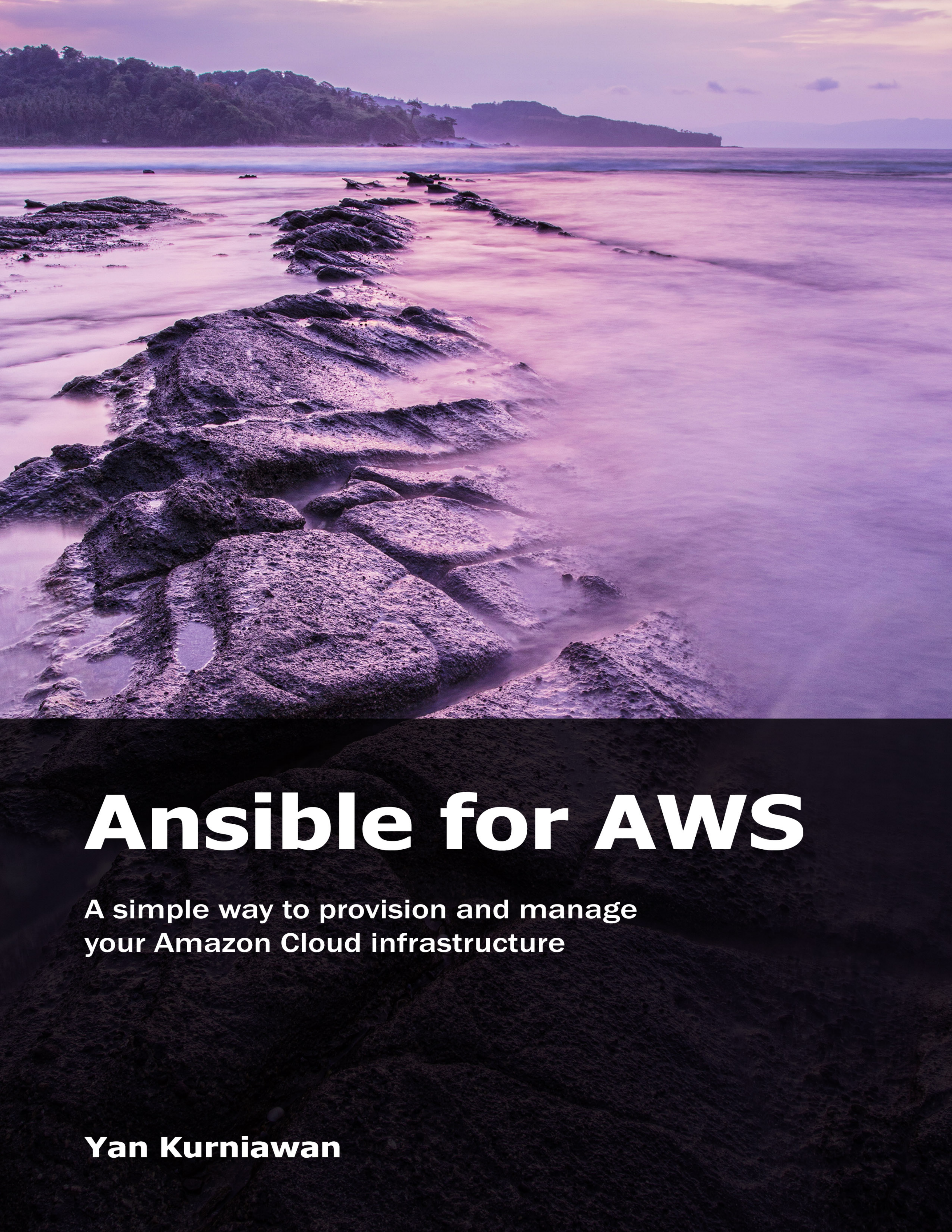


Ansible for AWS

A simple way to provision and manage your Amazon Cloud infrastructure

Yan Kurniawan

Ansible for AWS

A simple way to provision and manage your Amazon Cloud infrastructure

Yan Kurniawan

This book is for sale at http://leanpub.com/ansible-for-aws

This version was published on 2016-01-15

Tweet This Book!

Please help Yan Kurniawan by spreading the word about this book on Twitter!

The suggested hashtag for this book is #ansible4aws.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#ansible4aws *to my wife - who never thought I would write a book*

to my children - who still learn to write

Contents

Chapter 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 (AWS)?

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.

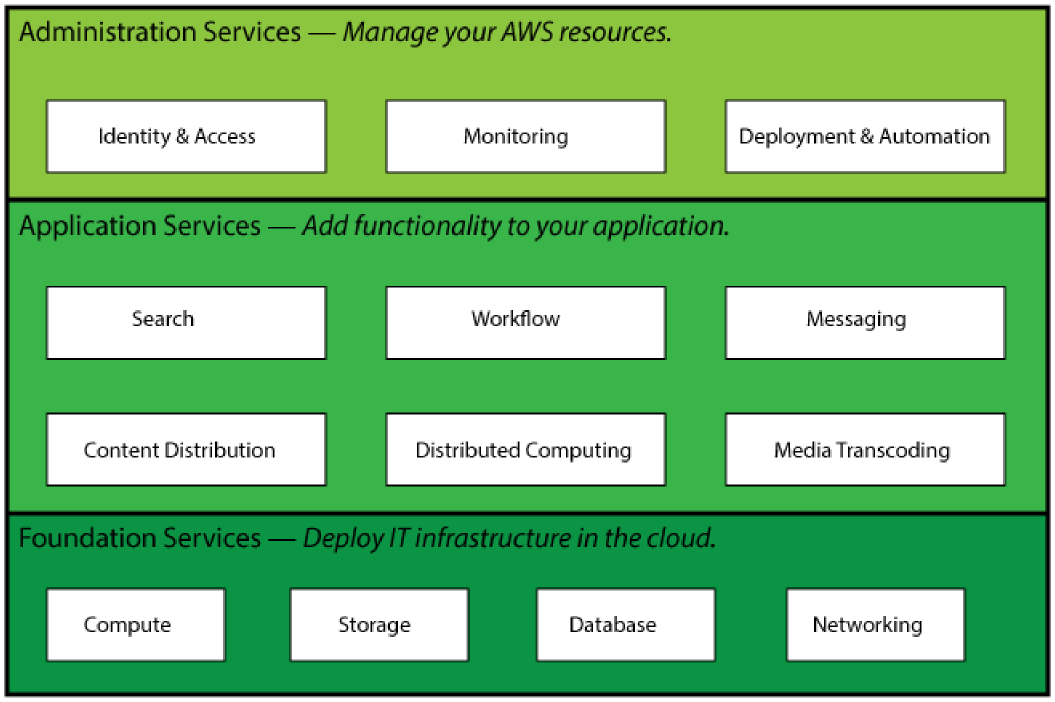
You pay for your virtual server only while it runs, with no up-front purchase costs or ongoing maintenance costs. Backed by the AWS network, your virtual server can do things no physical server can, such as automatically scaling into multiple servers when demand for your application increases.

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, email 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.[1]

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

[1] Getting Started with AWS


Administration Services — *Manage your AWS resources.*
Identity & Access
Monitoring
Deployment & Automation
Application Services — *Add functionality to your application.*
Search
Workflow
Messaging
Content Distribution
Distributed Computing
Media Transcoding
Foundation Services — *Deploy IT infrastructure in the cloud.*
Compute
Storage
Database
Networking

AWS Functionality

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[2]
- What is Amazon Web Services[3]

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)
- **ElastiCache**
- **Route 53**

[2]http://youtu.be/jOhbTAU4OPI
[3]http://youtu.be/mZ5H8sn_2ZI

EC2 (Elastic Compute Cloud)

EC2

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.[4]

VPC (Virtual Private Cloud)

VPC

Amazon Virtual Private Cloud (Amazon VPC) lets you provision a logically isolated section of the Amazon Web Services (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.[5]

RDS (Relational Database Service)

RDS

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

[4]http://aws.amazon.com/ec2
[5]http://aws.amazon.com/vpc

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.[6]

S3 (Simple Storage Service)

S3

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.[7]

ELB (Elastic Load Balancing)

ELB

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.[8]

[6] http://aws.amazon.com/rds
[7] http://aws.amazon.com/s3
[8] http://aws.amazon.com/elasticloadbalancing

ElastiCache

ElastiCache

ElastiCache is a web service that makes it easy to deploy, operate, and scale an in-memory cache in the cloud. The service improves the performance of web applications by allowing you to retrieve information from fast, managed, in-memory caches, instead of relying entirely on slower disk-based databases.

Amazon ElastiCache automatically detects and replaces failed nodes, reducing the overhead associated with self-managed infrastructures and provides a resilient system that mitigates the risk of overloaded databases, which slow website and application load times. Through integration with Amazon CloudWatch, Amazon ElastiCache provides enhanced visibility into key performance metrics associated with your Memcached or Redis nodes.

ElastiCache supports two open-source caching engines:

- Memcached - a widely adopted memory object caching system. ElastiCache is protocol compliant with Memcached, so popular tools that you use today with existing Memcached environments will work seamlessly with the service.
- Redis – a popular open-source in-memory key-value store that supports data structures such as sorted sets and lists. ElastiCache supports Redis master/slave replication which can be used to achieve cross AZ redundancy.

Using Amazon ElastiCache, you can add an in-memory caching layer to your infrastructure in a matter of minutes by using the AWS Management Console.[9]

Route 53

Route53

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.

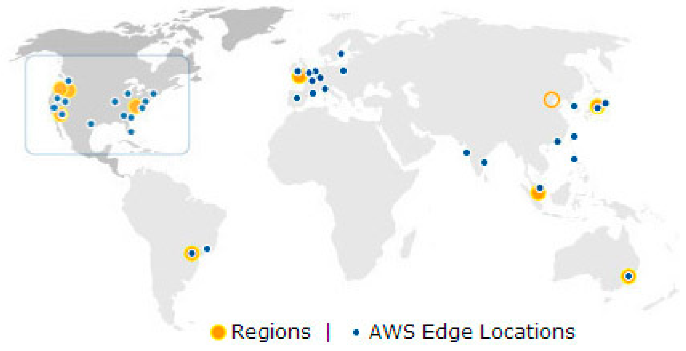
Route 53 is designed to be fast, easy to use, and cost-effective. It answers DNS queries with low latency by using a global network of DNS servers. Queries for your domain are automatically routed to the nearest DNS server, and thus answered with the best possible performance. With Route 53, you can create and manage your public DNS records with the AWS Management Console or with

[9] http://aws.amazon.com/elasticache

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.[10]

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 ten AWS regions around the world. This means you can provide a lower latency and better experience for your customers at minimal cost.[11]



See detailed list of offerings at all AWS locations[12]

[10]http://aws.amazon.com/route53

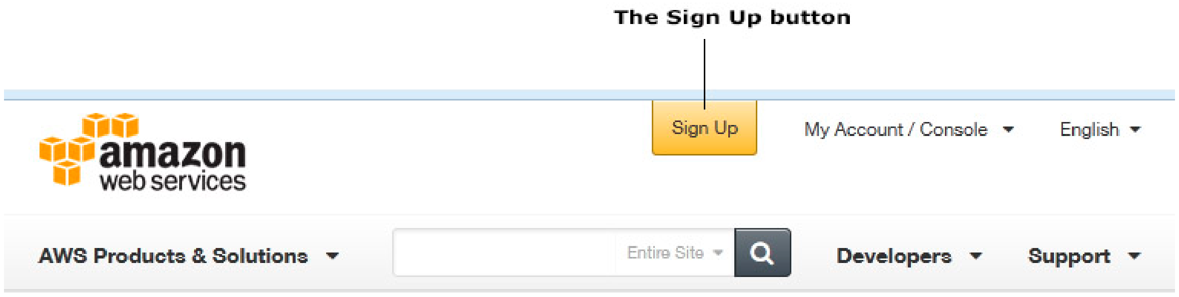
[11]http://aws.amazon.com/what-is-cloud-computing/#global-reach

[12]http://aws.amazon.com/about-aws/globalinfrastructure/regional-product-services

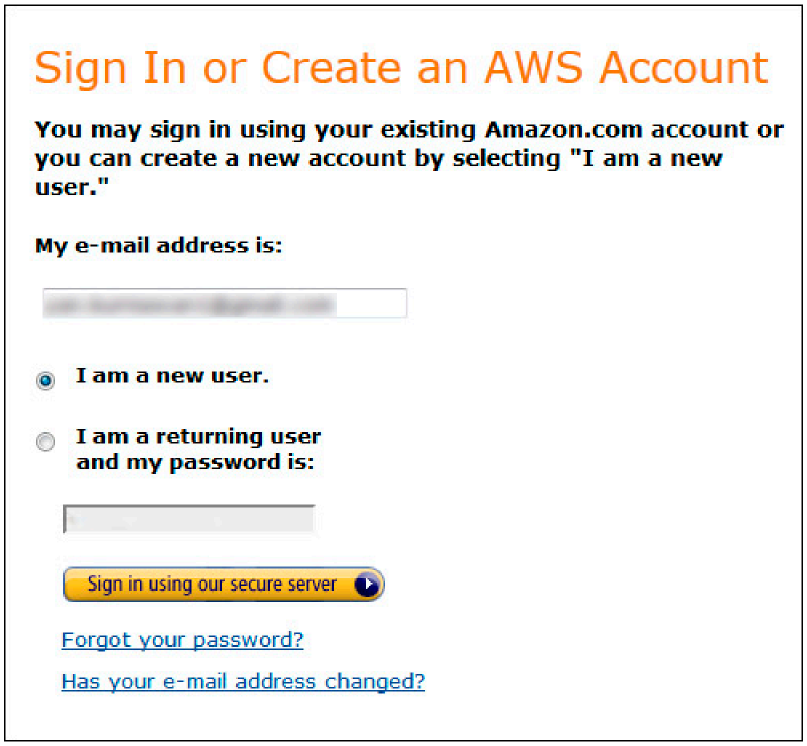
Setting Up Your AWS Account

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 below:

1. Click the Sign Up 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


Sign In or Create an AWS Account

You may sign in using your existing Amazon.com account or you can create a new account by selecting "I am a new user."

My e-mail address is:

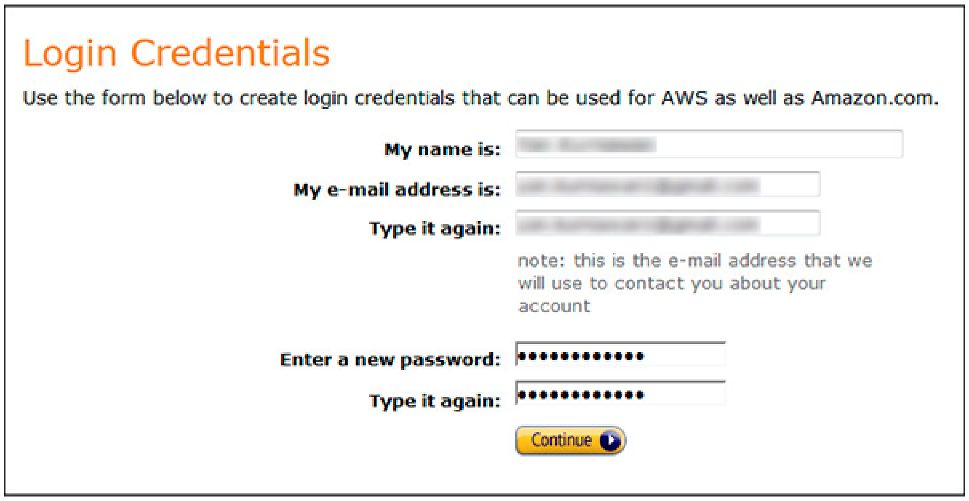
[✓ I am a new user.]

I am a returning user and my password is:

Sign in using our secure server
Forgot your password?
Has your e-mail address changed?

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.

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

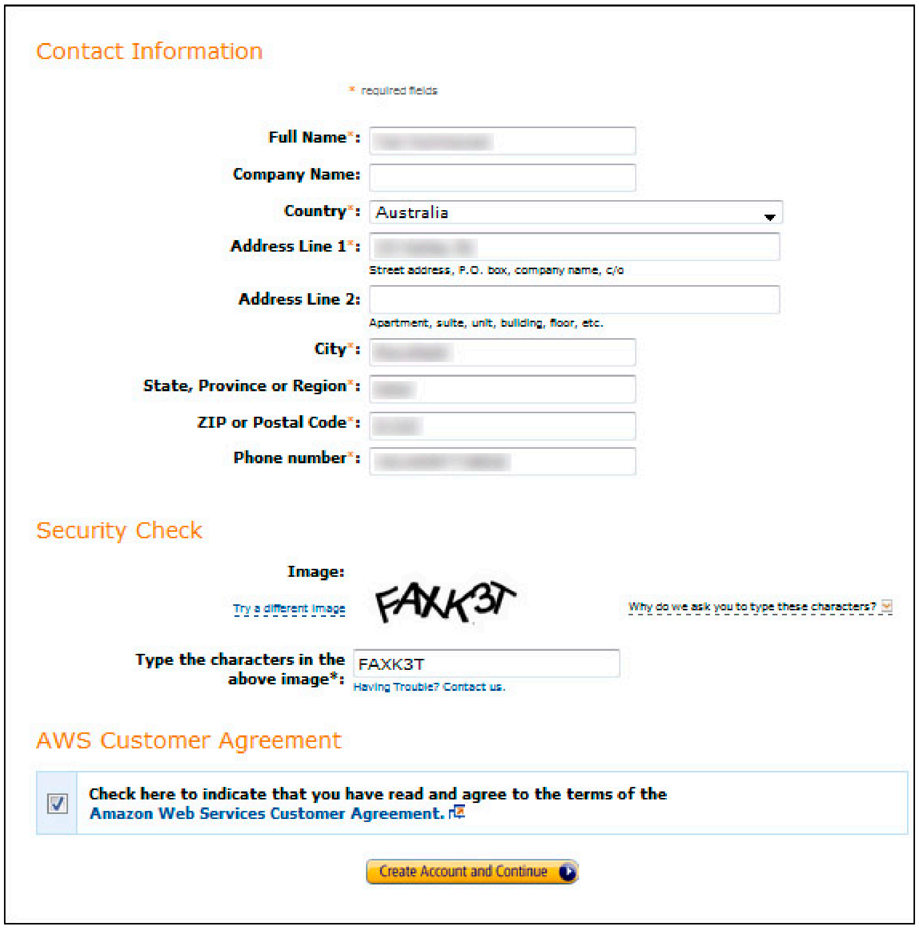


Login Credentials

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

My name is:	
My e-mail address is:	
Type it again:	
	note: this is the e-mail address that we will use to contact you about your account
Enter a new password:	•••••••••••••
Type it again:	•••••••••••••

4. On the next screen, enter the required personal information on the **Contact Information** form, type the **Security Check** characters, confirm your acceptance of the AWS customer agreement, and then click the **Create Account and Continue** button.



Contact Information

* required fields

Full Name*:	
Company Name:	
Country*:	Australia
Address Line 1*:	
	Street address, P.O. box, company name, c/o
Address Line 2:	
	Apartment, suite, unit, building, floor, etc.
City*:	
State, Province or Region*:	
ZIP or Postal Code*:	
Phone number*:	

Security Check

Image: FAXK3T

Try a different image

Why do we ask you to type these characters?

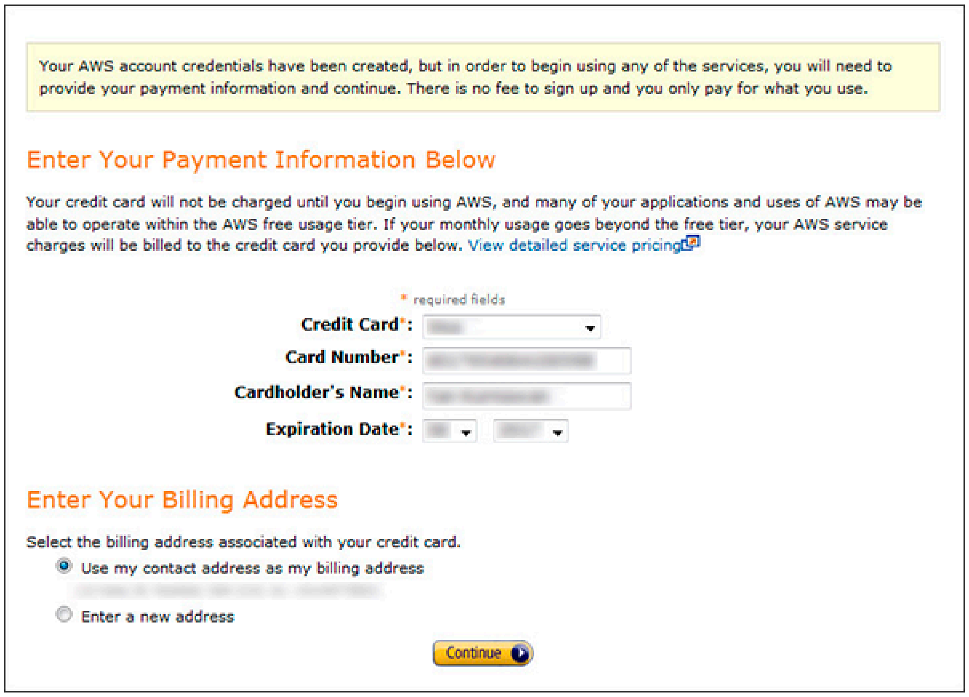
Type the characters in the above image*: FAXK3T

Having Trouble? Contact us.

AWS Customer Agreement

[✓] **Check here to indicate that you have read and agree to the terms of the Amazon Web Services Customer Agreement.**

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



Your AWS account credentials have been created, but in order to begin using any of the services, you will need to provide your payment information and continue. There is no fee to sign up and you only pay for what you use.

Enter Your Payment Information Below

Your credit card will not be charged until you begin using AWS, and many of your applications and uses of AWS may be able to operate within the AWS free usage tier. If your monthly usage goes beyond the free tier, your AWS service charges will be billed to the credit card you provide below. View detailed service pricing

* required fields

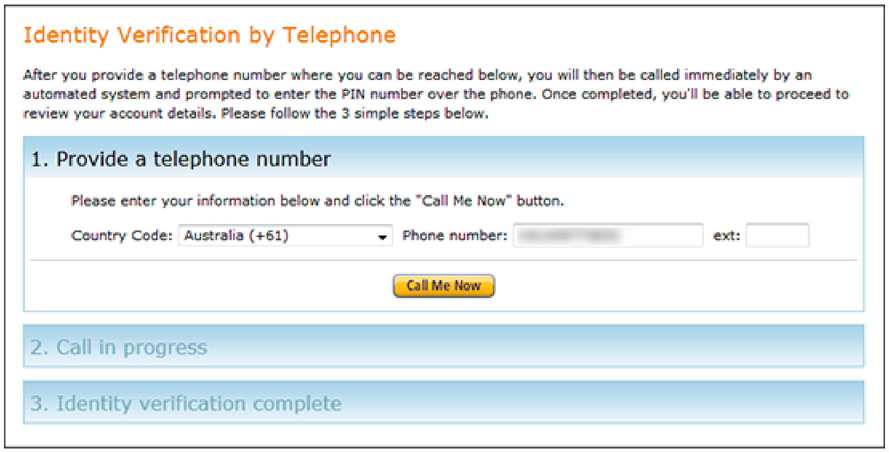
Credit Card*:	[redacted]
Card Number*:	[redacted]
Cardholder's Name*:	[redacted]
Expiration Date*:	[redacted] [redacted]

Enter Your Billing Address

Select the billing address associated with your credit card.

[✓ Use my contact address as my billing address]
[redacted]
Enter a new address

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 by Telephone

After you provide a telephone number where you can be reached below, you will then be called immediately by an automated system and prompted to enter the PIN number over the phone. Once completed, you'll be able to proceed to review your account details. Please follow the 3 simple steps below.

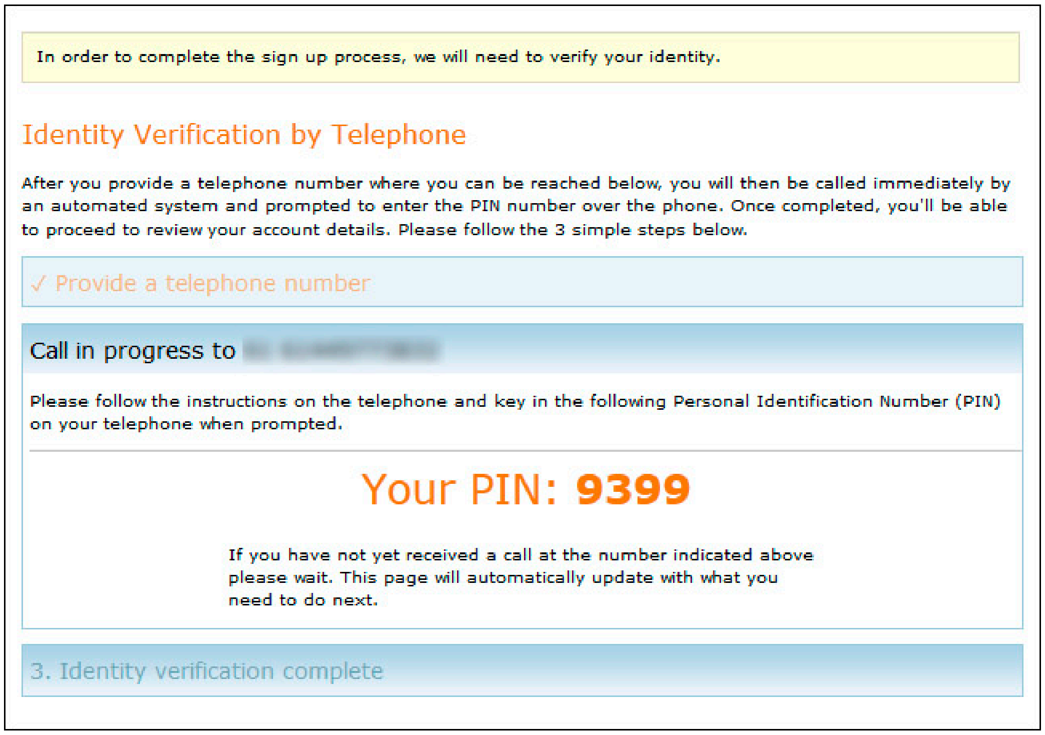
1. Provide a telephone number

Please enter your information below and click the "Call Me Now" button.

Country Code: Australia (+61) Phone number: [redacted] ext:

2. Call in progress

3. Identity verification complete



In order to complete the sign up process, we will need to verify your identity.

Identity Verification by Telephone

After you provide a telephone number where you can be reached below, you will then be called immediately by an automated system and prompted to enter the PIN number over the phone. Once completed, you'll be able to proceed to review your account details. Please follow the 3 simple steps below.

✓ Provide a telephone number

Call in progress to [redacted]

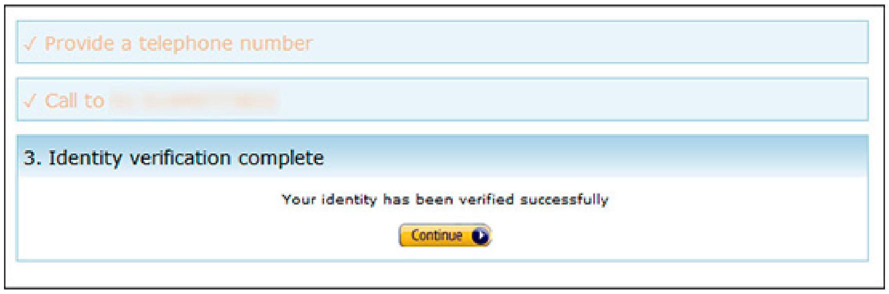
Please follow the instructions on the telephone and key in the following Personal Identification Number (PIN) on your telephone when prompted.

Your PIN: **9399**

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

Answer the phone and enter the displayed PIN code on the telephone keypad, or you can say the PIN numbers. After the identity verification completed successfully, click the **Continue** button.



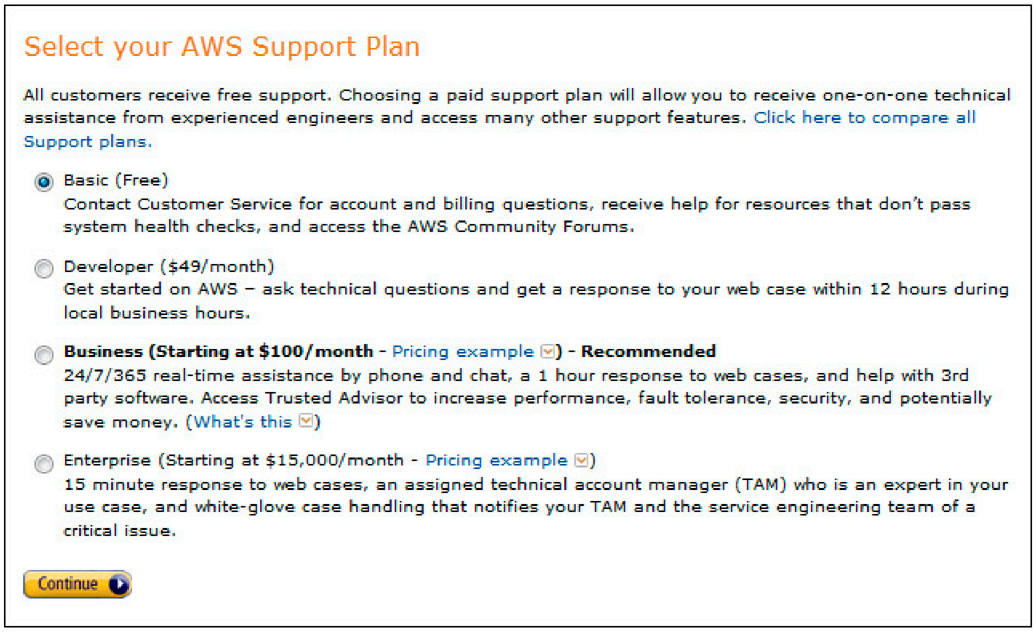
✓ Provide a telephone number

✓ Call to [redacted]

3. Identity verification complete

Your identity has been verified successfully

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



Select your AWS Support Plan

All customers receive free support. Choosing a paid support plan will allow you to receive one-on-one technical assistance from experienced engineers and access many other support features. Click here to compare all Support plans.

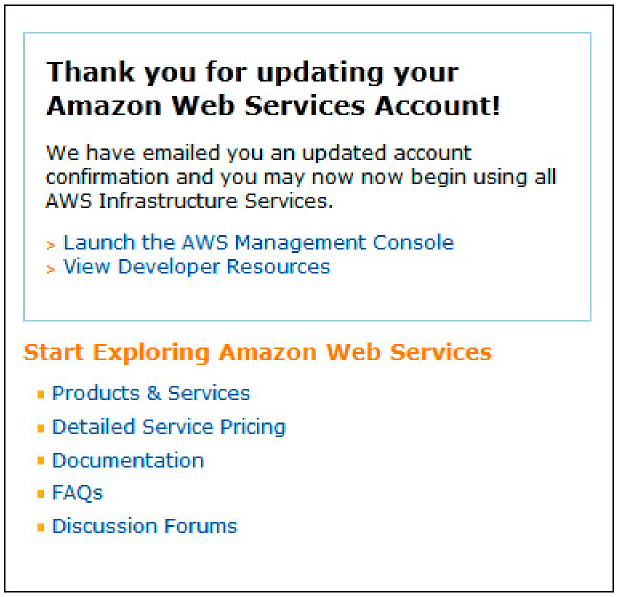
[✓ Basic (Free)]
Contact Customer Service for account and billing questions, receive help for resources that don't pass system health checks, and access the AWS Community Forums.

Developer ($49/month)
Get started on AWS – ask technical questions and get a response to your web case within 12 hours during local business hours.

Business (Starting at $100/month - Pricing example **) - Recommended**
24/7/365 real-time assistance by phone and chat, a 1 hour response to web cases, and help with 3rd party software. Access Trusted Advisor to increase performance, fault tolerance, security, and potentially save money. (What's this)

Enterprise (Starting at $15,000/month - Pricing example)
15 minute response to web cases, an assigned technical account manager (TAM) who is an expert in your use case, and white-glove case handling that notifies your TAM and the service engineering team of a critical issue.

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



Thank you for updating your Amazon Web Services Account!

We have emailed you an updated account confirmation and you may now now begin using all AWS Infrastructure Services.

> Launch the AWS Management Console
> View Developer Resources

Start Exploring Amazon Web Services

- Products & Services
- Detailed Service Pricing
- Documentation
- FAQs
- Discussion Forums

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[13]
S3 Pricing Scheme[14]

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[15] page.

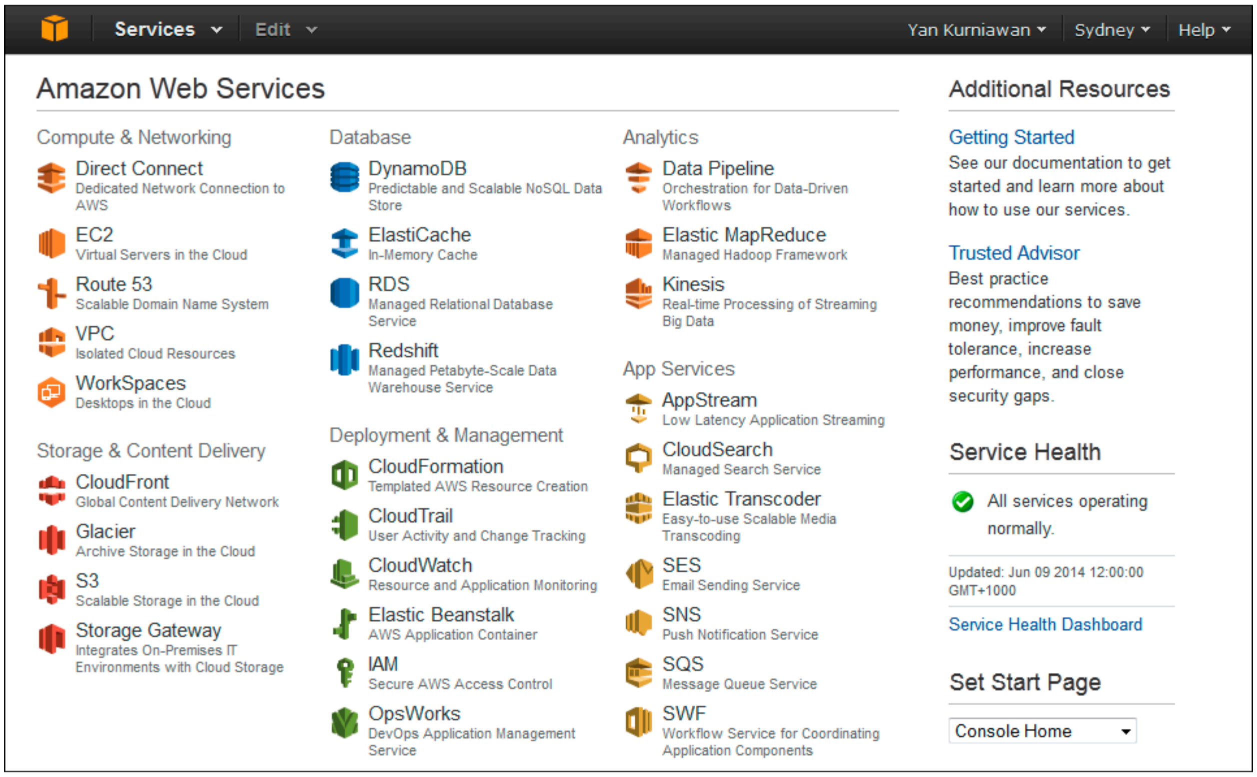
Amazon provides an online calculator[16] to estimate your monthly AWS bill.

[13]http://aws.amazon.com/ec2/pricing
[14]http://aws.amazon.com/s3/pricing
[15]http://aws.amazon.com/free
[16]http://calculator.s3.amazonaws.com/index.html

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.



Amazon Web Services

Compute & Networking
- Direct Connect — Dedicated Network Connection to AWS
- EC2 — Virtual Servers in the Cloud
- Route 53 — Scalable Domain Name System
- VPC — Isolated Cloud Resources
- WorkSpaces — Desktops in the Cloud

Storage & Content Delivery
- CloudFront — Global Content Delivery Network
- Glacier — Archive Storage in the Cloud
- S3 — Scalable Storage in the Cloud
- Storage Gateway — Integrates On-Premises IT Environments with Cloud Storage

Database
- DynamoDB — Predictable and Scalable NoSQL Data Store
- ElastiCache — In-Memory Cache
- RDS — Managed Relational Database Service
- Redshift — Managed Petabyte-Scale Data Warehouse Service

Deployment & Management
- CloudFormation — Templated AWS Resource Creation
- CloudTrail — User Activity and Change Tracking
- CloudWatch — Resource and Application Monitoring
- Elastic Beanstalk — AWS Application Container
- IAM — Secure AWS Access Control
- OpsWorks — DevOps Application Management Service

Analytics
- Data Pipeline — Orchestration for Data-Driven Workflows
- Elastic MapReduce — Managed Hadoop Framework
- Kinesis — Real-time Processing of Streaming Big Data

App Services
- AppStream — Low Latency Application Streaming
- CloudSearch — Managed Search Service
- Elastic Transcoder — Easy-to-use Scalable Media Transcoding
- SES — Email Sending Service
- SNS — Push Notification Service
- SQS — Message Queue Service
- SWF — Workflow Service for Coordinating Application Components

Additional Resources

Getting Started
See our documentation to get started and learn more about how to use our services.

Trusted Advisor
Best practice recommendations to save money, improve fault tolerance, increase performance, and close security gaps.

Service Health

✓ All services operating normally.

Updated: Jun 09 2014 12:00:00 GMT+1000

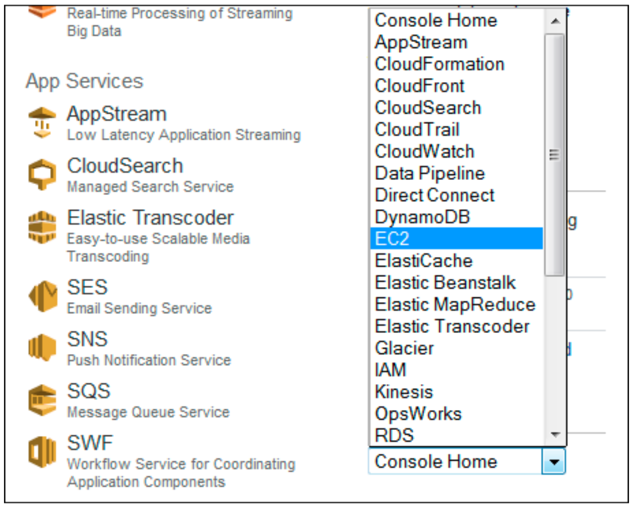
Service Health Dashboard

Set Start Page

Console Home

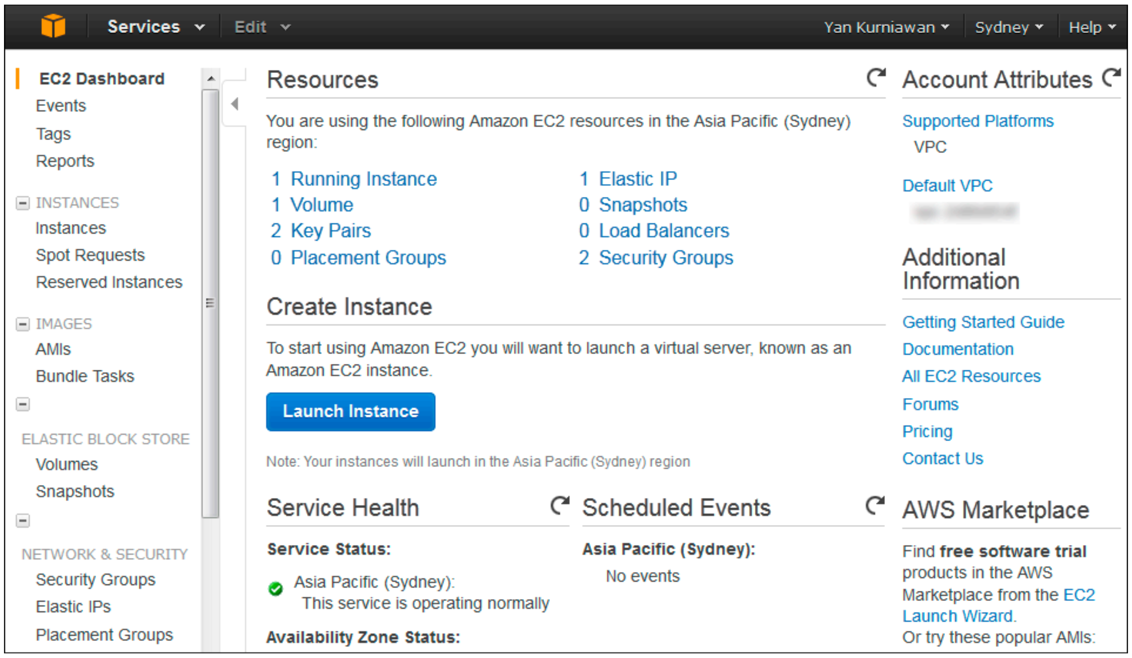
The Console Home

To set the start page or landing page after you log in, you can choose it from the **Set Start Page** pull-down menu. For example, if you work with EC2 instances most of the time, you can set your landing page to EC2 dashboard.



Set Start Page

After you choose EC2 Dashboard as your start page, the next time you log in to your AWS account you will land on the EC2 Dashboard.



You are using the following Amazon EC2 resources in the Asia Pacific (Sydney) region:

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

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

Service Status:

Asia Pacific (Sydney):
This service is operating normally

Availability Zone Status:

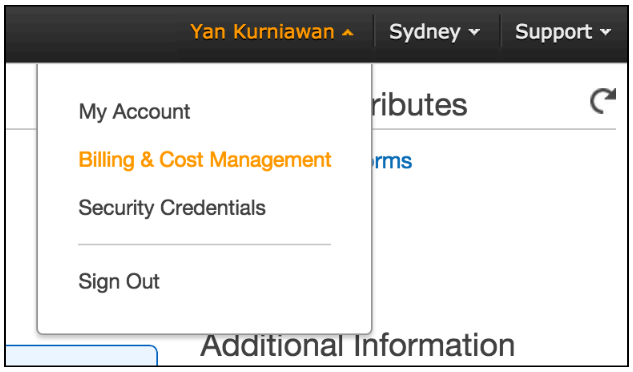
Asia Pacific (Sydney):
No events

Default VPC: [redacted]

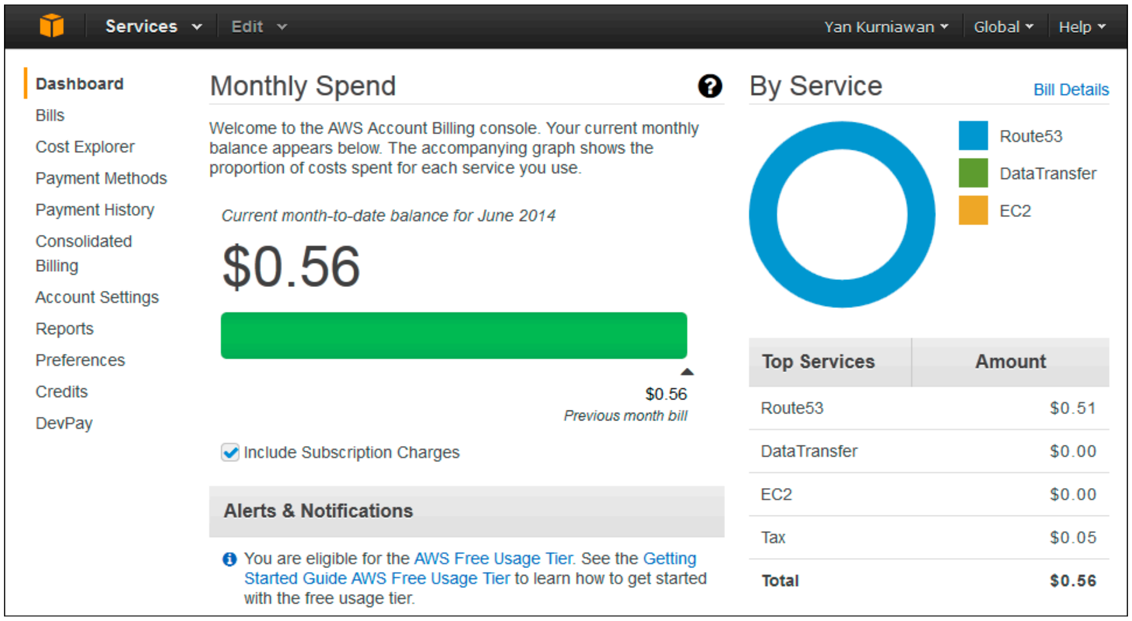
Find **free software trial** products in the AWS Marketplace from the EC2 Launch Wizard.
Or try these popular AMIs:

The EC2 Dashboard

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



Welcome to the AWS Account Billing console. Your current monthly balance appears below. The accompanying graph shows the proportion of costs spent for each service you use.

Current month-to-date balance for June 2014

$0.56

$0.56
Previous month bill

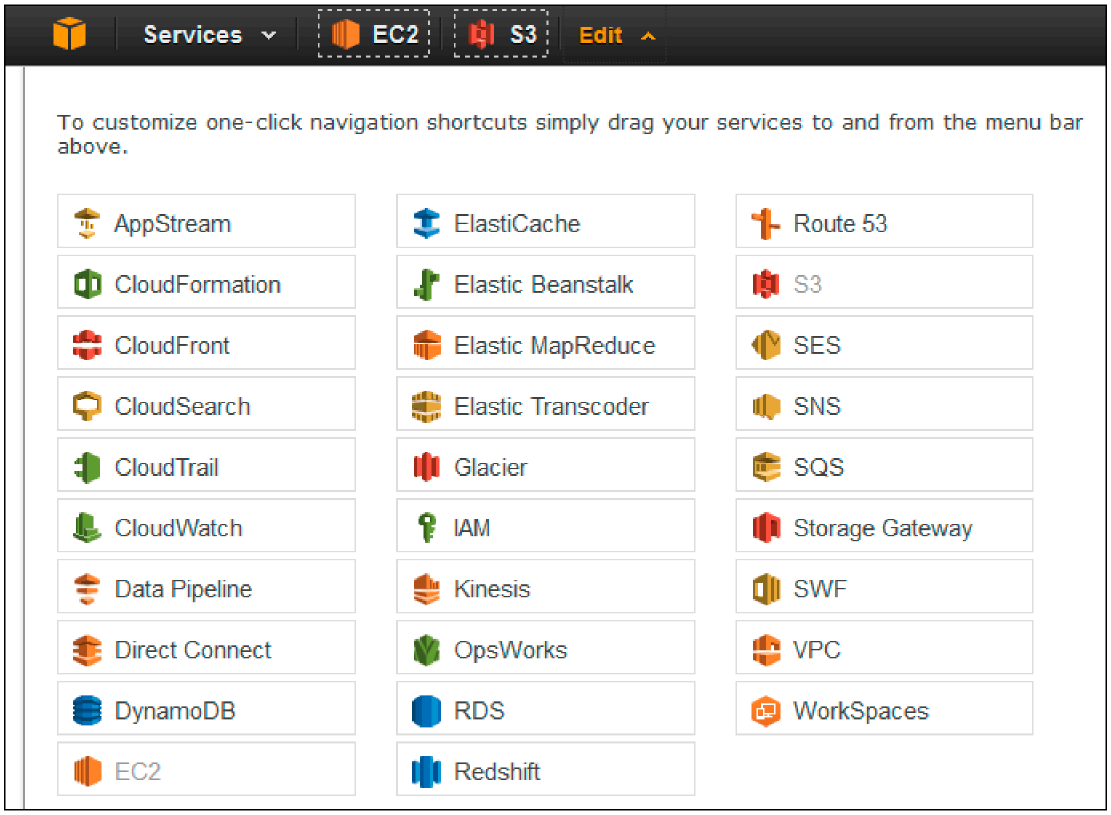
✓ Include Subscription Charges

You are eligible for the AWS Free Usage Tier. See the Getting Started Guide AWS Free Usage Tier to learn how to get started with the free usage tier.

Top Services	Amount
Route53	$0.51
DataTransfer	$0.00
EC2	$0.00
Tax	$0.05
Total	**$0.56**

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.



To customize one-click navigation shortcuts simply drag your services to and from the menu bar above.

Customize One-click Navigation

For a complete guide to AWS Management Console, visit AWS Management Console Getting Started Guide[17] page.

[17]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.

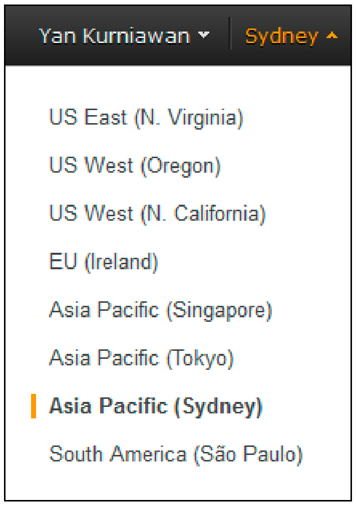
Let's create our first EC2 instance.

Create a Key Pair

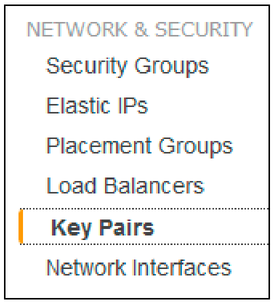
First, we need to create a key pair. AWS uses public-key cryptography to secure the login information for your instance. A Linux instance has no password; you use a key pair to log in to your instance securely. You specify the name of the key pair when you launch your instance, then provide the private 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

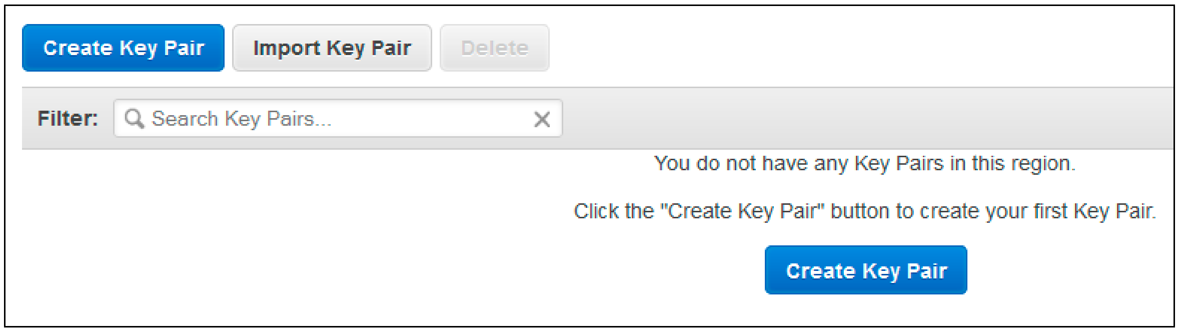
1. Open the Amazon EC2 Dashboard https://console.aws.amazon.com/ec2
2. From the navigation bar, select a region for the key pair.



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



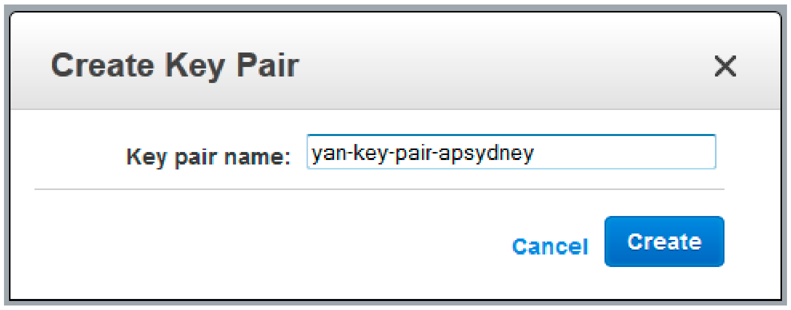
4. Click **Create Key Pair**



You do not have any Key Pairs in this region.

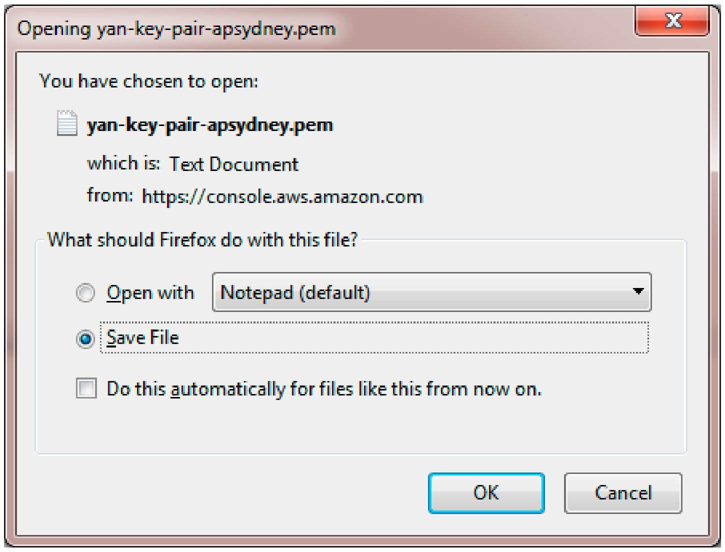
Click the "Create Key Pair" button to create your first Key Pair.

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`.



Key pair name: 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.
   Important
   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.


Opening yan-key-pair-apsydney.pem

You have chosen to open:

yan-key-pair-apsydney.pem

which is: Text Document

from: https://console.aws.amazon.com

What should Firefox do with this file?

[ ] Open with Notepad (default)

[✓ Save File]

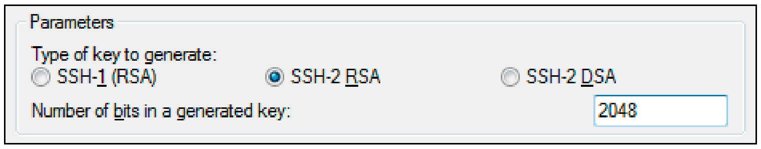
[ ] Do this automatically for files like this from now on.

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-apsydney.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**.

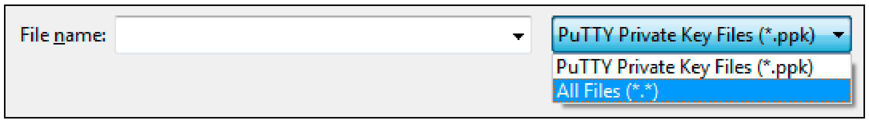

Parameters

Type of key to generate:

[ ] SSH-1 (RSA) [✓ SSH-2 RSA] [ ] SSH-2 DSA

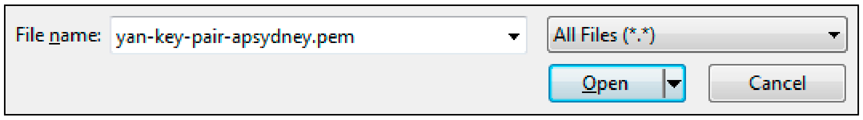
Number of bits in a generated key: 2048

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.


File name:

PuTTY Private Key Files (*.ppk)

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



6. Click **Save private key**. PuTTYgen displays a warning about saving the key without a passphrase. Click **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

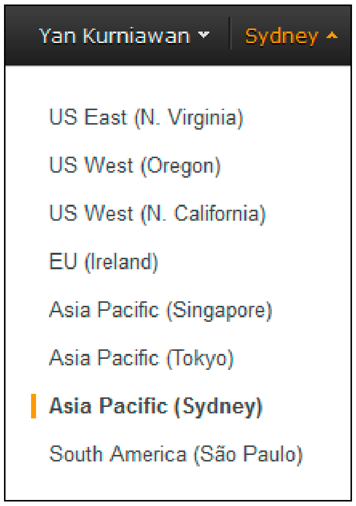
Security groups act as a firewall for associated instances, controlling both inbound and outbound traffic at the instance level. You must add rules to a security group that enable you to connect to your instance from your IP address using SSH. You can also add rules that allow inbound and outbound HTTP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

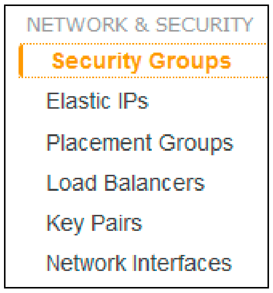
You'll need the public IP address of your local computer, which you can get using a service from Amazon AWS http://checkip.amazonaws.com. If you are connecting through an Internet service provider (ISP) or from behind a firewall without a static IP address, you need to find out the range of IP addresses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

To create a security group

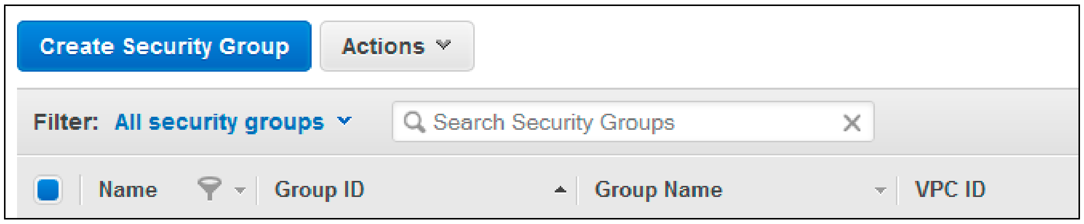
1. Open the Amazon EC2 Dashboard https://console.aws.amazon.com/ec2
2. From the navigation bar, select a region for the security group. Security groups are specific to a region; for example, if you plan to launch an instance in the Asia Pacific (Sydney) Region, you must create a security group for the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



3. Click **Security Groups** in the navigation pane.



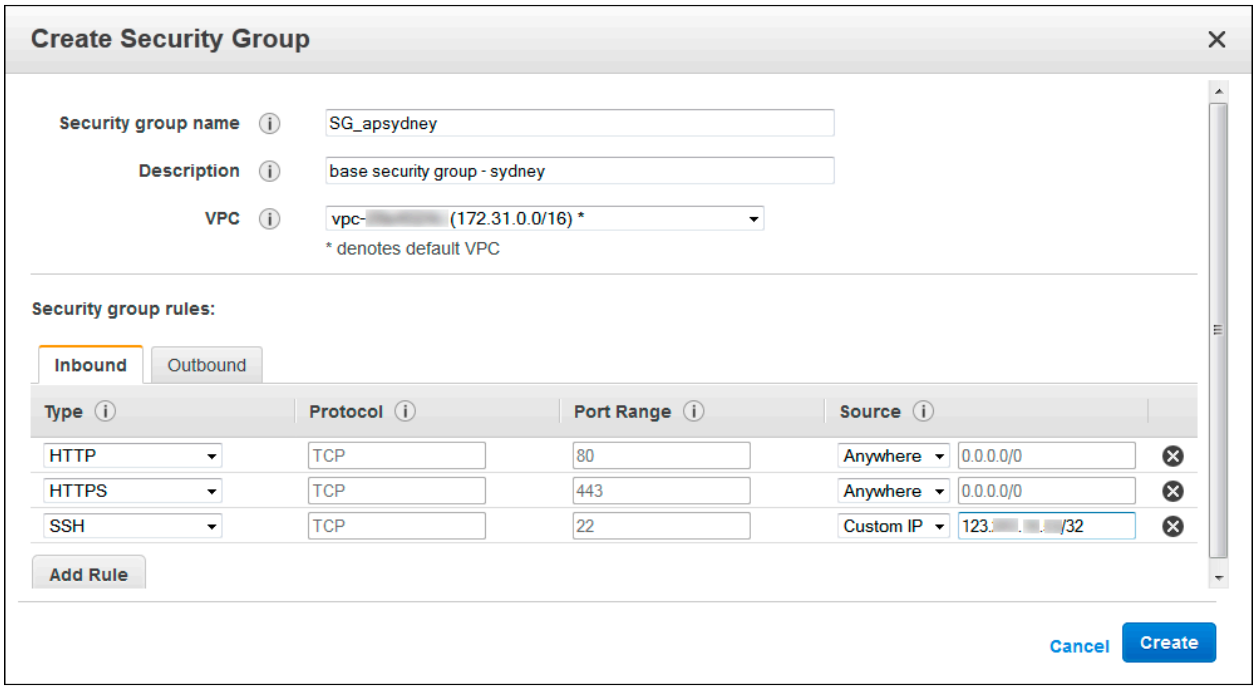
4. Click **Create Security Group**.



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.
Caution
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.



Security group name	SG_apsydney
Description	base security group - sydney
VPC	vpc-[redacted] (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.[redacted].[redacted].[redacted]/32

Add Rule

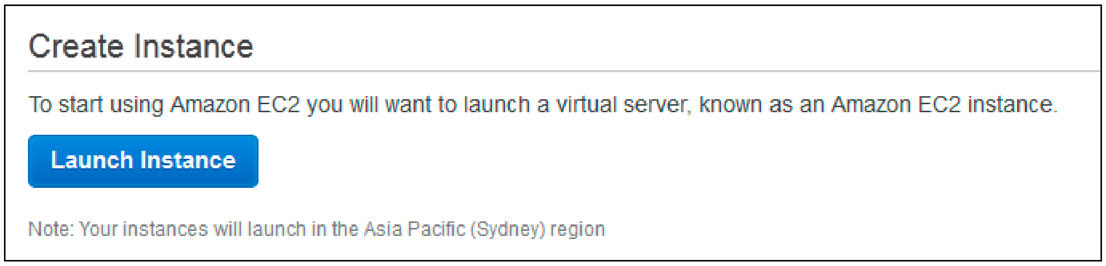
Create Security Group

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[18].

To launch an instance

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

[18]http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/launching-instance.html

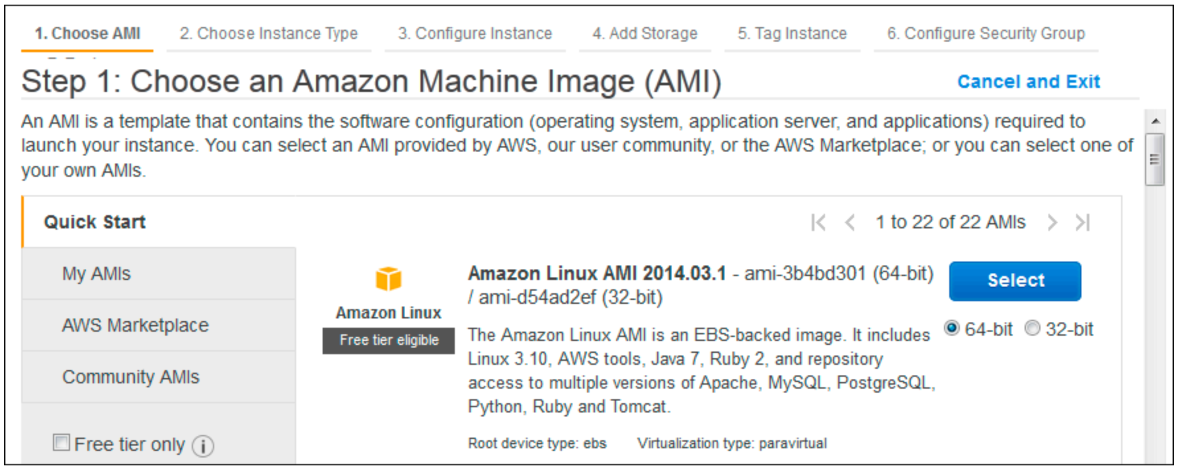


Create Instance

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 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 (AMIs) that serve as templates for your instance. Select the 64-bit Amazon Linux AMI. Notice that this configuration is marked "Free tier eligible". Click 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.

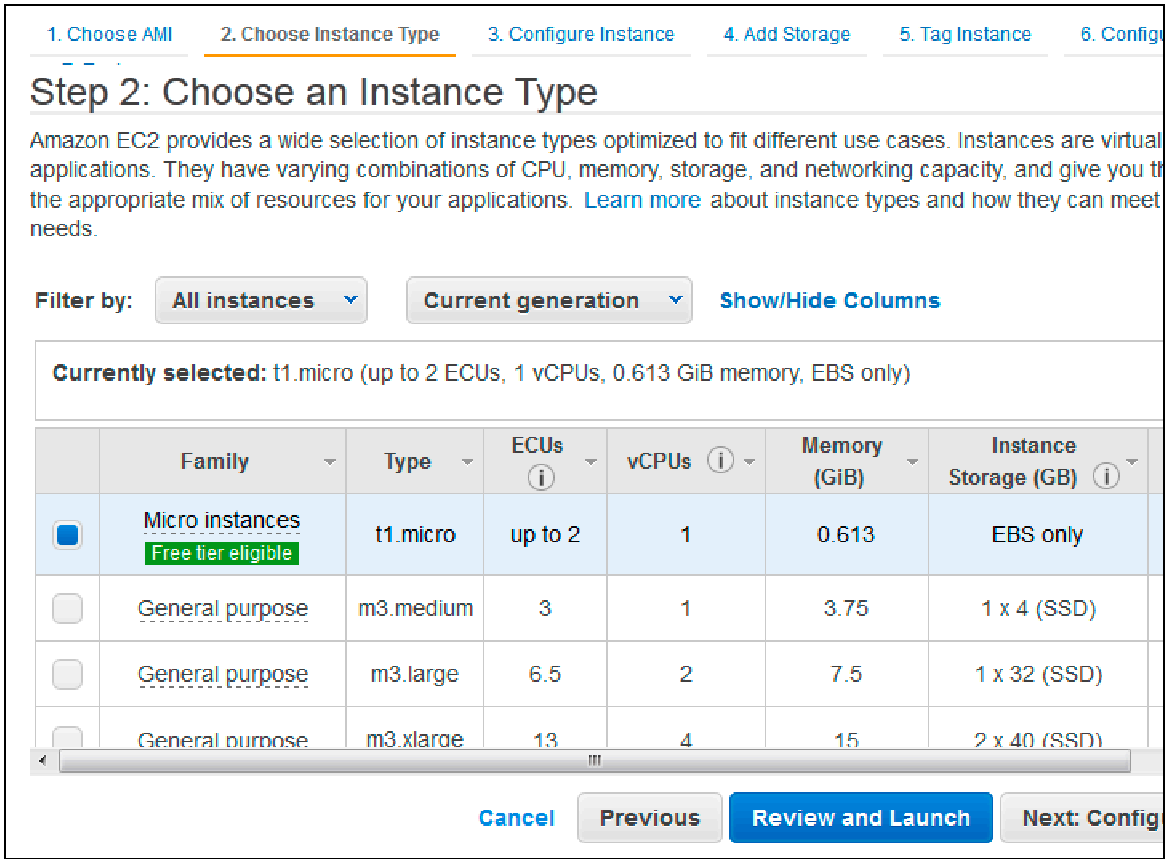
Amazon Linux AMI 2014.03.1 - ami-3b4bd301 (64-bit) / ami-d54ad2ef (32-bit)

The Amazon Linux AMI is an EBS-backed image. It includes Linux 3.10, AWS tools, Java 7, Ruby 2, and repository access to multiple versions of Apache, MySQL, PostgreSQL, Python, Ruby and Tomcat.

[✓ 64-bit] 32-bit

Root device type: ebs Virtualization type: paravirtual

4. On the **Choose an Instance Type** page, you can select the hardware configuration of your instance. The t1.micro 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

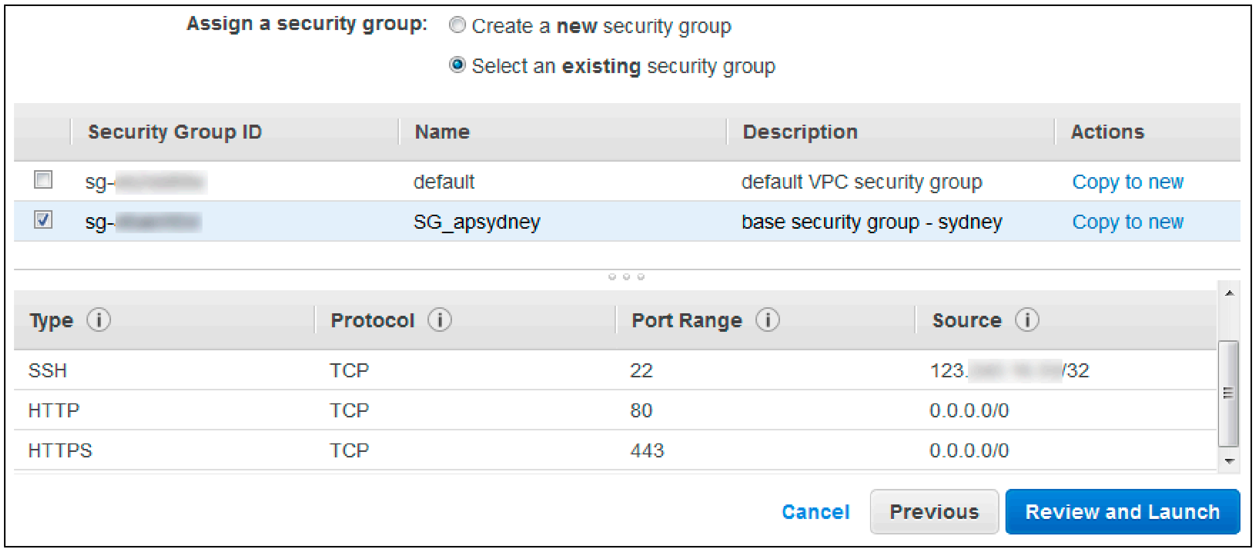
Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you t the appropriate mix of resources for your applications. Learn more about instance types and how they can meet needs.

Filter by: All instances Current generation Show/Hide Columns

Currently selected: t1.micro (up to 2 ECUs, 1 vCPUs, 0.613 GiB memory, EBS only)

	Family	Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)
■	Micro instances Free tier eligible	t1.micro	up to 2	1	0.613	EBS only
☐	General purpose	m3.medium	3	1	3.75	1 x 4 (SSD)
☐	General purpose	m3.large	6.5	2	7.5	1 x 32 (SSD)
☐	General purpose	m3.xlarge	13	4	15	2 x 40 (SSD)

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 **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**.

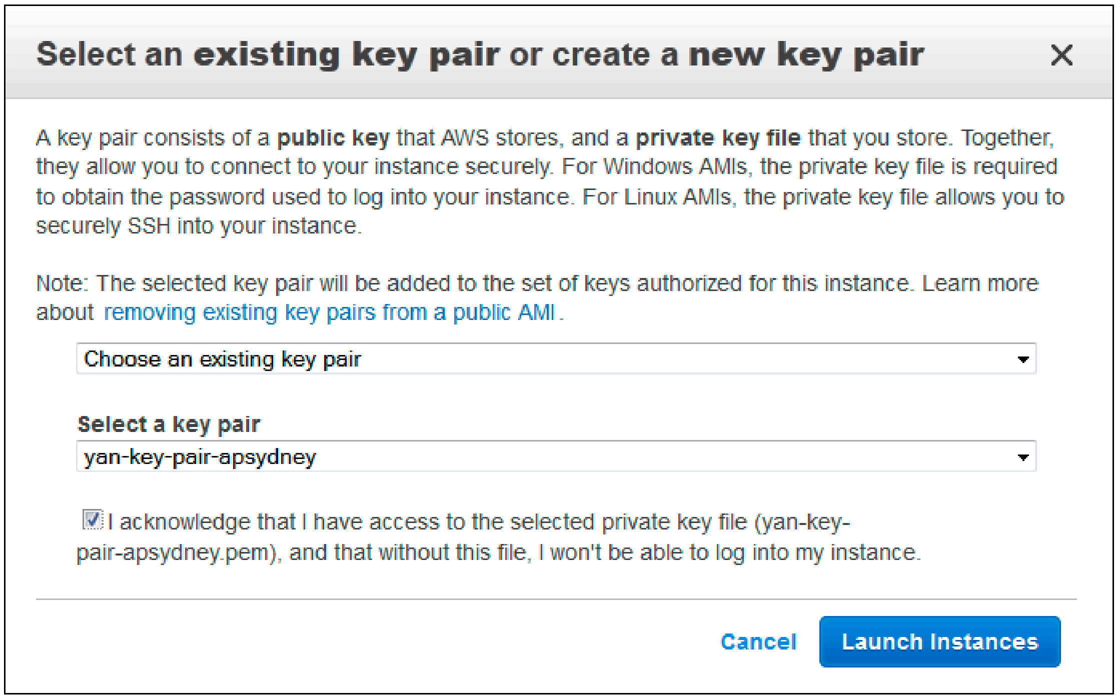


Assign a security group: Create a **new** security group [✓ Select an **existing** security group]

	Security Group ID	Name	Description	Actions
☐	sg-[redacted]	default	default VPC security group	Copy to new
✓	sg-[redacted]	SG_apsydney	base security group - sydney	Copy to new

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

6. On the **Review Instance Launch** page, click **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 **Launch Instances**.



Select an **existing key pair** or create a **new key pair**

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about removing existing key pairs from a public AMI .

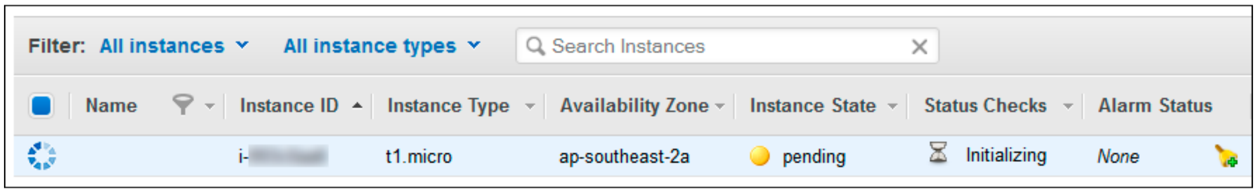
Choose an existing key pair

Select a key pair
yan-key-pair-apsydney

✓ I acknowledge that I have access to the selected private key file (yan-key-pair-apsydney.pem), and that without this file, I won't be able to log into my instance.

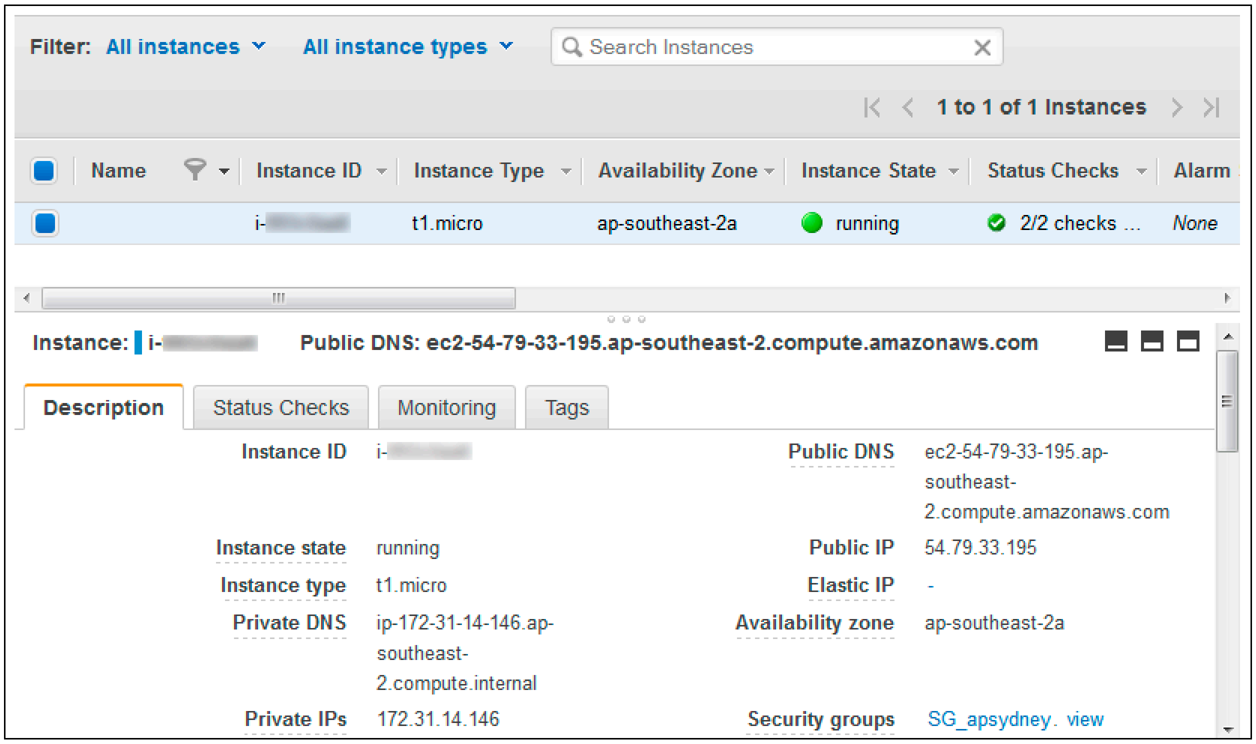
8. A confirmation page lets you know that your instance is launching. Click **View Instances** to close the confirmation page and return to the console.
9. On the **Instances** screen, you can view the status of your instance. It takes a short time for an instance to launch. When you launch an instance, 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 the **Show/Hide** icon and select **Public DNS**).



Filter: All instances All instance types Search Instances

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status
	i-[redacted]	t1.micro	ap-southeast-2a	pending	Initializing	None



Filter: All instances All instance types Search Instances

1 to 1 of 1 Instances

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm
	i-[redacted]	t1.micro	ap-southeast-2a	running	2/2 checks ...	None

Instance: i-[redacted] Public DNS: ec2-54-79-33-195.ap-southeast-2.compute.amazonaws.com

Description | Status Checks | Monitoring | Tags

Instance ID	i-[redacted]	Public DNS	ec2-54-79-33-195.ap-southeast-2.compute.amazonaws.com
Instance state	running	Public IP	54.79.33.195
Instance type	t1.micro	Elastic IP	-
Private DNS	ip-172-31-14-146.ap-southeast-2.compute.internal	Availability zone	ap-southeast-2a
Private IPs	172.31.14.146	Security groups	SG_apsydney. view

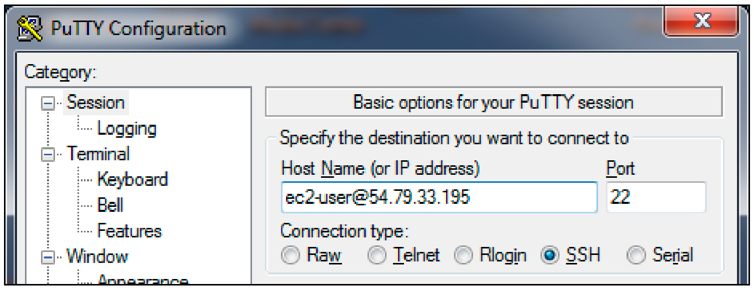
To learn more about Amazon EC2 instance type, see http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html

Connect to Your Instance

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

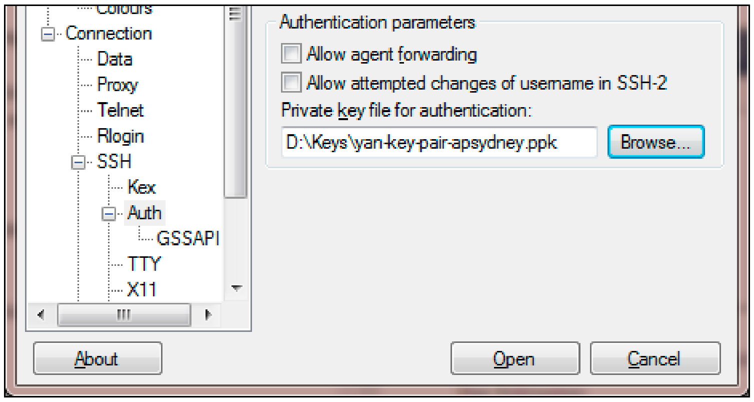
To connect to your Linux instance using PuTTY

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



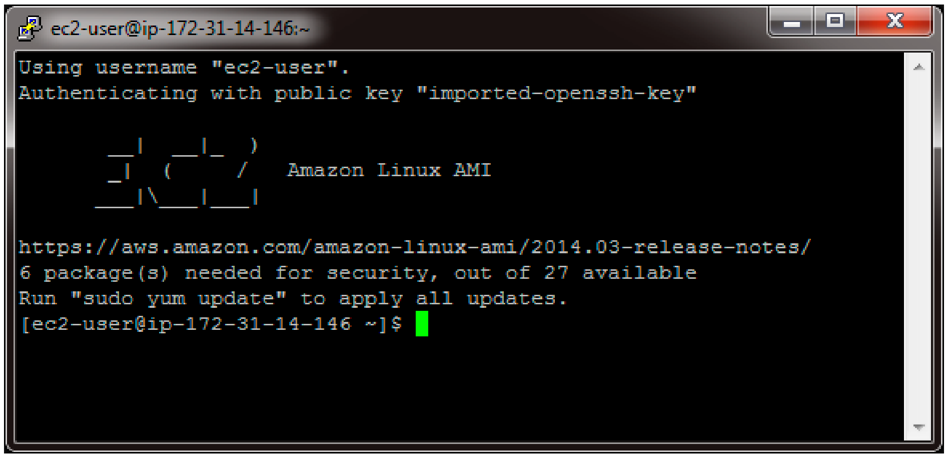
Host Name (or IP address): ec2-user@54.79.33.195
Port: 22
Connection type: Raw Telnet Rlogin [✓ SSH] Serial

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



Allow agent forwarding: ☐
Allow attempted changes of username in SSH-2: ☐
Private key file for authentication: D:\Keys\yan-key-pair-apsydney.ppk

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 **Yes**. A window opens and you are connected to your instance.



```
Using username "ec2-user".
Authenticating with public key "imported-openssh-key"

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

https://aws.amazon.com/amazon-linux-ami/2014.03-release-notes/
6 package(s) needed for security, out of 27 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-14-146 ~]$
```

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 and 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.

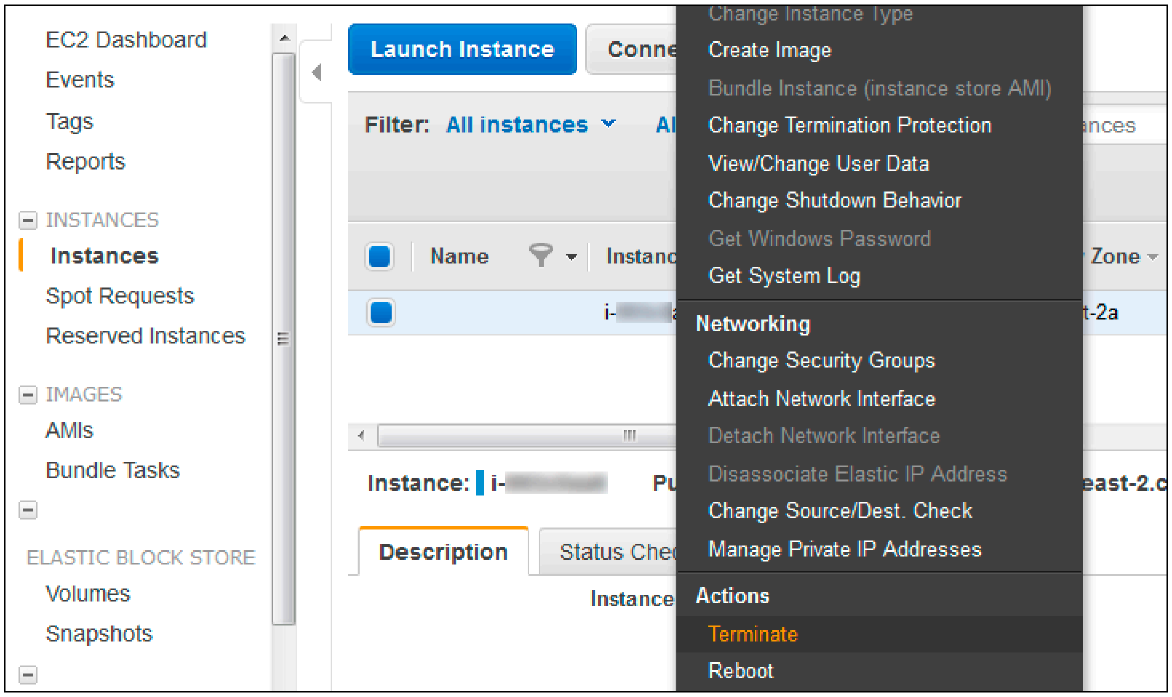
Terminate Your Instance

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.

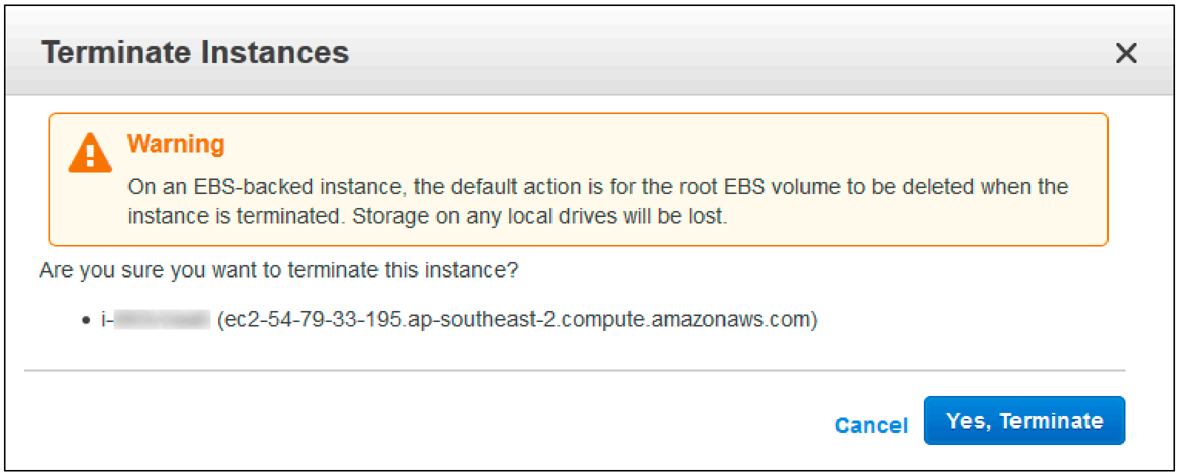
Terminating an instance effectively deletes it because you can't reconnect to an instance after you've terminated it. This differs from stopping the instance; when you stop an instance, it is shut down and you are not billed for hourly usage or data transfer. Also, you can restart a stopped instance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



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



Terminate Instances

Warning

On an EBS-backed instance, the default action is for the root EBS volume to be deleted when the instance is terminated. Storage on any local drives will be lost.

Are you sure you want to terminate this instance?

- i-[redacted] (ec2-54-79-33-195.ap-southeast-2.compute.amazonaws.com)

Most part 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[19]. For a complete list of Amazon AWS Documentation, visit AWS Documentation[20].

[19]http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html
[20]http://aws.amazon.com/documentation

Chapter 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 inter-relate, 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 allow 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.[21]

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 AMI (Amazon Machine Images) which is based on Red Hat distribution, similar to CentOS.

[21]http://www.ansible.com/how-ansible-works

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:

1. Install **RHEL EPEL** repository. The EPEL (Extra Packages for Enterprise Linux) 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 1.6.3
```

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:

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 (/root/.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 /root/.ssh/id_rsa.
Your public key has been saved in /root/.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
root@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. Login to remote host without entering the password
```
# ssh localhost
Last login: Sun Jun 22 10:17:35 2014 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.

The format for /etc/ansible/hosts 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` on the above example.

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_ssh_user=ec2-user
web2.example.com        ansible_ssh_user=ubuntu
```

Assuming the inventory file path is `/etc/ansible/hosts`, you can store host variables, in YAML format, in individual files:

```
/etc/ansible/host_vars/web1.example.com
/etc/ansible/host_vars/web2.example.com
```

For example, the data in the host variables file `/etc/ansible/host_vars/web1.example.com` might look like:

```
---
ansible_ssh_user: ec2-user
ansible_ssh_port: 5300
```

The following variables control how Ansible interacts with remote hosts:

`ansible_ssh_host`
: The name of the host to connect to, if different from the alias you wish to give to it.

`ansible_ssh_port`
: The ssh port number, if not 22

`ansible_ssh_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_sudo_pass`
: The sudo password to use (this is insecure, it's strongly recommended to use –ask-sudo-pass)

`ansible_connection`
: Connection type of the host. Candidates are local, ssh or paramiko. The default is paramiko before Ansible 1.2, and 'smart' afterwards which detects whether usage of 'ssh' would be feasible based on whether ControlPersist is supported.

`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_shell_type`
: The shell type of the target system. By default commands are formatted using 'sh'-style syntax by default. 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.

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 `/etc/ansible/hosts`, you can store group variables, in YAML format, in individual files:

`/etc/ansible/group_vars/sydney`

For example, the data in the group variables file `/etc/ansible/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 `/etc/ansible/hosts`. If the directory `/etc/ansible` does not exist, create the directory first.

```
# mkdir /etc/ansible
# vi /etc/ansible/hosts
```

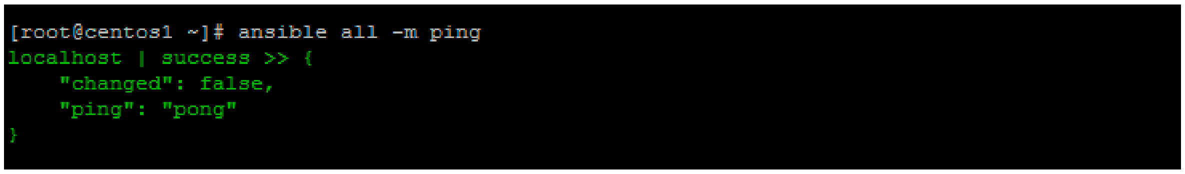
```
[local]
localhost

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

Now ping all your nodes:

```
# ansible 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.

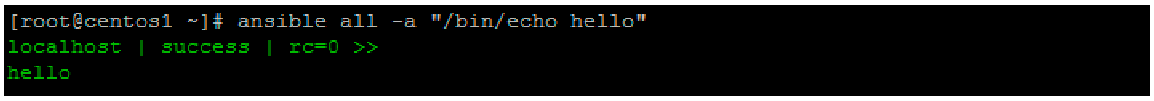


```
[root@centos1 ~]# ansible all -m ping
localhost | success >> {
    "changed": false,
    "ping": "pong"
}
```

Ping command

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

```
# ansible all -a "/bin/echo hello"
```



```
[root@centos1 ~]# ansible all -a "/bin/echo hello"
localhost | success | rc=0 >>
hello
```

Live command

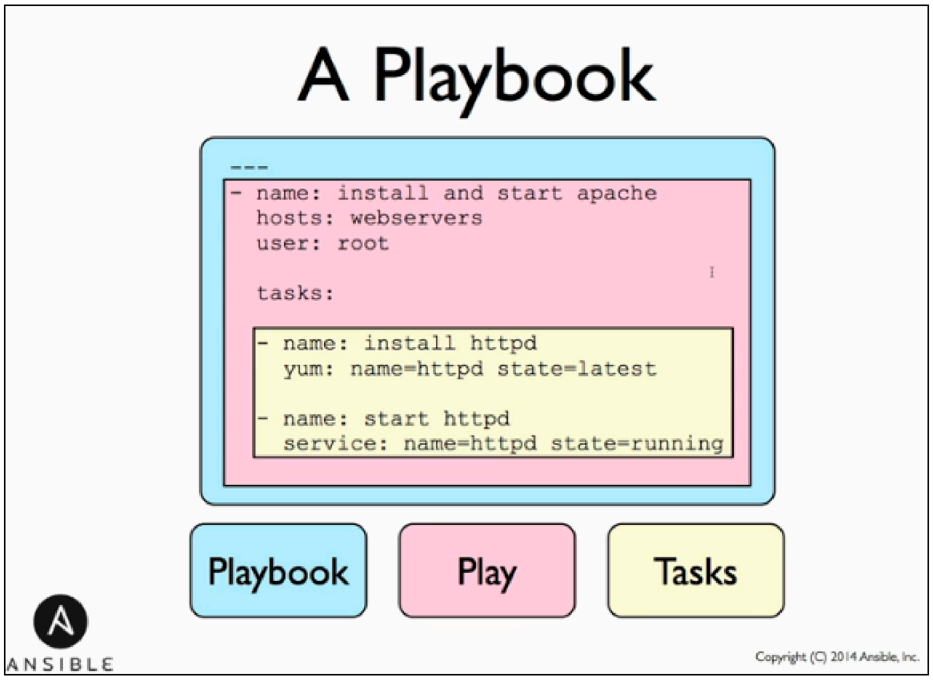
The above examples show how to use `/usr/bin/ansible` for running ad-hoc tasks. An ad-hoc command can be used to do quick things that you might not necessarily want to write a full playbook for. For example, if you want to restart some services on remote hosts.

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 allow 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.



```
---
- name: install and start apache
  hosts: webservers
  user: root

  tasks:

  - name: install httpd
    yum: name=httpd state=latest

  - name: start httpd
    service: name=httpd state=running
```

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[22] 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.

[22]http://docs.ansible.com/modules_by_category.html

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:

`sudo`
: Set this to `yes` if you want Ansible to running things from sudo for the whole play.

`remote_user`
: This defines the username Ansible will use to connect to targeted hosts.

`sudo_user`
: Sudo to different user than root.

`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:

Example:

```
---

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

  tasks:
    - ...
```

Run the playbook and pass the variables:

```
# ansible-playbook 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 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 somecommand 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 `/etc/ansible/hosts`:

```
[local]
localhost
```

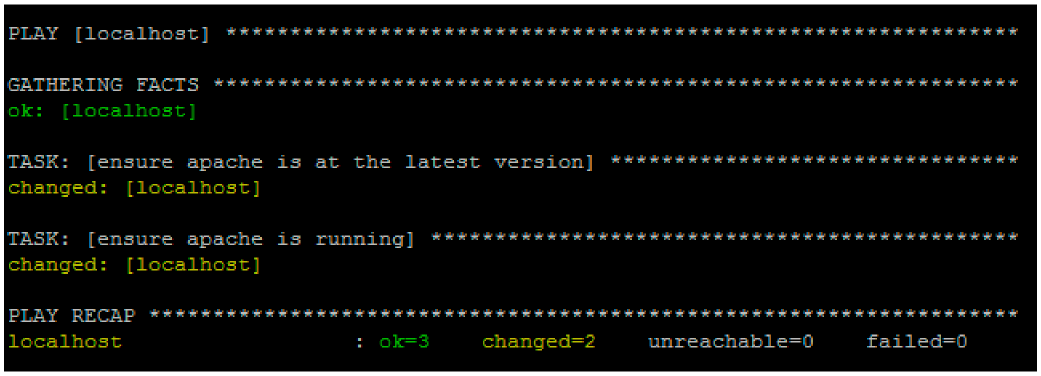
Create a playbook file `site.yml`:

```
# cd /etc/ansible
# 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 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
```

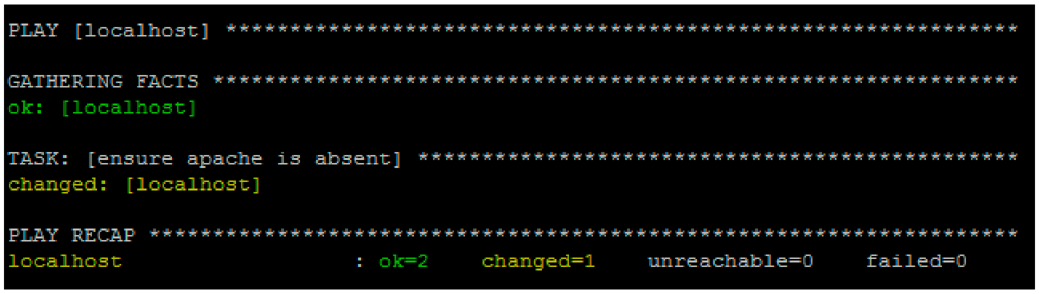
Now, let's uninstall Apache using our playbook

```
# 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 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 is 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[23] page.

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

[23]http://docs.ansible.com

Chapter 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. [24]

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 re-create your whole infrastructure, revert back to previous version of your infrastructure, and setup a consistent development, testing, and production environment.

In this chapter, you will learn about AWS Virtual Private Cloud (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 chapter 1, 3, and 4 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:

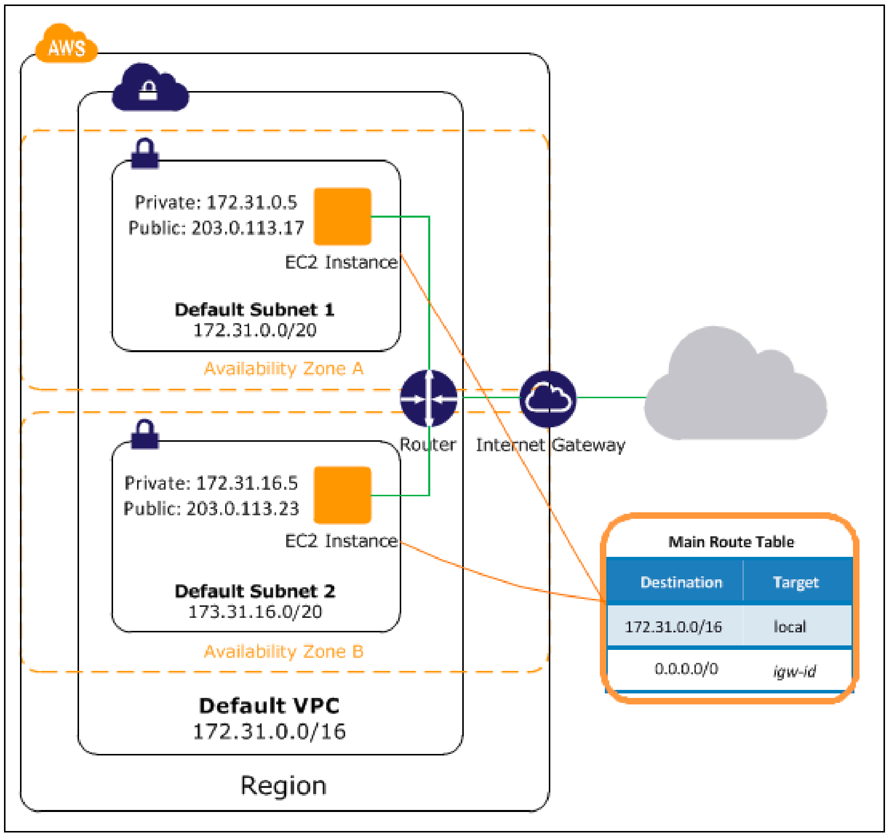
- a default subnet in each Availability Zone
- an Internet gateway connected to your default VPC
- a main route table for your default VPC with a rule that sends all traffic destined for the Internet to the Internet gateway
- a default security group associated with your default VPC
- a default network access control list (ACL) associated with your VPC
- a default DHCP options set for your AWS account associated with your default VPC

[24]http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Introduction.html

Amazon has multiple locations world-wide. 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 (ACL)* 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: [25]



Destination	Target
172.31.0.0/16	local
0.0.0.0/0	igw-id

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.

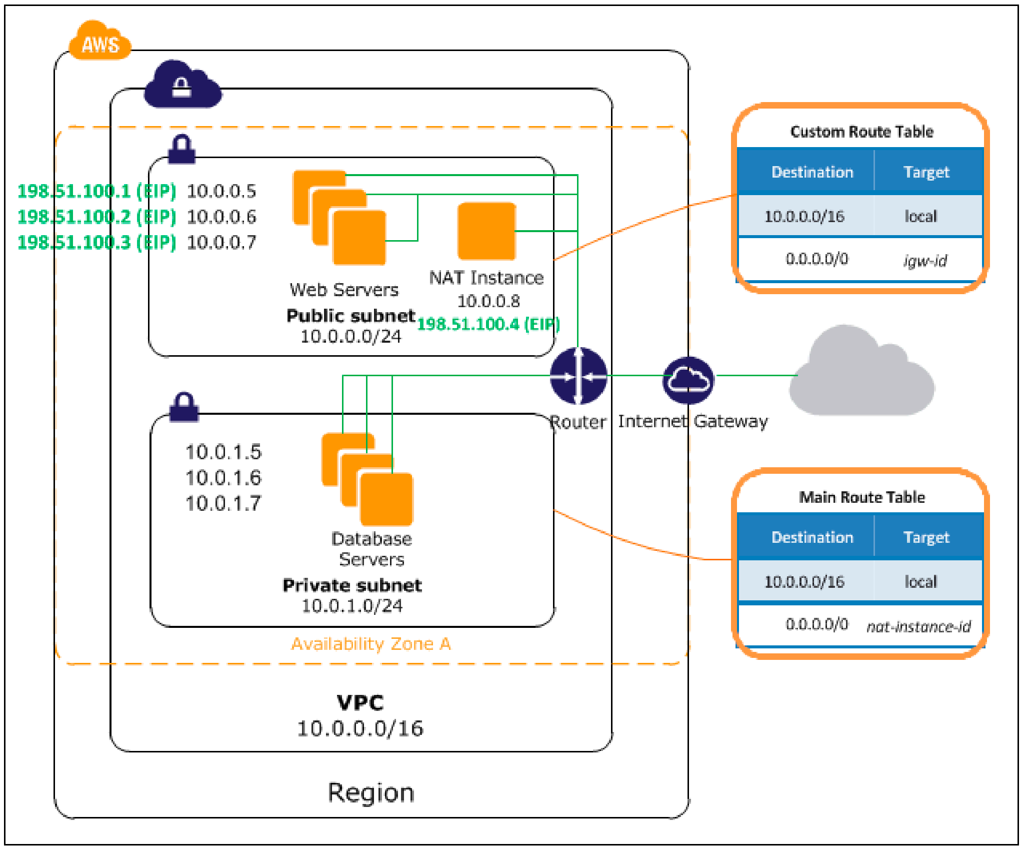
[25]http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/default-vpc.html

Getting Started with VPC

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.

The following diagram shows the key components of the configuration for this scenario. [26]



Custom Route Table

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	*igw-id*

Main Route Table

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	*nat-instance-id*

VPC Diagram

[26] http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Scenario2.html

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.

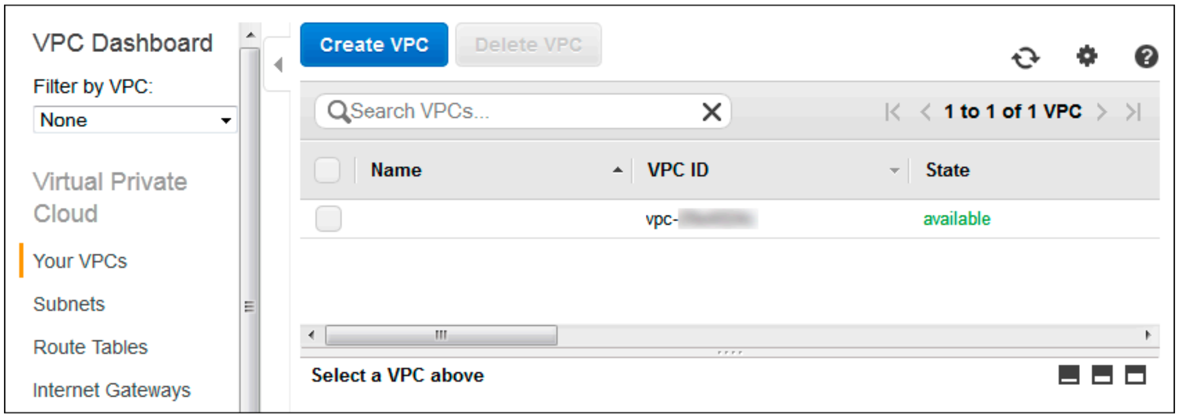
The following list describes the basic components presented in the configuration diagram for this scenario:

- 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 back-end servers that don't need to accept incoming traffic from the Internet; however, they can send requests to the Internet using the NAT instance (see the next bullet).
- A network address translation (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.

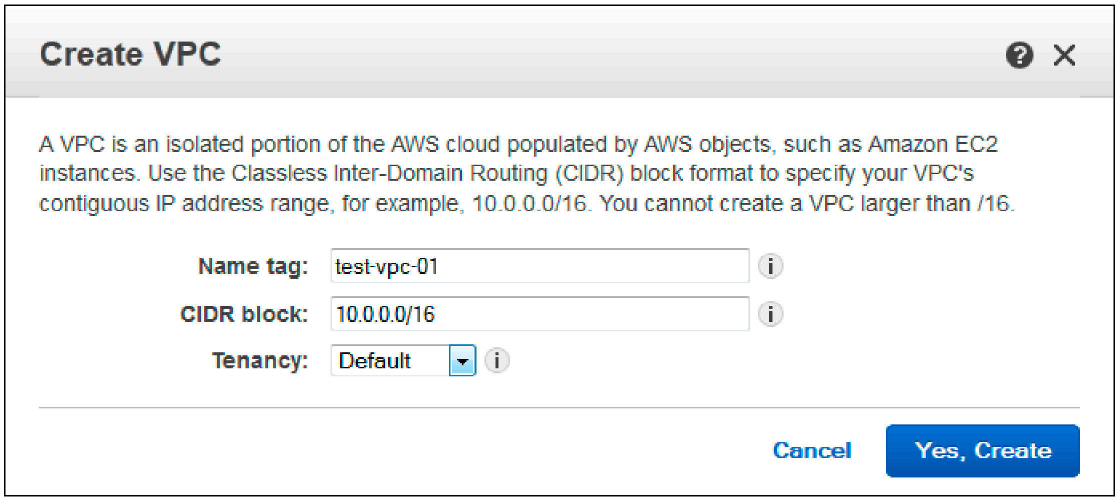
To create a VPC:

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**.



A VPC is an isolated portion of the AWS cloud populated by AWS objects, such as Amazon EC2 instances. Use the Classless Inter-Domain Routing (CIDR) block format to specify your VPC's contiguous IP address range, for example, 10.0.0.0/16. You cannot create a VPC larger than /16.

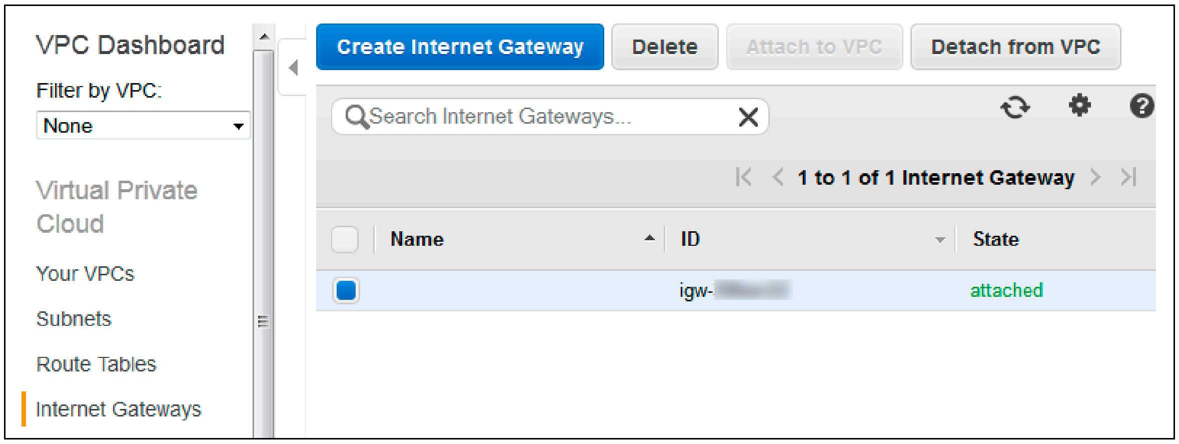
Name tag: test-vpc-01
CIDR block: 10.0.0.0/16
Tenancy: Default

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/.

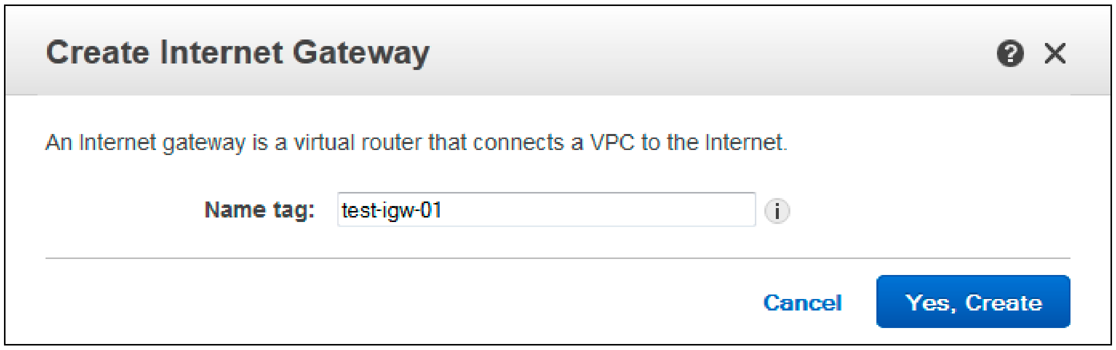
To create an Internet gateway and attach it to a VPC:

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



Name	ID	State
	igw-[redacted]	attached

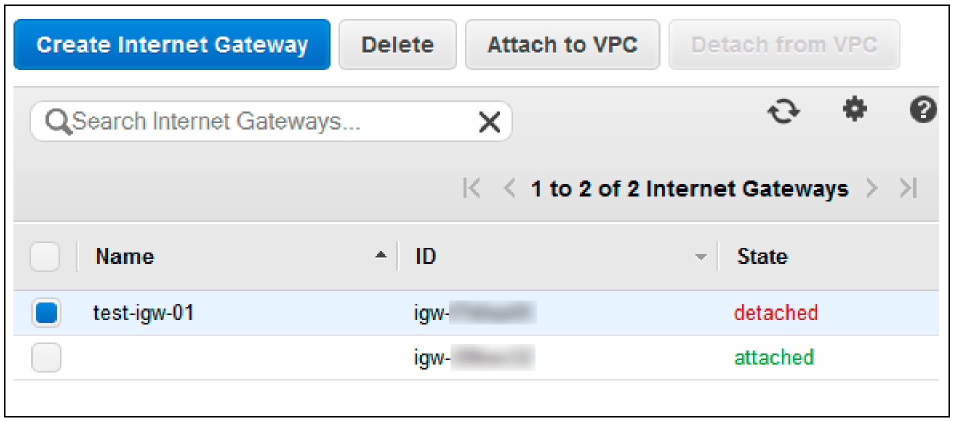
2. Click **Create Internet Gateway**.
3. Enter the desired **Name tag** for this Internet gateway.



An Internet gateway is a virtual router that connects a VPC to the Internet.

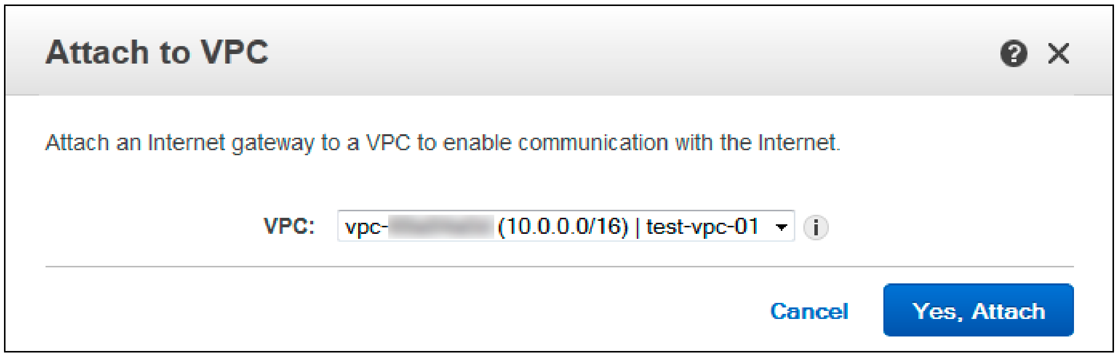
Name tag: test-igw-01

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



Name	ID	State
test-igw-01	igw-[redacted]	detached
	igw-[redacted]	attached

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



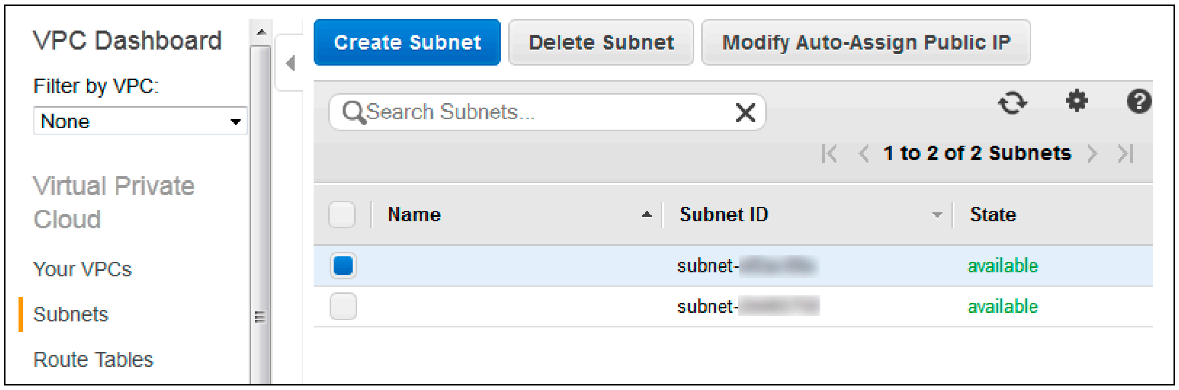
Attach to VPC

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

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

To create subnets on the VPC:

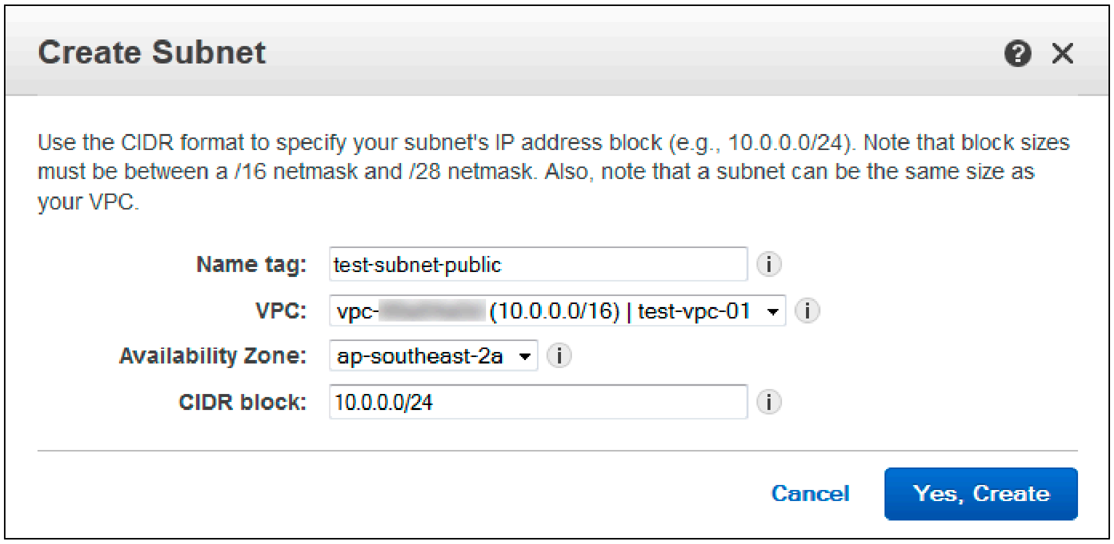
1. In the VPC Dashboard navigation pane, click **Subnets**.



1 to 2 of 2 Subnets

Name	Subnet ID	State
	subnet-[redacted]	available
	subnet-[redacted]	available

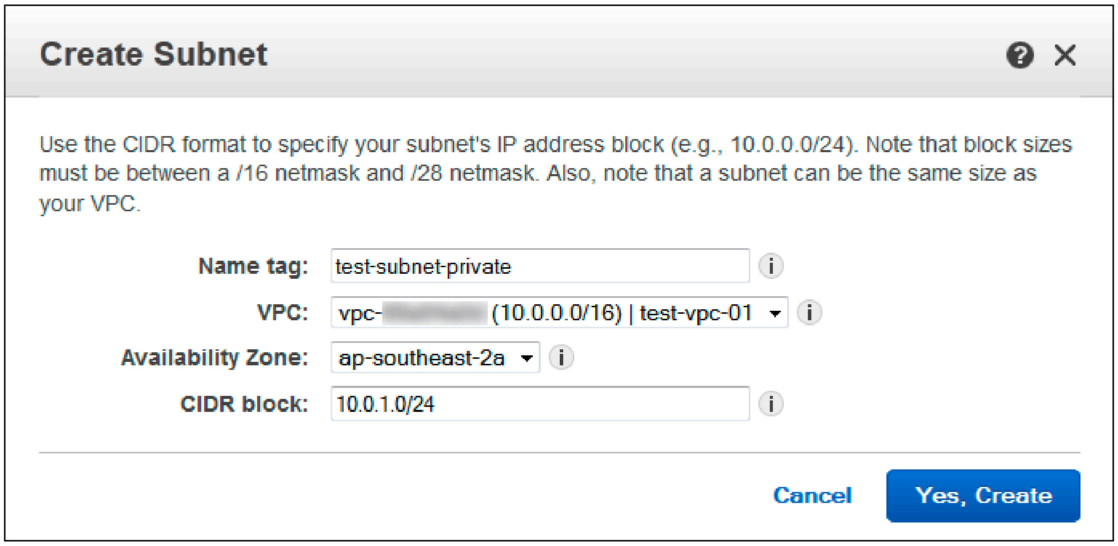
2. Click **Create Subnet**
3. Enter the desired **Name tag** and **CIDR block** for the public subnet, select VPC and Availability Zone.


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
VPC:	vpc-[redacted] (10.0.0.0/16) | test-vpc-01
Availability Zone:	ap-southeast-2a
CIDR block:	10.0.0.0/24

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
VPC:	vpc-[redacted] (10.0.0.0/16) | test-vpc-01
Availability Zone:	ap-southeast-2a
CIDR block:	10.0.1.0/24

6. Click **Yes, Create**.

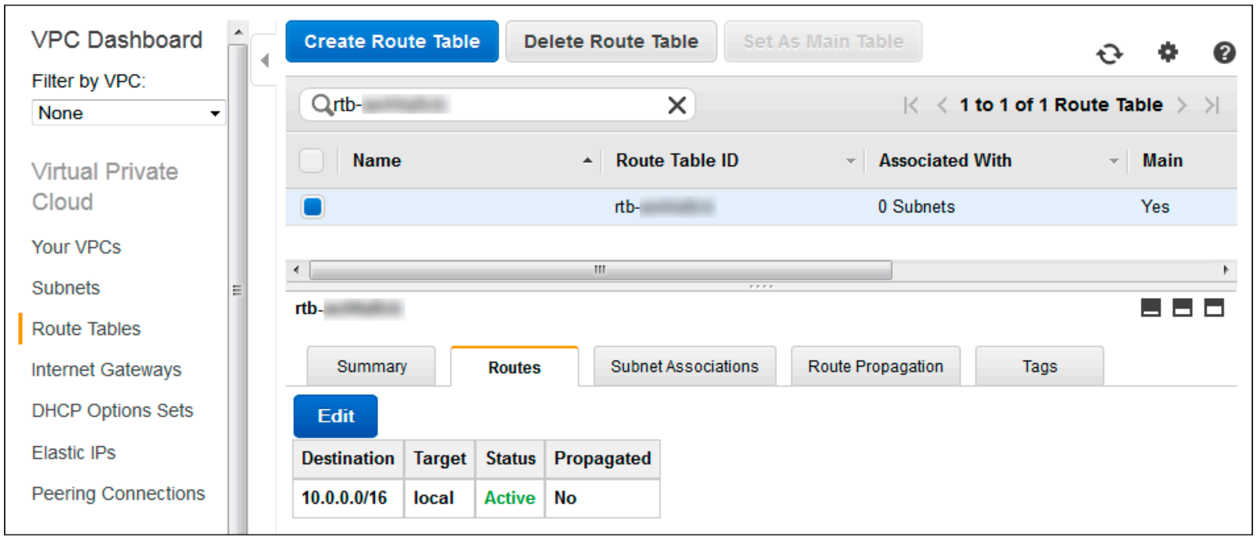
Route Tables

The following are the basic things that you need to know about route tables:[27]

- Your VPC has an implicit router.
- Your VPC automatically comes with a 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 a main route table. To see the main route tables for your VPC, in the VPC Dashboard navigation pane, click **Route Tables**.



Name	Route Table ID	Associated With	Main
	rtb-[redacted]	0 Subnets	Yes

Destination	Target	Status	Propagated
10.0.0.0/16	local	Active	No

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.

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

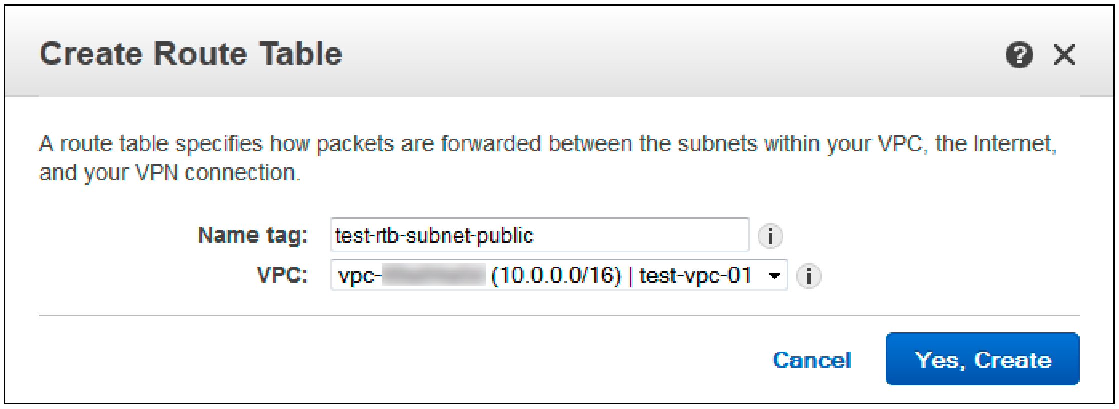
If you don't explicitly associate a subnet with a route table, the subnet is implicitly associated with the main route table. However, you can still explicitly associate a subnet with the main route table. You might do that if you change which table is the main route table. The console shows the number 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**.



Create Route Table

A route table specifies how packets are forwarded between the subnets within your VPC, the Internet, and your VPN connection.

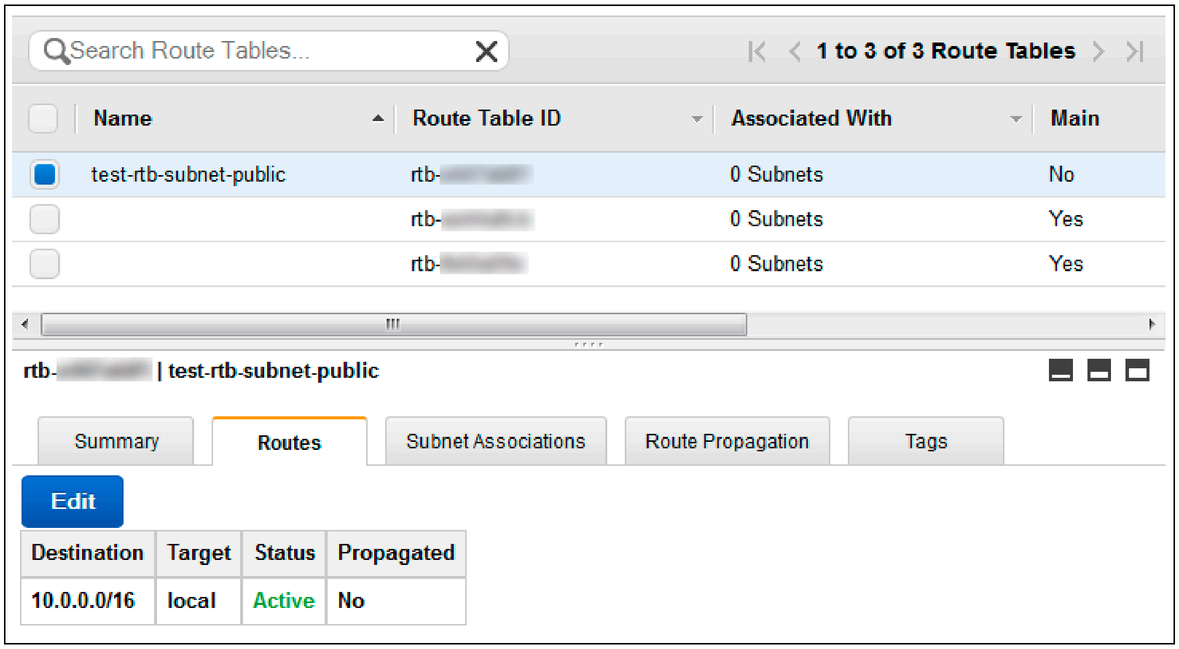
Name tag: test-rtb-subnet-public

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

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.

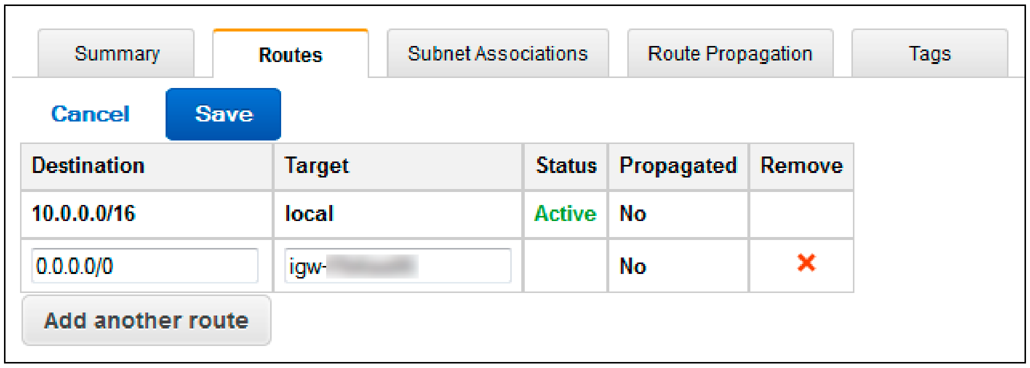


	Name	Route Table ID	Associated With	Main
🟦	test-rtb-subnet-public	rtb-[redacted]	0 Subnets	No
		rtb-[redacted]	0 Subnets	Yes
		rtb-[redacted]	0 Subnets	Yes

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

Destination	Target	Status	Propagated
10.0.0.0/16	local	Active	No

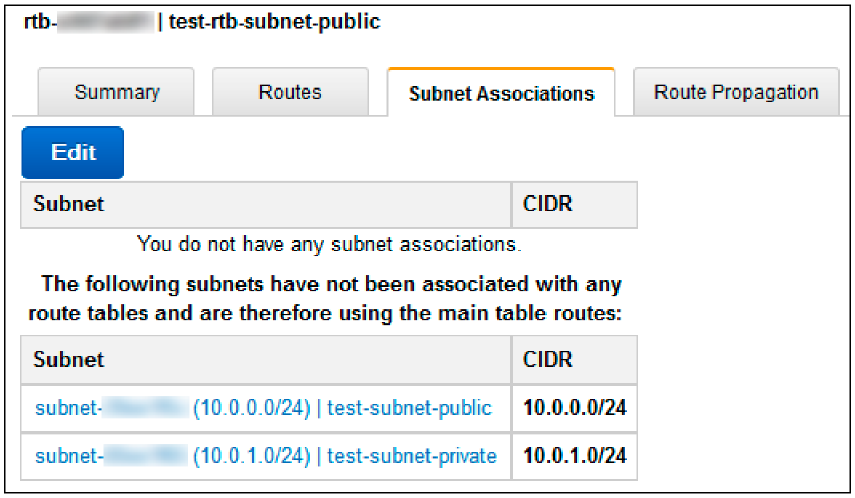
2. Click **Edit** button.
3. Enter `0.0.0.0/0` CIDR block in the **Destination** field and select the Internet gateway from the **Target** list.



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

4. Click **Save**.

5. Select **Subnet Associations** tab.



rtb- | test-rtb-subnet-public

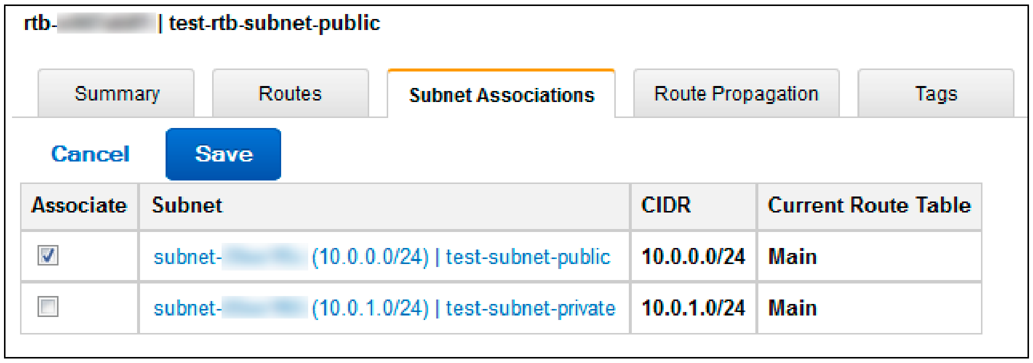
Subnet	CIDR

You do not have any subnet associations.

The following subnets have not been associated with any route tables and are therefore using the main table routes:

Subnet	CIDR
subnet- (10.0.0.0/24) | test-subnet-public	10.0.0.0/24
subnet- (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**.



rtb- | test-rtb-subnet-public

Associate	Subnet	CIDR	Current Route Table
☑	subnet- (10.0.0.0/24) | test-subnet-public	10.0.0.0/24	Main
☐	subnet- (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 Virtual Private Cloud (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
`aws_secret_key`	no			AWS secret key. If not set then the value of the `AWS_SECRET_KEY` environment variable is used
`cidr_block`	yes			The cidr block representing the VPC, e.g. 10.0.0.0/16
`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
`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
`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: [{ 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

parameter	required	default	choices	comments
				attached to the main table implicitly.
`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 /etc/ansible
# vi vpc_create.yml
```

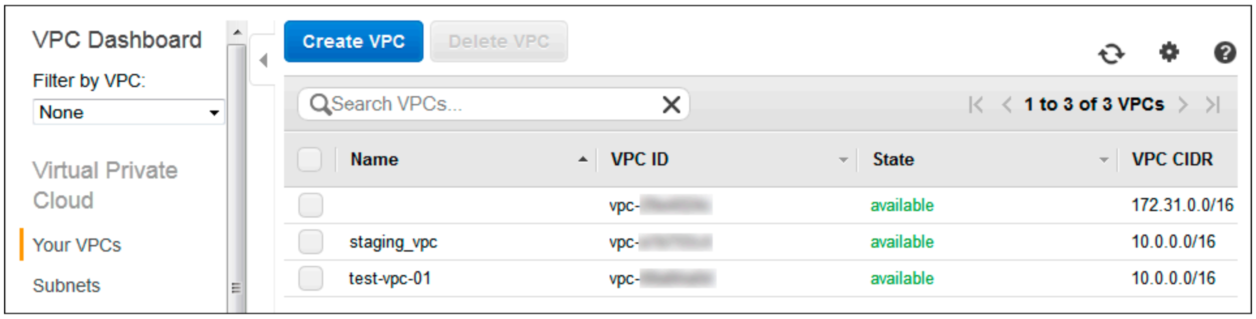
```
---
- hosts: localhost
  connection: local
  gather_facts: no
  vars:
    region: ap-southeast-2
    # prefix for naming
    prefix: staging
    # availability zone
    az: ap-southeast-2a
  tasks:
    - name: create vpc
      local_action:
        module: ec2_vpc
        region: "{{ region }}"
        cidr_block: 10.0.0.0/16
        resource_tags: '{"Name":"{{ prefix }}_vpc"}'
        subnets:
          - cidr: 10.0.0.0/24
            az: "{{ az }}"
            resource_tags: '{"Name":"{{ prefix }}_subnet_public"}'
          - cidr: 10.0.1.0/24
            az: "{{ az }}"
            resource_tags: '{"Name":"{{ prefix }}_subnet_private"}'
        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 }}_vpc_info file
      sudo: yes
      local_action: shell echo "{{ prefix }}"_vpc":" "{{ vpc.vpc_id }}"
                      > "{{ prefix }}"_vpc_info
    - name: write subnets id to {{ prefix }}_vpc_info file
      sudo: yes
      local_action: shell echo "{{ item.resource_tags.Name }}"":" "{{ item.id }}\
"
                      >> "{{ prefix }}"_vpc_info
      with_items: vpc.subnets
```

Run the playbook:

```
# ansible-playbook 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, etc simply by changing the `prefix` variable and run the playbook again.

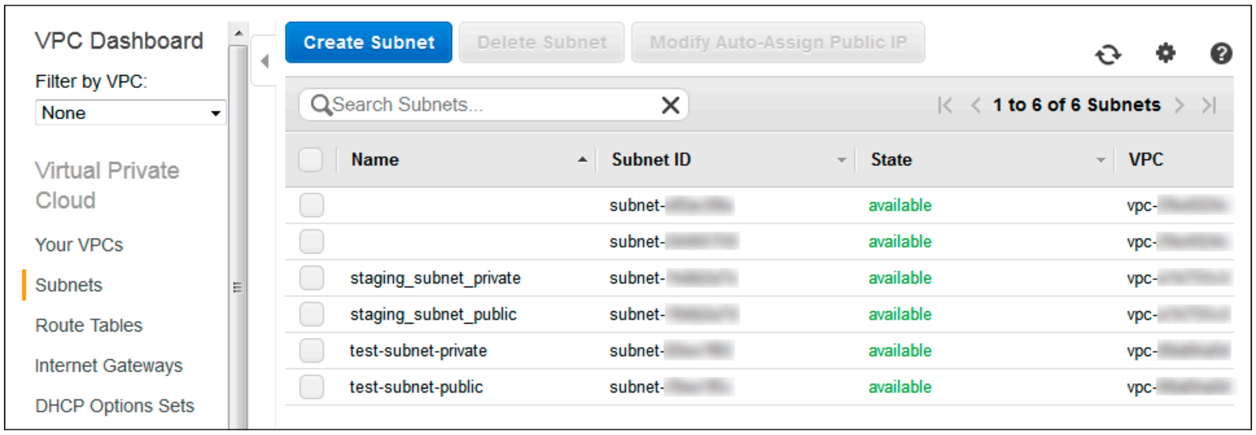
You should see the `staging_vpc` created and listed on the VPC console. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/ and select **Your VPCs** in the navigation pane.



Name	VPC ID	State	VPC CIDR
	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

staging_vpc

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



Name	Subnet ID	State	VPC
	subnet-[redacted]	available	vpc-[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]

New Subnets

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-xxxxxxxx
staging_subnet_public: subnet-xxxxxxxx
staging_subnet_private: subnet-xxxxxxxx
```

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
      local_action:
        module: 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. 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. [28]

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. To see the options for this module, please refer to the table in Chapter 3 - Security Groups section. 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.

[28] http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_SecurityGroups.html

- 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 /etc/ansible
# 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
      local_action:
        module: ec2_group
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        name: "{{ prefix }}_sg_web"
        description: security group for webservers
    - name: create empty security group for databases
      local_action:
        module: ec2_group
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        name: "{{ prefix }}_sg_database"
        description: security group for databases
    - name: create empty security group for nat
      local_action:
        module: ec2_group
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        name: "{{ prefix }}_sg_nat"
        description: security group for nat
```

Run the playbook:

```
# ansible-playbook 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
      local_action:
        module: ec2_group
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        #your security group name
        name: "{{ prefix }}_sg_web"
        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 }}_sg_database"
    - name: modify sg_database rules
      local_action:
        module: ec2_group
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        name: "{{ prefix }}_sg_database"
        description: security group for databases
        rules:
          - proto: tcp
            from_port: 3306
            to_port: 3306
            group_name: "{{ prefix }}_sg_web"
        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
      local_action:
        module: ec2_group
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
```

```
        name: "{{ prefix }}_sg_nat"
        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 }}_sg_web
      local_action:
        module: ec2_group
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        name: "{{ prefix }}_sg_web"
        description: security group for webservers
        state: absent
    - name: delete {{ prefix }}_sg_database
      local_action:
        module: ec2_group
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        name: "{{ prefix }}_sg_database"
        description: security group for databases
        state: absent
    - name: delete {{ prefix }}_sg_nat
      local_action:
        module: ec2_group
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        name: "{{ prefix }}_sg_nat"
        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 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
      local_action:
        module: ec2
        region: "{{ region }}"
        key_name: "{{ key }}"
        instance_type: "{{ instance_type }}"
        image: "{{ image }}"
        wait: yes
        group: "{{ prefix }}_sg_web"
        instance_tags:
          Name: "{{ prefix }}_web"
          class: web
          environment: staging
        id: web_launch_01
        vpc_subnet_id: "{{ staging_subnet_public }}"
      register: ec2
    - name: associate new EIP for the instance
```

```
      local_action:
        module: 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-apsydney
    instance_type: t2.micro
    image: ami-d9fe9be3
    prefix: staging
  tasks:
    - name: database instance provisioning
      local_action:
        module: ec2
        region: "{{ region }}"
        key_name: "{{ key }}"
        instance_type: "{{ instance_type }}"
        image: "{{ image }}"
        wait: yes
        group: "{{ prefix }}_sg_database"
        instance_tags:
          Name: "{{ prefix }}_database"
          class: database
          environment: staging
        id: db_launch_01
        vpc_subnet_id: "{{ staging_subnet_private }}"
        assign_public_ip: no
```

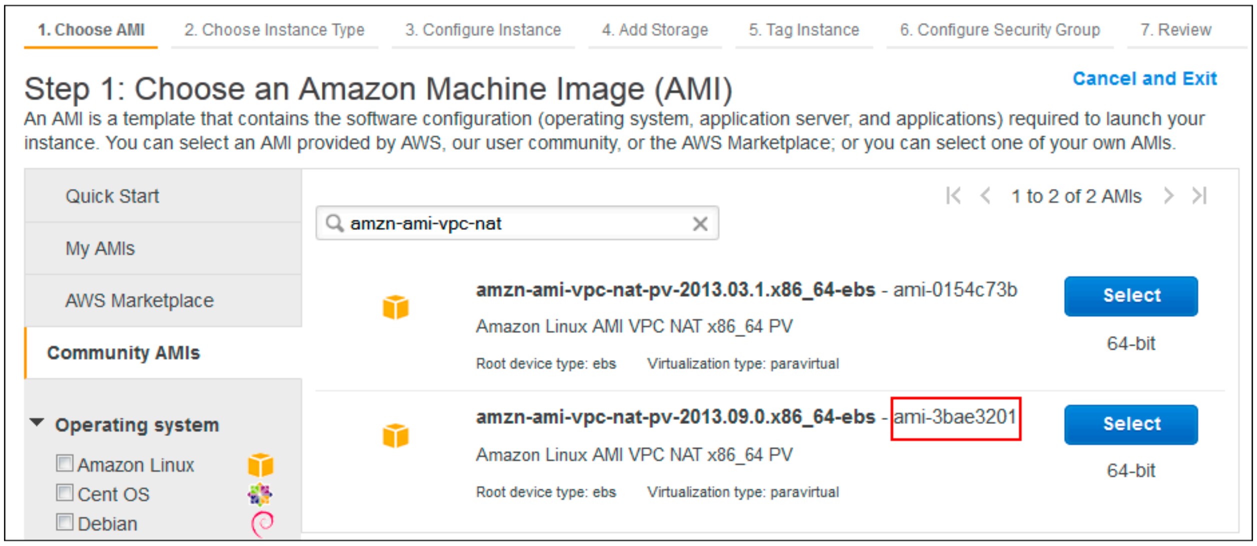
NAT Instance

Instances that you launch into a private subnet in a VPC can't communicate with the Internet. You can optionally use a network address translation (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. [29]

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.

amzn-ami-vpc-nat

1 to 2 of 2 AMIs

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

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

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
```

[29] http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_NAT_Instance.html

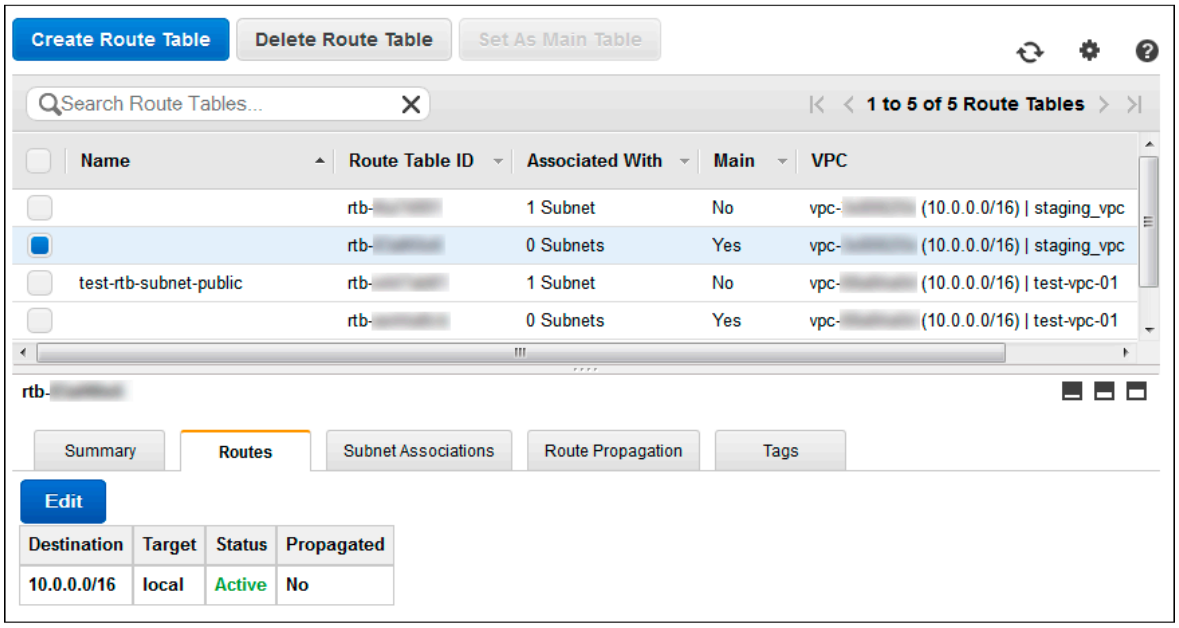
```
---
- hosts: localhost
  connection: local
  gather_facts: no
  vars_files:
    - staging_vpc_info
  vars:
    region: ap-southeast-2
    key: yan-key-pair-apsydney
    instance_type: t1.micro
    image: ami-3bae3201
    prefix: staging
  tasks:
    - name: NAT instance provisioning
      local_action:
        module: ec2
        region: "{{ region }}"
        key_name: "{{ key }}"
        instance_type: "{{ instance_type }}"
        image: "{{ image }}"
        wait: yes
        group: "{{ prefix }}_sg_nat"
        instance_tags:
          Name: "{{ prefix }}_nat"
          class: nat
          environment: staging
        id: nat_launch_01
        vpc_subnet_id: "{{ staging_subnet_public }}"
        source_dest_check: no
        wait: yes
      register: ec2
    - name: associate new EIP for the instance
      tags: eip
      local_action:
        module: 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.

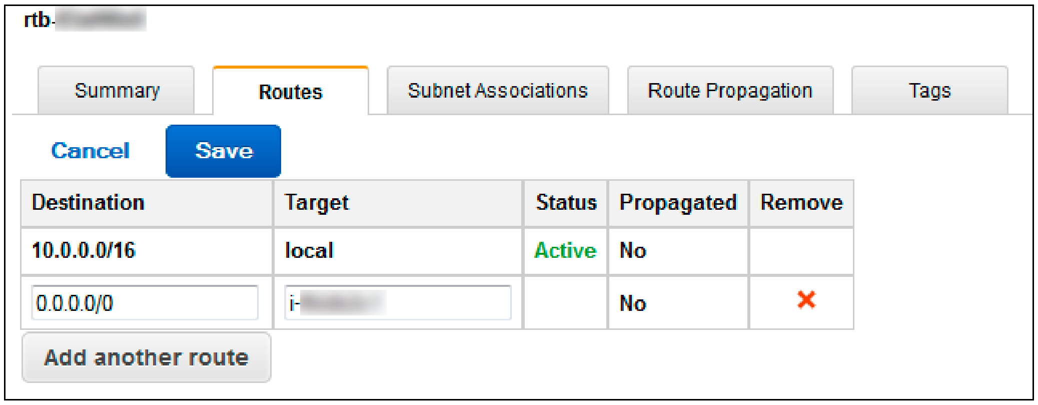


Name	Route Table ID	Associated With	Main	VPC
	rtb-[redacted]	1 Subnet	No	vpc-[redacted] (10.0.0.0/16) | staging_vpc
	rtb-[redacted]	0 Subnets	Yes	vpc-[redacted] (10.0.0.0/16) | staging_vpc
test-rtb-subnet-public	rtb-[redacted]	1 Subnet	No	vpc-[redacted] (10.0.0.0/16) | test-vpc-01
	rtb-[redacted]	0 Subnets	Yes	vpc-[redacted] (10.0.0.0/16) | test-vpc-01

rtb-[redacted]

Destination	Target	Status	Propagated
10.0.0.0/16	local	Active	No

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.



rtb-[redacted]

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

5. Click **Save**

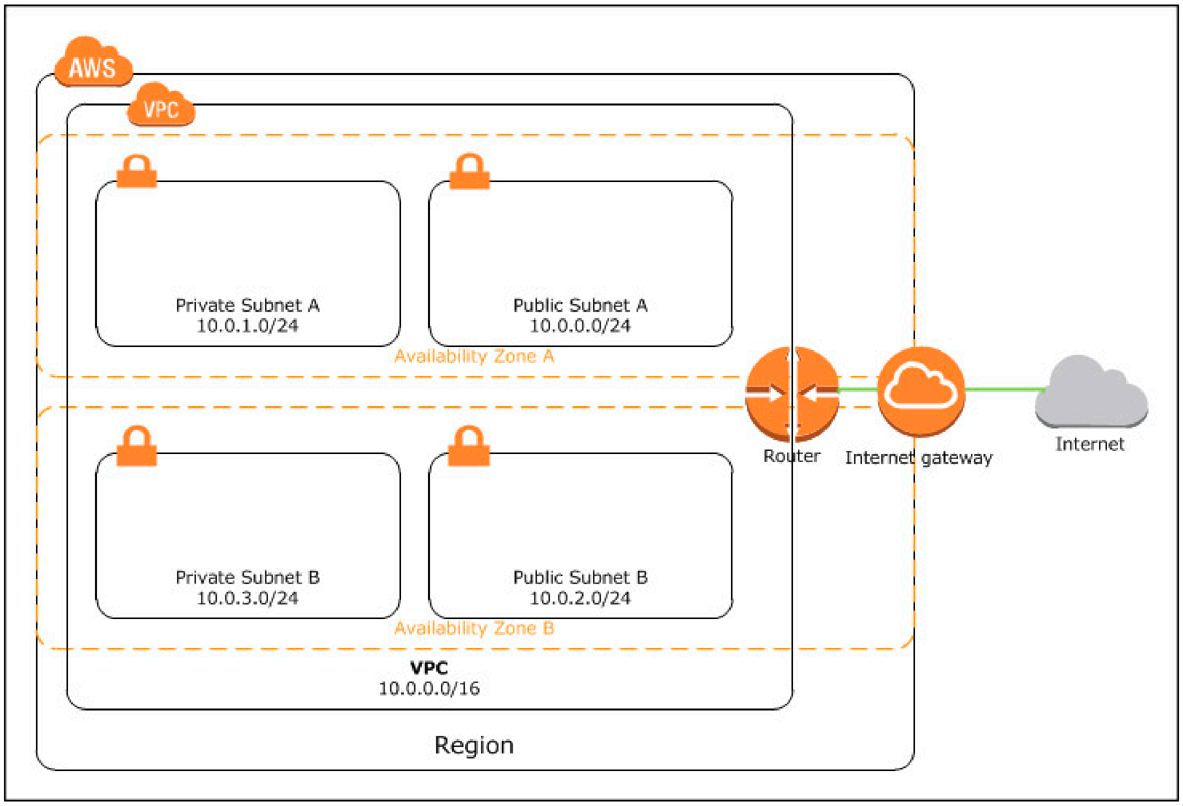
Now we have completed the VPC infrastructure provisioning using Ansible. At the time of writing, there is not yet a module for *Network ACLs* provisioning. If you want to add some Network ACLs for your VPC subnet, you could do it from the VPC console: select **Network ACLs** in the navigation pane and 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 (AZ) 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 /etc/ansible
# vi vpc_create_multi_az.yml
```

```
---
- hosts: localhost
  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
      local_action:
        module: 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 }}"
            resource_tags: '{"Name":"{{ prefix }}_subnet_public_0"}'
          - cidr: 10.0.1.0/24
            az: "{{ az0 }}"
            resource_tags: '{"Name":"{{ prefix }}_subnet_private_0"}'
          - cidr: 10.0.2.0/24
            az: "{{ az1 }}"
            resource_tags: '{"Name":"{{ prefix }}_subnet_public_1"}'
          - cidr: 10.0.3.0/24
            az: "{{ az1 }}"
            resource_tags: '{"Name":"{{ prefix }}_subnet_private_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 }}_vpc_info file
      sudo: yes
      local_action: shell echo "{{ prefix }}"_vpc":" "{{ vpc.vpc_id }}"
```

```
                        > "{{ prefix }}"_vpc_info
    - name: write subnets id to {{ prefix }}_vpc_info file
      sudo: yes
      local_action: shell echo "{{ item.resource_tags.Name }}"":" "{{ item.id }}"
                        >> "{{ prefix }}"_vpc_info
      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`.

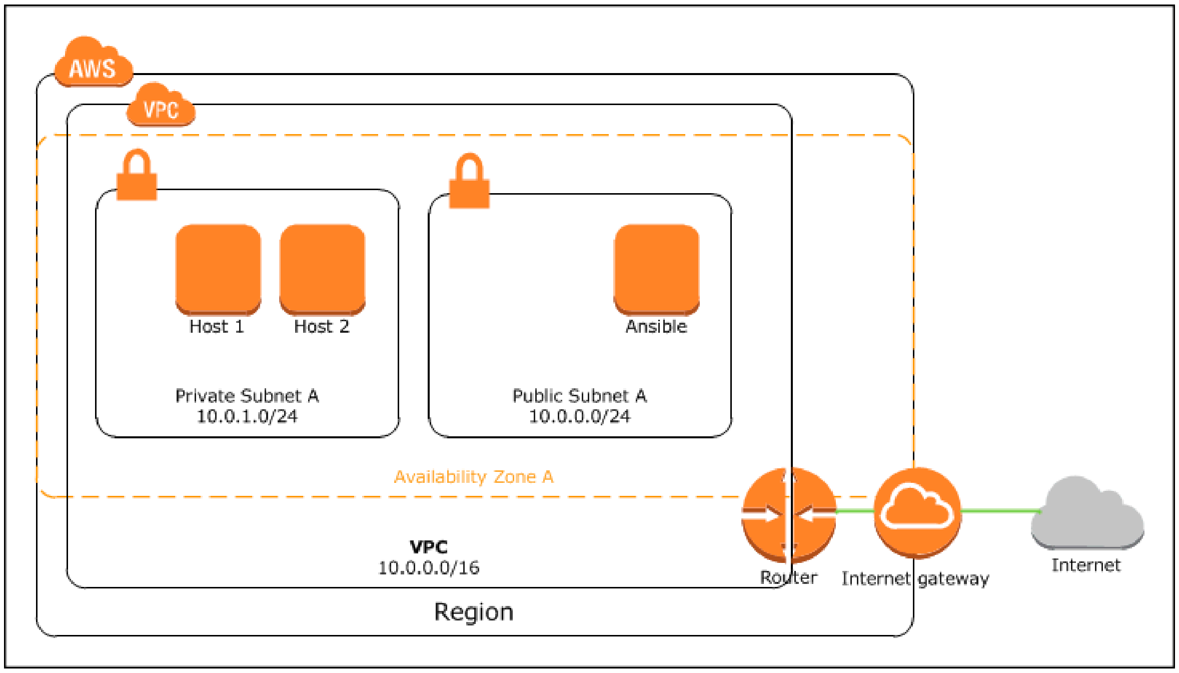
Run `sg_empty.yml` playbook, and then `sg_modify.yml` playbook to re-create our VPC security groups.

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 RDS (Relational Database Service), deployed in 2 (or more) availability zones (`staging_subnet_private_0` and `staging_subnet_private_1`).

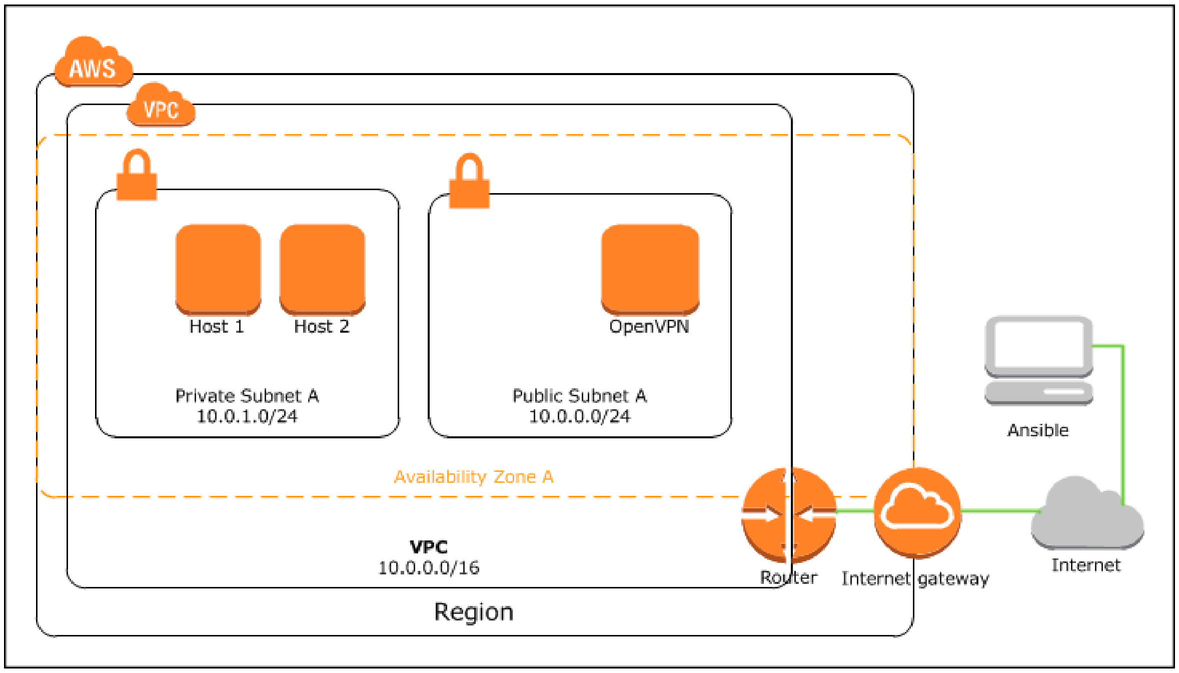
Ansible in VPC

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 VPN (Virtual Private Network) 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 /etc/ansible
# vi sg_jumpbox.yml
```

```
---
- hosts: localhost
  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
```

```
      local_action:
        module: ec2_group
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        #your security group name
        name: "{{ prefix }}_sg_jumpbox"
        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 sg_jumpbox.yml
```

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

```
# vi ec2_vpc_jumpbox.yml
```

```
---
- hosts: localhost
  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
    vpc_subnet_id: "{{ staging_subnet_public_0 }}"
  tasks:
    - name: jump box instance provisioning
      local_action:
        module: ec2
        region: "{{ region }}"
        key_name: "{{ key }}"
        instance_type: "{{ instance_type }}"
        image: "{{ image }}"
        wait: yes
        group: "{{ prefix }}_sg_jumpbox"
        instance_tags:
          Name: "{{ prefix }}_jumpbox"
          class: jumpbox
          environment: "{{ prefix }}"
        id: jumpbox_launch_01
        vpc_subnet_id: "{{ vpc_subnet_id }}"
      register: ec2
    - name: associate new EIP for the instance
      local_action:
        module: 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
  sudo: 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
      local_action:
        module: ec2_group
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        name: "{{ prefix }}_sg_database"
        description: security group for databases
        rules:
          # allow ssh from the jump box
          - proto: tcp
            from_port: 22
            to_port: 22
            group_name: "{{ prefix }}_sg_jumpbox"
          # allow mysql access from web servers
          - proto: tcp
            from_port: 3306
            to_port: 3306
            group_name: "{{ prefix }}_sg_web"
        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

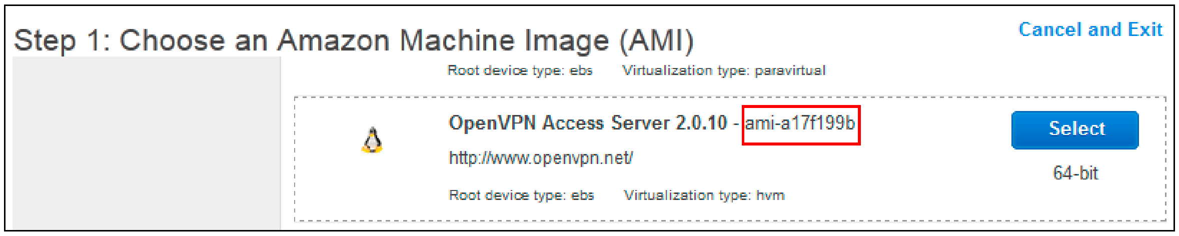
This section will show you how to launch an OpenVPN EC2 instance in the public subnet using Ansible, and configure the server from its web UI.

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.[30]

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.


Step 1: Choose an Amazon Machine Image (AMI)
Cancel and Exit
Root device type: ebs Virtualization type: paravirtual
OpenVPN Access Server 2.0.10 - ami-a17f199b
http://www.openvpn.net/
Root device type: ebs Virtualization type: hvm
Select
64-bit

5. Select **Cancel and Exit**.

Create the security group for our OpenVPN server:

```
# vi sg_openvpn.yml
```

[30] http://openvpn.net/index.php/access-server/overview.html

```
---
- hosts: localhost
  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
      local_action:
        module: ec2_group
        region: "{{ region }}"
        vpc_id: "{{ vpc_id }}"
        #your security group name
        name: "{{ prefix }}_sg_openvpn"
        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 sg_openvpn.yml
```

The following playbook will launch our OpenVPN server instance in public subnet A:

```
# vi ec2_vpc_openvpn.yml
```

```
---
- hosts: localhost
  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
      local_action:
        module: ec2
        region: "{{ region }}"
        key_name: "{{ key }}"
        instance_type: "{{ instance_type }}"
        image: "{{ image }}"
        source_dest_check: no
        wait: yes
        group: "{{ prefix }}_sg_openvpn"
        instance_tags:
          Name: "{{ prefix }}_openvpn"
          class: openvpn
          environment: "{{ prefix }}"
        id: openvpn_launch_01
        vpc_subnet_id: "{{ vpc_subnet_id }}"
      register: ec2
    - name: associate new EIP for the instance
      local_action:
        module: 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 `sudo 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`.
   Static IP Address Network: (leave empty)
   Group Default IP Address Network: (leave empty)
4. Click **Save settings**.
5. 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 spesific 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 /etc/ansible
# 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 VPCConnection
    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
      local_action:
        module: vpc_lookup
        region: "{{ region }}"
        tags:
          Name: test-vpc
      register: vpc

    - name: delete vpc
      local_action:
        module: 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`.