of Auto Scaling group 255
sns
 reference 210
source
 Ansible, installing from 321
sqsh
 reference 210
SSH keys 45
ssm
 reference 210
Statement element, IAM policy document 315
storagegateway
 reference 210
sts
 reference 210
subnet CIDR blocks
 reference 127
subnet ID
 obtaining 166
subnet sizing
 reference 127
subnets
 creating, on VPC 132
support
 reference 210
swf
 reference 210

T
template formatting
 reference 91
template module, Ansible
 reference 89
V
valid object key names
 examples 195
Version Control System (VCS) 124
Version element, IAM policy document 315
Virtual Private Cloud (VPC)
 about 124, 126
 creating 129, 130
 obtaining 166
Virtual Private Cloud
 about 10
 reference 10
Virtual Private Network (VPN) 157
volume types, Amazon EBS
 General Purpose (SSD) 100
 magnetic volumes 101
 Provisioned IOPS (SSD) 101
 reference 101
VPC CIDR block
 reference 127
VPC provisioning 137, 139
VPC security groups
 about 142
 characteristics 143
 managing 143, 144, 147
 reference 142
W
waf
 reference 210
WordPress site
 directory structure 115
 directory, creating for roles 109
 mysql roles 110
 playbook, provisioning 106
 playbook, running 108
 site.yml 114
 site.yml playbook, running 116

tasks, for common roles 110
tasks, for web roles 110
variables file, creating 109
wordpress roles 111
workspaces
 reference 210

Y

YAML
 about 43
 reference 43



•
•
•
•
•

- (A)
(B)
(C)
(D)
(E)

*



*

*

*

*



*

*

*

*

*

*
*



- -

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

2.

- (A)
- (B)
- (C)
- (D)
- (E)

3.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

4.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

5.

- (A)
- (B)
- (C)
- (D)
- (E)

6.

- (A)
- (B)
- (C)
- (D)
- (E)

7.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

8.

- (A)
- (B)

-year-

9.

- (A)
- (B)
- (C)
- (D)
- (E)

10.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

11.

-
- (A)
 - (B)
 - (C)
 - (D)
 - (E)
 - (F)

12.

- (A)
- (B)
- (C)
- (D)
- (E)

13.

- (A)
- (B)
- (C)
- (D)
- (E)

14.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-
15.

- (A)
- (B)
- (C)
- (D)
- (E)

16.

- (A)
- (B)
- (C)
- (D)
- (E)

17.

- (A)
- (B)
- (C)
- (D)

18.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

19.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

20.

- (A)
- (B)
- (C)
- (D)
- (E)

21.

- (A)
- (B)
- (C)
- (D)
- (E)

22.

- (A)
- (B)
- (C)

-year-
(D)
(E)

23.

- (A)
(B)
(C)
(D)
(E)
-

24.

- (A)
(B)
(C)
(D)
(E)

25.

- (A)
(B)
(C)
(D)
(E)

26.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

27.

- (A)
- (B)
- (C)
- (D)
- (E)

28.

- (A)
- (B)
- (C)
- (D)
- (E)

29.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

30.

- (A)
- (B)
- (C)
- (D)
- (E)

31.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

32.

(A)

- (A)
- (B)
- (C)
- (D)
- (E)

33.

- (A)
- (B)
- (C)
- (D)
- (E)

34.

- (A)
- (B)
- (C)
- (D)
- (E)

35.

- (A)
- (B)
- (C)
- (D)

36.

- (A)
- (B)
- (C)

37.

- (A)
- (B)
- (C)
- (D)
- (E)

38.

- (A)
- (B)
- (C)
- (D)
- (E)

39.

- (A)
- (B)
- (C)
- (D)
- (E)

40.

- (A)
- (B)
- (C)
- (D)
- (E)

41.

- (C)
- (D)
- (E)

42.

- (A)
- (B)
- (C)
- (D)

43.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

44.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

45.

- (A)
- (B)
- (C)
- (D)
- (E)

46.

- (A)
- (B)
- (C)
- (D)
- (E)

47.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-
48.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

49.

-
- (A)
 - (B)
 - (C)
 - (D)
 - (E)
 - (F)

50.

- (A)
- (B)
- (C)
- (D)
- (E)

51.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

52.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)
- (G)

53.

- (A)
- (B)
- (C)
- (D)
- (E)

54.

- (A)
- (B)
- (C)
- (D)

55.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

56.

- (A)
- (B)
- (C)
- (D)
- (E)

57.

- (A)
- (B)
- (C)
- (D)
- (E)

58.

- (A)
- (B)
- (C)
- (D)
- (E)

59.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

60.

- (A)
- (B)
- (C)
- (D)
- (E)

61.

- (A)
- (B)
- (C)
- (D)
- (E)

62.

- (A)
- (B)
- (C)
- (D)

63.

- (A)
- (B)
- (C)
- (D)
- (E)

64.

-year-

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

65.

- (A)
- (B)
- (C)
- (D)
- (E)

66.

- (A)
- (B)
- (C)
- (D)
- (E)

67.

- (A)
- (B)
- (C)
- (D)
- (E)

68.

- (A)
- (B)
- (C)
- (D)
- (E)

69.

- (A)
- (B)
- (C)
- (D)
- (E)

70.

-
- (A)
 - (B)
 - (C)
 - (D)

71.

- (A)
- (B)
- (C)
- (D)

72.

(A)

- (A)
- (B)
- (C)
- (D)
- (E)

73.

- (A)
- (B)
- (C)
- (D)
- (E)

74.

- (A)
- (B)
- (C)
- (D)
- (E)

75.

- (A)
- (B)
- (C)
- (D)
- (E)

76.

- (A)
- (B)
- (C)
- (D)
- (E)

77.

-
- (A)
 - (B)
 - (C)
 - (D)
 - (E)

78.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

79.

- (A)
- (B)
- (C)
- (D)
- (E)

80.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)
- (G)
- (H)

81.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

82.

- (A)
- (B)
- (C)
- (D)
- (E)

83.

- (A)
- (B)
- (C)
- (D)
- (E)

84.

- (A)
- (B)
- (C)
- (D)
- (E)

(F)

85.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
 - (F)
-

86.

- (A)
- (B)
- (C)
- (D)
- (E)

87.

- (A)
- (B)
- (C)
- (D)
- (E)

88.

- (A)
- (B)
- (C)
- (D)
- (E)

89.

90.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

91.

- (A)
- (B)
- (C)
- (D)
- (E)

92.

- (A)
- (B)
- (C)
- (D)
- (E)

93.

- (A)
- (B)
- (C)
- (D)

94.

- (A)
- (B)
- (C)
- (D)
- (E)

95.

- (A)
- (B)
- (C)
- (D)

96.

- (A)
- (B)
- (C)
- (D)

(E)

97.

(A)
(B)
(C)
(D)

98.

(A)
(B)
(C)
(D)
(E)

99.

(A)
(B)
(C)
(D)

(E)

100.

(A)

(B)

101.

(A)

(B)

(C)

(D)

102.

(A)

(B)

(C)

(D)

(E)

103.

(A)

(B)

(C)

104.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

105.

- (A)
- (B)
- (C)
- (D)
- (E)

106.

- (A)
- (B)
- (C)
- (D)
- (E)

107.

- (A)
- (B)
- (C)
- (D)

(E)

108.

- (A)
- (B)
- (C)
- (D)

109.

- (A)
- (B)
- (C)
- (D)
- (E)

110.

- (A)
- (B)
- (C)
- (D)
- (E)

111.

- (A)
- (B)
- (C)
- (D)
- (E)

112.

- (A)
- (B)
- (C)
- (D)
- (E)

113.

- (A)
 - (B)
 - (C)
 - (D)
-

114.

- (A)
- (B)
- (C)
- (D)
- (E)

115.

- (A)

- (B)
- (C)
- (D)
- (E)

116.

- (A)
- (B)
- (C)
- (D)
- (E)

117.

- (A)
- (B)
- (C)
- (D)
- (E)

49.

50.

87.

88.

89.

90.

1.
2.
4.
5.
6.
7.
8.
9.
10.

41.
42.
43.
46.
47.
48.
49.
50.

81.
82.
83.
84.
85.
86.
87.
88.
89.
90.



•
•
•
•
•

- (A)
(B)
(C)
(D)
(E)

*



*

*

*

*



*

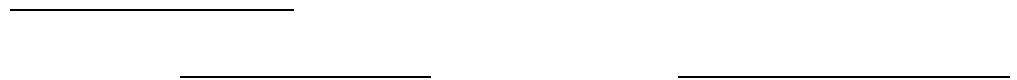
*

*

*

*

*
*



- -

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

2.

- (A)
- (B)
- (C)
- (D)
- (E)

3.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

4.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

5.

- (A)
- (B)
- (C)
- (D)
- (E)

6.

- (A)
- (B)
- (C)
- (D)
- (E)

7.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

8.

- (A)
- (B)

-year-

9.

- (A)
- (B)
- (C)
- (D)
- (E)

10.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

11.

-
- (A)
 - (B)
 - (C)
 - (D)
 - (E)
 - (F)

12.

- (A)
- (B)
- (C)
- (D)
- (E)

13.

- (A)
- (B)
- (C)
- (D)
- (E)

14.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-
15.

- (A)
- (B)
- (C)
- (D)
- (E)

16.

- (A)
- (B)
- (C)
- (D)
- (E)

17.

- (A)
- (B)
- (C)
- (D)

18.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

19.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

20.

- (A)
- (B)
- (C)
- (D)
- (E)

21.

- (A)
- (B)
- (C)
- (D)
- (E)

22.

- (A)
- (B)
- (C)

-year-
(D)
(E)

23.

- (A)
(B)
(C)
(D)
(E)
-

24.

- (A)
(B)
(C)
(D)
(E)

25.

- (A)
(B)
(C)
(D)
(E)

26.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

27.

- (A)
- (B)
- (C)
- (D)
- (E)

28.

- (A)
- (B)
- (C)
- (D)
- (E)

29.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

30.

- (A)
- (B)
- (C)
- (D)
- (E)

31.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

32.

(A)

- (A)
- (B)
- (C)
- (D)
- (E)

33.

- (A)
- (B)
- (C)
- (D)
- (E)

34.

- (A)
- (B)
- (C)
- (D)
- (E)

35.

- (A)
- (B)
- (C)
- (D)

36.

- (A)
- (B)
- (C)

37.

- (A)
- (B)
- (C)
- (D)
- (E)

38.

- (A)
- (B)
- (C)
- (D)
- (E)

39.

- (A)
- (B)
- (C)
- (D)
- (E)

40.

- (A)
- (B)
- (C)
- (D)
- (E)

41.

- (C)
- (D)
- (E)

42.

- (A)
- (B)
- (C)
- (D)

43.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

44.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

45.

- (A)
- (B)
- (C)
- (D)
- (E)

46.

- (A)
- (B)
- (C)
- (D)
- (E)

47.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-
48.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

49.

-
- (A)
 - (B)
 - (C)
 - (D)
 - (E)
 - (F)

50.

- (A)
- (B)
- (C)
- (D)
- (E)

51.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

52.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)
- (G)

53.

- (A)
- (B)
- (C)
- (D)
- (E)

54.

- (A)
- (B)
- (C)
- (D)

55.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

56.

- (A)
- (B)
- (C)
- (D)
- (E)

57.

- (A)
- (B)
- (C)
- (D)
- (E)

58.

- (A)
- (B)
- (C)
- (D)
- (E)

59.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

60.

- (A)
- (B)
- (C)
- (D)
- (E)

61.

- (A)
- (B)
- (C)
- (D)
- (E)

62.

- (A)
- (B)
- (C)
- (D)

63.

- (A)
- (B)
- (C)
- (D)
- (E)

64.

-year-

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

65.

- (A)
- (B)
- (C)
- (D)
- (E)

66.

- (A)
- (B)
- (C)
- (D)
- (E)

67.

- (A)
- (B)
- (C)
- (D)
- (E)

68.

- (A)
- (B)
- (C)
- (D)
- (E)

69.

- (A)
- (B)
- (C)
- (D)
- (E)

70.

-
- (A)
 - (B)
 - (C)
 - (D)

71.

- (A)
- (B)
- (C)
- (D)

72.

(A)

- (A)
- (B)
- (C)
- (D)
- (E)

73.

- (A)
- (B)
- (C)
- (D)
- (E)

74.

- (A)
- (B)
- (C)
- (D)
- (E)

75.

- (A)
- (B)
- (C)
- (D)
- (E)

76.

- (A)
- (B)
- (C)
- (D)
- (E)

77.

-
- (A)
 - (B)
 - (C)
 - (D)
 - (E)

78.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

79.

- (A)
- (B)
- (C)
- (D)
- (E)

80.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)
- (G)
- (H)

81.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

82.

- (A)
- (B)
- (C)
- (D)
- (E)

83.

- (A)
- (B)
- (C)
- (D)
- (E)

84.

- (A)
- (B)
- (C)
- (D)
- (E)

(F)

85.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
 - (F)
-

86.

- (A)
- (B)
- (C)
- (D)
- (E)

87.

- (A)
- (B)
- (C)
- (D)
- (E)

88.

- (A)
- (B)
- (C)
- (D)
- (E)

89.

90.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

91.

- (A)
- (B)
- (C)
- (D)
- (E)

92.

- (A)
- (B)
- (C)
- (D)
- (E)

93.

- (A)
- (B)
- (C)
- (D)

94.

- (A)
- (B)
- (C)
- (D)
- (E)

95.

- (A)
- (B)
- (C)
- (D)

96.

- (A)
- (B)
- (C)
- (D)

(E)

97.

(A)
(B)
(C)
(D)

98.

(A)
(B)
(C)
(D)
(E)

99.

(A)
(B)
(C)
(D)

(E)

100.

(A)

(B)

101.

(A)

(B)

(C)

(D)

102.

(A)

(B)

(C)

(D)

(E)

103.

(A)

(B)

(C)

104.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

105.

- (A)
- (B)
- (C)
- (D)
- (E)

106.

- (A)
- (B)
- (C)
- (D)
- (E)

107.

- (A)
- (B)
- (C)
- (D)

(E)

108.

- (A)
- (B)
- (C)
- (D)

109.

- (A)
- (B)
- (C)
- (D)
- (E)

110.

- (A)
- (B)
- (C)
- (D)
- (E)

111.

- (A)
- (B)
- (C)
- (D)
- (E)

112.

- (A)
- (B)
- (C)
- (D)
- (E)

113.

- (A)
 - (B)
 - (C)
 - (D)
-

114.

- (A)
- (B)
- (C)
- (D)
- (E)

115.

- (A)

- (B)
- (C)
- (D)
- (E)

116.

- (A)
- (B)
- (C)
- (D)
- (E)

117.

- (A)
- (B)
- (C)
- (D)
- (E)

49.

50.

87.

88.

89.

90.

1.
2.
4.
5.
6.
7.
8.
9.
10.

41.
42.
43.
46.
47.
48.
49.
50.

81.
82.
83.
84.
85.
86.
87.
88.
89.
90.



•
•
•
•
•

- (A)
(B)
(C)
(D)
(E)

*



*

*

*

*



*

*

*

*

*

*

*



- -

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

2.

- (A)
- (B)
- (C)
- (D)
- (E)

3.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

4.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

5.

- (A)
- (B)
- (C)
- (D)
- (E)

6.

- (A)
- (B)
- (C)
- (D)
- (E)

7.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

8.

- (A)
- (B)

-year-

9.

- (A)
- (B)
- (C)
- (D)
- (E)

10.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

11.

-
- (A)
 - (B)
 - (C)
 - (D)
 - (E)
 - (F)

12.

- (A)
- (B)
- (C)
- (D)
- (E)

13.

- (A)
- (B)
- (C)
- (D)
- (E)

14.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-
15.

- (A)
- (B)
- (C)
- (D)
- (E)

16.

- (A)
- (B)
- (C)
- (D)
- (E)

17.

- (A)
- (B)
- (C)
- (D)

18.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

19.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

20.

- (A)
- (B)
- (C)
- (D)
- (E)

21.

- (A)
- (B)
- (C)
- (D)
- (E)

22.

- (A)
- (B)
- (C)

-year-
(D)
(E)

23.

- (A)
(B)
(C)
(D)
(E)
-

24.

- (A)
(B)
(C)
(D)
(E)

25.

- (A)
(B)
(C)
(D)
(E)

26.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

27.

- (A)
- (B)
- (C)
- (D)
- (E)

28.

- (A)
- (B)
- (C)
- (D)
- (E)

29.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

30.

- (A)
- (B)
- (C)
- (D)
- (E)

31.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

32.

(A)

- (A)
- (B)
- (C)
- (D)
- (E)

33.

- (A)
- (B)
- (C)
- (D)
- (E)

34.

- (A)
- (B)
- (C)
- (D)
- (E)

35.

- (A)
- (B)
- (C)
- (D)

36.

- (A)
- (B)
- (C)

37.

- (A)
- (B)
- (C)
- (D)
- (E)

38.

- (A)
- (B)
- (C)
- (D)
- (E)

39.

- (A)
- (B)
- (C)
- (D)
- (E)

40.

- (A)
- (B)
- (C)
- (D)
- (E)

41.

- (C)
- (D)
- (E)

42.

- (A)
- (B)
- (C)
- (D)

43.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

44.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

45.

- (A)
- (B)
- (C)
- (D)
- (E)

46.

- (A)
- (B)
- (C)
- (D)
- (E)

47.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-
48.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

49.

-
- (A)
 - (B)
 - (C)
 - (D)
 - (E)
 - (F)

50.

- (A)
- (B)
- (C)
- (D)
- (E)

51.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

52.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)
- (G)

53.

- (A)
- (B)
- (C)
- (D)
- (E)

54.

- (A)
- (B)
- (C)
- (D)

55.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

56.

- (A)
- (B)
- (C)
- (D)
- (E)

57.

- (A)
- (B)
- (C)
- (D)
- (E)

58.

- (A)
- (B)
- (C)
- (D)
- (E)

59.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

60.

- (A)
- (B)
- (C)
- (D)
- (E)

61.

- (A)
- (B)
- (C)
- (D)
- (E)

62.

- (A)
- (B)
- (C)
- (D)

63.

- (A)
- (B)
- (C)
- (D)
- (E)

64.

-year-

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

65.

- (A)
- (B)
- (C)
- (D)
- (E)

66.

- (A)
- (B)
- (C)
- (D)
- (E)

67.

- (A)
- (B)
- (C)
- (D)
- (E)

68.

- (A)
- (B)
- (C)
- (D)
- (E)

69.

- (A)
- (B)
- (C)
- (D)
- (E)

70.

-
- (A)
 - (B)
 - (C)
 - (D)

71.

- (A)
- (B)
- (C)
- (D)

72.

(A)

- (A)
- (B)
- (C)
- (D)
- (E)

73.

- (A)
- (B)
- (C)
- (D)
- (E)

74.

- (A)
- (B)
- (C)
- (D)
- (E)

75.

- (A)
- (B)
- (C)
- (D)
- (E)

76.

- (A)
- (B)
- (C)
- (D)
- (E)

77.

-
- (A)
 - (B)
 - (C)
 - (D)
 - (E)

78.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

79.

- (A)
- (B)
- (C)
- (D)
- (E)

80.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)
- (G)
- (H)

81.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

82.

- (A)
- (B)
- (C)
- (D)
- (E)

83.

- (A)
- (B)
- (C)
- (D)
- (E)

84.

- (A)
- (B)
- (C)
- (D)
- (E)

(F)

85.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
 - (F)
-

86.

- (A)
- (B)
- (C)
- (D)
- (E)

87.

- (A)
- (B)
- (C)
- (D)
- (E)

88.

- (A)
- (B)
- (C)
- (D)
- (E)

89.

90.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

91.

- (A)
- (B)
- (C)
- (D)
- (E)

92.

- (A)
- (B)
- (C)
- (D)
- (E)

93.

- (A)
- (B)
- (C)
- (D)

94.

- (A)
- (B)
- (C)
- (D)
- (E)

95.

- (A)
- (B)
- (C)
- (D)

96.

- (A)
- (B)
- (C)
- (D)

(E)

97.

(A)
(B)
(C)
(D)

98.

(A)
(B)
(C)
(D)
(E)

99.

(A)
(B)
(C)
(D)

(E)

100.

(A)

(B)

101.

(A)

(B)

(C)

(D)

102.

(A)

(B)

(C)

(D)

(E)

103.

(A)

(B)

(C)

104.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
 - (F)
-

105.

- (A)
- (B)
- (C)
- (D)
- (E)

106.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

107.

- (A)
- (B)
- (C)
- (D)

(E)

108.

- (A)
- (B)
- (C)
- (D)

109.

- (A)
- (B)
- (C)
- (D)
- (E)

110.

- (A)
- (B)
- (C)
- (D)
- (E)

111.

- (A)
- (B)
- (C)
- (D)
- (E)

112.

- (A)
- (B)
- (C)
- (D)
- (E)

113.

- (A)
 - (B)
 - (C)
 - (D)
-

114.

- (A)
- (B)
- (C)
- (D)
- (E)

115.

- (A)

- (B)
- (C)
- (D)
- (E)

116.

- (A)
- (B)
- (C)
- (D)
- (E)

117.

- (A)
- (B)
- (C)
- (D)
- (E)

49.

50.

87.

88.

89.

90.

1.
2.
4.
5.
6.
7.
8.
9.
10.

41.
42.
43.
46.
47.
48.
49.
50.

81.
82.
83.
84.
85.
86.
87.
88.
89.
90.



•
•
•
•
•

- (A)
(B)
(C)
(D)
(E)

*



*

*

*

*



*

*

*

*

*

*
*



- -

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

2.

- (A)
- (B)
- (C)
- (D)
- (E)

3.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

4.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

5.

- (A)
- (B)
- (C)
- (D)
- (E)

6.

- (A)
- (B)
- (C)
- (D)
- (E)

7.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

8.

- (A)
- (B)

-year-

9.

- (A)
- (B)
- (C)
- (D)
- (E)

10.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

11.

-
- (A)
 - (B)
 - (C)
 - (D)
 - (E)
 - (F)

12.

- (A)
- (B)
- (C)
- (D)
- (E)

13.

- (A)
- (B)
- (C)
- (D)
- (E)

14.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-
15.

- (A)
- (B)
- (C)
- (D)
- (E)

16.

- (A)
- (B)
- (C)
- (D)
- (E)

17.

- (A)
- (B)
- (C)
- (D)

18.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

19.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

20.

- (A)
- (B)
- (C)
- (D)
- (E)

21.

- (A)
- (B)
- (C)
- (D)
- (E)

22.

- (A)
- (B)
- (C)

-year-
(D)
(E)

23.

- (A)
(B)
(C)
(D)
(E)
-

24.

- (A)
(B)
(C)
(D)
(E)

25.

- (A)
(B)
(C)
(D)
(E)

26.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

27.

- (A)
- (B)
- (C)
- (D)
- (E)

28.

- (A)
- (B)
- (C)
- (D)
- (E)

29.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

30.

- (A)
- (B)
- (C)
- (D)
- (E)

31.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

32.

(A)

- (A)
- (B)
- (C)
- (D)
- (E)

33.

- (A)
- (B)
- (C)
- (D)
- (E)

34.

- (A)
- (B)
- (C)
- (D)
- (E)

35.

- (A)
- (B)
- (C)
- (D)

36.

- (A)
- (B)
- (C)

37.

- (A)
- (B)
- (C)
- (D)
- (E)

38.

- (A)
- (B)
- (C)
- (D)
- (E)

39.

- (A)
- (B)
- (C)
- (D)
- (E)

40.

- (A)
- (B)
- (C)
- (D)
- (E)

41.

- (C)
- (D)
- (E)

42.

- (A)
- (B)
- (C)
- (D)

43.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

44.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

45.

- (A)
- (B)
- (C)
- (D)
- (E)

46.

- (A)
- (B)
- (C)
- (D)
- (E)

47.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-
48.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

49.

-
- (A)
 - (B)
 - (C)
 - (D)
 - (E)
 - (F)

50.

- (A)
- (B)
- (C)
- (D)
- (E)

51.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

52.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)
- (G)

53.

- (A)
- (B)
- (C)
- (D)
- (E)

54.

- (A)
- (B)
- (C)
- (D)

55.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

56.

- (A)
- (B)
- (C)
- (D)
- (E)

57.

- (A)
- (B)
- (C)
- (D)
- (E)

58.

- (A)
- (B)
- (C)
- (D)
- (E)

59.

- (A)
- (B)
- (C)
- (D)
- (E)

-year-

60.

- (A)
- (B)
- (C)
- (D)
- (E)

61.

- (A)
- (B)
- (C)
- (D)
- (E)

62.

- (A)
- (B)
- (C)
- (D)

63.

- (A)
- (B)
- (C)
- (D)
- (E)

64.

-year-

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

65.

- (A)
- (B)
- (C)
- (D)
- (E)

66.

- (A)
- (B)
- (C)
- (D)
- (E)

67.

- (A)
- (B)
- (C)
- (D)
- (E)

68.

- (A)
- (B)
- (C)
- (D)
- (E)

69.

- (A)
- (B)
- (C)
- (D)
- (E)

70.

-
- (A)
 - (B)
 - (C)
 - (D)

71.

- (A)
- (B)
- (C)
- (D)

72.

(A)

- (A)
- (B)
- (C)
- (D)
- (E)

73.

- (A)
- (B)
- (C)
- (D)
- (E)

74.

- (A)
- (B)
- (C)
- (D)
- (E)

75.

- (A)
- (B)
- (C)
- (D)
- (E)

76.

- (A)
- (B)
- (C)
- (D)
- (E)

77.

-
- (A)
 - (B)
 - (C)
 - (D)
 - (E)

78.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

79.

- (A)
- (B)
- (C)
- (D)
- (E)

80.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)
- (G)
- (H)

81.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

82.

- (A)
- (B)
- (C)
- (D)
- (E)

83.

- (A)
- (B)
- (C)
- (D)
- (E)

84.

- (A)
- (B)
- (C)
- (D)
- (E)

(F)

85.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
 - (F)
-

86.

- (A)
- (B)
- (C)
- (D)
- (E)

87.

- (A)
- (B)
- (C)
- (D)
- (E)

88.

- (A)
- (B)
- (C)
- (D)
- (E)

89.

90.

- (A)
- (B)
- (C)
- (D)
- (E)
- (F)

91.

- (A)
- (B)
- (C)
- (D)
- (E)

92.

- (A)
- (B)
- (C)
- (D)
- (E)

93.

- (A)
- (B)
- (C)
- (D)

94.

- (A)
- (B)
- (C)
- (D)
- (E)

95.

- (A)
- (B)
- (C)
- (D)

96.

- (A)
- (B)
- (C)
- (D)

(E)

97.

(A)
(B)
(C)
(D)

98.

(A)
(B)
(C)
(D)
(E)

99.

(A)
(B)
(C)
(D)

(E)

100.

(A)

(B)

101.

(A)

(B)

(C)

(D)

102.

(A)

(B)

(C)

(D)

(E)

103.

(A)

(B)

(C)

104.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
 - (F)
-

105.

- (A)
- (B)
- (C)
- (D)
- (E)

106.

- (A)
 - (B)
 - (C)
 - (D)
 - (E)
-

107.

- (A)
- (B)
- (C)
- (D)

(E)

108.

- (A)
- (B)
- (C)
- (D)

109.

- (A)
- (B)
- (C)
- (D)
- (E)

110.

- (A)
- (B)
- (C)
- (D)
- (E)

111.

- (A)
- (B)
- (C)
- (D)
- (E)

112.

- (A)
- (B)
- (C)
- (D)
- (E)

113.

- (A)
 - (B)
 - (C)
 - (D)
-

114.

- (A)
- (B)
- (C)
- (D)
- (E)

115.

- (A)

- (B)
- (C)
- (D)
- (E)

116.

- (A)
- (B)
- (C)
- (D)
- (E)

117.

- (A)
- (B)
- (C)
- (D)
- (E)

49.

50.

87.

88.

89.

90.

1.
2.
4.
5.
6.
7.
8.
9.
10.

41.
42.
43.
46.
47.
48.
49.
50.

81.
82.
83.
84.
85.
86.
87.
88.
89.
90.