

Lucas Chan, Rowan Udell

# AWS Administration Cookbook

Harness the full capability of AWS



Packt



# Title Page

## AWS Administration Cookbook

Harness the full capability of AWS

Lucas Chan  
Rowan Udell

**Packt**

BIRMINGHAM - MUMBAI



# AWS Administration Cookbook

Copyright © 2017 Packt Publishing

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

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

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

First published: April 2017

Production reference: 1140417

Published by Packt Publishing Ltd.  
Livery Place  
35 Livery Street  
Birmingham  
B3 2PB, UK.

ISBN 978-1-78712-763-0

[www.packtpub.com](http://www.packtpub.com)



# Credits

<b>Authors</b>	<b>Copy Editors</b>
Lucas Chan Rowan Udell	Safis Editing Madhusudan Uchil
<b>Reviewer</b>	<b>Project Coordinator</b>
Michael Kelly	Virginia Dias
<b>Commissioning Editor</b>	<b>Proofreader</b>
Kartikey Pandey	Safis Editing
<b>Acquisition Editor</b>	<b>Indexer</b>
Meeta Rajani	Tejal Daruwale Soni
<b>Content Development Editor</b>	<b>Graphics</b>
Sweeny Dias	Kirk D'Penha
<b>Technical Editor</b>	<b>Production Coordinator</b>
Khushbu Sutar	Nilesh Mohite



# About the Authors

**Lucas Chan** has been working in tech since 1995 in a variety of development, systems admin, and DevOps roles. He is currently a senior consultant and engineer at Versent and technical director at Stax. He's been running production workloads on AWS for over 10 years. He's also a member of the APAC AWS warriors program and holds all five of the available AWS certifications.

*I'd like to thank Rowan Udell and everyone at Packt Publishing for giving me the opportunity to write my first book! Dr. Michael Kelly, for graciously offering his spare time to review and critique our work. My mum, dad and sister, for putting up with my absence and occasional crankiness for the past few months. Hannah, for simply being an all-around amazing human. Damian Wilson, who gave me my first tech job way back in 1995: I wouldn't be where I am today without the opportunities he gave me. Thor Essman and James Coxon, who created the amazing company that we both work for and whose support knows no bounds. Trang, for keeping me sane and occasionally dragging me to the beach when I needed a break from writing. Lastly, a shout out to those who suffer from vestibular migraines—may you walk with a steady foot, see the world with clarity and hear with grace and harmony.*

**Rowan Udell** has been working in development and operations for 15 years. He has held a variety of positions, such as SRE, frontend developer, backend developer, consultant, technical lead, and team leader. His travels have seen him work in start-ups and enterprises in the finance, education, and web industries in Australia and Canada. He currently works as a senior consultant with Versent, an AWS Advanced Partner in Sydney. He specializes in serverless applications and architectures on AWS, and contributes actively in the Serverless Framework community. He regularly blogs at <http://blog.rowanudell.com>.

*Firstly I'd like to thank my partner, Marie-Pier, for making this book possible for me—having two small children does not make writing a book easy! Thanks to my family for all their encouragement over the years. To all my colleagues and peers at Versent, thank you for setting the bar so high and making a great company to work for. Much thanks to our colleague Dr. Michael Kelly for reviewing the book, and the team at Packt for their support. Finally, thanks to Lucas for the great work, and picking up my slack.*



# About the Reviewer

**Michael Kelly** is a DevOps engineer and consultant in the financial services industry at Versent in Sydney. In his past, Michael worked as a developer and DevOps engineer at start-ups, developing Infrastructure as Code solutions. Michael is AWS certified and holds a PhD in computer science. In his downtime, Michael also blogs about cloud solutions at <https://blog.ashiny.cloud>.

*I'd like to thank Rowan and Lucas for letting me be a part of this. It has really come together to be an excellent book—congratulations. To Anna and my family, thank you for always being there whenever I have needed you. To everyone at Versent, I am very proud to be in the company of such skilled craftsmen and women who put quality into everything they do. Thank you.*



# **www.PacktPub.com**

For support files and downloads related to your book, please visit [www.PacktPub.com](http://www.PacktPub.com).

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

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



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

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

## Why subscribe?

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



# Customer Feedback

Thanks for purchasing this Packt book. At Packt, quality is at the heart of our editorial process. To help us improve, please leave us an honest review on this book's Amazon page at <https://www.amazon.com/dp/178712763X>.

If you'd like to join our team of regular reviewers, you can e-mail us at [customerreviews@packtpub.com](mailto:customerreviews@packtpub.com). We award our regular reviewers with free eBooks and videos in exchange for their valuable feedback. Help us be relentless in improving our products!



# Table of Contents

## Preface

[What this book covers](#)

[What you need for this book](#)

[Who this book is for](#)

## Sections

[Getting ready](#)

[How to do it...](#)

[How it works...](#)

[There's more...](#)

[See also](#)

## Conventions

[Reader feedback](#)

[Customer support](#)

[Downloading the example code](#)

[Errata](#)

[Piracy](#)

[Questions](#)

## 1. AWS Fundamentals

[Introduction](#)

[Creating an account](#)

[Regions and Availability Zones](#)

[The AWS web console](#)

[CloudFormation templates](#)

[Infrastructure as Code](#)

[Visibility](#)

[Consistency](#)

[Troubleshooting](#)

[Scale](#)

[Costs](#)

DevOps

Server configuration

IaC on AWS

CloudFormation

What is CloudFormation?

Why is CloudFormation important?

The layer cake

CloudFormation templates

YAML versus JSON

A closer look at CloudFormation templates

Parameters

Resources

Outputs

Mappings

Dependencies and ordering

Functions

Fn::Join

Fn::Sub

Conditionals

Permissions and service roles

Custom resources

Cross-stack references

Updating resources

Change sets

Other things to know

Name collisions

Rollback

Limits

Circular dependencies

DSLs and generators

Credentials

## Stack policies

The command-line interface tool

Installation

Upgrade

Configuration

Default profile

Named profiles

Environment variables

Instance roles

Usage

Commands

Subcommands

Options

Output

JSON

Table

Text

Querying

Generate CLI skeleton

Input

Output

Pagination

Autocomplete

Related tools

jq

## 2. Managing AWS Accounts

Introduction

Setting up a master account

How to do it...

How it works...

There's more...

Multi-factor authentication

Using the CLI

[See also](#)

[Creating a member account](#)

[Getting ready](#)

[How to do it...](#)

[How it works...](#)

[There's more...](#)

[Accessing the member account](#)

[Service control policies](#)

[Root credentials](#)

[Deleting accounts](#)

[See also](#)

[Inviting an account](#)

[Getting ready](#)

[How to do it...](#)

[How it works...](#)

[There's more...](#)

[Removing accounts](#)

[Consolidated billing](#)

[See also](#)

[Managing your accounts](#)

[Getting ready](#)

[How to do it...](#)

[Getting the root ID for your organization](#)

[Creating an OU](#)

[Getting the ID of an OU](#)

[Adding an account to an OU](#)

[Removing an account from an OU](#)

[Deleting an OU](#)

[How it works...](#)

[There's more...](#)

[See also](#)

[Adding a service control policy](#)

[Getting ready](#)

[How to do it...](#)

[How it works...](#)

[There's more...](#)

[See also](#)

### [3. Storage and Content Delivery](#)

[Introduction](#)

[Storage](#)

[Elastic Block Store](#)

[Elastic File System](#)

[Simple Storage Service](#)

[Glacier](#)

[Content delivery](#)

[Hosting a static website](#)

[How to do it...](#)

[Creating S3 buckets and hosting content](#)

[Creating a hosted zone](#)

[Creating DNS records](#)

[Uploading website content](#)

[How it works...](#)

[There's more...](#)

[Delegating your domain to AWS](#)

[Cross-origin resource sharing](#)

[Caching a website](#)

[Getting ready](#)

[About dynamic content](#)

[Configuring CloudFront distributions](#)

[How to do it...](#)

[Working with network storage](#)

[Getting ready](#)

[How to do it...](#)

[How it works...](#)

[There's more...](#)

Backing up data for compliance  
How to do it...

How it works...

#### 4. Using AWS Compute

##### Introduction

Creating a key pair  
Getting ready

How to do it...

How it works...

Launching an instance  
Getting ready

How to do it...

How it works...

There's more...

See also

Attaching storage  
Getting ready

How to do it...

How it works...

See also

Securely accessing private instances  
Getting ready

How to do it...  
Configuration

How it works...

There's more...

Auto scaling an application server  
Getting ready

How to do it...

How it works...  
Scaling policies

Alarms

Creating machine images  
Getting ready

How to do it...

How it works...

Template

Validate the template

Build the AMI

There's more...

Debugging

Orphaned resources

Deregistering AMIs

Other platforms

Creating security groups

Getting ready

How to do it...

There's more...

Differences from traditional firewalls

Creating a load balancer

How to do it...

How it works...

There's more...

HTTPS/SSL

Path-based routing

## 5. Management Tools

Introduction

Auditing your AWS account

How to do it...

How it works...

There's more...

Recommendations with Trusted Advisor

How to do it...

How it works...

There's more...

Creating e-mail alarms

How to do it...

How it works...

There's more...

Existing topics

Other subscriptions

See also

[Publishing custom metrics in CloudWatch](#)

Getting ready

How to do it...

How it works...

There's more...

Cron

[Auto scaling](#)

[Backfilling](#)

See also

[Creating monitoring dashboards](#)

Getting ready

How to do it...

There's more...

Widget types

See also

[Creating a budget](#)

Getting ready

How to do it...

How it works...

[Feeding log files into CloudWatch logs](#)

Getting ready

How to do it...

How it works...

There's more...

## 6. Database Services

Introduction

[Creating a database with automatic failover](#)

Getting ready

How to do it...

How it works...

There's more...

[Creating a NAT gateway](#)

Getting ready

How to do it...

How it works...

See also

[Creating a database read-replica](#)

Getting ready

How to do it...

How it works...

There's more...

[Promoting a read-replica to master](#)

Getting ready

How to do it...

[Creating a one-time database backup](#)

Getting ready

How to do it...

[Restoring a database from a snapshot](#)

Getting ready

How to do it...

There's more...

[Migrating a database](#)

Getting ready

How to do it...

How it works...

There's more...

Database engines

Ongoing replication

Multi-AZ

[Calculating DyanmoDB performance](#)

Getting ready

How to do it...

How it works...

There's more...

Burst capacity

Metrics

Eventually consistent reads

## 7. Networking

Introduction

Building a secure network

Getting ready

How to do it...

How it works...

There's more...

See also...

Creating a NAT gateway

Getting ready

How to do it...

How it works...

See also

Canary deployment via DNS

Getting ready

How to do it...

How it works...

Hosting a domain

Getting ready

How to do it...

How it works...

There's more...

See also...

Routing based on location with failover

Getting ready

How to do it...

How it works...

Normal operation (geolocation routing)

[Region A failure](#)

[Region B failure](#)

[There's more...](#)

[See also...](#)

[Network logging and troubleshooting](#)

[Getting ready](#)

[How to do it...](#)

[How it works...](#)

[There's more...](#)

[Log format](#)

[Updates](#)

[Omissions](#)

[See also](#)

## 8. Security and Identity

[Introduction](#)

[Federating with your AWS account](#)

[Getting ready](#)

[How to do it...](#)

[Active Directory configuration](#)

[Auth Account policy configuration](#)

[Auth Account role configuration](#)

[Simple AD configuration](#)

[App Account role configuration](#)

[How it works...](#)

[There's more...](#)

[See also](#)

[Creating SSL certificates](#)

[How to do it...](#)

[How it works...](#)

[There's more...](#)

[EC2 instances](#)

[Importing certificates](#)

[Active Directory as a service](#)

[Getting ready](#)

[How to do it...](#)

[How it works...](#)

[There's more...](#)

[See also](#)

[Creating users](#)

[Getting ready](#)

[How to do it...](#)

[How it works...](#)

[See also](#)

[Creating instance roles](#)

[How to do it...](#)

[How it works...](#)

[There's more...](#)

[See also](#)

[Cross-account user roles](#)

[Getting ready](#)

[How to do it...](#)

[How it works...](#)

[There's more...](#)

[AWS CLI profiles](#)

[See also](#)

[Storing secrets](#)

[Getting ready](#)

[How to do it...](#)

[How it works...](#)

[There's more...](#)

[Key aliases](#)

[Secret reader role](#)

[Secret writer role](#)

## The put-file command

### Versioning

See also

## 9. Estimating Costs

Introduction

Calculating costs

Getting ready

How to do it...

How it works...

See also

Estimating CloudFormation template costs

Getting ready

How to do it...

How it works...

See also

Purchasing reserved instances

Getting ready

How to do it...

How it works...

There's more...

Estimating total cost of ownership

Getting ready

How to do it...

How it works...

There's more...

See also



# Preface

The AWS platform is growing at a rapid rate, and it's being increasingly adopted across all industries and sectors. As the saying goes, friends don't let friends build data centers. No matter how you look at it, the model of pay-as-you-go compute, network, and storage is here to stay. It's also becoming increasingly hard to argue against *standing on the shoulders of giants*, especially when you look the rate with which features and enhancements are added to the AWS platform compared to what you'd typically get out of other cloud providers or a so-called *private* cloud.

We work with a lot of technical professionals who are highly knowledgeable in their domain, but often completely new to the AWS platform. Alternatively, they might be familiar with AWS but new to automation and infrastructure code practices.

We wanted to write a book for these people.

This book is intended to kick-start your AWS journey by providing recipes, patterns, and best practices across the areas we are often asked to help with on our consulting engagements. All the recipes and recommendations contained in this book are based on our personal experiences and observations from our time helping customers on the AWS platform.

CloudFormation is the AWS-native method for automating the (repeatable and reliable) deployment of AWS resources, and we use it extensively throughout this book. The recipes that follow will help you get well acquainted with CloudFormation, and you'll soon be on your way to customizing and building your own templates. With so much power at your fingertips, there's a lot of potential for finding yourself in a rabbit hole. This book aims to steer you in the right direction and help you adopt the platform in a sustainable and maintainable way.



# What this book covers

[Chapter 1](#), *AWS Fundamentals*, is an overview of Infrastructure as Code, CloudFormation, and the AWS CLI tools.

[Chapter 2](#), *Managing AWS Accounts*, covers everything you need to know to manage your accounts and get started with AWS organizations.

[Chapter 3](#), *Storage and Content Delivery*, shows you how to back up your data and serve file objects to your users.

[Chapter 4](#), *Using AWS Compute*, dives deep into how to run VMs (EC2 instances) on AWS, how to auto scale them, and how to create and manage load balancers.

[Chapter 5](#), *Management Tools*, provides an overview of how to audit your account and monitor your infrastructure.

[Chapter 6](#), *Database Services*, shows you how to create, manage, and scale databases on the AWS platform.

[Chapter 7](#), *Networking*, introduces private networks, routing, and DNS.

[Chapter 8](#), *Security and Identity*, offers advice and practical solutions for managing identities and role-based access.

[Chapter 9](#), *Estimating Costs*, provides an overview of how to estimate your spend on the AWS platform as well as how to reduce your costs by purchasing reserved instance capacity.



# What you need for this book

The recipes in this book show you how to deploy a wide variety of resources on AWS, so you'll need at least one AWS account with full administrative access. You'll also need a text editor to edit YAML/JSON CloudFormation templates, and the AWS CLI tools, which are supported on common operating systems (macOS/Linux/Windows).



# **Who this book is for**

This book is for anyone with a technical background who is interested in using AWS, either for moving existing workloads or deploying completely new applications. Those who want to learn CloudFormation will also find this book useful.



# Sections

In this book, you will find several headings that appear frequently (*Getting ready*, *How to do it...*, *How it works...*, *There's more...*, and *See also*).

To give clear instructions on how to complete a recipe, we use these sections as follows:



# Getting ready

This section tells you what to expect in the recipe, and describes how to set up any software or any preliminary settings required for the recipe.



# **How to do it...**

This section contains the steps required to follow the recipe.



# **How it works...**

This section usually consists of a detailed explanation of what happened in the previous section.



# **There's more...**

This section consists of additional information about the recipe in order to make the reader more knowledgeable about the recipe.



## See also

This section provides helpful links to other useful information for the recipe.



# Conventions

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

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "You should now have an active session under `PowerUserRole` in the application account."

A block of code is set as follows:

```
Parameters:  
  EC2KeyName:  
    Type: String  
    Description: EC2 Key Pair to launch with
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
Parameters:  
  EC2KeyName:  
    Type: String  
    Description: EC2 Key Pair to launch with
```

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

```
| aws ec2 describe-availability-zones --output json
```

**New terms** and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "Clicking the Next button moves you to the next screen."



*Warnings or important notes appear in a box like this.*



*Tips and tricks appear like this.*



# Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail [feedback@packtpub.com](mailto:feedback@packtpub.com), and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at [www.packtpub.com/authors](http://www.packtpub.com/authors).



# **Customer support**

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



# Downloading the example code

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

You can download the code files by following these steps:

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

You can also download the code files by clicking on the Code Files button on the book's webpage at the Packt Publishing website. This page can be accessed by entering the book's name in the Search box.

Please note that you need to be logged in to your Packt account.

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

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

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



# Errata

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

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



# Piracy

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

Please contact us at [copyright@packtpub.com](mailto:copyright@packtpub.com) with a link to the suspected pirated material.

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



# Questions

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



# AWS Fundamentals

In this chapter, we will cover:

- Infrastructure as Code
- AWS CloudFormation
- The AWS command-line tool



# Introduction

**Amazon Web Services (AWS)** is a public cloud provider. It provides infrastructure and platform services at a pay-per-use rate. This means you get on-demand access to resources that you used to have to buy outright. You can get access to enterprise-grade services while only paying for what you need, usually down to the hour.

AWS prides itself on providing the primitives to developers so that they can build and scale the solutions that they require.



# Creating an account

In order to follow along with the recipes, you will need an AWS account. Create an account at <https://aws.amazon.com/> by clicking on the Sign Up button and entering your details.



*Even though we will be taking advantage of the free tier wherever possible, you will need a valid credit card to complete the signup process. Go to <https://aws.amazon.com/free/> for more information. Note that the free tier only applies for the first year of your account's lifetime.*



# Regions and Availability Zones

A fundamental concept of AWS is that its services and the solutions built on top of them are *architected for failure*. This means that a failure of the underlying resources is a scenario actively planned for, rather than avoided until it cannot be ignored.

Due to this, all the services and resources available are divided up into geographically diverse **Regions**. Using specific regions means you can provide services to your users that are optimized for speed and performance.

Within a region, there are always multiple **Availability Zones** (a.k.a. **AZ**). Each AZ represents a geographically distinct—but still close—physical data center. AZs have their own facilities and power source, so an event that might take a single AZ offline is unlikely to affect the other AZs in the region.

The smaller regions have at least two AZs, and the largest has five.

At the time of writing, the following regions are active:

Code	Name	Availability Zones
us-east-1	N. Virginia	5
us-east-2	Ohio	3
us-west-1	N. California	3
us-west-2	Oregon	3
ca-central-1	Canada	2
eu-west-1	Ireland	3
eu-west-2	London	2
eu-central-1	Frankfurt	2
ap-northeast-1	Tokyo	3
ap-northeast-2	Seoul	2
ap-southeast-1	Singapore	2
ap-southeast-2	Sydney	3
ap-south-1	Mumbai	2
sa-east-1	Sao Paulo	3



# The AWS web console

The web-based console is the first thing you will see after creating your AWS account, and you will often refer to it when viewing and confirming your configuration.

The screenshot shows the AWS web console homepage. At the top, there's a navigation bar with 'Services' and 'Resource Groups' dropdowns, and various service icons. Below the navigation is a search bar and a 'Recently visited services' section with links to S3, EC2, Lambda, and CloudFormation. To the right, there's a 'Featured next steps' section with links to manage spend and get best practices. A large central area is titled 'Build a solution' with six quick-start guides: 'Launch a virtual machine', 'Build a web app', 'Deploy a serverless microservice', 'Host a static website', 'Create a backend for your mobile app', and 'Register a domain'. Below this is a 'Learn to build' section with three categories: 'Websites' (3 videos, 3 tutorials, 3 labs), 'DevOps' (6 videos, 2 tutorials, 3 labs), and 'Backup and recovery' (3 videos, 2 tutorials, 3 labs). To the right of the 'Learn to build' section is a link to 'AWS Marketplace' and a 'Have feedback?' section.

The AWS web console

The console provides an overview of all the services available as well as associated billing and cost information. Each service has its own section, and the information displayed depends on the service being viewed. As new features and services are released, the console will change and improve. Don't be surprised if you log in and things have changed from one day to the next.



*Keep in mind that the console always shows your resources by region. If you cannot see a resource that you created, make sure you have the right region selected.*

Choose the region closest to your physical location for the fastest response times. Note that not all regions have the same services available. The larger, older regions generally have the most services available. Some of the newer or smaller regions (that might be closest to you) might not have all services enabled yet. While services are continually being released to regions, you may have to use another region if you simply must use a newer service.

 *The us-east-1 (a.k.a. North Virginia) region is special given its status as the first region. All services are available there, and new services are always released there.*

As you get more advanced with your use of AWS, you will spend less time in the console and more time controlling your services programmatically via the AWS CLI tool and CloudFormation, which we will go into in more detail in the next few topics.



# CloudFormation templates

Where possible, we have based the recipes around a CloudFormation template. CloudFormation is the *Infrastructure as Code* service from AWS.



*Where CloudFormation was not applicable, we have used the AWS CLI to make the process repeatable and automatable.*

Since the recipes are based on CloudFormation templates, you can easily combine different templates to achieve your desired outcomes. By editing the templates or joining them, you can create more useful and customized configurations with minimal effort.



# Infrastructure as Code

**Infrastructure as Code (IaC)** is the practice of managing infrastructure through code definitions.

On an **Infrastructure-as-a-Service (IaaS)** platform such as AWS, IaC is needed to get the most utility and value. IaC differs primarily from traditional *interactive* methods of managing infrastructure because it is machine processable. This enables a number of benefits:

- Improved visibility of resources
- Higher levels of consistency between deployments and environments
- Easier troubleshooting of issues
- The ability to scale more with less effort
- Better control over costs

On a less tangible level, all of these factors contribute to other improvements for your developers: you can now leverage tried-and-tested software development practices for your infrastructure and enable DevOps practices in your teams.



# Visibility

As your infrastructure is represented in machine-readable files, you can treat it like you do your application code. You can take the best-practice approaches to software development and apply them to your infrastructure. This means you can store it in version control (for example, Git and SVN) just like you do your code, along with the benefits that it brings:

- All changes to infrastructure are recorded in commit history
- You can review changes before accepting/merging them
- You can easily compare different configurations
- You can pick and use specific point-in-time configurations



# Consistency

Consistent configuration across your environments (for example, dev, test, and prod) means that you can more confidently deploy your infrastructure. When you know what configuration is in use, you can easily test changes in other environments due to a common baseline.

IaC is not the same as *just writing scripts* for your infrastructure. Most tools and services will leverage higher-order languages and DSLs to allow you to focus on your higher-level requirements. It enables you to use advanced software development techniques, such as static analysis, automated testing, and optimization.



# Troubleshooting

IaC makes replicating and troubleshooting issues easier: since you can duplicate your environments, you can accurately reproduce your production environment for testing purposes.

In the past, test environments rarely had exactly the same infrastructure due to the prohibitive cost of hardware. Now that it can be created and destroyed on demand, you are able to duplicate your environments only when they are needed. You only need to pay for the time that they are running for, usually down to the hour. Once you have finished testing, simply turn your environments off and stop paying for them.

Even better than troubleshooting is fixing issues before they cause errors. As you refine your IaC in multiple environments, you will gain confidence that is difficult to obtain without it. By the time you deploy your infrastructure in to production, you have done it multiple times already.



# Scale

Configuring infrastructure by hand can be a tedious and error-prone process. By automating it, you remove the potential variability of a manual implementation: computers are good at boring, repetitive tasks, so use them for it!

Once automated, the labor cost of provisioning more resources is effectively zero—you have already done the work. Whether you need to spin up one server or a thousand, it requires no additional work.

From a practical perspective, resources in AWS are effectively unconstrained. If you are willing to pay for it, AWS will let you use it.



# Costs

AWS have a vested (commercial) interest in making it as easy as possible for you to provision infrastructure. The benefit to you as the customer is that you can create *and destroy* these resources on demand.

Obviously, destroying infrastructure on-demand in a traditional, physical hardware environment is simply not possible. You would be hard-pressed to find a data center that will allow you to stop paying for servers and space simply because you are not currently using them.

Another use case where on-demand infrastructure can make large cost savings is your development environment. It only makes sense to have a development environment while you have developers to use it. When your developers go home at the end of the day, you can switch off your development environments so that you no longer pay for them. Before your developers come in in the morning, simply schedule their environments to be created.



# DevOps

DevOps and IaC go hand in hand. The practice of storing your infrastructure (traditionally the concern of Operations) as code (traditionally the concern of Development) encourages a sharing of responsibilities that facilitates collaboration.

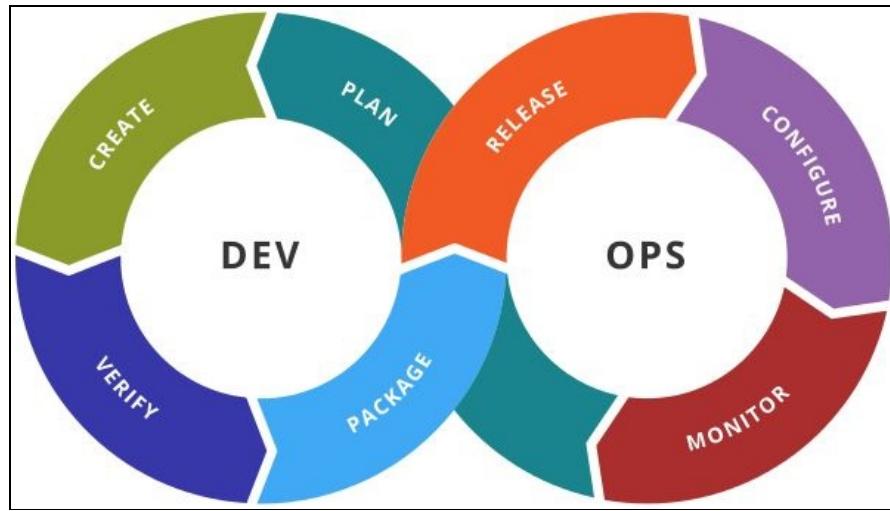


Image courtesy: Wikipedia

By automating the **PACKAGE**, **RELEASE**, and **CONFIGURE** activities in the software development life cycle (as pictured), you increase the speed of your releases while also increasing confidence.

Cloud-based IaC encourages *architecture for failure*: as your resources are virtualized, you must plan for the chance of physical (host) hardware failure, however unlikely.

Being able to recreate your entire environment in minutes is the ultimate recovery solution.

Unlike physical hardware, you can easily simulate and test failure in your software architecture by deleting key components—they are all virtual anyway!



# Server configuration

Server-side examples of IaC are configuration-management tools such as Ansible, Chef, and Puppet.

While important, these configuration-management tools are not specific to AWS, so we will not be covering them in detail here. There are a myriad of books and courses devoted to this topic if you need to know more.



# IaC on AWS

CloudFormation is the IaC service from AWS.

Templates written in a specific format and language define the AWS resources that should be provisioned. CloudFormation is declarative and can not only provision resources, but also update them.

We will go into CloudFormation in greater detail in the next topic.



# CloudFormation

We'll use CloudFormation extensively throughout this book, so it's important that you have an understanding of what it is and how it fits in to the AWS ecosystem. There should easily be enough information here to get you started, but where necessary, we'll refer you to AWS' own documentation.



# What is CloudFormation?

The **CloudFormation** service allows you to provision and manage a collection of AWS resources in an automated and repeatable fashion. In AWS terminology, these collections are referred to as **stacks**. Note however that a stack can be as large or as small as you like. It might consist of a single S3 bucket, or it might contain everything needed to host your three-tier web app.

In this chapter, we'll show you how to define the resources to be included in your CloudFormation stack. We'll talk a bit more about the composition of these stacks and why and when it's preferable to divvy up resources between a number of stacks. Finally, we'll share a few of the tips and tricks we've learned over years of building countless CloudFormation stacks.



*Be warned!*

*Pretty much everyone incurs at least one or two flesh wounds along their journey with CloudFormation. It is all very much worth it, though.*



# Why is CloudFormation important?

By now, the benefits of automation should be starting to become apparent to you. But don't fall in to the trap of thinking CloudFormation will be useful only for large collections of resources. Even performing the simplest task of, say, creating an S3 bucket can get very repetitive if you need to do it in every region.

We work with a lot of customers who have very tight controls and governance around their infrastructure, and especially in the network layer (think VPCs, NACLs, and security groups). Being able to express one's cloud footprint in YAML (or JSON), store it in a source code repository, and funnel it through a high-visibility pipeline gives these customers confidence that their infrastructure changes are peer-reviewed and will work as expected in production. Discipline and commitment to IaC SDLC practices are of course a big factor in this, but CloudFormation helps bring us out of the era of following 20-page run-sheets for manual changes, navigating untracked or unexplained configuration drift, and unexpected downtime caused by fat fingers.



# The layer cake

Now is a good time to start thinking about your AWS deployments in terms of layers. Your layers will sit atop one another, and you will have well-defined relationships between them.

Here's a bottom-up example of how your layer cake might look:

- VPC with CloudTrail
- Subnets, routes, and NACLs
- NAT gateways, VPN or bastion hosts, and associated security groups
- App stack 1: security groups, S3 buckets
- App stack 1: cross-zone RDS and read replica
- App stack 1: app and web server auto scaling groups and ELBs
- App stack 1: CloudFront and WAF config

In this example, you may have many occurrences of the app stack layers inside your VPC, assuming you have enough IP addresses in your subnets! This is often the case with VPCs living inside development environments. So immediately, you have the benefit of multi-tenancy capability with application isolation.

One advantage of this approach is that while you are developing your CloudFormation template, if you mess up the configuration of your app server, you don't have to wind back all the work CFN did on your behalf. You can just turf that particular layer (and the layers that depend on it) and restart from there. This is not the case if you have everything contained in a single template.

We commonly work with customers for whom ownership and management of each layer in the cake reflects the structure of the technology divisions within a company. The traditional infrastructure, network, and cyber security folk are often really interested in creating a safe place for digital teams to deploy their apps, so they like to heavily govern the foundational layers of the cake. **Conway's Law**, coined by Melvin Conway, starts to come in to play here:

*"Any organization that designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure."*

Finally, even if you are a single-person infrastructure coder working in a small team, you will benefit from this approach. For example, you'll find that it dramatically reduces your exposure to things such as AWS limits, timeouts, and circular dependencies.



# CloudFormation templates

This is where we start to get our hands dirty. CloudFormation template files are the codified representation of your stack, expressed in either YAML or JSON. When you wish to create a CloudFormation stack, you push this template file to CloudFormation, through its API, web console, command line tools, or some other method (such as the SDK).

Templates can be replayed over and over again by CloudFormation, creating many instances of your stack.



# YAML versus JSON

Up until recently, JSON was your only option. We'll actually encourage you to adopt YAML, and we'll be using it for all of the examples shown in this book. Some of the reasons are as follows:

- It's just nicer to look at. It's less syntax heavy, and should you choose to go down the path of generating your CloudFormation templates, pretty much every language has a YAML library of some kind.
- The size of your templates will be much smaller. This is more practical from a developer's point of view, but it also means you're less likely to run into the CloudFormation size limit on template files (50 KB).
- The string-substitution features are easier to use and interpret.
- Your EC2 `userData` (the script that runs when your EC2 instance boots) will be much easier to implement and maintain.

AWSTemplateFormatVersion: '2010-09-09'

Parameters:

EC2KeyName: Type: String Description: EC2 Key Pair to launch with Mappings:

RegionMap: us-east-1: AMIID: ami-9be6f38c ap-southeast-2: AMIID: ami-28cff44b

Resources:

ExampleEC2Instance: Type: AWS::EC2::Instance Properties: InstanceType: t2.nano

UserData: Fn::Base64: Fn::Sub': |

```
#!/bin/bash -ex /opt/aws/bin/cfn-signal '${ExampleWaitHandle}'
```

ImageId:

Fn::FindInMap: [ RegionMap, Ref: 'AWS::Region', AMIID ]

KeyName:

Ref: EC2KeyName ExampleWaitHandle: Type:

AWS::CloudFormation::WaitConditionHandle Properties: ExampleWaitCondition: Type:

AWS::CloudFormation::WaitCondition DependsOn: ExampleEC2Instance Properties: Handle:

Ref: ExampleWaitHandle Timeout: 600

Outputs:

ExampleOutput: Value:

Fn::GetAtt: ExampleWaitCondition.Data Description: The data signaled with the WaitCondition



# Parameters

CloudFormation parameters are the input values you define when creating or updating your stack, similar to how you provide parameters to any command-line tools you might use. They allow you to customize your stack without making changes to your template. Common examples of what parameters might be used for are as follows:

- **EC2 AMI ID:** You may wish to redeploy your stack with a new AMI that has the latest security patches installed.
- **Subnet IDs:** You could have a list of subnets that an auto scaling group should deploy servers in. These subnet IDs will be different between your dev, test, and production environments.
- **Endpoint targets and credentials:** These include things such as API hostnames, usernames, and passwords.

You'll find that there are a number of parameter types. In brief, they are:

- String
- Number
- List
- CommaDelimitedList

In addition to these, AWS provides some AWS-specific parameter types. These can be particularly handy when you are executing your template via the CloudFormation web console. For example, a parameter type of `AWS::EC2::AvailabilityZone::Name` will cause the web console to display a drop-down list of valid Availability Zones for this parameter. In the `ap-southeast-2` region, the list would look like this:

- ap-southeast-2a
- ap-southeast-2b
- ap-southeast-2c

The list of AWS-specific parameter types is steadily growing and is large enough that we can't list them here. We'll use many of them throughout this book, however, and they can easily be found in the AWS CloudFormation documentation.

When creating or updating a stack, you will need to provide values for all the parameters you've defined in your template. Where it makes sense, you can define default values for a parameter. For example, you might have a parameter called `debug` that tells your application to run in debug mode. You typically don't want this mode enabled by default, so you can set the default value for this parameter to `false`, `disabled`, or something else your application understands. Of course, this value can be overridden when creating or updating your stack.

You can and should provide a short, meaningful description for each parameter. These are displayed in the web console next to each parameter field. When used properly, they provide hints and context to whoever is trying to run your CloudFormation template.

At this point, we need to introduce the inbuilt `Ref` function. When you need to reference a parameter value, you use this function to do so:

```
| KeyName:  
|   Ref: EC2KeyName
```

While `Ref` isn't the only inbuilt function you'll need to know, it's almost certainly going to be the one you'll use the most. We'll talk more about inbuilt functions later in this chapter.



# Resources

Resources are your actual pieces of AWS infrastructure. These are your EC2 instances, S3 buckets, ELBs, and so on. Almost any resource type you can create by pointing and clicking in the AWS web console can also be created using CloudFormation.



*It's not practical to list all the AWS resource types in this chapter, although you will get familiar with the most common types as you work your way through the recipes in this book. AWS keeps a definitive list of resources types here*

*<http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html>.*

There are a few important things to keep in mind about CloudFormation resources:

- New or bleeding-edge AWS resources are often not immediately supported. CloudFormation support typically lags a few weeks (sometimes months) behind the release of new AWS features. This used to be quite frustrating for anyone to whom infrastructure automation is key. Fast-forward to today, and this situation is somewhat mitigated by the ability to use custom resources. These are discussed further on in this chapter.
- Resources have a default return value. You can use `Ref` to fetch these return values for use elsewhere in your template. For example, the `AWS::EC2::VPC` resource type has a default return value that is the ID of the VPC. They look something like this: `vpc-11aa111a`.
- Resources often contain additional return values. These additional values are fetched using the inbuilt `Fn::GetAtt` function. Continuing from the previous example, the `AWS::EC2::VPC` resource type also returns the following:
  - `CidrBlock`
  - `DefaultNetworkAcl`
  - `DefaultSecurityGroup`
  - `Ipv6CidrBlocks`



# Outputs

Just like AWS resources, CloudFormation stacks can also have return values, called **outputs**. These values are entirely user defined. If you don't specify any outputs, then nothing is returned when your stack is completed.

Outputs can come in handy when you are using a CI/CD tool to create your CloudFormation stacks. For example, you might like to output the public hostname of an ELB so your CI/CD tool can turn it into a clickable link within the job output.

You'll also use them when you are linking together pieces of your layer cake. You may want to reference an S3 bucket or security group created in another stack. This is much easier to do with the new cross-stack references feature, which we'll discuss later in this chapter. You can expect to see the `Ref` and `Fn::GetAtt` functions a lot in the output section of any CloudFormation template.

Mappings: RegionMap: us-east-1: AMIID: ami-9be6f38c ap-southeast-2: AMIID: ami-28cff44b



# Dependencies and ordering

When executing your template, CloudFormation will automatically work out which resources depend on each other and order their creation accordingly. Additionally, resource creation is parallelized as much as possible so that your stack execution finishes in the timeliest manner possible. Things occasionally become unstuck, however.

Let's take an example where an app server depends on a DB server. In order to connect to the database, the app server needs to know its IP address or hostname. This situation would actually require you to create the DB server first so that you can use `Ref` to fetch its IP and provide it to your app server. CloudFormation has no way of knowing about the coupling between these two resources, so it will go ahead and create them in any order it pleases (or in parallel if possible).

To fix this situation, we use the `DependsOn` attribute to tell CloudFormation that our app server depends on our DB server. In fact, `DependsOn` can actually take a list of strings if a resource happens to depend on multiple resources before it can be created. So if our app server were to also depend on, say, a Memcached server, then we use `DependsOn` to declare both dependencies.

If necessary, you can take this further. Let's say that after your DB server boots, it will automatically start the database, set up a schema, and import a large amount of data. It may be necessary to wait for this process to complete before we create an app server that attempts to connect to a DB expecting a complete schema and data set. In this scenario, we want a way to signal to CloudFormation that the DB server has completed its initialization so it can go ahead and create resources that depend on it. This is where `WaitCondition` and `WaitConditionHandle` come in.

Firstly, you create an `AWS::CloudFormation::WaitConditionHandle` type, which you can later reference via `Ref`.

Next, you create an `AWS::CloudFormation::WaitCondition` type. In our case, we want the wait period to start as soon as the DB server is created, so we specify that this `WaitCondition` resource `DependsOn` our DB server.

After the DB server has finished importing data and is ready to accept connections, it calls the callback URL provided by the `WaitConditionHandle` resource to signal to CloudFormation that it can stop waiting and start executing the rest of the CloudFormation stack. The URL is supplied to the DB server via `userData`, again using `Ref`. Typically, `curl`, `wget` or some equivalent is used to call the URL.

A `WaitCondition` resource can have a `Timeout` period too. This is a value specified in seconds. In our example, we might supply a value of `900` because we know that it should never take more than 15 minutes to boot our DB and import the data.

Here's an example of what `DependsOn`, `WaitConditionHandle`, and `WaitCondition` look like combined:

```
ExampleWaitHandle:
  Type: AWS::CloudFormation::WaitConditionHandle
  Properties:
    ExampleWaitCondition:
      Type: AWS::CloudFormation::WaitCondition
      DependsOn: ExampleEC2Instance
      Properties:
        Handle:
```

| Ref: ExampleWaitHandle  
| Timeout: 600



# Functions

CloudFormation provides some inbuilt functions to make composing your templates a lot easier. We've already looked at `Ref` and `Fn::GetAtt`. Let's look at some others you are likely to encounter.



# Fn::Join

Use `Fn::Join` to concatenate a list of strings using a specified delimiter, like this, for example:

```
| "Fn::Join": [ ".", [ 1, 2, 3, 4 ] ]
```

This would yield the following value:

```
| "1.2.3.4"
```



# Fn::Sub

Use `Fn::Sub` to perform string substitution. Consider this: DSN: "Fn::Sub" - `mysql://${db_user}:${db_pass}@${db_host}:3306/wordpress` - { db\_user: lchan, db\_pass: ch33s3, db\_host: localhost }

This would yield the following value:

```
| mysql://lchan:ch33s3@localhost:3306/wordpress
```

When you combine these functions with `Ref` and `Fn::GetAtt`, you can start doing some really powerful stuff, as we'll be seeing in the recipes throughout this book.

Other available inbuilt functions include:

- `Fn::Base64`
- `Fn::FindInMap`
- `Fn::GetAZs`
- `Fn::ImportValue`
- `Fn::Select`



*Documentation on all of these functions is available here <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/intrinsic-function-reference.html>.*



# Conditionals

It's reasonably common to provision a similar but distinct set of resources based on which environment your stack is running in. In your development environment, for example, you may not wish to create an entire fleet of database servers (HA master and read slaves), instead opting for just a single database server. You can achieve this by using conditionals:

- Fn::And
- Fn::Equals
- Fn::If
- Fn::Not
- Fn::Or



# Permissions and service roles

One important thing to remember about CloudFormation is that it's more or less just making API calls on your behalf. This means that CloudFormation will assume the very same permissions or role you use to execute your template. If you don't have permission to create a new hosted zone in Route 53, for example, any template you try to run that contains a new Route 53-hosted zone will fail.

On the flip side, this has created a somewhat tricky situation where anyone developing CloudFormation typically has a very elevated level of privileges, and these privileges are somewhat unnecessarily granted to CloudFormation each time a template is executed.

If my CloudFormation template contains only one resource, which is a Route 53-hosted zone, it doesn't make sense for that template to be executed with full admin privileges to my AWS account. It makes much more sense to give CloudFormation a very slim set of permissions to execute the template with, thus limiting the blast radius if a bad template were to be executed (that is, a bad copy-and-paste operation resulting in deleted resources).

Thankfully, service roles have recently been introduced, and you can now define an IAM role and tell CloudFormation to use this role when your stack is being executed, giving you a much safer space to play in.



# Custom resources

As discussed previously in this chapter, it's common for there to be a lengthy wait between the release of a new AWS feature and your ability to use that feature in CloudFormation.

Before custom resources, this led AWS developers down the path of doing over 95 percent of their automation in CloudFormation and then running some CLI commands to fill in the gaps. It was often difficult to tell exactly which resources belonged to which stack, and knowing exactly when your stack had finished execution became a guessing game.

Fast forward to today, and the emerging pattern is to use a custom resource to delegate to a **AWS Lambda** function. Lambda can fill in the gaps by making API calls on your behalf, and it becomes much easier to track the heritage and completion of these resources.

 *With any luck, you won't need to use this feature for a while. In the meantime, the AWS custom resource documentation is quite comprehensive. If you are trying to use CloudFormation to create a resource that you can't find in the AWS docs, then it's likely that it's not supported in CloudFormation yet and using custom resources is your answer. For more information, refer to <http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-custom-resources.html>.*



# Cross-stack references

When using the layered cake approach, it's very common to want to use outputs from one stack as inputs in another stack. For example, you may create a VPC in one stack and require its VPC ID when creating resources in another.

For a long time, one needed to provide some glue around stack creation to pass output between stacks. AWS recently introduced cross-stack references, which provide a more native way of doing this.

You can now *export* one or more outputs from your stack. This makes those outputs available to other stacks. Note that the name of this value needs to be unique, so it's probably a good idea to include the CloudFormation stack name in the name you're exporting to achieve this.

Once a value is exported, it becomes available to be imported in another stack using the `Fn::ImportValue` function—very handy!



*Make sure, however, that during the time an exported value is being referenced, you are not able to delete or modify it. Additionally, you won't be able to delete the stack containing the exported value. Once something is referencing an exported value, it's there to stay until there are no stacks referencing it at all.*



# Updating resources

One of the principles of IaC is that all changes should be represented as code for review and testing. This is especially important where CloudFormation is concerned.

After creating a stack for you, the CloudFormation service is effectively hands off. If you make a change to any of the resources created by CloudFormation (in the web console, command line, or by some other method), you're effectively causing configuration drift. CloudFormation no longer knows the exact state of the resources in your stack.

The correct approach is to make these changes in your CloudFormation template and perform an update operation on your stack. This ensures that CloudFormation always knows the state of your stack and allows you to maintain confidence that your infrastructure code is a complete and accurate representation of your running environments.



# Change sets

When performing a stack update, it can be unclear exactly what changes are going to be made to your stack. Depending on which resource you are changing, you may find that it will need to be deleted and recreated in order to implement your change. This, of course, is completely undesired behavior if the resource in question contains data you'd like to keep. Keep in mind that RDS databases can be a particular pain point.

To mitigate this situation, CloudFormation allows you to create and review a *change set* prior to executing the update. The change set shows you which operations CloudFormation intends to perform on your resources. If the change set looks good, you can choose to proceed. If you don't like what you see, you can delete the change set and choose another course of action—perhaps choosing to create and switch to an entirely new stack to avoid a service outage.



# Other things to know

There are a few other things you should keep in the back of your mind as you start to build out your own CloudFormation stacks. Let's take a look.



# Name collisions

Often, if you omit the name attribute from a resource, CloudFormation will generate a name for you. This can result in weird-looking resource names, but it will increase the *replayability* of your template. Using `AWS::S3::Bucket` as an example, if you specify the `BucketName` parameter but don't ensure its uniqueness, CloudFormation will fail to execute your template the second time around because the bucket will already exist. Omitting `BucketName` fixes this. Alternatively, you may opt to generate your own unique name each time the template is run. There's probably no right or wrong approach here, so just do what works for you.



# Rollback

When creating a CloudFormation stack, you are given the option of disabling rollback. Before you go ahead and set this to `true`, keep in mind that this setting persists beyond stack creation. We've ended up in precarious situations where updating an existing stack has failed (for some reason) but rollback has been disabled. This is a fun situation for no one.



# Limits

The limits most likely to concern you are as follows:

- The maximum size allowed for your CloudFormation template is 50 KB. This is quite generous, and if you hit this limit, you almost certainly need to think about breaking up your template into a series of smaller ones. If you absolutely need to exceed the 50 KB limit, then the most common approach is to first upload your template to S3 and then provide an S3 URL to CloudFormation to execute.
- The maximum number of parameters you can specify is 60. If you need more than this then again, consider whether or not you need to add more layers to your cake. Otherwise, lists or mappings might get you out of trouble here.
- Outputs are also limited to 60. If you've hit this limit, it's probably time to resort to a series of smaller templates.
- Resources are limited to 200. The same rules apply here as before.
- By default, you're limited to a total of 200 CloudFormation stacks. You can have this limit increased simply by contacting AWS.



# Circular dependencies

Something to keep in the back of your mind is that you may run in to a circular dependency scenario, where multiple resources depend on each other for creation. A common example is where two security groups reference each other in order to allow access between themselves.

A workaround for this particular scenario is to use the `AWS::EC2::SecurityGroupEgress` and `AWS::EC2::SecurityGroupIngress` types instead of the ingress and egress rule types for `AWS::EC2::SecurityGroup`.



# DSLs and generators

DSLs and generators can be a point of hot debate among infrastructure coders. Some love them, some hate them. Some of the reasons why people love them include the following:

- They allow CloudFormation to be written in a language that is more native to them or their team.
- They allow the use some advanced programming constructs. Iteration is a particularly well-cited example.
- Until YAML was supported by CloudFormation, using a DSL usually resulted in code that was easier to read and far less verbose.

Some of the reasons people dislike them are:

- DSLs have a history of becoming abandonware or significantly lagging behind CloudFormation, although there are a couple of well-supported DSLs out there
- Developers are potentially required to learn a new language and navigate another new set of documentation, on top of learning CloudFormation and navigating the AWS documentation
- Google and Stack Overflow become a little less useful because one needs to translate questions and answers

Beyond what is written here, this topic won't come up again in this book. We can't give specific advice as to which road you should take because it's almost always a highly personal and situational choice.

However, a sensible approach, especially while coming to grips with AWS and CloudFormation, would be to stick with YAML (or JSON) until you get to the point where you think a DSL or generator might be useful.



# Credentials

Under no circumstances do you want to have credentials hardcoded in your templates or committed to your source code repository. Doing this doesn't just increase the chance your credentials will be stolen, it also reduces the portability of your templates. If your credentials are hardcoded and you need to change them, that obviously requires you to edit your CloudFormation template.

Instead, you should add credentials as parameters in your template. Be sure to use the `NoEcho` parameter when you do this so that CloudFormation masks the value anywhere the parameters are displayed.



# Stack policies

If there are resources in your stack you'd like to protect from accidental deletion or modification, applying a stack policy will help you achieve this. By default, all resources are able to be deleted or modified. When you apply a stack policy, all resources are protected unless you explicitly allow them to be deleted or modified in the policy. Note that stack policies do not apply during stack creation—they only take effect when you attempt to update a stack.



# The command-line interface tool

The AWS **command-line interface (CLI)** tool is an important piece of the AWS administrator's toolkit.

The CLI tool is often one of the quickest and easiest ways to interact with the API. As a text-based tool, it scales much easier than using the web console. Unlike the console, it can be automated, for example, via scripts. The AWS **application programming interface (API)** represents all the functionality available to you as an AWS administrator. It is also easier to keep a track of through your command-line history. Like all good CLI tools, simple individual commands can be chained (or *piped*) together to perform complex tasks.



*The CLI tool is open source software, maintained on GitHub <https://github.com/aws/aws-cli>. For more detailed documentation, refer to the AWS CLI homepage <https://aws.amazon.com/cli>.*



# Installation

The CLI tool requires Python 2.6.5 or greater.

The easiest way to install it is to use the Python package manager, `pip`:

```
| pip install awscli
```

This will make the command `aws` available on your system.



# Upgrade

AWS frequently releases new services and functionality. In order to use the new features, you will need to upgrade the CLI tool.

To upgrade, run the following `pip` command periodically:

```
| pip install --upgrade awscli
```



# Configuration

Authentication between the CLI tool and the AWS API is done via two pieces of information:

- Access key ID
- Secret access key



*As the name suggests, you should keep your secret access key a secret! Be careful where you store or send it.*

Once you have created a user, you can configure the tool to use it for authentication purposes.

While you can configure the CLI tool with access keys directly, this should be avoided. Instead, you should use profiles to store your credentials. Using profiles gives you a more consistent and manageable centralized location to secure your secret keys.



# Default profile

Without any additional configuration or options, your CLI tool commands will use the default profile.

To set up the default profile, you can use the following command:

```
| aws configure
```

This will prompt you for an access key ID, secret access key, region, and output format.

<strong>aws configure --profile <profile-name></strong>

Once you have responded to the prompts, you can reference the named profile by passing the `--profile <profile-name>` option with your command.

<strong>export AWS\_PROFILE=<profile-name></strong>

<strong>export AWS\_ACCESS\_KEY\_ID=<access-key-id></strong> <strong>export AWS\_SECRET\_ACCESS\_KEY=<secret-access-key></strong>



# Instance roles

When running the CLI tool on an EC2 instance, you can leverage the instance's IAM role to make calls. This means you do not need to configure credentials or set environment variables (manually).

Behind the scenes, the instance will retrieve and set its own AWS environment variables that allow API calls. You do need to ensure the instance has appropriate permissions.



*The AWS CLI tool comes preinstalled on AWS Linux-based instances.*



# Usage

All CLI tool commands are service based. Using service commands and subcommands, you can make calls directly to the AWS API.



# Commands

Each command represents an AWS service. While most services have one command associated with them, some services have multiple commands (for example, S3 has `s3` and `s3api`).



*Run `aws help` to see all the commands/services that are available—they will probably have changed by the time this book prints!*



# Subcommands

Each command has a selection of subcommands to perform service-specific actions.



*Run `aws <command> help` to see all subcommands.*



# Options

Subcommands take options, which start with `--`.



*See all options and their purpose with `aws <command> <subcommand> help`.*

While most are optional (hence the name), those that are *not* surrounded by square brackets (`[]`) are required. You will get an error message (with appropriate details) if you do not include them.

The built-in documentation is the best place to start looking for answers. There are usually examples after all of the options have been described. Otherwise, there are plenty of examples available online.

Some options are available to all or most commands, so they are particularly useful to know.



# Output

The CLI tool can be configured to output in JSON, table, or text format. To control the output type, use the `--output` option.

To set a default output type for all your commands, set the `output` parameter for your profile.

```
aws ec2 describe-availability-zones --output json {  
  "AvailabilityZones": [  
    {  
      "State": "available", "RegionName": "us-east-1", "Messages": [], "ZoneName": "us-east-1a"  
    }, {  
      "State": "available", "RegionName": "us-east-1", "Messages": [], "ZoneName": "us-east-1c"  
    }, {  
      "State": "available", "RegionName": "us-east-1", "Messages": [], "ZoneName": "us-east-1d"  
    }, {  
      "State": "available", "RegionName": "us-east-1", "Messages": [], "ZoneName": "us-east-1e"  
    }  
  ]  
}
```



# Table

The table format displays a text/ASCII table of results. This can be useful for generating printable reports:

```
rowan — bash — 114x31
$ aws ec2 describe-availability-zones --output table
+-----+
| DescribeAvailabilityZones |
+-----+
| AvailabilityZones          |
+-----+
| RegionName | State   | ZoneName |
+-----+
| us-east-1  | available | us-east-1a |
+-----+
| AvailabilityZones          |
+-----+
| RegionName | State   | ZoneName |
+-----+
| us-east-1  | available | us-east-1c |
+-----+
| AvailabilityZones          |
+-----+
| RegionName | State   | ZoneName |
+-----+
| us-east-1  | available | us-east-1d |
+-----+
| AvailabilityZones          |
+-----+
| RegionName | State   | ZoneName |
+-----+
| us-east-1  | available | us-east-1e |
+-----+
$ █
```



# Text

The text output format only displays the resulting key/value response. No additional formatting or display characters are added.

```
rowan — -bash — 114x31
$ aws ec2 describe-availability-zones --output text
AVAILABILITYZONES      us-east-1      available      us-east-1a
AVAILABILITYZONES      us-east-1      available      us-east-1c
AVAILABILITYZONES      us-east-1      available      us-east-1d
AVAILABILITYZONES      us-east-1      available      us-east-1e
$ 
```



# Querying

The CLI tool supports transforming the response from the API with the `--query` option. This option takes a JMESPath query as a parameter and returns the query result.



*JMESPath is a query language for JSON. For more information, visit <http://jmespath.org/>.*

As the query is processed as part of the command, it takes place on the server, not the client. By offloading work to the server, you can reduce the size of the resulting payload and improve response times.

JMESPath can be used to transform the response that you receive:

```
$ aws ec2 describe-availability-zones \
--output json \
--query 'AvailabilityZones[] .ZoneName'
[
    "us-east-1a",
    "us-east-1c",
    "us-east-1d",
    "us-east-1e"
]
```

It can also be used to filter the data that is received:

```
$ aws ec2 describe-availability-zones
--output json
--query "AvailabilityZones[?ZoneName == 'us-east-1a'].State"
[
    "available"
]
```



# Generate CLI skeleton

When performing complex tasks with the CLI tool, it may be easier to pass a JSON object of options. This kind of interaction may signify that you should use one of the AWS **software development kits (SDKs)**.

```
<strong>$ aws ec2 describe-availability-zones --generate-cli-skeleton </strong> <strong>{ </strong> <strong>"DryRun": true, </strong> <strong>"ZoneNames": [ </strong> <strong>"" </strong> ], </strong> <strong>"Filters": [ </strong> <strong>{ </strong> <strong>"Name": "", </strong> <strong>"Values": [ </strong> <strong>"" </strong> <strong> ] </strong> <strong> } </strong> <strong> ] </strong> <strong>} </strong>
```

You can then copy, edit, and use this object to define your command options without passing lots of individual options. It works best for commands with arrays of options or a variable number of options.

```
<strong>$ aws ec2 describe-availability-zones --generate-cli-skeleton output </strong>
<strong>{ </strong>

<strong>"AvailabilityZones": [ </strong> <strong> { </strong>
<strong>"ZoneName": "ZoneName", </strong> <strong>"State": "State", </strong>
<strong>"RegionName": "RegionName", </strong> <strong>"Messages": [ </strong>
<strong> { </strong>

<strong>"Message": "Message" </strong> <strong> } </strong>
<strong> ] </strong>
<strong> } </strong>
<strong> ] </strong>
<strong>}</strong>
```



# Pagination

Results returned by the CLI tool are limited to 1,000 resources by default.

This is not normally an issue, but at a certain scale, you may run into pagination issues. A common example is files in an S3 bucket.



*If you are absolutely sure you should be seeing a particular resource in a response but cannot, check your pagination. The resource may be included in the matching resources, just not in the part of the response that was returned to you.*

The following options allow you to control the number and starting point of the results returned to you from the API:

- `--page-size`: This limits how many resources *will be displayed to you*, but does not actually limit the number returned. The default number of items (that is, 1,000) will still be processed and returned to you.
- `--max-items`: This sets an upper limit on how many items will actually be returned in the response. You may receive fewer items, but you will not receive more than this number.
- `--starting-token`: This changes where the response starts. Use this to display subsequent results, beyond the first page.

```
| aws s3 ls --bucket bucket-name --max-items 100 --starting-token None____100
```



# Autocomplete

You can enable tab-completion of commands, subcommands, and options by configuring the completer included with the CLI tool.

On OS X, Linux, and Windows systems with a bash shell, you can load the completer with the following command: `complete -C 'which aws_completer'aws`

By default, the `aws_completer` program is installed in `/usr/local/bin`. If your tool is installed to a non-standard location, you will need to find it and change the `which aws_completer` command to the relevant path.



# Related tools

The following program work nicely with the AWS CLI tool, and may come in handy.

```
<strong>$ aws ec2 describe-availability-zones --output json | jq  
.AvailabilityZones[].ZoneName'</strong> <strong>"us-east-1a"</strong>  
  
<strong>"us-east-1c"</strong>  
  
<strong>"us-east-1d"</strong>  
  
<strong>"us-east-1e"</strong>
```



# Managing AWS Accounts

In this chapter, we will cover the following topics:

- Setting up a master account
- Creating a member account
- Inviting an account
- Managing your accounts
- Adding a service control policy



# Introduction

We work with a lot of companies who maintain a large, ever-growing number of AWS accounts. Keeping a handle on all these accounts has typically been quite difficult to do—even for the most seasoned AWS users.

With the release of AWS Organizations, you now have the ability to centrally manage your AWS accounts, to arrange them into logical groupings and hierarchies, and to apply controls to them in ways which haven't previously been possible on the AWS platform.



# Setting up a master account

All accounts that use AWS Organizations for billing and control purposes must have a *master account*. This account controls membership to the organization, and pays the bills of all the members (someone's got to do it).



# How to do it...

To set up a master account, perform the following steps:

1. Go to the My Organization section of the account you want to become the master. You must be logged in with your root credentials (that is, those you created the account with):

The screenshot shows the AWS Management Console homepage. At the top right, the user is signed in as 'rowan udell' in the 'N. Virginia' region. A dropdown menu is open, showing options like 'My Account', 'My Organization' (which is highlighted in yellow), 'My Billing Dashboard', 'My Security Credentials', 'Sign Out', and a 'Get best practices' section. Below the header, there's a search bar and a 'Services' dropdown. The main content area is divided into sections: 'AWS services', 'Build a solution', and 'Learn to build'. The 'Build a solution' section contains six quick-start guides with icons and descriptions: 'Launch a virtual machine' (With EC2, ~1 minute), 'Build a web app' (With Elastic Beanstalk, ~6 minutes), 'Deploy a serverless microservice' (With Lambda, API Gateway, ~2 minutes), 'Host a static website' (With S3, CloudFront, Route 53, ~5 minutes), 'Create a backend for your mobile app' (With Mobile Hub, ~5 minutes), and 'Register a domain' (With Route 53, ~3 minutes). The 'Learn to build' section has a 'See all' link. At the bottom, there are links for 'https://console.aws.amazon.com/organizations/home?region=us-east-1', 'One', and 'Backup and recovery'.

2. In the AWS Organizations section of the AWS console, click on Create organization, as shown in the following screenshot:

Secure https://console.aws.amazon.com/organizations/home?region=us-east-1#/getting-started

Guest

AWS Organizations

rowan u dell Support

## Welcome

You can work with AWS Organizations in any of the following ways:

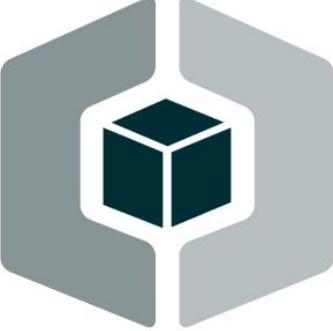
- AWS SDKs
- AWS Management Console
- AWS Command Line Tools
- AWS Organizations HTTPS Query API

Invitations 0

# AWS Organizations

AWS Organizations enables you to centrally apply policy-based controls across multiple accounts in the AWS Cloud. You can consolidate all your AWS accounts into an organization, and arrange all AWS accounts into distinct organizational units.

Create organization User guide



1 2 3

Create accounts Organize accounts Apply policies

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

3. Unless you have a specific requirement, choose ENABLE ALL FEATURES to get the full benefit of organizations, as shown in the following screenshot:

The screenshot shows the 'Create new organization' step of the AWS Organizations wizard. A modal window is open, prompting the user to choose between enabling all features or enabling only consolidated billing. The 'ENABLE ALL FEATURES' option is selected, listing benefits like policy-based controls and hierarchical account management. The 'ENABLE ONLY CONSOLIDATED BILLING' option is also listed, showing benefits like single payer and centralized cost tracking. A large 'Create organization' button is at the bottom of the modal. The background shows the 'Welcome' screen of the AWS Organizations console, which includes links for AWS SDKs, Management Console, Command Line Tools, and HTTPS Query API, along with an 'Invitations' section. At the bottom, there are tabs for 'Create accounts', 'Organize accounts', and 'Apply policies', with the 'Create accounts' tab currently active.

4. Now that your account has been converted, you can return to the AWS Organizations page to see a list of all your accounts:

AWS Organizations

Secure https://console.aws.amazon.com/organizations/home?region=us-east-1#/accounts

Guest

AWS Organizations

rowan udell Support

Accounts Organize accounts Policies Invitations Settings

Add account Remove account Filter

<input type="checkbox"/>	Account name	Account ID	Status
<input type="checkbox"/>	rowan udell	833595768461	Joined on 3/5/2017

Select an account to see its details.

Feedback English

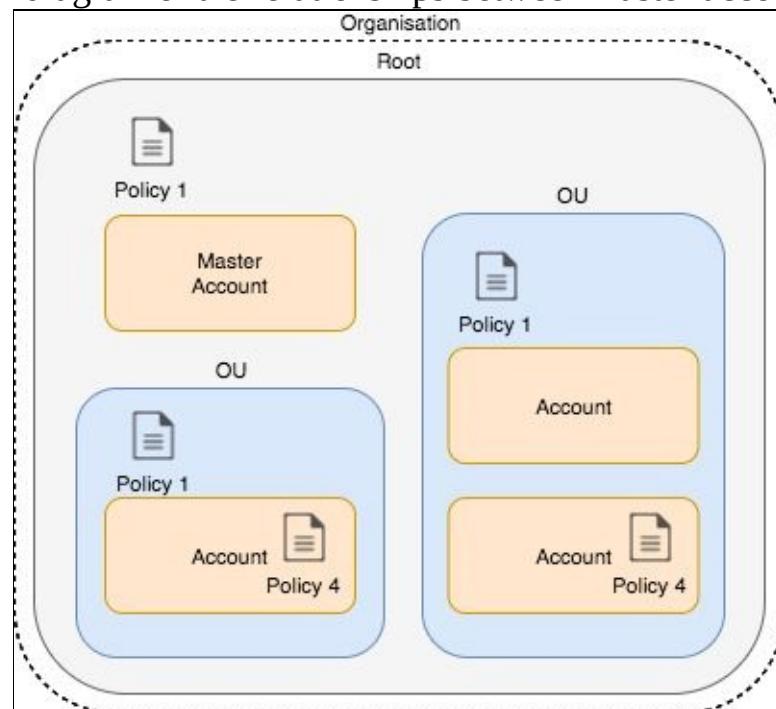
© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use



# How it works...

While this is a very simple recipe, it's the first thing you must do before you can use any of the useful features of AWS Organizations.

Here you can see a high-level diagram of the relationships between master accounts, members, and



organizational units (OUs):

We deliberately enable all the features of organizations. The consolidated billing option is available for backward compatibility—before organizations, consolidated billing was your only option to link accounts.



*Do not use your master account for day-to-day tasks. Since it is so important, it doesn't make sense to risk using it and/or having access keys for it. If your master account was to become compromised somehow, it would impact all of your member accounts. Just don't do it.*

The master account will always have a star next to its name.



# **There's more...**

All of the organization's functionality is exposed via the API. This means you can use the AWS SDKs or the CLI tool to do the same things you would in the web console.



# Multi-factor authentication

As mentioned in the consolidated billing confirmation e-mail, it is advisable to configure **multi-factor authentication (MFA)** on your console. To do this, log in as your root user (that is, the credentials you used when first creating your account), go to the **Identity and Access Management (IAM)** console, and follow the Activate MFA on your root account prompts.



# Using the CLI

You can easily create your master account with the CLI tool. The following command will turn your account into a master account, with all organizations features enabled:

```
| aws organizations create-organization
```



## See also

- The *Inviting an account* recipe
- The *Creating a member account* recipe



# Creating a member account

Once your organization is up and running, the most common use you will have for it is automating the account creation process. Accounts created inside an organization are referred to as **member accounts**.

All charges incurred by a member account will be billed to the master account.



# Getting ready

Obviously, you will need an organization to perform this recipe. See the other recipes in this chapter to get started.

```
<strong> aws organizations create-account </strong> <strong> --email <member-  
account-owners@email.com> </strong> <strong> --account-name <member-account-  
name> \ </strong> <strong> --query 'CreateAccountStatus.Id'</strong>  
  
<strong> aws organizations describe-create-account-status \ </strong> <strong> --create-  
account-request-id <your-create-account-status-id></strong>
```



# How it works...

The command to create a member account in your organization is extremely simple.



*The e-mail address used cannot be associated with any other AWS accounts.*

The account creating process takes some time, so it is done *asynchronously*. This means that you won't receive an immediate status to your `create-account` command. Instead, the command in this recipe will return a request ID.

This ID is then passed to another account to check the status of the creation. When the status is `CREATED`, you can start to use the new account.



# There's more...

While this functionality is definitely useful, the AWS Organizations service is relatively new. This means there are a few *features* you should be aware of.

<strong> aws organizations list-accounts</strong>

<strong> arn:aws:iam::<your-member-account-id>:role/OrganizationAccountAccessRole</strong>

3. If you have changed the created role's name, update the last part of the ARN accordingly.

See the recipes in [Chapter 8](#), *Security and Identity* for information on how to best manage multiple accounts.



# Service control policies

The **service control policies (SCPs)** are another major feature of AWS Organizations. You can apply them at multiple levels/resources, including accounts (both member accounts and invited accounts). Check the other recipes in this chapter for more details.



# Root credentials

Some activities still require the root credentials of the account. An example activity would be closing (or deleting) an account (see the next section for more details).

In order to do this, you will need to do the password recovery process for the e-mail that was associated with the account when the `create-account` request was sent.



# Deleting accounts

At the time of writing, *there is no way to delete an account created in your organization via the API*. We can only imagine that being able to programmatically delete a member account created in an organization will be a *highly requested* feature, and will be addressed soon. You can still go into the member account and close it using the root credentials, but these don't exist by default.



*While you can technically delete your organization via the API, you cannot do it if you have created any member accounts in your organization (because you can't delete them, your organization will never be empty). This should improve in the near future, but is still worth being aware of now.*



## See also

- The *Setting up a master account* recipe
- The *Adding a service control policy* recipe
- The *Cross-account user roles* recipe in [Chapter 8, Security and Identity](#)



# Inviting an account

While it makes sense to create new accounts in your organization, what do you do with all the other accounts you have now?

You can invite existing accounts to your organization, which means you can treat them just like a member account from an administrative point of view. This greatly simplifies the administrative overhead of your accounts, as there isn't a separate process for *old* and *new* accounts.

As this is generally performed once for each existing account, we will use the console.



*All the AWS organizations functionality is available via the SDKs and AWS CLI tool. If you need to automate this process, you can.*



# Getting ready

You must have enabled AWS Organizations for one of your accounts (your master account), and have another account that has not been made part of an organization yet (that you will invite).



# How to do it...

1. From the AWS console of the master account, click on your username, and select My Organization from the drop-down menu:

The screenshot shows the AWS Management Console interface. At the top right, the user's name "rowan udell" is displayed with a dropdown arrow. A tooltip or expanded menu is shown for "My Organization". Other options in the dropdown include "My Account", "My Billing Dashboard", "My Security Credentials", "Sign Out", and "Get best practices".

**AWS services**

Find a service by name (for example, EC2, S3, Elastic Beanstalk).

> All services

**Build a solution**

Get started with simple wizards and automated workflows.

- Launch a virtual machine**  
With EC2  
~1 minute
- Build a web app**  
With Elastic Beanstalk  
~6 minutes
- Deploy a serverless microservice**  
With Lambda, API Gateway  
~2 minutes
- Host a static website**  
With S3, CloudFront, Route 53  
~5 minutes
- Create a backend for your mobile app**  
With Mobile Hub  
~5 minutes
- Register a domain**  
With Route 53  
~3 minutes

**Learn to build**

Learn to deploy your solutions through step-by-step guides, labs, and videos.

See all

https://console.aws.amazon.com/organizations/home?region=us-east-1

2. You will be taken to the AWS Organizations console, where you will see your current account:

The screenshot shows the AWS Organizations console with the 'Accounts' tab selected. The main area displays a table with columns: Account name, Account ID, and Status. One account is listed: 'rowan udell' with Account ID '833595768461' and status 'Joined on 3/5/2017'. There are buttons for 'Add account' and 'Remove account'. A search bar labeled 'Filter' is present. On the right, a sidebar titled 'Account details' says 'Select an account to see its details.' The bottom navigation bar includes 'Feedback', 'English', 'Privacy Policy', and 'Terms of Use'.

3. Click on the Invitations tab in the top-right of the console:

The screenshot shows the AWS Organizations console with the 'Invitations' tab selected. The main area displays a table with columns: Email or Account ID, Request Date, Expiration, Status, and Actions. A message at the top says 'No items to show'. There is a button for 'Invite account'. The bottom navigation bar includes 'Feedback', 'English', 'Privacy Policy', and 'Terms of Use'.

4. Click on the Invite account button. Specify the account ID (or main e-mail address) of the account to invite:

The screenshot shows the AWS Organizations console at the URL <https://console.aws.amazon.com/organizations/home#/accounts/add/invite>. The top navigation bar includes 'AWS Organizations', 'Guest', 'rowan udell', 'Support', and a menu icon. A central message asks, "How do you want to add an account to your organization?". Two options are presented: "Invite account" (selected) and "Create account". The "Invite account" section contains the sub-instruction "Invite an existing AWS account to join your organization." Below this is a form field labeled "Account ID or email\*" containing the value "365407189188", with a placeholder "Enter multiple email addresses or account IDs separated by commas." To the right is a "Notes" text area with the instruction "You may optionally include a note with your request.". At the bottom of the form are required field indicators (\* Required fields), copyright information ("© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved."), and links for "Feedback", "English", "Privacy Policy", and "Terms of Use".

5. Once you click Invite, you will be taken to a list of invitations where you can view the status:

The screenshot shows the AWS Organizations console at the URL <https://console.aws.amazon.com/organizations/home#/sent-invitations>. The top navigation bar includes 'AWS Organizations', 'Guest', 'rowan udell', 'Support', and a menu icon. The main content area displays a table titled "Invitations" under the "Accounts" tab. The table has columns: "Email or Account ID", "Request Date", "Expiration", "Status", and "Actions". One row is shown, corresponding to the invite sent earlier: "365407189188", "3/15/17 5:06 PM", "3/30/17 5:06 PM", "OPEN", and a "Cancel" link. The bottom of the screen features a footer with "Feedback", "English", "Privacy Policy", and "Terms of Use" links, along with copyright information ("© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.").

6. At this stage, the target/invited account will receive an e-mail notifying them of the invite:

# Your AWS account has been invited to join an AWS organization



no-reply <no-reply@amazon.com>  
to aws+alpha Show details

5:06 PM (19 minutes ago)

Inbox

Reply to All

More ▾



Hello,

rowan udell (owned by aws+orgs@rowanudell.com) would like to add your AWS account (365407189188) to their AWS organization.

If you accept the invitation, all activity in your AWS account will be billed to the AWS account of rowan udell, and rowan udell will be able to view the AWS usage and charges from your account.

If the AWS organization has enabled all features, rowan udell will have full control of your account. For example, rowan udell could limit your access to AWS services and APIs or prevent you from removing your account from the organization. You can verify if the organization has enabled all features (or is just using Consolidated Billing) through the console link below.

To view the invitation, including what features have been enabled, click this link:

<https://console.aws.amazon.com/organizations/home#/invites>

To learn more about AWS Organizations, see [What is AWS Organizations?](#)

Thank you for using Amazon Web Services.

7. Log in to the invited account and go to the My Organization link under the user menu:

Secure https://console.aws.amazon.com/console/home?region=us-east-1

Rowan Udell N. Virginia Support

AWS services

Find a service by name (for example, EC2, S3, Elastic Beanstalk).

All services

Compute: EC2, EC2 Container Service, Lightsail, Elastic Beanstalk, Lambda, Batch

Developer Tools: CodeCommit, CodeBuild, CodeDeploy, CodePipeline, X-Ray

Internet of Things: AWS IoT

Storage: S3, EFS, Glacier, Storage Gateway

Management Tools: CloudWatch, CloudFormation, CloudTrail, Config, OpsWorks, Service Catalog, Trusted Advisor, Managed Services

Mobile Services: Mobile Hub, Cognito, Device Farm, Mobile Analytics, Pinpoint

Database: RDS, DynamoDB, ElastiCache

Security, Identity & Compliance: Security, Identity &

Game Development: Amazon GameLift

Application Services: Step Functions, SWF, API Gateway

My Account, My Organization, My Billing Dashboard, My Security Credentials, Sign Out, Get best practices, Announcing AWS Batch, Announcing Amazon Lightsail, See all

8. In the AWS Organizations console, you can see the pending invite on the left:

Secure https://console.aws.amazon.com/organizations/home?region=us-east-1#/getting-started

Rowan Udell Support

Welcome

You can work with AWS Organizations in any of the following ways:

AWS Organizations Query API, AWS Management Console, AWS Command Line Tools, AWS SDKs

Invitations 1

AWS Organizations

AWS Organizations enables you to centrally apply policy-based controls across multiple accounts in the AWS Cloud. You can consolidate all your AWS accounts into an organization, and arrange all AWS accounts into distinct organizational units.

Create organization User guide

1 Create accounts 2 Organize accounts 3 Apply policies

Feedback English © 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

9. Clicking on the invite, you can see its details:

The screenshot shows the AWS Organizations Invitations page. At the top, there's a header with the AWS logo, the title "AWS Organizations", and user information for "Rowan Udell". Below the header, the main title is "Invitations". A message states: "You have invitations to join other organizations. Review the details to respond to the invitations. You can only join one organization at a time." On the left, a card displays invitation details:

Organization ID	o-vtxz71iefy
Master account name	rowan udell
Master account email	aws+orgs@rowanudell.com
Requested controls	<b>Enable all features</b> The master account pays the charges accrued by all member accounts and can attach policy-based controls to the member accounts.

At the bottom of the card are two buttons: "Accept" and "Decline".

At the very bottom of the page, there are links for "Feedback", "English", and "Privacy Policy / Terms of Use".

10. When the invite includes all features, you will be asked to confirm your acceptance:

The screenshot shows the same AWS Organizations Invitations page as before, but with a modal dialog box overlaid. The dialog has a title "Confirm joining the organization" and a message: "You are about to join the AWS Organization with the following ID: o-vtxz71iefy". Below this, there is a detailed description of what accepting the invitation entails: "If you accept the invitation, the administrator of the organization can attach policy-based controls to your AWS account. The organization administrator can control which AWS services and APIs are allowed in this account for business reasons such as security or budgetary controls. These controls can include preventing this account from leaving the organization." At the bottom of the dialog is a blue "Confirm" button.

11. Once confirmed, you can now see the details of the organization you have joined:

AWS Organizations

Secure https://console.aws.amazon.com/organizations/home?region=us-east-1#/organization/overview

Guest

AWS Organizations

Rowan Udell Support

Your account belongs to the following organization:

Organization ID:  
o-vtxz71iefy

Master account email:  
aws+orgs@rowanudell.com

[Leave organization](#)

Organization features enabled

All features enabled: The organization that your account is in pays for your account and can apply organization policies that can restrict what your account can do.

[Learn more](#)

Feedback English | © 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. | Privacy Policy | Terms of Use

12. At this stage, the master account will be notified of the accepted invite:

# An AWS account has joined your AWS Organization



no-reply <no-reply@amazon.com>  
to aws+orgs Show details

5:09 PM (17 minutes ago)

Inbox

Reply to All

More ▾



Greetings from Amazon Web Services,

Rowan Udell has accepted your request to join and is now a member of your AWS organization.

AWS Account ID (last 4 digits): -9188

AWS Account Name: Rowan Udell

AWS Account E-mail: aws+alpha@rowanudell.com

Effective Date: 2017-03-15

You can remove this AWS account at any time by going to the Accounts section of the AWS Organizations console:

<https://console.aws.amazon.com/organizations/home/accounts>

Thank you for using Amazon Web Services.

Sincerely,

Amazon Web Services

13. Back in the master account, you can now see the new account alongside the master:

AWS Organizations

Secure https://console.aws.amazon.com/organizations/home?region=us-east-1#/accounts

Guest

AWS Organizations

rowan udell Support

Accounts Organize accounts Policies Invitations Settings

Add account Remove account Filter

	Account name	Account ID	Status
<input type="checkbox"/>	rowan udell	833595768461	Joined on 3/5/2017
<input type="checkbox"/>	Rowan Udell	365407189188	Joined on 3/15/2017

Select an account to see its details.

Feedback English

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use



# How it works...

While there are many steps involved, the process of inviting an existing account is a relatively simple *handshake* process. This means that both sides must actively initiate/accept the invite, in order for it to succeed—an invite cannot be forced.

After specifying the target account's account ID (or e-mail address), the e-mail address associated will be notified.

As part of the handshake process, the invited account must explicitly accept the invite.



*It is important to note that the default invite type (and what we have used in the recipe) is to use the full feature set for AWS Organizations. As noted in the console, this means that the invited account could be prevented from leaving the organization if the relevant policies are configured.*

After confirmation, both parties will receive an e-mail detailing the membership. From this point forward, the bill for the invited account will be paid by the master account.



# **There's more...**

Invited accounts are treated differently to accounts created via the organizations functionality.



# Removing accounts

Unlike *member accounts* (which are created via the AWS Organizations API), invited accounts can be removed from an organization.



# Consolidated billing

As an alternative to the *full feature* invite, it is possible to specify just *consolidated billing* mode for an organization. In this mode, no OUs or policies will be available, only the billing relationship will be shared between the accounts (that is, the master account will pay the bill of the member accounts).



*Any pre-existing accounts that were configured to use consolidated billing will have been automatically migrated to AWS Organizations in consolidated billing mode.*



## See also

- The *Creating a member account* recipe

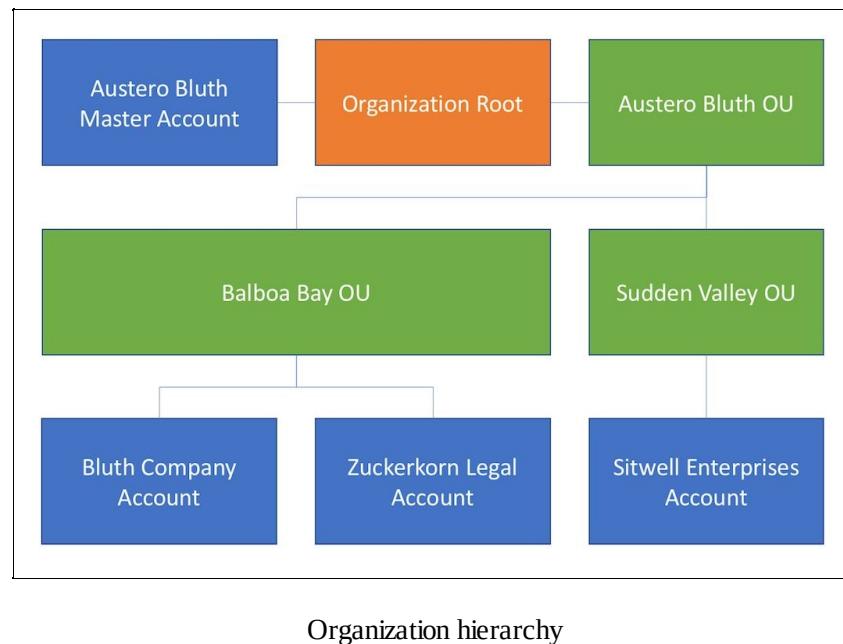


# Managing your accounts

There are a number of ways to group and arrange your AWS accounts. How you do this is completely up to you, but here are a few examples to consider:

- **Business unit (BU) or location:** You may wish to allow each BU to work in isolation on their own products or services, on their own schedule, without impacting other parts of the business
- **Cost center:** Grouping according to cost may help you track spend versus allocated budget
- **Environment type:** It may make sense to group your development, test, and production environments together in a way which helps you manage the controls across each environment
- **Workload type or data classification:** Your company may want to isolate workload types from each other, or ensure that particular controls are applied to all accounts containing a particular kind of data

In the following fictitious example, we have isolated the **Sitwell Enterprises Account** from the rest of the organization by placing it in an OU called **Sudden Valley**. Perhaps they operate in a different geographical location and have different regulatory requirements around controls and access.



Note that while it's also technically possible for us to put the master account inside an OU, we avoid doing this to make it obvious that:

- It's the master account and has control over the entire organization
- The rules we set, using SCPs for the member accounts in our organization, do not apply to the master account (because they can't)

Learn more about SCPs in the *Adding a service control policy* recipe in this chapter.



# Getting ready

Before we can proceed, you should have already done the following:

- Set up a master AWS account
- Created an organization
- Created member accounts in your organization, or manually added member accounts (by invitation) to your organization



# How to do it...

We'll now cover the one-line commands you'll need to perform the common tasks required to manage your OU. These commands can only be performed in your master account.

<strong>aws organizations list-roots</strong>

```
<strong> </strong><strong>aws organizations create-organizational-unit </strong>
<strong> --parent-id <root-id or parent-ou-id> \
</strong><strong> --name <desired-ou-name></strong>
```

<strong>aws organizations list-organizational-units-for-parent \</strong>

<strong>--parent-id <root-id or parent-ou-id></strong>

```
<strong> </strong><strong>aws organizations move-account \
</strong><strong> --account-id <twelve-digit-account-id> \</strong> <strong> --source-
parent-id <root-id> \</strong> <strong> --destination-parent-id <new-parent-ou-id>
</strong>
```

2. If you wish to move an account from one OU to another, simply use the same command but use the existing parent OU ID instead of the root ID.

```
<strong> aws organizations move-account </strong> <strong> --account-id <twelve-digit-account-id> </strong> <strong> --source-parent-id <existing-parent-ou-id> </strong> <strong> --destination-parent-id <root-id></strong>
```



# Deleting an OU

To delete an OU, you'll first need to make sure it's empty by removing its child accounts (as mentioned previously). You can then go ahead and delete the OU like so:

```
| aws organizations delete-organizational-unit \
|   --organizational-unit-id <ou-id>
```



# How it works...

If done right, grouping your accounts together using OUs will help you simplify the way you manage and administer them. Try to use only *just enough* OUs to get the job done. The idea is to use OUs to make your life easier, not harder.



# There's more...

- The **organizational control policies (OCPs)** can be attached to your root, OU, or AWS accounts. At this time, only one kind of OCP is supported: SCP.
- Accounts can only belong to one OU or root.
- Similarly, OUs can only belong to one OU or root.
- It's best to avoid deploying resources in the master account because this account can't be controlled with SCPs. The master account should be treated as a management account for audit, control, and billing purposes only.



## See also

- The *Adding a service control policy* recipe



# Adding a service control policy

Before we begin, we should talk through what SCPs are and how they apply to your organization.

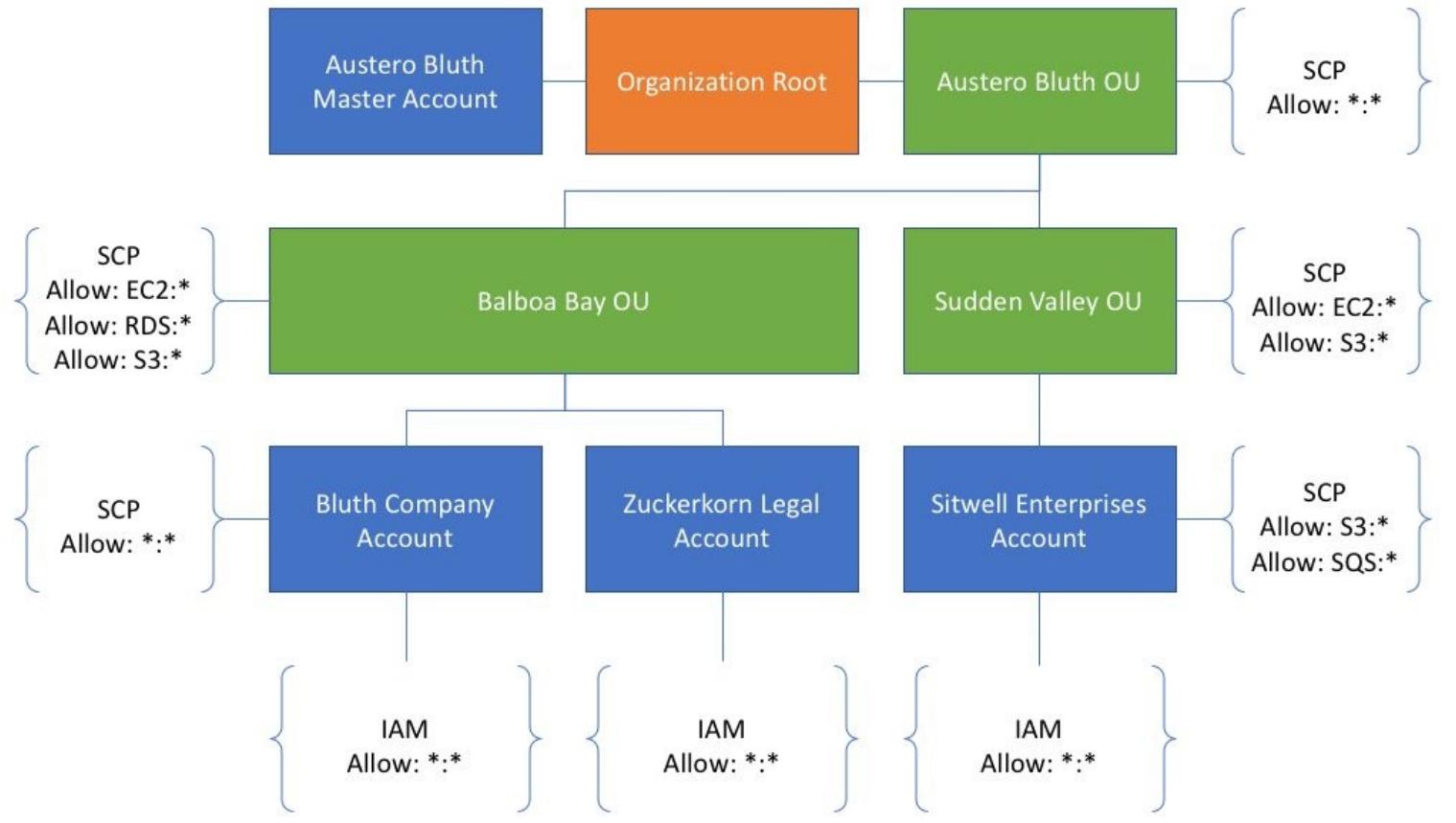
An SCP consists of a policy document which defines (by way of filtering) the services and actions which are able to be used and performed within an OU or in an AWS account. If you've previously configured an IAM policy, then you will have more than enough background knowledge to get started with SCPs. Apart from a couple of minor exceptions, they look exactly the same.

SCPs can be applied at different levels throughout an organization. These are the levels, starting from the bottom and going up:

- **AWS account level:** An SCP applied to an AWS account takes effect on only that account. It's important to note that the SCP is very separate from the IAM policies which live inside the account. For example, an SCP might allow full access to S3 for an AWS account but the IAM policies inside the account may deny it (for certain roles and/or users).
- **OU level:** An SCP applied at the OU level will apply to all the AWS accounts which live inside the OU as well as any child OUs (remember that an OU can be a member of an OU).
- **Root level:** If an SCP is applied at this level, it will apply to all AWS accounts inside the organization.

Things can start to get really interesting when you have an SCP applied at multiple levels. The *intersection* of the policies at the root, OU, account, and IAM levels is evaluated and will determine whether or not an API call is allowed to be made. For example, someone belonging to an IAM role which has full administrator access to an account still won't be able to call any EC2 APIs if any of the SCPs above it (account, OU, root) deny EC2 access.

In the following example, we have a top-level OU, Austero Bluth, with an SCP which allows access to all AWS resources for all OUs and accounts underneath it:



Organization hierarchy and policies

Austero Bluth has two child OUs; let's focus on Sudden Valley. It has an SCP which allows only EC2 and S3. By using a whitelisting approach, anything except these two services will be denied. Remembering that SCPs act like a filter, any OU or AWS account living underneath the Sudden Valley OU will, at most, have access to EC2 and S3.

The **Sitwell Enterprises Account** also has an SCP attached to it. This particular SCP allows S3 and SQS. Note that the SQS statement will have no effect here because the Sitwell account is inside an OU which does not allow SQS. Also note that this account has no access to EC2 despite the **Sudden Valley OU** allowing it; this is because EC2 wasn't explicitly allowed in the SCP attached to the account.

At the IAM level, we have a role in the Sitwell AWS account which allows full administrator privileges. But, because the intersection of the SCPs governing this account will only allow S3, anyone using this role will be denied access if they attempt to use EC2 or SQS, for example.

Let's also take a look at the **Bluth Company Account**. The SCP which is attached to it allows full AWS access; however, it lives inside an OU (Balboa Bay) which only allows EC2, RDS, and S3. There is an IAM role inside this account which also allows full admin access but, again, administrators in this account will be limited to EC2, RDS, and S3.

{

"Version":"2012-10-17", "Statement":[

{

"Effect":"Allow", "Action":["EC2:\*","S3:\*"], "Resource":"\*"

}

]

}

```
<strong> aws organizations create-policy </strong> <strong> --content "$(tr -d '\n' < my-  
policy-file.json)" </strong> <strong> --description "A policy description goes here" </strong>  
<strong> --name "My policy" </strong>  
  
<strong> --type SERVICE_CONTROL_POLICY</strong>  
  
<strong> aws organizations attach-policy </strong> <strong> --target-id <ou-or-aws-  
account-id> </strong> <strong> --policy-id <policy-id></strong>  
  
<strong> aws organizations list-targets-for-policy </strong> <strong> --policy-id  
<policy-id></strong>
```



# How it works...

Again, the policies you add will act as a filter at each level of your organizational structure. With this in mind, it might be a good time to point out that testing your policies on a single account before applying them organization-wide will save you a lot of heartache. Making sweeping changes to an SCP living at the top of your organization may create an unforeseen situation at the AWS account level at the bottom of the chain. A local admin in an AWS account is not able to override SCPs.



# There's more...

- At the time of publishing, you are only able to have a single root inside an organization (it's created automatically for you when you create an organization).
- For obvious reasons, the master account is not affected by any SCPs which are attached to it. You may also notice that it's technically possible to place the master account in an OU; again, it will be unaffected by any SCPs which have been attached to that OU.
- Since the master account is unaffected by SCPs, it's a good idea to leave it as empty as possible and to not create any resources in it. Use child AWS accounts instead so you can apply fine-grained controls to them.
- SCPs are required on each OU and account but shouldn't be considered the only form of access control for your AWS accounts. Apply IAM where appropriate.
- When creating our policy, we have to specify a `--type` parameter. At the time of publishing, AWS only supports one variant of OCP: `SERVICE_CONTROL_POLICY`.
- As much as possible, follow the principle of least privilege. You want to give your AWS accounts access to only the services they need. This helps you mitigate damage caused by misclicks, programming errors, or compromised accounts.
- In the long run, you may find it advantageous to not assign controls at the root level. Instead, you may be better off adding *all* accounts to an OU and applying your controls to the OU instead.
- Your policies can take a whitelisting or blacklisting approach. In this recipe, we've used a whitelist approach, but you may instead prefer to allow your OUs and accounts to use all services except the ones you explicitly disallow. You should choose one of these approaches and stick with it, as mixing the two will cause you lots of confusion down the road.
- Unlike IAM policies, you can't specify conditions in SCP documents and `Resource` *must* be `*`.



## See also

- The *Federating with your AWS account* recipe in [Chapter 8, Security and Identity](#) for some discussion around IAM roles



# Storage and Content Delivery

In this chapter, we will cover the following recipes:

- Hosting a static website
- Caching a website
- Working with network storage
- Backing up data for compliance



# Introduction

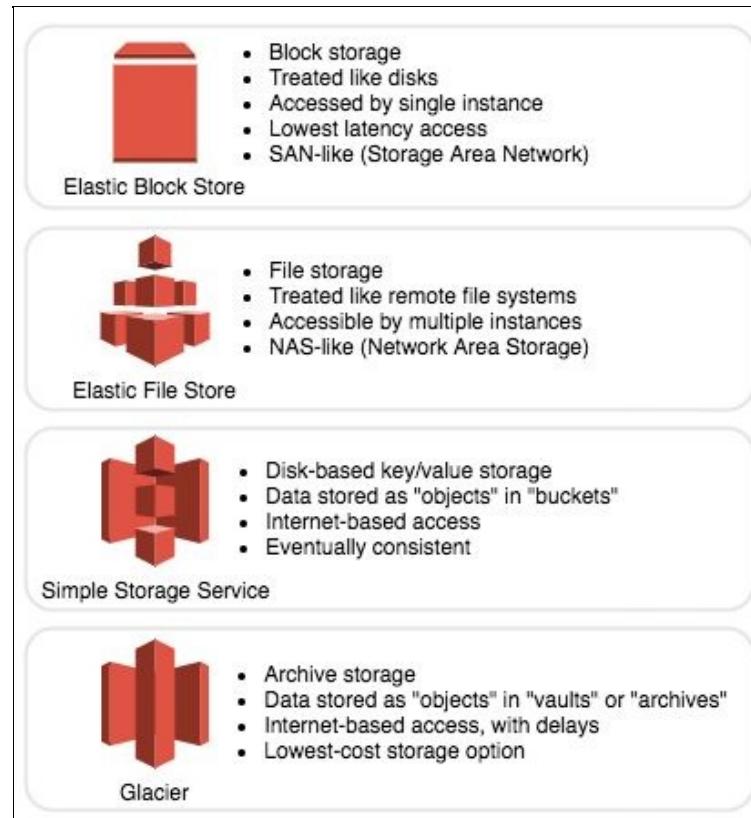
Each of these recipes is backed by a CloudFormation template that makes them quick and easy to reproduce and modify.



# Storage

Storage is an integral part of any organization's cloud usage. When used correctly, servers are short-lived and replaceable. This means that having a durable, available storage service is critical to persisting and sharing state.

Here is a high-level summary of the storage services AWS offers:



Storage services from AWS



# Elastic Block Store

**Elastic Block Store (EBS)** provides block-device storage as volumes to EC2 instances. It behaves similarly to a **storage area network (SAN)** and offers the lowest-latency access of the various storage services offered. EBS volumes can only be accessed by one instance at a time. The size of a volume must be specified when they are provisioned, and cannot be changed after.

Volumes are hosted on redundant hardware in a specific AZ, but they do not offer redundancy across AZs.

Some recommended use cases for EBS are:

- Instance boot volumes
- Intensive data processing
- Transactional writes

We will cover EBS in more detail in the [Chapter 4, Using AWS Compute](#), as its primary use is as the underlying storage for EC2 instances.



# Elastic File System

**Elastic File System (EFS)** provides a file-storage service that can be accessed simultaneously by many instances, similar to **Network Attached Storage (NAS)**. While not as fast as EBS, it still provides low-latency access. As it may be accessed by multiple clients at a time, it can reach much higher levels of throughput than EBS. EFS filesystems also in size scale dynamically and so do not need to be preallocated or modified during use. Filesystems are stored redundantly across AZs.

Some recommended use cases for EFS are:

- Home directories
- Serving shared web content
- Content management



*EFS performance scales according to the filesystem size. As the filesystem size is not preallocated, the only way to increase your performance is to add more data to it.*



# Simple Storage Service

**Simple Storage Service (S3)** provides a web-based service for hosting files. Files are referred to as **objects** and grouped in **buckets**. Objects are effectively a key-value pair, similar to a document database. Keys are used like file paths, with / used as a separator and grouping character. Buckets can be easily accessed like a website via an automatically generated domain name.



*Due to being associated with a domain name, bucket names must be globally unique.*

Some recommended usecases for S3 are:

- Static website assets
- Sharing large files
- Short-term (a.k.a. *warm*) backups



# Glacier

**Glacier** is a companion service to S3, but is the **cold** storage option. Cold storage is a service where you are not able to directly access your data; you must lodge a request for data to be restored (to S3), and you are notified when it is ready. A physical example of cold storage might be backup tapes that are stored in a secure location. Similar to S3, files are referred to as *objects*. Files are grouped together and stored in **archives**. Archives can be created and deleted, but never modified. Archives are grouped together in to **vaults**, which allow you to control access.

The shortest restoration time is 1-5 minutes (with limitations). Standard restoration times take 3-5 hours, with some other options available.

Some recommended usecases for Glacier are:

- Long-term (a.k.a. *cold*) backups
- Compliance backups



# Content delivery

Content delivery is aimed at quickly and efficiently distributing your content to users. The best practice way to do this is to leverage a **Content Delivery Network (CDN)**. Amazon's CDN service is **Amazon CloudFront**.

While AWS currently has 14 regions, it has an additional 68 edge locations that can be used as part of CloudFront. This gives you a massive global network of resources you can use to improve your users' experience of your application.

CloudFront works closely with S3 to serve static assets. In addition to this, it can be configured to cache dynamic content. This gives you an easy way to improve the performance of applications that are not even aware of CloudFront.

CloudFront websites are referred to as **distributions** which speaks to their CDN role.



*Distributions can also be used to provide a common frontend for multiple, disparate, sources of content.*



# Hosting a static website

It's really easy to host a static website on AWS. It turns out it's also dirt cheap, fast, reliable, and massively scalable too.

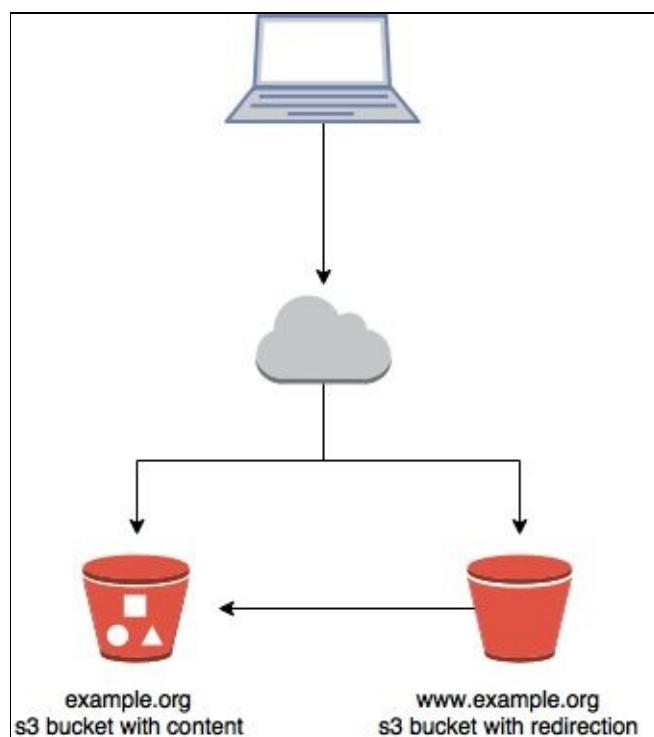
You do this by storing your content in an S3 bucket and configuring that bucket to behave like a website.

It's important to note that we're talking about static content only. This method doesn't work for websites requiring server-side processing or some other backend functionality. WordPress, for example, requires PHP which means you need a fully functional web server to run it. S3 won't interpret PHP pages for you, it will just serve files straight to the browser.

So, why would you want to host a static website in S3? Common scenarios we see are:

- Simply, your website is completely static and you don't change it very often.
- Your company is launching a new product or service. You're expecting very large numbers of customers to visit a mini-site within a short time period; likely more traffic than your existing web hosting environment can handle.
- You need somewhere to host a failover or *down for maintenance* style page which is separate from your existing web hosting environment.

 *HTTPS is not supported by S3 when it is used to serve static content.*





# How to do it...

This recipe provides you with the CloudFormation necessary to create:

- An S3 bucket for hosting your content
- A Route 53 hosted zone and necessary DNS records
- A redirection from `www` to `root/apex` for your domain

After running this CloudFormation you will of course need to upload your content to the buckets which CloudFormation created for you.

Resources:

ApexBucket:

Type: AWS::S3::Bucket Properties:

BucketName: !Ref DomainName WWWBucket:

Type: AWS::S3::Bucket Properties:

BucketName: !Sub - www.\${Domain}

- Domain: !Ref DomainName

<strong> aws s3 website s3://example.org/

--index-document index.html --error-document error.html</strong>

{

"Version": "2012-10-17", "Statement": [{

    "Sid": "Allow Public Access to everything in our bucket", "Effect": "Allow",  
    "Principal": "\*", "Action": "s3:GetObject", "Resource": "arn:aws:s3:::example.org/\*"

}

]

}

ApexBucket:

Type: AWS::S3::Bucket Properties:

BucketName:

    Ref: DomainName AccessControl: PublicRead WebsiteConfiguration: IndexDocument:  
    index.html ErrorDocument: error.html WWWBucket:

Type: AWS::S3::Bucket Properties:

BucketName:

Fn::Join: [ ., [ www, Ref: DomainName ] ]

AccessControl: BucketOwnerFullControl WebsiteConfiguration:

RedirectAllRequestsTo: HostName:

Ref: ApexBucket

DNSHostedZone:

Type: "AWS::Route53::HostedZone"

Properties: Name: Ref: DomainName

us-east-1:

S3HostedZoneID: Z3AQBSTGFYJSTF

S3AliasTarget: s3-website-us-east-1.amazonaws.com us-east-2:

S3HostedZoneID: Z2O1EMRO9K5GLX

S3AliasTarget: s3-website.us-east-2.amazonaws.com

DNSRecords:

Type: "AWS::Route53::RecordSetGroup"

Properties:

HostedZoneId: Ref: DNSHostedZone RecordSets:

- Name:

Ref: DomainName Type: A

AliasTarget: HostedZoneId: Fn::FindInMap: [ RegionMap, Ref: "AWS::Region", S3HostedZoneID ]

DNSName:

Fn::FindInMap: [ RegionMap, Ref: "AWS::Region", S3AliasTarget ]

- Name:

Fn::Join: [ ., [ www, Ref: DomainName ] ]

Type: CNAME

TTL: 300

ResourceRecords: - Fn::GetAtt: WWWBucket.DomainName

<strong> aws cloudformation create-stack \ </strong> <strong> --stack-name static-website-1 \ </strong> <strong> --template-body file://03-hosting-a-static-website.yaml \ </strong> <strong> --parameters \</strong> <strong>

ParameterKey=DomainName,ParameterValue=<your-domain-name></strong>

```
<html>
```

```
 <head> <title>Welcome to exmaple.org</title> </head> <body>
<h1>example.org</h1> <p>Hello World!</p> </body> </html>
```

```
<html>
```

```
 <head> <title>Error</title> </head> <body> <h1>example.org</h1> <p>Something
went wrong!</p> </body> </html>
```



# How it works...

That's it! As soon as S3 has an `index.html` file to serve up, you will be hosting a single-page website on S3. Go ahead and test it out. The supplied CloudFormation example will output a URL you can use to see your new website. After you've verified it's working, you can go ahead and upload your real static website and enjoy fast, cheap, and server-free hosting.



# **There's more...**

Let's look at some additional things to consider.



# Delegating your domain to AWS

While we've created a hosted zone and some DNS records in Route 53, no one can actually see them yet. In order to send your website visitors to your new S3 static website, you'll need to delegate your domain to Route 53. This is left to you as an exercise; however, there are some important things to remember:

- The DNS servers to delegate your domain to can be found in the NS record for your hosted zone.
- If your domain is already live and production-like, you'll need to make sure all your DNS records for your zone are recreated in Route 53, including things such as MX records, which are critical for the continuity of your e-mail service.
- Before delegating to AWS, you may consider reducing the TTL values on your DNS records. This will be useful if for some reason you need to re-delegate or make changes to them. Once your DNS setup is stable, you can increase TTLs.

ApexBucket:

Type: AWS::S3::Bucket Properties: BucketName: !Ref DomainName AccessControl: PublicRead WebsiteConfiguration: IndexDocument: index.html ErrorDocument: error.html CorsConfiguration: CorsRules: - AllowedOrigins: - example.net - www.example.net - example.com - www.example.com AllowedMethods: - GET

MaxAge: 3000

AllowedHeaders: - Content-\*

- Host



# Caching a website

In this recipe, we'll show you how to use AWS CloudFront to cache your website.

The primary reasons you'll want to consider doing this are as follows:

- Copies of your content will be geographically located closer to your end users, thus improving their experience and delivering content to them faster.
- The burden for serving content will be removed from your fleet of servers. This could potentially result in a large cost saving if you're able to turn off some servers or reduce your bandwidth bill.
- You may need to be shielded from large and unexpected spikes in traffic.
- While not the focus of this chapter, CloudFront gives you the ability to implement **Web Application Firewall (WAF)** as an added layer of protection from the bad guys.



*Unlike most AWS services, which are region specific, CloudFront is a global service.*



# Getting ready

First of all, you're going to need a publicly accessible website. This could be a static website hosted in S3, or it could be a dynamically generated website hosted in EC2. In fact, your website doesn't even need to be hosted in AWS in order to use CloudFront. As long as your website is publicly accessible, you should be good to go.

You'll also need to have the ability to modify the DNS records for your website. Instead of pointing to your web server (or S3 bucket), we'll eventually point them to CloudFront.



# About dynamic content

If your website consists of mostly dynamic content, you can still benefit from implementing CloudFront.

First of all, CloudFront will maintain a pool of persistent connections with your origin servers. This lessens the time it takes for files to be served to your users because the number of three-way handshakes they'll need to perform is reduced.

Second, CloudFront implements some additional optimizations around TCP connections for high performance. More data is able to initially be transferred over the wire because CloudFront uses a wider initial TCP window.

Finally, implementing a CDN such as CloudFront *does* give you the opportunity to review your caching strategy and how you use cache-control headers. If your home page is dynamically generated, you'll get some benefit straight away by serving it via CloudFront, but the benefits will be much greater if you were to let CloudFront cache it for a few minutes. Again, cost, end user, and backend performance are all things you should take into consideration.



# Configuring CloudFront distributions

Distributions can be configured with a fairly wide array of options. Our recipe is going to be quite simple so that you can get up and running as quickly as possible. But we will talk through some of the more common configuration options:

- Origins: A distribution needs to have at least one origin. An origin, as the name indicates, is where your website content originates from your public-facing website. The properties you'll most likely be concerned with are:
  - Origin Domain Name: This is the hostname of your public-facing website. The CloudFormation template we supply accepts this hostname as a parameter.
  - Origin Path: It's possible to configure the distribution to fetch content from a directory or subfolder at the origin, for example, `/content/images` if you were using CloudFront to cache images only. In our case, we are caching our entire website, so we don't specify an Origin Path at all.
  - Origin ID: This is particularly important when you are using nondefault cache behavior settings and therefore have configured multiple origins. You need to assign a unique ID to the origins so that the cache behaviors know which origin to target. There'll be more discussion on cache behaviors later.
  - HTTP Port, HTTPS Port: If your origin is listening on nonstandard ports for HTTP or HTTPS, you would use these parameters to define those ports.
  - Origin Protocol Policy: You are able to configure the distribution to talk to your origin via:
    - HTTP Only
    - HTTPS Only
    - Match Viewer

The Match Viewer option forwards requests to the origin based on which protocol the user requested with in their browser. Again, we are keeping things quite simple in this recipe, so we'll be opting for HTTP Only.

- Logging: Keep in mind that because less traffic will be hitting your origin, fewer access logs will also be captured. It makes sense to have CloudFront keep these logs for us in an S3 bucket. This is included in the CloudFormation provided with this recipe:
  - **Cache behaviors:** In this recipe, we'll configure a single (default) cache behavior, which will forward all requests to our origin.
  - **CloudFront:** It allows you to get quite fine grained with the behaviors you configure. You might, for example, want to apply a rule to all the `.js` and `.css` files on your origin. Perhaps you want to forward query strings to the origin for these file types. Similarly, you might want to ignore the TTL the origin is trying to set for image files, instead telling CloudFront to cache for a minimum of 24 hours.
- Aliases: These are additional hostnames you want the distribution to serve traffic for. For example, if your Origin Domain Name is configured to `loadbalancer.example.org`, then you probably want aliases

that look something like this:

- example.org
- www.example.org

The CloudFormation template provided with this recipe expects one or more aliases to be provided in the form of a comma-delimited list of strings.

- Allowed HTTP Methods: By default, CloudFront will only forward GET and HEAD requests to your origin. This recipe doesn't change those defaults, so we don't declare this parameter in the provided template. If your origin is serving dynamically generated content, then you will likely want to declare this parameter and set its values to GET, HEAD, OPTIONS, PUT, POST, PATCH, and DELETE.
- TTLs (minimum/maximum/default): You can optionally define how long you'd like objects to stay in CloudFront's caches before expiring and being refetched from the origin. Again, we've opted to stick to CloudFront's default values to keep this recipe simple, so we've omitted this parameter from our template. The defaults are as follows:
  - Minimum TTL: 0 seconds
  - Default TTL: 1 day
  - Maximum TTL: 1 year
- Price Class: By default, CloudFront will serve your content from all of its edge locations, giving you the maximum performance possible. We're going to deploy our distribution using the lowest possible price class, Price Class 100. This corresponds to edge locations in the United States, Canada, and Europe. Users from Australia would not benefit too much from this Price Class, but you're also paying less for it. Price Class 200 adds Asian regions, and Price Class All adds South America and Australia.

A comprehensive list and detailed explanation on which values can be specified when creating a CloudFront distribution can be found here at <http://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/distribution-web-values-specify.html>.

AWSTemplateFormatVersion: '2010-09-09'

Parameters:

OriginDomainName:

Description: The hostname of your origin (i.e. www.example.org.s3-website-ap-southeast-2.amazonaws.com) Type: String

Aliases:

Description: Comma delimited list of aliases (i.e. example.org,www.example.org)

Type: CommaDelimitedList Resources:

DistributionALogBucket: Type: AWS::S3::Bucket DistributionA:

Type: AWS::CloudFront::Distribution Properties:

DistributionConfig: Origins:

- DomainName:

Ref: OriginDomainName Id: OriginA

CustomOriginConfig: OriginProtocolPolicy: http-only Enabled: true

Logging:

IncludeCookies: false Bucket:

Fn::GetAtt: DistributionALogBucket.DomainName Prefix: cf-distribution-a Aliases:

Ref: Aliases

DefaultCacheBehavior: TargetOriginId: OriginA ForwardedValues:

QueryString: false ViewerProtocolPolicy: allow-all PriceClass: PriceClass\_100

Outputs:

DistributionDomainName: Description: The domain name of the CloudFront Distribution Value:

Fn::GetAtt: DistributionA.DomainName LogBucket:

Description: Bucket where CloudFront logs will be stored Value:

Ref: DistributionALogBucket

```
<strong> aws cloudformation create-stack \ </strong> <strong> --stack-name cloudfont-cache-1 \ </strong> <strong> --template-body file://03-caching-a-website.yaml \ </strong>
<strong> --parameters \ </strong> <strong>
ParameterKey=OriginDomainName,ParameterValue=<your-domain-name> \ </strong>
<strong> ParameterKey=Aliases,ParameterValue='<alias-1>\,<alias-2>'</strong>
```



# Working with network storage

In this recipe, we will use the Amazon EFS to provide network-based storage to instances.

Some of the benefits of using EFS compared to other AWS services are as follows:

- Guaranteed write order between distributed clients
- Automatic resizing—no need to preallocate and no need to downsize
- You only pay for the space you use (per GB)—no transfer or extra costs



# Getting ready

This example works with the default VPC and subnets, present in all AWS accounts when they are created. Even if you have changed your network configuration, all you need is a working VPC with two or more subnets in different AZs for this recipe.

AWSTemplateFormatVersion: "2010-09-09"

Description: Create an EFS file system and endpoints.

VpcId:

Description: VPC ID that contains the subnets that will access the file system Type: AWS::EC2::VPC::Id SubnetIds:

Description: Subnet IDs allowed to access the EFS file system Type: List<AWS::EC2::Subnet::Id>

FileSystem:

Type: AWS::EFS::FileSystem Properties:

FileSystemTags:

- Key: Name

Value:

Fn::Sub: "\${AWS::StackName} EFS File System"

PerformanceMode: generalPurpose

MountTargetA:

Type: AWS::EFS::MountTarget Properties:

FileSystemId:

Ref: FileSystem

SecurityGroups:

- Fn::GetAtt: MountTargetSecurityGroup.GroupId SubnetId:

Fn::Select: [ 0, Ref: SubnetIds ]

MountTargetB:

Type: AWS::EFS::MountTarget Properties:

FileSystemId:

Ref: FileSystem

SecurityGroups:

- Fn::GetAtt: MountTargetSecurityGroup.GroupId SubnetId:

Fn::Select: [ 1, Ref: SubnetIds ]

MountTargetSecurityGroup:

Type: AWS::EC2::SecurityGroup Properties:

GroupDescription: EFS endpoint security group Tags:

- Key: Name

Value: MountTargetSecurityGroup VpcId:

Ref: VpcId

MountTargetAccessSecurityGroup:

Type: AWS::EC2::SecurityGroup Properties:

GroupDescription: EFS endpoint access security group Tags:

- Key: Name

Value: MountTargetAccessSecurityGroup VpcId:

Ref: VpcId

MountTargetIngress:

Type: AWS::EC2::SecurityGroupIngress Properties:

FromPort: 2049

GroupId:

Fn::GetAtt: MountTargetSecurityGroup.GroupId IpProtocol: tcp

SourceSecurityGroupId: Fn::GetAtt: MountTargetAccessSecurityGroup.GroupId

ToPort: 2049

MountTargetEgress: Type: AWS::EC2::SecurityGroupEgress Properties:

DestinationSecurityGroupId: Fn::GetAtt: MountTargetAccessSecurityGroup.GroupId

FromPort: 2049

GroupId:

Fn::GetAtt: MountTargetSecurityGroup.GroupId IpProtocol: tcp

ToPort: 2049

MountTargetAccessIngress:

Type: AWS::EC2::SecurityGroupIngress Properties:

FromPort: 22

GroupId:

Fn::GetAtt: MountTargetAccessSecurityGroup.GroupId IpProtocol: tcp

CidrIp: 0.0.0.0/0

ToPort: 22

MountTargetAccessEgress: Type: AWS::EC2::SecurityGroupEgress Properties:

DestinationSecurityGroupId: Fn::GetAtt: MountTargetSecurityGroup.GroupId

FromPort: 2049

GroupId:

Fn::GetAtt: MountTargetAccessSecurityGroup.GroupId IpProtocol: tcp

ToPort: 2049

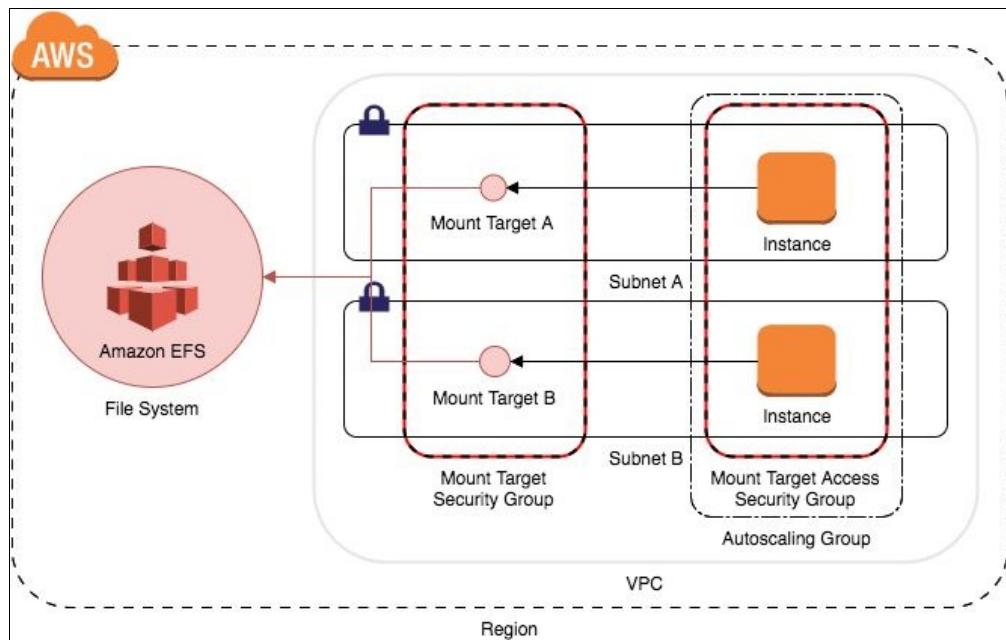
```
<strong> aws cloudformation create-stack </strong> <strong> --stack-name wwns1 \
</strong> <strong> --template-body file://03-working-with-network-storage.yaml </strong>
```

</strong> <strong> --parameters \</strong> <strong>  
ParameterKey=VpcId,ParameterValue=<your-vpc-id> \ </strong> <strong>  
ParameterKey=SubnetIds,ParameterValue="<sub>1</sub><subnet-id-1>\, \  
  
<sub\_2>"</strong>



# How it works...

Here is what the created resources will look like at the end of the recipe:



Working with network storage We start by creating the standard CloudFormation template properties in step 1.

In step 2, you define the template's parameters that will be used when configuring the resources.

Steps 3 and 4 are where the EFS resources are specified. They consist of an EFS filesystem and mount targets in each of the AZs that will access it.

We then create the security groups in steps 5 and 6: one for the mount targets and one for the instances that are allowed to connect to the mount targets.

As these two security groups contain two-way (or circular) references to each other, we must define the rules between them in separate resources in steps 7 and 8.

In step 9, you save the template with a specific filename so that it can be referenced in the command to launch the stack in step 10.

MountPoint:

Description: The path on disk to mount the EFS file system Type: String

Default: /mnt/efs KeyName:

Description: The SSH key pair allowed to connect to the client instance

Type: AWS::EC2::KeyPair::KeyName

AutoScalingGroup:

Type: AWS::AutoScaling::AutoScalingGroup DependsOn: MountTargetA Properties:

MinSize: 2

MaxSize: 2

LaunchConfigurationName: Ref: LaunchConfiguration Tags:

- Key: Name

Value:

Fn::Sub: "\${AWS::StackName} EFS Client"

PropagateAtLaunch: true VPCZoneIdentifier: Ref: SubnetIds

LaunchConfiguration:

Type: AWS::AutoScaling::LaunchConfiguration DependsOn: FileSystem Properties:

ImageId: ami-1e299d7e SecurityGroups:

- Ref: MountTargetAccessSecurityGroup InstanceType: t2.micro KeyName:

Ref: KeyName

UserData:

Fn::Base64:

Fn::Sub: |

```
#!/bin/bash -xe

mkdir -p ${MountPoint}

echo 'Waiting for mount target DNS to propagate'

sleep 90

echo '${FileSystem}.efs.${AWS::Region}.amazonaws.com:/

${MountPoint} nfs4

nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600, retrans=2 0 0' >> /etc/fstab

mount -a\nchown ec2-user: ${MountPoint}\n"

<strong> aws cloudformation create-stack \ </strong> <strong> --stack-name wwns1 \
</strong> <strong> --template-body file://03-working-with-network-storage.yaml \
</strong> <strong> --parameters </strong> <strong>
ParameterKey=VpcId,ParameterValue=<vpc-id> </strong> <strong>
ParameterKey=SubnetIds,ParameterValue='<subnet-id-1>\, \
<subnet-id-1>' </strong> <strong> ParameterKey=MountPoint,ParameterValue=
<local-path-to-mount-efs> </strong> <strong>
ParameterKey=KeyName,ParameterValue=<existing-key-pair-name></strong>
```

Once the new stack is ready, you will be able to SSH to your instances and verify that they have mounted the EFS filesystem.



# Backing up data for compliance

We work with a lot of companies (especially in the finance industry) that have strict rules around the minimum time data needs to be kept for. This can become quite onerous and expensive if you need to keep customer records for a minimum of 7 years, for example.

Using S3, Glacier, and life cycle rules, we can create a flexible long-term backup solution while also automating the archiving and purging of backups and reducing costs.

We are also going to utilize *versioning* in order to mitigate the damage caused by a file being accidentally deleted or overwritten in our backup bucket.

AWSTemplateFormatVersion: '2010-09-09'

Parameters:

ExpirationInDays:

Description: The maximum amount of time to keep files for

Type: Number

Default: 2555

TransitionToInfrequentAccessInDays: Description: How many days until files are moved to the Infrequent Access class Type: Number

Default: 30

TransitionToGlacierInDays: Description: How many days until files are moved to Glacier

Type: Number

Default: 60

PreviousVersionsExpirationInDays: Description: The maximum amount of time to keep previous versions of files for

Type: Number

Default: 60

PreviousVersionsToInfrequentAccessInDays: Description: How many days until previous versions of files are moved to the Infrequent Access class Type: Number

Default: 30

Resources:

BackupBucket:

Type: AWS::S3::Bucket

Properties:

VersioningConfiguration: Status: Enabled

LifecycleConfiguration: Rules:

- Status: Enabled

ExpirationInDays:

Ref: ExpirationInDays

Transitions:

- StorageClass: STANDARD\_IA TransitionInDays:

Ref: TransitionToInfrequentAccessInDays - StorageClass: GLACIER

TransitionInDays:

Ref: TransitionToGlacierInDays NoncurrentVersionExpirationInDays: Ref: PreviousVersionsExpirationInDays NoncurrentVersionTransitions: - StorageClass: STANDARD\_IA TransitionInDays:

Ref: PreviousVersionsToInfrequentAccessInDays

Outputs:

BackupBucket:

Description: Bucket where backups are stored Value:

Ref: BackupBucket

```
<strong>aws cloudformation create-stack </strong> <strong> --stack-name backup-s3-glacier-1 </strong> <strong> --template-body file://03-backing-up-data-for-compliance.yaml</strong>
```

Once the stack has been created, you'll be all set to start copying backups to the S3 bucket and to start worrying less about your backups' life cycle and management. If you decide that the expiry or transition times need to change after you've created the bucket, you can do this by simply updating the parameters for the stack.



# Using AWS Compute

In this chapter, we will cover:

- Creating a key pair
- Launching an instance
- Attaching storage
- Securely accessing private instances
- Auto scaling an application server
- Creating machine images
- Creating security groups
- Creating a load balancer



# Introduction

**Elastic Cloud Compute (EC2)** is by far the most utilized and complex service in the AWS catalogue. More than *just virtual machines*, EC2 provides a framework of sub-services to help you secure and manage your instances elastically.



# **Creating a key pair**

A key pair is used to access your instances via SSH. This is the quickest and easiest way to access your instances.



# Getting ready

To perform this recipe, you must have your AWS CLI tool configured correctly.



# How to do it...

1. Create the key pair, and save it to disk:

```
aws ec2 create-key-pair \
--key-name MyEC2KeyPair \
--query 'KeyMaterial' \
--output text > ec2keypair.pem
```

2. Change the permissions on the created file:

```
chmod 600 ec2keypair.pem
```



# How it works...

This call requests a new private key from EC2. The response is then parsed using a JMESPath query, and the private key (in the `KeyMaterial` property) is saved to a new key file with the `.pem` extension.

Finally, we change the permissions on the key file so that it cannot be read by other users—this is required before SSH will allow you to use it.



# Launching an instance

There will be scenarios—usually when testing and developing your infrastructure code—when you need quick access to an instance. Creating it via the AWS CLI is the quickest and most consistent way to create one-off instances.

There are other recipes in the book that will require a running instance. This recipe will get you started.



# Getting ready

For this recipe, you must have an existing key pair.

In this recipe, we are launching an instance of AWS Linux using an AMI ID in the `us-east-1` region. If you are working in a different region, you will need to update your `image-id` parameter.

You must have configured your AWS CLI tool with working credentials.



# How to do it...

Run the following AWS CLI command, using your own key-pair name:

```
aws ec2 run-instances \
--image-id ami-9be6f38c \
--instance-type t2.micro \
--key-name <your-key-pair-name>
```



# How it works...

While you can create an instance via the AWS web console, it involves many distracting options. When developing and testing, the CLI tool is the best way to provision instances.

While the `key-name` argument is optional, you will not be able to connect to your instance unless you have pre-configured some other way of logging in.



*The `t2.micro` instance type used in this recipe is included in the AWS free tier. You can run one micro instance per month for free during the first 12 months of your usage. See <https://aws.amazon.com/free> for more information.*

As no VPC or security groups are specified, the instance will be launched in your account's default VPC and security group. The default security group allows access from anywhere, on all ports, and so is not suitable for long-lived instances. You can modify an instance's security groups after it is launched, without stopping it.



# There's more...

If you have created your own AMI, then you can change the `image-id` argument to quickly launch your specific AMI.

You may also want to take note of the `InstanceId` value in the response from the API, as you may need it for future commands.



## See also

- The *Creating a key pair* recipe
- The *Creating machine images* recipe



# Attaching storage

Ideally, you will have defined all your storage requirements up-front as code using a service such as CloudFormation. However, sometimes that is not possible due to application restrictions or changing requirements.

You can easily add additional storage to your instances while they are running by attaching a new volume.



# Getting ready

For this recipe, you will need the following:

- A running instance's ID. It will start with `i-` followed by alphanumeric characters.
- The AZ the instance is running in. This looks like the region name with a letter after it; for example, `us-east-1a`.

In this recipe, we are using an AWS Linux instance. If you are using a different operating system, the steps to mount the volume will be different. We will be running an instance in the AZ `us-east-1a`.

You must have configured your AWS CLI tool with working credentials.

<strong> aws ec2 create-volume --availability-zone us-east-1a </strong>  
<strong> aws ec2 attach-volume </strong> <strong>--volume-id <your-volume-id> </strong> <strong>--instance-id <your-instance-id> </strong> <strong>--device /dev/sdf</strong>

<strong> mount /dev/xvdf /mnt/volume</strong>



# How it works...

In this recipe, we start by creating a volume. Volumes are created from snapshots. If you do not specify a snapshot ID it uses a blank snapshot, and you get a blank volume.

While volumes are hosted redundantly, they are only hosted in a single AZ, so must be provisioned in the same AZ the instance is running in.

The `create-volume` command returns a response that includes the newly created volume's `VolumeId`. We then use this ID in the next step.



*It can sometimes take a few seconds for a volume to become available. If you are scripting these commands, use the `aws ec2 wait` command to wait for the volume to become available.*

In step 3, we attach a volume to the instance. When attaching to an instance, you must specify the name of the device that it will be presented to the operating system as. Unfortunately, this does not guarantee what the device will appear as. In the case of AWS Linux, `/dev/sdf` becomes `/dev/xvdf`.



*Device naming is kernel-specific, so if you are using something other than AWS Linux, the device name may be different. See [http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/device\\_naming.html](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/device_naming.html) for full details.*



## See also

- The *Launching an instance* recipe
- The *Working with network storage* recipe in [Chapter 3, Storage and Content Delivery](#)

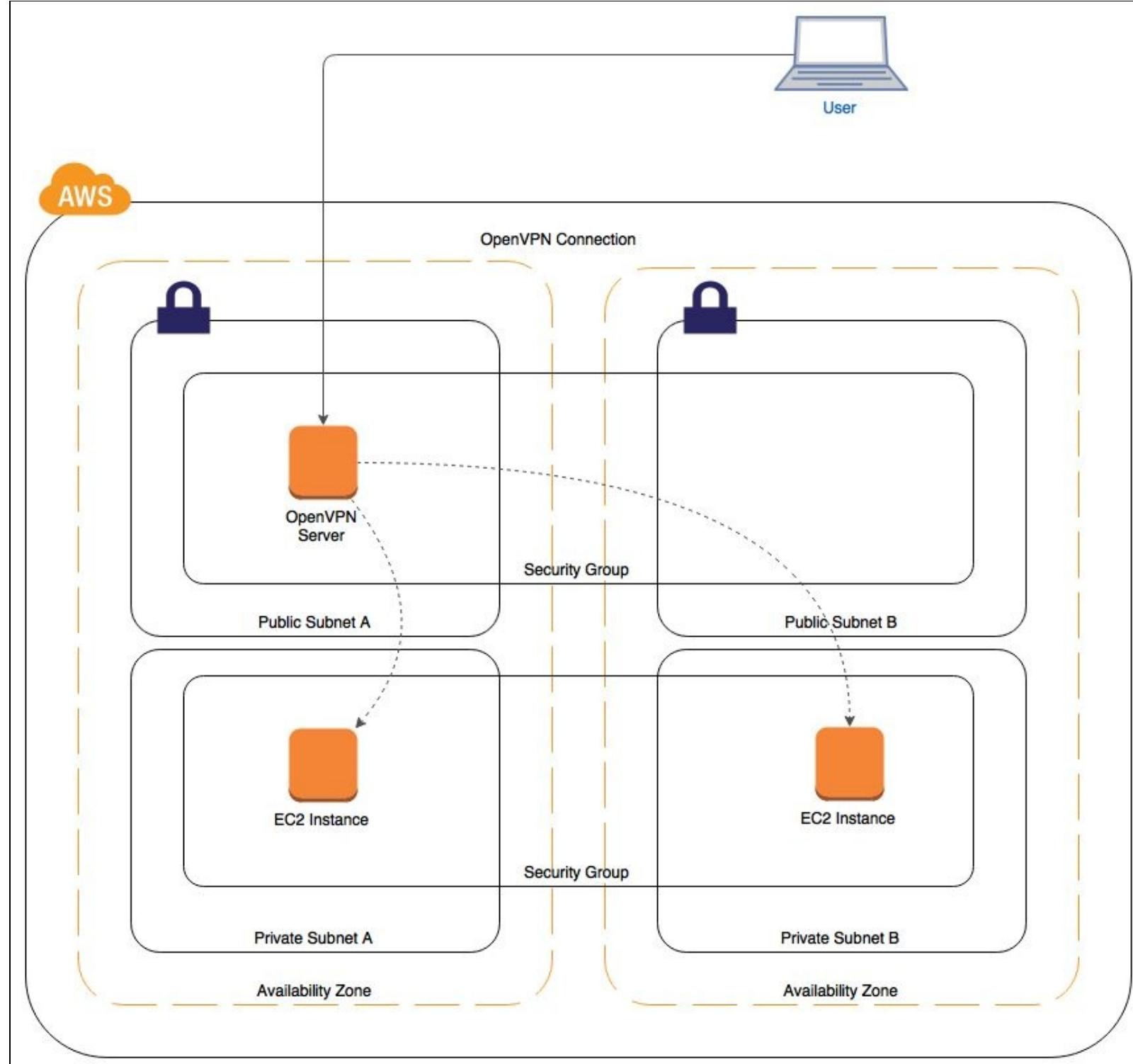


# Securely accessing private instances

Any instance or resource living in a private subnet in your VPC will be inaccessible from the Internet. This makes good sense from a security perspective because it gives your instances a higher level of protection.

Of course, if they can't be accessed from the Internet, then they're not going to be easy to administer.

One common pattern is to use a VPN server as a single, highly controlled, entry point to your private network. This is what we're going to show you in this recipe, as pictured in the following diagram:





# Getting ready

We're going to use OpenVPN for this example. They provide a free (for up to two users) AMI in the AWS marketplace, which has OpenVPN already installed and configured. You'll need to accept the terms and conditions for using this AMI. You can do so by visiting the AMI's marketplace page at <https://aws.amazon.com/marketplace/pp/B00MI40CAE/>.

You need to decide on a password, which will be your *temporary* admin password. We'll feed this password into a CloudFormation template and then change it after we create our stack.

 *You can use the default VPC for this example.*

Mappings:

AWSRegion2AMI: # Latest OpenVPN AMI at time of publishing: 2.1.4

us-east-1:

AMI: ami-bc3566ab us-east-2:

AMI: ami-10306a75

us-west-2:

AMI: ami-d3e743b3

us-west-1:

AMI: ami-4a02492a eu-west-1:

AMI: ami-f53d7386

eu-central-1: AMI: ami-ad1fe6c2

ap-southeast-1: AMI: ami-a859ffcb ap-northeast-1: AMI: ami-e9da7c88

ap-southeast-2: AMI: ami-89477aea sa-east-1:

AMI: ami-0c069b60

VpcId:

Type: AWS::EC2::VPC::Id Description: VPC where load balancer and instance will launch SubnetId:

Type: List<AWS::EC2::Subnet::Id> Description: Subnet where OpenVPN server will launch (pick at least 1)

InstanceType:

Type: String

Description: OpenVPN server instance type Default: m3.medium KeyName:

Type: AWS::EC2::KeyPair::KeyName Description: EC2 KeyPair for SSH access

AdminPassword:

Type: String

Description: Password for 'openvpn' user Default: openvpn NoEcho: true

AllowAccessFromCIDR:

Type: String

Description: IP range/address to allow VPN connections from Default: "0.0.0.0/0"

VPNSecurityGroup:

Type: AWS::EC2::SecurityGroup Properties:

GroupDescription: Inbound access to OpenVPN server VpcId: !Ref VpcId

SecurityGroupIngress: - CidrIp: !Ref AllowAccessFromCIDR

FromPort: 443

IpProtocol: tcp ToPort: 443

- CidrIp: !Ref AllowAccessFromCIDR

FromPort: 22

IpProtocol: tcp ToPort: 22

- CidrIp: !Ref AllowAccessFromCIDR

FromPort: 1194

IpProtocol: udp ToPort: 1194

OpenVPNInstance:

Type: AWS::EC2::Instance Properties:

ImageId: !FindInMap [ AWSRegion2AMI, !Ref "AWS::Region", AMI ]

InstanceType: !Ref InstanceType KeyName: !Ref KeyName NetworkInterfaces: - AssociatePublicIpAddress: true DeviceIndex: "0"

## GroupSet:

- !Ref VPNSecurityGroup SubnetId: !Select [ 0, Ref: SubnetId ]

## Tags:

- Key: Name

Value: example-openvpn-server UserData:

Fn::Base64: !Sub - |

public\_hostname=openvpn admin\_user=openvpn admin\_pw=\${admin\_pw}

reroute\_gw=1

reroute\_dns=1

- admin\_pw: !Ref AdminPassword

## Outputs:

OpenVPNAdministration: Value:

Fn::Join:

- ""

- - https://

- !GetAtt OpenVPNInstance.PublicIp - /admin/

Description: Admin URL for OpenVPN server OpenVPNClientLogin: Value:

Fn::Join:

- ""

- - https://

- !GetAtt OpenVPNInstance.PublicIp - /

Description: Client login URL for OpenVPN server OpenVPNServerIPAddress: Value:  
!GetAtt OpenVPNInstance.PublicIp Description: IP address for OpenVPN server

<strong> aws cloudformation create-stack \

--template-body file://04-securely-access-private-instances.yaml \

--stack-name example-vpn \

--parameters \

ParameterKey=KeyName,ParameterValue=<key-pair-name> \

ParameterKey=VpcId,ParameterValue=<your-vpc-id> \

ParameterKey=SubnetId,ParameterValue=<your-public-subnet-id> </strong>



# Configuration

- Once your stack is created, you'll want to change the password for the `openvpn` user (administrator). Go to the admin control panel and do this now: `https://<ip-or-hostname-of-vpn-server>/admin`. If your VPN server is operating as expected you'll be greeted with a status page after logging in, as pictured in the following screenshot:

The screenshot shows the 'Status Overview' page of the OpenVPN Access Server. The left sidebar has a 'Status' section with 'Status Overview', 'Current Users', and 'Log Reports'. Below that is a 'Configuration' section with 'License', 'SSL Settings', 'Server Network Settings' (which includes 'VPN Mode' and 'VPN Settings'), 'Advanced VPN', 'Web Server', 'Client Settings', and 'Failover'. Under 'User Management' are 'User Permissions', 'Group Permissions', and 'Revoke Certificates'. The 'Authentication' section includes 'General', 'PAM', 'RADIUS', and 'LDAP'. At the bottom are 'Tools' and 'Profiles'. The main content area shows 'Server Status' (server is currently ON) and a 'Stop the Server' button. It also displays 'Active Configuration' details like Access Server version (2.1.4b), Server Name (openvpn), Authenticate users with (local), Accepting VPN client connections on IP address (eth0: 172.30.0.147), Port for VPN client connections (tcp/443, udp/1194), OSI Layer (3 (routing/NAT)), Clients access private subnets using (NAT), and Node (openvpnras2). To the right is a 'At a glance' summary showing Server Status (on), License (2 users), Current Users (0), and links for Logout and Help.

While you're there, you should create a non-administrator user account. This will be the account you'll use to connect to the VPN. Add this account on the User Permissions page as pictured in the following screenshot:

The screenshot shows the OpenVPN Access Server User interface. The left sidebar contains navigation links for Status, Configuration, User Management, Authentication, and Tools. The main content area is titled "User Permissions" and displays a table of users. The table columns are: Username, Group, More Settings, Allow Admin, Auto-login, Deny Access, and Delete. A new user entry "joebloggs" is shown with a "No Default Group" group selection. Below the table is a checkbox for "Require user permissions record for VPN access" and a "Save Settings" button. The top right corner shows "At a glance" information: Server Status: on, License: 2 users, and Current Users: 0.

Username	Group	More Settings	Allow Admin	Auto-login	Deny Access	Delete
openvpn	No Default Group	Show	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
New Username: joebloggs	No Default Group	Show	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

- Under Server Network Settings, in the Hostname or IP address field, enter the hostname or IP address of the server. This step is important, and when you download your OpenVPN config file from the server (next step), it will make your life much easier if it has the correct hostname or IP address in it. The next screenshot shows what you can expect to see on the Server Network Settings page:

[Logout](#) [Help](#)**Status**[Status Overview](#)  
[Current Users](#)  
[Log Reports](#)**Configuration**[License](#)  
[SSL Settings](#)  
**Server Network Settings**  
[VPN Mode](#)  
[VPN Settings](#)  
[Advanced VPN](#)  
[Web Server](#)  
[Client Settings](#)  
[Failover](#)**User Management**[User Permissions](#)  
[Group Permissions](#)  
[Revoke Certificates](#)**Authentication**[General](#)  
[PAM](#)  
[RADIUS](#)  
[LDAP](#)**Tools**[Profiles](#)

## Server Network Settings

**VPN Server**

**Warning:** Changing the Hostname, Protocol or Port Number after VPN clients are deployed will cause the existing clients to be unusable (until a new client configuration or VPN installer is downloaded from the Client Web Server)

**Hostname or IP Address:** 54.208.141.59**Interface and IP Address**

- Listen on all interfaces  
 eth0: 172.30.0.147

**Protocol**

- TCP  
 UDP  
 Both (Multi-daemon mode)

**Multi-Daemon Mode**

In Multi-Daemon mode, the Access Server will load-balance connecting VPN clients across multiple OpenVPN daemons to fully leverage the capability of multi-core servers. NOTE: It is not recommended to set the number of TCP and UDP daemons to a higher value than the number of processor cores on the machine. Doing so may result in resource exhaustion and system instability.

**Number of TCP daemons:** 1**TCP Port number:** 443**Number of UDP daemons:** 1**UDP Port number:** 1194**Service Forwarding**

When TCP or Multi-daemon mode is chosen for the VPN Server protocol, the VPN Server can optionally provide access to these services through its IP address and port:

- Admin Web Server  
 Client Web Server

**At a glance**Server Status: **on** [More](#)License: **2 users** [Info](#)Current Users: **0** [List](#)



# How it works...

You should now be able to connect to your VPN server. Go to the user login page and log in with the credentials you gave to the previously mentioned non-administrator user:

`https://<ip-or-hostname-of-vpn-server>/`

After logging in, you will have the option to download the OpenVPN client with configuration which is specific to your account. Alternatively, if you already have a VPN client installed, you can just download the configuration on its own.



# There's more...

There are a couple of important points you'll need to keep in mind now that you are up and running with an OpenVPN server:

- If you need to SSH to the instance, you must connect with the username `openvpnas`
- To access your other instances, you'll need to allow connections from the VPN security group created in this recipe



# Auto scaling an application server

**Auto scaling** is a fundamental component of compute in the cloud. It provides not only the ability to scale up and down in response to application load, but also redundancy, by ensuring that capacity is always available. Even in the unlikely event of an AZ outage, the auto scaling group will ensure that instances are available to run your application.

Auto scaling also allows you to pay for only the EC2 capacity you need, because underutilized servers can be automatically de-provisioned.



# Getting ready

You must supply two or more subnet IDs for this recipe to work.

The following example uses an AWS Linux AMI in the `us-east-1` region. Update the parameters as required if you are working in a different region.

AWSTemplateFormatVersion: "2010-09-09"

Description: Create an Auto Scaling Group

Parameters:

SubnetIds:

Description: Subnet IDs where instances can be launched Type: List<AWS::EC2::Subnet::Id>

<strong> </strong>AmiId:

Description: The application server's AMI ID

Type: AWS::EC2::Image::Id Default: ami-9be6f38c # AWS Linux in us-east-1

InstanceType:

Description: The type of instance to launch Type: String

Default: t2.micro

MinSize:

Description: Minimum number of instances in the group Type: Number

Default: 1

MaxSize:

Description: Maximum number of instances in the group Type: Number

Default: 4

ThresholdCPUHigh: Description: Launch new instances when CPU utilization is over this threshold Type: Number

Default: 60

ThresholdCPULow: Description: Remove instances when CPU utilization is under this threshold Type: Number

Default: 40

ThresholdMinutes: Description: Launch new instances when over the CPU threshold for this many minutes Type: Number

Default: 5

Resources:

AutoScalingGroup: Type: AWS::AutoScaling::AutoScalingGroup Properties:

MinSize: !Ref MinSize MaxSize: !Ref MaxSize LaunchConfigurationName: !Ref LaunchConfiguration Tags:

- Key: Name

Value: !Sub "\${AWS::StackName} server"

PropagateAtLaunch: true VPCZoneIdentifier: !Ref SubnetIds

LaunchConfiguration:

Type: AWS::AutoScaling::LaunchConfiguration Properties:

ImageId: !Ref AmiId InstanceType: !Ref InstanceType UserData:

Fn::Base64: !Sub |

```
#!/bin/bash -xe # This will be run on startup, launch your application here
```

ScaleUpPolicy:

Type: AWS::AutoScaling::ScalingPolicy Properties:

AdjustmentType: ChangeInCapacity AutoScalingGroupName: !Ref AutoScalingGroup

Cooldown: 60

ScalingAdjustment: 1

ScaleDownPolicy: Type: AWS::AutoScaling::ScalingPolicy Properties:

    AdjustmentType: ChangeInCapacity AutoScalingGroupName: !Ref AutoScalingGroup  
Cooldown: 60

    ScalingAdjustment: -1

CPUHighAlarm:

Type: AWS::CloudWatch::Alarm Properties:

    ActionsEnabled: true AlarmActions:

        - !Ref ScaleUpPolicy AlarmDescription: Scale up on CPU load ComparisonOperator:  
GreaterThanThreshold Dimensions:

            - Name: AutoScalingGroupName Value: !Ref AutoScalingGroup EvaluationPeriods:  
!Ref ThresholdMinutes MetricName: CPUUtilization Namespace: AWS/EC2

    Period: 60

    Statistic: Average Threshold: !Ref ThresholdCPUHigh

CPULowAlarm:

Type: AWS::CloudWatch::Alarm Properties:

    ActionsEnabled: true AlarmActions:

        - !Ref ScaleDownPolicy AlarmDescription: Scale down on CPU load  
ComparisonOperator: LessThanThreshold Dimensions:

            - Name: AutoScalingGroupName Value: !Ref AutoScalingGroup EvaluationPeriods:  
!Ref ThresholdMinutes MetricName: CPUUtilization Namespace: AWS/EC2

    Period: 60

Statistic: Average Threshold: !Ref ThresholdCPULow

```
<strong> aws cloudformation create-stack \
--stack-name asg \
--template-body file://04-auto-scaling-an-application-server.yaml \
--parameters \
ParameterKey=SubnetIds,ParameterValue='<subnet-id-1>\, \
<subnet-id-2>'

</strong>
```



# How it works...

This example defines an auto scaling group and the dependent resources. These include the following:

- A launch configuration to use when launching new instances
- Two scaling policies, one to scale the number of instances up, and an inverse policy to scale back down
- An alarm to alert when the CPU crosses a certain threshold, for a certain number of minutes

The auto scaling group and launch-configuration resource objects in this example use mostly default values. You will need to specify your own `securityGroups` and a `KeyName` parameter in the `LaunchConfiguration` resource configuration if you want to be able to connect to the instances (for example, via SSH).

AWS will automatically take care of spreading your instances evenly over the subnets you have configured, so make sure they are in different AZs! When scaling down, the oldest instances will be removed before the newer ones.



# Scaling policies

The scaling policies detail how many instances to create or delete when they are triggered. It also defines a `cooldown` value, which helps prevent *flapping* servers—when servers are created and deleted before they have finished starting and are useful.



*While the scaling policies in this example use equal values, you might want to change that so your application can scale up quickly, and scale down slowly for the best user experience.*



# Alarms

The `CPUHighAlarm` parameter will alert when the average CPU utilization goes over the value set in the `ThresholdCPUHigh` parameter. This alert will be sent to the `ScaleUpPolicy` resource provisioning more instances, which will bring the average CPU utilization down across the whole auto scaling group. As the name suggests, the `CPULowAlarm` parameter does the reverse when the average CPU utilization goes under the `ThresholdCPULow` parameter.

This means that new instances will be launched until the CPU utilization across the auto scaling group stabilizes somewhere between 40-60% (based on the default parameter values), or the `MaxSize` of instances is reached.



*It is very important to leave a gap between the high and low alarms thresholds. If they are too close together, the alarms will not stabilize and you will see instances created and destroyed almost continually.*

The minimum charge for an instance is *one hour*, so creating and destroying them multiple times in one hour may result in higher than expected charges.



# Creating machine images

Creating or *baking* your own **Amazon Machine Images (AMIs)** is a key part of systems administration in AWS. Having a pre-baked image helps you provision your servers faster, easier, and more consistently than configuring it by hand.

Packer is the de facto standard tool that helps you make your own AMIs. By automating the launch, configuration, and clean-up of your instances, it makes sure you get a repeatable image every time.

In this recipe, we will create an image with the Apache web server pre-installed and configured. While this is a simple example, it is also a very common use-case.

By baking-in your web server, you can scale up your web serving layer to dynamically match the demands on your websites. Having the software already installed and configured means you get the fastest and most reliable start-up possible.



# Getting ready

For this recipe, you must have the Packer tool available on your system. Download and install Packer from the project's website <https://www.packer.io/downloads.html>.

```
"builders": [
  {
    "type": "amazon-ebs", "instance_type": "t2.micro", "region": "us-east-1", "source_ami": "ami-9be6f38c", "ssh_username": "ec2-user", "ami_name": "aws-linux-apache {{timestamp}}"
  }
],
"provisioners": [
  {
    "type": "shell", "inline": [
      "sudo yum install -y httpd", "sudo chkconfig httpd on"
    ]
  }
]
```

<strong> packer validate 04-creating-machine-images.json</strong>

<strong> packer build 04-creating-machine-images.json</strong>

6. Wait until the process is complete. While it is running, you will see an output similar to the following:

```
awsac — packer build src/04-creating-machine-images.json — 114x31
$ packer build src/04-creating-machine-images.json
amazon-ebs output will be in this color.

==> amazon-ebs: Prevalidating AMI Name...
amazon-ebs: Found Image ID: ami-9be6f38c
==> amazon-ebs: Creating temporary keypair: packer_587a9f63-1bcc-7e85-b9c9-eeec160cc172
==> amazon-ebs: Creating temporary security group for this instance...
==> amazon-ebs: Authorizing access to port 22 the temporary security group...
==> amazon-ebs: Launching a source AWS instance...
amazon-ebs: Instance ID: i-0c249b526d0cabeb9b
==> amazon-ebs: Waiting for instance (i-0c249b526d0cabeb9b) to become ready...
==> amazon-ebs: Waiting for SSH to become available...
==> amazon-ebs: Connected to SSH!
==> amazon-ebs: Provisioning with shell script: /var/folders/vt/1kw7w5ns6h16vt8j_tk08pzm0000gn/T/packer-shell21831
1435
amazon-ebs: Loaded plugins: priorities, update-motd, upgrade-helper
amazon-ebs: Resolving Dependencies
amazon-ebs: --> Running transaction check
amazon-ebs: ---> Package httpd.x86_64 0:2.2.31-1.8.amzn1 will be installed
```

## 7. Take note of the AMI ID returned by Packer so that you can use it when launching instances in the future:

```
awsac — bash — 114x31
amazon-ebs: Verifying : httpd-tools-2.2.31-1.8.amzn1.x86_64 1/5
amazon-ebs: Verifying : apr-1.5.1-1.12.amzn1.x86_64 2/5
amazon-ebs: Verifying : httpd-2.2.31-1.8.amzn1.x86_64 3/5
amazon-ebs: Verifying : apr-util-ldap-1.4.1-4.17.amzn1.x86_64 4/5
amazon-ebs: Verifying : apr-util-1.4.1-4.17.amzn1.x86_64 5/5
amazon-ebs:
amazon-ebs: Installed:
amazon-ebs: httpd.x86_64 0:2.2.31-1.8.amzn1
amazon-ebs:
amazon-ebs: Dependency Installed:
amazon-ebs: apr.x86_64 0:1.5.1-1.12.amzn1      apr-util.x86_64 0:1.4.1-4.17.amzn1
amazon-ebs: apr-util-ldap.x86_64 0:1.4.1-4.17.amzn1 httpd-tools.x86_64 0:2.2.31-1.8.amzn1
amazon-ebs:
amazon-ebs: Complete!
==> amazon-ebs: Stopping the source instance...
==> amazon-ebs: Waiting for the instance to stop...
==> amazon-ebs: Creating the AMI: aws-linux-apache 1484431202
amazon-ebs: AMI: ami-fb816ded
==> amazon-ebs: Waiting for AMI to become ready...
==> amazon-ebs: Terminating the source AWS instance...
==> amazon-ebs: Cleaning up any extra volumes...
==> amazon-ebs: No volumes to clean up, skipping
==> amazon-ebs: Deleting temporary security group...
==> amazon-ebs: Deleting temporary keypair...
Build 'amazon-ebs' finished.

==> Builds finished. The artifacts of successful builds are:
--> amazon-ebs: AMIs were created:

us-east-1: ami-fb816ded
$ 
```



# **How it works...**

While this is a very simple recipe, there is a lot going on behind the scenes. This is why we recommend you use Packer to create your machine images.



# Template

In the `builders` section of the template, we define our build details.

We are using the most common type of AMI builder: `amazon-ebs`. There are other types of AWS builders, for instance, storage-backed instance types.

Next, we define the type of instance to use when baking.



*Make sure that you can often decrease the time it takes to bake your instance by using a larger instance size. Remember that the minimum price paid for an instance is one hour of billable time.*

The `source_ami` property in this recipe is an AWS Linux AMI ID in the `region` we have specified. The `ssh_username` allows you to set the username used to connect and run `provisioners` on the instance. This will be determined by your operating system, which in our case is `ec2-user`.

Finally, the `ami_name` field includes the built-in Packer variable `{{timestamp}}`. This ensures the AMI you create will always have a unique name.



# Validate the template

The `packer validate` command is a quick way to ensure your template is free of syntax errors before you launch any instances.



# Build the AMI

Once you have created and validated your template, the `packer build` command does the following for you:

- Creates a one-time key pair for SSH access to the instance
- Creates a dedicated security group to control access to the instance
- Launches an instance
- Waits until SSH is ready to receive connections
- Runs the provisioner steps on the instance
- Stops the instance
- Generates an AMI from the stopped instance
- Terminates the instance



*Check the Packer documentation for more provisioners and functionality at <https://www.packer.io/docs/>.*



# **There's more...**

While Packer makes the administration of images much easier on AWS, there are still a few things to watch out for.



# Debugging

Obviously, with so many steps being automated for you, there are many things that can potentially go wrong. Packer gives you a few different ways to debug issues with your builds.

One of the most useful arguments to use with Packer is the `-debug` flag. This will force you to manually confirm each step *before* it takes place. Doing this makes it easy to work out exactly which step in the command is failing, which in turn usually makes it obvious what needs to be changed.

Another useful thing to do is to raise the level of logging output during a Packer command. You can do this by setting the `PACKER_LOG` variable to `true`. The easiest way to do this is with `PACKER_LOG=1` at the beginning of your Packer command line. This will mean you get a lot more information printed to the console (for example, SSH logs, AWS API calls, and so on) during the command. You may even want to run with this level of logging normally in your builds, for auditing purposes.



# Orphaned resources

Packer does a great job of managing and cleaning up the resource it uses, but it can only do that while it is running.

If your Packer job aborts for any reason (most likely network issues) then there may be some resources left **orphaned**, or **unmanaged**. It is good practice to check for any Packer instances (they will have *Packer* in their name), and stop them if there are no active Packer jobs running.

You may also need to clean up any leftover key pairs and security groups, but this is less of an issue as there is no cost associated with them (unlike instances).



# Deregistering AMIs

As it becomes easier to create AMIs, you may find you end up with more than you need!

AMIs are made up of EC2 snapshots, which are stored in S3. There is a cost associated with storing snapshots, so you will want to clean them up periodically. Given the size of most AMIs (usually a few GBs), it is unlikely to be one of your major costs.

An even greater cost is the administrative overhead of managing too many AMIs. As your images improve and fixes are applied (especially security fixes), you may want to prevent people from using them.

To remove an AMI, you must first *deregister* it, and then remove the underlying snapshots.



*Make sure you do not deregister AMIs that are currently in use. For example, an auto scaling group that references a deregistered AMI will fail to launch new instances!*

You can easily deregister snapshots through the web console or using the AWS CLI tool.

Once an AMI is no longer registered, you can remove the associated snapshots. Packer automatically adds the AMI ID to the snapshots' description. By searching your snapshots for the deregistered AMI ID, you can find which ones need to be deleted.

You will not be able to delete snapshots if the AMI has not been deregistered, or if the deregistration is still taking place (it can take a few minutes).



# Other platforms

It is also worth noting that Packer can build for more platforms than just AWS. You can also build images for VMWare, Docker, and many others.

This means you could build almost exactly the same machine image locally (for example, using Docker) as you do in AWS. This makes it much more convenient when setting up local development environments, for example.

Check the `builders` section of the Packer documentation for details.



# Creating security groups

AWS describes security groups as *virtual firewalls*. While this analogy helps newcomers to the EC2 platform understand their purpose and function, it's probably more accurate to describe them as a *firewall-like* method of authorizing traffic. They don't offer all the functionality you'd find in a traditional firewall, but this simplification also makes them extremely powerful, particularly when combined with Infrastructure as Code and modern SDLC practices.

We're going to go through a basic scenario involving a web server and load balancer. We want the load balancer to respond to HTTP requests from everywhere, and we want to isolate the web server from everything except the load balancer.



# Getting ready

Before we get started there's a small list of things you'll need to have ready:

- `AmiId`: This is the ID of an AMI in your region. For this recipe, we'd recommend using an AWS Linux AMI because our instance will attempt to run some `yum` commands on startup.
- `vPCID`: This is the ID of the VPC you wish to launch the EC2 server into.
- `SubnetIDs`: These are the subnets which our EC2 instance can launch in.

AWSTemplateFormatVersion: '2010-09-09'

Parameters:

AmiId:

Type: AWS::EC2::AMI::Id Description: AMI ID to launch instances from VPC

VPCID: Type: AWS::EC2::VPC::Id Description: VPC where load balancer and instance will launch SubnetIDs:

SubnetIDs: Type: List<AWS::EC2::Subnet::Id> Description: Subnets where load balancer and instance will launch (pick at least 2)

ExampleELBSecurityGroup:

Type: AWS::EC2::SecurityGroup Properties:

GroupDescription: Security Group for example ELB

SecurityGroupIngress: - IpProtocol: tcp CidrIp: 0.0.0.0/0

FromPort: 80

ToPort: 80

ExampleEC2InstanceSecurityGroup:

Type: AWS::EC2::SecurityGroup Properties:

GroupDescription: Security Group for example Instance SecurityGroupIngress: - IpProtocol: tcp SourceSecurityGroupName: Ref: ExampleELBSecurityGroup FromPort: 80

ToPort: 80

ExampleLoadBalancer:

Type: AWS::ElasticLoadBalancingV2::LoadBalancer Properties:

Subnets:

- Fn::Select: [ 0, Ref: SubnetIDs ]

- Fn::Select: [ 1, Ref: SubnetIDs ]

SecurityGroups: - Fn::GetAtt: ExampleELBSecurityGroup.GroupId ExampleListener:  
Type: AWS::ElasticLoadBalancingV2::Listener Properties:

LoadBalancerArn: Ref: ExampleLoadBalancer DefaultActions: - Type: forward

TargetGroupArn: Ref: ExampleTargetGroup Port: 80

Protocol: HTTP

ExampleTargetGroup: Type: AWS::ElasticLoadBalancingV2::TargetGroup Properties:

Port: 80

Protocol: HTTP

VpcId:

Ref: VPCID

Targets:

- Id:

Ref: ExampleEC2Instance

ExampleEC2Instance:

Type: AWS::EC2::Instance Properties:

InstanceType: t2.nano UserData:

Fn::Base64:

Fn::Sub: |

```
#!/bin/bash -ex yum install -y nginx service nginx start exit 0
```

ImageId:

Ref: AmiId

SecurityGroupIds: - Fn::GetAtt: ExampleEC2InstanceSecurityGroup.GroupId  
SubnetId:

Fn::Select: [ 0, Ref: SubnetIDs ]

Outputs:

ExampleEC2InstanceHostname: Value:

Fn::GetAtt: [ ExampleEC2Instance, PublicDnsName ]

ExampleELBURL:

Value:

Fn::Join:

- "

- [ 'http://', { 'Fn::GetAtt': [ ExampleLoadBalancer, DNSName ] }, '/' ]

7. Go ahead and launch this template using the CloudFormation web console or the AWS CLI.

ExampleEC2InstanceIngress:

Type: AWS::EC2::SecurityGroupIngress Properties: IpProtocol: tcp

SourceSecurityGroupName: Ref: ExampleEC2InstanceSecurityGroup GroupName: Ref:  
ExampleEC2InstanceSecurityGroup FromPort: 22

ToPort: 22



# Differences from traditional firewalls

- Security groups can't be used to explicitly block traffic. Only rules of a permissive kind can be added; deny style rules are not supported. Essentially, all inbound traffic is denied unless you explicitly allow it.
- Your rules also may not refer to source ports; only destination ports are supported.
- When security groups are created, they will contain a rule which allows all outbound connections. If you remove this rule, new outbound connections will be dropped. It's a common pattern to leave this rule in place and filter all your traffic using inbound rules only.
- If you do replace the default outbound rule, it's important to note that only new outbound connections will be filtered. Any outbound traffic being sent in response to an inbound connection will still be allowed. This is because security groups are *stateful*.
- Unlike security groups, network ACLs are not stateful and do support `DENY` rules. You can use them as a complementary layer of security inside your VPC, especially if you need to control traffic flow between subnets.



# Creating a load balancer

AWS offers two kinds of load balancers:

- Classic load balancer
- Application load balancer

We're going to focus on the application load balancer. It's effectively an upgraded, second generation of the ELB service, and it offers a lot more functionality than the classic load balancer. HTTP/2 and WebSockets are supported natively, for example. The hourly rate also happens to be cheaper.



*Application load balancers do not support layer-4 load balancing. For this kind of functionality, you'll need to use a classic load balancer.*

AWSTemplateFormatVersion: '2010-09-09'

Parameters:

VPCID:

Type: AWS::EC2::VPC::Id Description: VPC where load balancer and instance will launch SubnetIDs:

Type: List<AWS::EC2::Subnet::Id> Description: Subnets where load balancer and instance will launch (pick at least 2)

Mappings:

ELBAccountMap: us-east-1:

ELBAccountID: 127311923021

ap-southeast-2: ELBAccountID: 783225319266

Resources:

ExampleLogBucket: Type: AWS::S3::Bucket ExampleBucketPolicy: Type: AWS::S3::BucketPolicy Properties:

Bucket:

Ref: ExampleLogBucket PolicyDocument: Statement:

-

Action:

- "s3:PutObject"

Effect: "Allow"

Resource:

Fn::Join:

- ""

- "arn:aws:s3:::"
- Ref: ExampleLogBucket - "/"\*

Principal:

AWS:

Fn::FindInMap: [ ELBAccountMap, Ref: "AWS::Region", ELBAccountID ]

ExampleELBSecurityGroup:

Type: AWS::EC2::SecurityGroup Properties:

GroupDescription: Security Group for example ELB

SecurityGroupIngress: -

IpProtocol: tcp CidrIp: 0.0.0.0/0

FromPort: 80

ToPort: 80

ExampleTargetGroup:

Type: AWS::ElasticLoadBalancingV2::TargetGroup Properties:

Port: 80

Protocol: HTTP

HealthCheckIntervalSeconds: 30

HealthCheckProtocol: HTTP

HealthCheckPort: 80

HealthCheckPath: /

HealthCheckTimeoutSeconds: 5

HealthyThresholdCount: 5

UnhealthyThresholdCount: 2

Matcher:

HttpCode: '200'

VpcId:

Ref: VPCID

ExampleListener:

Type: AWS::ElasticLoadBalancingV2::Listener Properties:

LoadBalancerArn: Ref: ExampleLoadBalancer DefaultActions: - Type: forward TargetGroupArn: <strong> </strong>Ref: ExampleTargetGroup Port: 80

Protocol: HTTP

ExampleLoadBalancer:

Type: AWS::ElasticLoadBalancingV2::LoadBalancer Properties:

LoadBalancerAttributes: - Key: access\_logs.s3.enabled Value: true

- Key: access\_logs.s3.bucket Value:

Ref: ExampleLogBucket - Key: idle\_timeout.timeout\_seconds Value: 60

Scheme: internet-facing Subnets:

- Fn::Select: [ 0, Ref: SubnetIDs ]

- Fn::Select: [ 1, Ref: SubnetIDs ]

SecurityGroups: - Fn::GetAtt: ExampleELBSecurityGroup.GroupId

Outputs:

ExampleELBURL: Value:

Fn::Join:

- "

- [ 'http://', { 'Fn::GetAtt': [ ExampleLoadBalancer, DNSName ] }, '/' ]

ExampleLogBucket: Value:

Ref: ExampleLogBucket



# How it works...

As you can see, we're applying a logging configuration which points to the S3 bucket we've created. We're configuring this load balancer to be Internet-facing, with an idle timeout of 60 seconds (the default).

All load balancers are Internet-facing by default, so it's not strictly necessary to define a `Scheme` in our example; however, it can be handy to include this anyway. This is especially the case if your CloudFormation template contains a mix of public and private load balancers.



*If you specify a logging configuration but the load balancer can't access the S3 bucket, your CloudFormation stack will fail to complete.*

Private ELBs are not Internet-facing and are available only to resources which live inside your VPC.

That's it! You now have a working application load balancer configured to ship logs to an S3 bucket.



# **There's more...**

Load balancers on AWS are highly configurable and there are many options available to you. Here are some of the more frequent ELB options you'll encounter:

ExampleHTTPSLListener:

Type: AWS::ElasticLoadBalancingV2::Listener Properties:

Certificates:

- CertificateArn: arn:aws:acm:ap-southeast-2:123456789012: certificate/12345678-1234-1234-123456789012

LoadBalancerArn: Ref: ExampleLoadBalancer DefaultActions: - Type: forward TargetGroupArn: Ref: ExampleTargetGroup Port: 443

Protocol: HTTPS

The listener will need to reference a valid **Amazon Resource Name (ARN)** for the certificate you wish to use. It's really easy to have AWS Certificate Manager create a certificate for you, but it does require validation of the domain name you're generating the certificate for. You can, of course, bring your own certificate if you wish. You'll need to import it in to AWS Certificate Manager before you can use it with your ELB (or CloudFront distribution).

Unless you have specific requirements around ciphers, a good starting approach is to not define an SSL Policy and let AWS choose what is currently *best of breed*.



# Path-based routing

Once you are comfortable with ELB configuration, you can start to experiment with path-based routing. In a nutshell, it provides a way to inspect a request and proxy it to different targets based on the path requested.

One common scenario you might encounter is needing to route requests for `/blog` to a different set of servers running WordPress, instead of to your main server pool, which is running your Ruby on Rails application.



# Management Tools

In this chapter, we will cover:

- Auditing your AWS account
- Recommendations with Trusted Advisor
- Creating e-mail alarms
- Publishing custom metrics in CloudWatch
- Creating monitoring dashboards
- Creating a budget
- Feeding log files into CloudWatch logs



# Introduction

As with all administration, monitoring and alerting is a critical part of using AWS-based infrastructure. If anything, due to the ephemeral nature of cloud resources, keeping track and measuring your usage is even more important than when using on-premises systems.



# Auditing your AWS account

We're now going to show you how to set up CloudTrail in your AWS account. Once CloudTrail has been enabled, it will start to record all of the API calls made in your account to the AWS service and then deliver them to you as log files in an S3 bucket.

When we talk about API calls we mean things like:

- Actions performed in the AWS console.
- Calls made to AWS APIs using the CLI or SDKs.
- Calls made on your behalf by AWS services. Think CloudFormation or the auto scaling service.

Each entry in the log will contain useful information, such as:

- The service that was called
- The action that was requested
- The parameters sent with the request
- The response that was returned by AWS
- The identity of the caller (including IP address)
- The date and time of the request

ExampleTrailBucket:

Type: AWS::S3::Bucket

ExampleBucketPolicy:

Type: AWS::S3::BucketPolicy Properties:

Bucket: !Ref ExampleTrailBucket PolicyDocument:

Statement:

- Sid: AWSCloudTrailAclCheck20150319

Effect: Allow

Principal:

Service: cloudtrail.amazonaws.com Action: s3:GetBucketAcl Resource: !Join

- ""

-

- "arn:aws:s3:::"

- !Ref ExampleTrailBucket - Sid: AWSCloudTrailWrite20150319

Effect: Allow

Principal:

Service: cloudtrail.amazonaws.com Action: s3:PutObject Resource: !Join

- ""

-

- "arn:aws:s3:::"

- !Ref ExampleTrailBucket - "/AWSLogs/"

- !Ref AWS::AccountId - "/\*"

Condition:

StringEquals:

s3:x-amz-acl: bucket-owner-full-control

ExampleTrail:

Type: AWS::CloudTrail::Trail Properties:

EnableLogFileValidation: true IncludeGlobalServiceEvents: true IsLogging: true

IsMultiRegionTrail: true S3BucketName: !Ref ExampleTrailBucket DependsOn:

- ExampleTrailBucket - ExampleBucketPolicy

Outputs:

ExampleBucketName: Value: !Ref ExampleTrailBucket Description: Bucket where CloudTrail logs will be stored

<strong> aws cloudformation create-stack \</strong> <strong>--template-body file://05-auditing-your-aws-account.yaml \</strong> <strong>--stack-name example-cloudtrail</strong>



# How it works...

This template will set up CloudTrail with the following configuration:

- CloudTrail will be turned on for all regions in your account. This is a sensible place to start because it gives you visibility over where your AWS resources are being created. Even if you are the sole user of your AWS account it can be handy to know if you are making API calls to other regions by mistake (it's easy to do). When you create a multi region trail, new regions will automatically be included when they come online with no additional effort on your part.
- Global service events will also be logged. Again this is a sensible default because it includes services that aren't region-specific. CloudFront and IAM are two examples of AWS services that aren't region-specific.
- Log file validation is turned on. With this feature enabled, CloudTrail will deliver a digest file on an hourly basis that you can use to determine if your CloudTrail logs have been tampered with. CloudTrail uses SHA-256 for hashing and signing (RSA). The AWS CLI can be used to perform ad hoc validation of CloudTrail logs.

For a quick view of your CloudTrial logs, with some basic search and filter functionality, you can head to the AWS web console:

The screenshot shows the AWS CloudTrail Management Console interface. The top navigation bar includes links for Services, Resource Groups, and Support. On the left, there's a sidebar with 'API activity history' and 'Trails'. The main content area is titled 'API activity history' and displays a message about the last 7 days of API activity for supported services, mentioning 'create, modify, and delete API calls'. It also notes that for read-only API activity, users should go to their Amazon S3 bucket or CloudWatch Logs. Below this, there's a message about filtering the list using available attributes and a link to learn more. At the bottom, there are filter and time range controls, and a table listing API events from January 29, 2017. The table columns are Event time, User name, Event name, Resource type, and Resource name. The listed events include ConsoleLogin, CreateLogStream, CreateLogGroup, CreateTags, and RunInstances.

Event time	User name	Event name	Resource type	Resource name
2017-01-29, 09:12:19 ...	root	ConsoleLogin		
2017-01-29, 09:12:05 ...	i-045e22ec0913a61f6	CreateLogStream		
2017-01-29, 09:12:05 ...	i-045e22ec0913a61f6	CreateLogGroup		
2017-01-29, 09:12:00 ...	i-045e22ec0913a61f6	CreateLogGroup		
2017-01-29, 09:12:00 ...	i-045e22ec0913a61f6	CreateLogStream		
2017-01-29, 09:11:29 ...	root	CreateTags		i-045e22ec0913a61f6
2017-01-29, 09:11:13 ...	lucasbookair14	RunInstances	EC2 Instance and 7 more	i-045e22ec0913a61f6 ...

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.

[Privacy Policy](#) [Terms of Use](#)

CloudTrail web console



# There's more...

- Log files are encrypted using server side encryption in S3. This encryption is transparent to you, but you can opt to encrypt these files with your own **customer master key (CMK)** if you wish.
- API calls are logged by CloudTrail in under 15 minutes.
- Logs are shipping to your S3 bucket every five minutes.
- It's possible to aggregate CloudTrail events across many accounts into a single bucket. This is a pattern often used to log AWS activity into a *SecOps* or similar account for auditing.
- Logging aside, CloudTrail keeps your API activity for seven days.
- You can create more than one trail. You might consider creating a trail for your developers that is separate from the trail consumed by security.
- If a CloudFormation stack creates an S3 bucket and that S3 bucket has objects in it the delete operation will fail if and when you choose to delete the stack. You can manually delete the S3 bucket in the S3 web console if you wish to work around this.



# Recommendations with Trusted Advisor

Trusted Advisor covers four main areas and it is designed to give you some guidance around what are considered best practices for your cloud deployment. The areas covered are:

- Cost Optimization
- Performance
- Security
- Fault Tolerance

It's available to everyone and free to use—with one fairly large catch. Unless you are paying for Business or Enterprise level support with AWS you only get access to four checks. At the time of publishing there are 55 possible checks.



# **How to do it...**

The good news is you don't need to do anything at all to turn on Trusted Advisor. It's automatically enabled when your AWS account is created and will continue to update for the lifetime of your account.

Go ahead and navigate to the Trusted Advisor section of the AWS web console.



# How it works...

The four checks provided for free with this service are:

- **Unrestricted ports:** This is a check on the highest risk ports in your security groups. They'll be flagged if they're open to everyone (0.0.0.0/0).
- **IAM usage:** This is a fairly rudimentary check. If there isn't at least one IAM user in your account this check won't pass. It's considered good practice to not use your root login credentials for your AWS account and instead create IAM users with least privilege access.
- **MFA on root account:** This is also a fairly rudimentary check. You need to have MFA enabled for your root login in order for this check to pass. It's obviously a good idea to enable MFA for your IAM users too.
- **Service limits:** This one is quite handy: if you're approaching 80% of your service limits, this check won't pass. For example, it's nice to know if you're about to hit the cap of CloudFormation stacks or EC2 instances before you attempt to create them.

Even though there's only four checks here, these are some of the more useful ones so we'd encourage you to pay attention to them.

The console uses a color scheme to denote the status of each check:

- **Red:** It's recommended that you take action to remedy this check
- **Yellow:** This check requires investigation and possible remediation
- **Green:** This check is passing and needs no attention



*Visit the Preferences page in the Trusted Advisor web console if you'd like to have a weekly report e-mailed to you.*



Services

Resource Groups



Lucas Chan

Global

Support

## Dashboard

Cost Optimization

Performance

Security

Fault Tolerance

Preferences

## Trusted Advisor Dashboard



### Cost Optimization



0 ✓ 0 ▲ 0 !

### Performance



1 ✓ 0 ▲ 0 !

### Security



3 ✓ 0 ▲ 0 !

### Fault Tolerance



0 ✓ 0 ▲ 0 !

## Recommended Actions

### ► Security Groups - Specific Ports Unrestricted

Refreshed: a few seconds ago  
Previous status: Yellow

Checks security groups for rules that allow unrestricted access (0.0.0.0/0) to specific ports.

0 of 1 security group rules allow unrestricted access to a specific port.

### ► IAM Use

Refreshed: a few seconds ago



Checks for your use of AWS Identity and Access Management (IAM).

At least one IAM user has been created for this account.

### ► MFA on Root Account

Refreshed: a few seconds ago



Trusted Advisor console



# There's more...

As well as opening up the entire suite of Trusted Advisor checks, a Business or Enterprise level support arrangement gives you access to the following:

- **Notifications:** You are able to have notifications delivered to you at a higher frequency using a number of delivery methods. Since Trusted Advisor is an available source in CloudWatch Events you'll be able to create notifications that can be handled by SNS (e-mail, push, SMS) or even notifications that will trigger Lambda functions.
- **API access:** You'll have access to a number of Trusted Advisor API methods such as `DescribeTrustedAdvisorCheckResult` and `DescribeTrustedAdvisorCheckSummaries`. You can use these to integrate the results from checks into your own dashboards or monitoring systems. You'll also be able to use the APIs to refresh Trusted Advisor checks (after you've taken corrective action on them, for example).
- **Exclusion:** You can selectively mute checks that are failing. You'll sometimes want to do this for things such as RDS instances in your development environments that aren't in multi-AZ mode or don't have backups enabled.

Finally, some of the more useful checks we see for our Business and Enterprise level support customers are:

- **Reserved Instances:** A nice cost optimization if you have a reasonably static workload.
- **Unassociated Elastic IPs:** If IP addresses are not associated with a network interface (on an EC2 instance for example) you will still be charged for them. Also if there are unassociated IPs floating around, that is usually a sign that they are being allocated manually instead of with CloudFormation. Remember that the goal here is for more automation, not less.
- **Idle load balancers:** Again, these cost money and are often easily orphaned in low automation environments.
- **S3 bucket permissions:** It's not always obvious if the permissions on an S3 bucket have been misconfigured. This check helps you avoid unintentionally leaking data.



# Creating e-mail alarms

While e-mail alarms may not be the most scalable of all alarms (due to the amount of e-mail most people get), they are the easiest to integrate—almost everyone has an e-mail address!

This recipe uses two AWS services:

- **CloudWatch (CW)**
- **Simple Notification Service (SNS)**

As you will often want to create alarms for metrics after viewing them through the CloudWatch dashboard, this recipe will use the console to create the alarms.



# How to do it...

1. In the CloudWatch console, go to the Alarms section:

The screenshot shows the CloudWatch Management Console interface. The left sidebar has a 'CloudWatch' section with links for Dashboards, Alarms, Metrics (with a 'NEW' badge), and other services like Events, Rules, and Logs. The main content area has two sections: 'Metric Summary' and 'Alarm Summary'. 'Metric Summary' displays operational and performance metrics for AWS resources, mentioning 80 available metrics in the Asia Pacific (Sydney) region. It includes a 'Browse Metrics' button and a search bar. 'Alarm Summary' indicates no alarms are created yet and provides a link to manage billing alarms in the US East (N. Virginia) region. A 'Create Alarm' button is located in this section. Below these is the 'Service Health' section, which shows the 'Amazon CloudWatch Service' is operating normally. The bottom of the page includes standard AWS navigation links for Feedback, English, Privacy Policy, and Terms of Use.

2. Click Create Alarm to start the wizard:

CloudWatch Management Con... Guest

Secure https://ap-southeast-2.console.aws.amazon.com/cloudwatch/home?region=ap-southeast-2#alarm:alarmFilter=ANY

Services Resource Groups

CloudWatch Dashboards Alarms ALARM INSUFFICIENT OK Billing Events Rules Logs Metrics NEW

Create Alarm Modify Copy Delete Filter: All alarms Search Alarms State Name Threshold Config Status No records found.

0 Alarms selected Select an alarm above

Feedback English © 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

This screenshot shows the AWS CloudWatch Alarms page. The left sidebar is titled 'Alarms' and lists categories: ALARM, INSUFFICIENT, OK, Billing, Events, Rules, Logs, and Metrics (NEW). The main area has buttons for 'Create Alarm', 'Modify', 'Copy', and 'Delete'. A search bar is present. The table header includes columns for 'State', 'Name', 'Threshold', and 'Config Status'. A message at the bottom states 'No records found.'

3. Select the metric you are interested in alerting on. In this case, we will choose By Function Name under Lambda Metrics:

CloudWatch Management Con... Guest

Secure https://ap-southeast-2.console.aws.amazon.com/cloudwatch/home?region=ap-southeast-2#alarm:alarmFilter=ANY

Services

CloudWatch Dashboards Alarms ALARM INSUFFICIENT OK Billing Events Rules Logs Metrics NEW

Create Alarm

1. Select Metric 2. Define Alarm

Browse Metrics Search Metrics

CloudWatch Metrics by Category

Your CloudWatch metric summary has loaded. Total metrics: 67

ApiGateway Metrics: 10	DynamoDB Metrics: 7	Lambda Metrics: 28
By Api Name: 5 By Stage: 5	Table Metrics: 3 Table Operation Metrics: 3 Account Metrics: 1	Across All Functions: 4 By Function Name: 12 By Resource: 12
Logs Metrics: 8	S3 Metrics: 4	Custom Metrics: 10
Account Metrics: 2 Log Group Metrics: 6	Storage Metrics: 4	Choose namespace...

Update Graph

Cancel Previous Next Create Alarm

Feedback Terms of Use

This screenshot shows the 'Create Alarm' wizard. It's on Step 1: Select Metric. The left sidebar is identical to the previous screenshot. The main area shows a summary of CloudWatch metrics: Total metrics: 67. It then lists metrics by category: ApiGateway Metrics (10), DynamoDB Metrics (7), Lambda Metrics (28), Logs Metrics (8), S3 Metrics (4), and Custom Metrics (10). Each category has a breakdown of specific metrics like Table Metrics, Account Metrics, etc. Buttons for 'Update Graph', 'Cancel', 'Previous', 'Next', and 'Create Alarm' are at the bottom.

4. Select the specific metric. You can filter by any of the values in the table. In this case, we will select

## Errors and click Next:

The screenshot shows the 'Create Alarm' wizard in the CloudWatch Management Console. The user is on step 1, 'Select Metric'. They have chosen the 'example-lambda-function' and selected the 'Errors' metric. A graph displays the metric's average value over the last 5 minutes, which is currently at 0. Below the graph, the alarm threshold is defined as 'Errors > 0' for 1 consecutive period(s). The 'Next' button is available at the bottom right.

5. Define the alarm, giving at least a name and a threshold. In this case, we will alert if there are ever *any* errors (such as  $> 0$ ):

The screenshot shows the 'Create Alarm' wizard in the CloudWatch Management Console, step 2: 'Define Alarm'. The user has named the alarm 'FunctionErrors' and set the threshold to 'Errors > 0' for 1 consecutive period(s). The 'Alarm Preview' section shows a graph where the blue line crosses the red threshold line at 0.8 at 08:00 on 1/23. The 'Create Alarm' button is available at the bottom right.

6. In the Actions section, create a new list by giving the e-mail address you want to be notified on of a breach, and a topic name (in this example, we use EmailMe), and then click Create Alarm:

The screenshot shows the 'Create Alarm' dialog box in the AWS CloudWatch Management Console. The left sidebar shows 'Alarms' is selected. The main form has two tabs: '1. Select Metric' (disabled) and '2. Define Alarm' (selected). Under 'Define Alarm':

- Name:** FunctionErrors
- Description:** (empty)
- Whenever:** Errors
  - is:** > 0
  - for:** 1 consecutive period(s)
- Actions**:
  - Notification**: Whenever this alarm: State is ALARM, Send notification to: EmailMe (Email list: rowan@rowanudell.com), + Notification, + AutoScaling Action, + EC2 Action.
- Metric Details**: Namespace: AWS/Lambda, FunctionName: example-lambda-func!, Metric Name: Errors, Period: 5 Minutes, Statistic: Standard (Average).

At the bottom are buttons: Cancel, Previous, Next, and a prominent blue **Create Alarm** button.

7. You will be asked to confirm the e-mail address, and no notifications will be given until it is verified.

S CloudWatch Management Con X Guest

Secure <https://ap-southeast-2.console.aws.amazon.com/cloudwatch/home?region=ap-southeast-2#alarm:alarmFilter=ANY>

Services Resource Groups Rowan U dell Sydney Support

CloudWatch Dashboards Alarms ALARM INSUFFICIENT OK Billing Events Rules Logs Metrics NEW

Your alarm [FunctionErrors](#) has been saved.

Create Alarm Modify Copy Delete Filter: All alarms Search Alarms

State INSUFFICIENT

Config Status Pending confirmation

1 to 1 of 1 alarms

Confirm new email addresses

Check your email inbox for a message with the subject "AWS Notification - Subscription Confirmation" and click the included link to confirm that you are willing to receive alerts to that address. AWS can only send notifications to confirmed addresses

Waiting for confirmation of 1 new email address

rowan@rowanudell.com Resend confirmation link

Note: You have 72 hours to confirm these email addresses

I will do it later View Alarm

Feedback English © 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

8. The confirmation e-mail will look like this:

## AWS Notification - Subscription Confirmation



AWS Notifications <no-reply@sns.amazonaws.com>  
to rowan Show details

8:03 PM (less than a minute ago)  
Inbox Reply More

You have chosen to subscribe to the topic:  
**arn:aws:sns:ap-southeast-2:365407189188:EmailMe**

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):  
[Confirm subscription](#)

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)

[Reply](#) [Forward](#)

9. Once you have clicked on the Confirm subscription link in the e-mail, you will see a confirmation message as follows:



## Subscription confirmed!

You have subscribed rowan@rowanudell.com to the topic:  
[EmailMe](#).

Your subscription's id is:

arn:aws:sns:ap-southeast-2:365407189188:EmailMe:04a3340f-c222-433a-8ec1-4fa7f68acb41

If it was not your intention to subscribe, [click here to unsubscribe](#).

- Back in the console the status will update, showing that you have successfully confirmed your subscription:

The screenshot shows the AWS CloudWatch Management Console interface. On the left, there is a sidebar with navigation links: CloudWatch, Dashboards, Alarms (selected), ALARM, INSUFFICIENT, OK, Billing, Events, Rules, Logs, and Metrics. The main area displays a list of alarms under the 'Alarms' section. A success message at the top states: "Your alarm FunctionErrors has been saved." Below this, there is a button labeled "Create Alarm" and a search bar for "Search Alarms". A modal dialog box is open in the center, titled "Confirm new email addresses". It contains instructions: "Check your email inbox for a message with the subject 'AWS Notification - Subscription Confirmation' and click the included link to confirm that you are willing to receive alerts to that address. AWS can only send notifications to confirmed addresses". It also shows a list of email addresses: "Waiting for confirmation of 0 new email address" and "rowan@rowanudell.com". A note below says: "Note: You have 72 hours to confirm these email addresses". At the bottom of the dialog are two buttons: "I will do it later" and "View Alarm". The footer of the page includes links for Feedback, English, Privacy Policy, and Terms of Use.

- You will then see your newly created alarm in the console, and can view its status and history:

CloudWatch Management Con X Guest

Secure https://ap-southeast-2.console.aws.amazon.com/cloudwatch/home?region=ap-southeast-2#alarm:alarmFilter=ANY;name=FunctionErrors

Services Resource Groups Rowan Udell Sydney Support

CloudWatch Dashboards Alarms ALARM INSUFFICIENT OK Billing Events Rules Logs Metrics NEW

Your alarm FunctionErrors has been saved.

Create Alarm Modify Copy Delete Filter: All alarms Search Alarms State Name Threshold Config Status 1 to 1 of 1 alarms

OK FunctionErrors Errors > 0 for 5 minutes

1 Alarm selected

**Alarm:FunctionErrors**

Details History

**State Details:** State changed to OK at 2017/01/23. Reason: Threshold Crossed: 1 datapoint (0.0) was not greater than the threshold (0.0).

**Description:**

**Threshold:** Errors > 0 for 5 minutes

**Actions:** In ALARM: • Send message to topic "EmailMe" (rowan@rowanudell.com)

**Namespace:** AWS/Lambda

**Metric Name:** Errors

**Dimensions:** FunctionName = example-lambda-function

**Statistic:** Average

**Period:** 5 minutes

FunctionErrors Errors > 0

Feedback English © 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

12. In the SNS console, you can see the topic that was created for you as follows:

AWS SNS Guest

Secure https://ap-southeast-2.console.aws.amazon.com/sns/v2/home?region=ap-southeast-2#/topics

Services Resource Groups Rowan Udell Sydney Support

SNS dashboard Topics Applications Subscriptions Text messaging (SMS)

## Topics

Publish to topic Create new topic Actions

Filter

Name	ARN
EmailMe	arn:aws:sns:ap-southeast-2:365407189188:EmailMe



# How it works...

While we normally prefer the CLI (or CloudFormation) for creating AWS resources the wizard for creating alarms does a lot of work for you, so it is a good place to start. Once you know what kinds of alarms you are interested in, you can automate them.

The CloudWatch console is a great place to keep an eye on the performance of your resources. Often when looking at the metrics you might find a scenario that you would want to be notified of, and quickly create an alarm on it.



*While e-mail is probably the easiest way to get started with alarms, it doesn't scale all that well (Do you really want more e-mail?). For very important metrics you might want a CloudWatch dashboard instead, or a different notification protocol/target.*

We start by selecting the metric we are interested in; in this case, it is errors from the example-lambda-function, but the process would work the same regardless of the metric you select.

You must define a name for the alarm, and you can optionally create a description. One of the most important parts of the alarm is how you define the threshold that will trigger it. You can choose not only the value and comparison operator used (for example, greater than ( $>$ ), less than ( $<$ ), greater than or equal to ( $\geq$ ), and so on), but also the number of failing data points that must occur before the alarm is triggered. This can stop you being alerted unnecessarily for temporary *spikes* in metric values. In this scenario we want to know if there are *any* errors, so we set the value to  $1$ .

On the right-hand side you can define the check period and the statistic used (for example, Average, Maximum, Minimum, and so on). You can also see the recent history of the selected metric in the top-right corner. The red line on the graph is where the currently defined threshold will sit, so you can quickly see if the alarm would have been triggered.

In the Actions section of the alarm, you define what action will be taken when triggered. While you can select an existing SNS topic, we will define a new one by clicking on New list. You are then prompted for the details of the new topic; you must give both a name and an e-mail address to subscribe to the topic.

When you click Create Alarm, you will see the status of the subscription. After receiving the e-mail and clicking on the confirmation link, the status will automatically update. It doesn't matter if you navigate away from the window before you confirm the subscription. Just remember that your target e-mail address won't receive any notifications if you do not confirm the subscription.

Viewing the newly created alarm shows its current state, and its recent history. An alarm has three possible states:

- ALARM: The metric is over the defined threshold
- INSUFFICIENT\_DATA: There were not enough data points to determine if the metric is under or over the threshold
- OK: The metric is under the defined threshold

You can filter alarms by their state by the links on the side menu, which also show an updated view of how many alarms are in each state.

Behind the scenes, the wizard has created an SNS topic for you. The topic is what handles converting the alarm message to an e-mail, and sending it. Without the SNS topic the alarm would still alert (that is change state), but there would be no way to tell without looking at the metric in the CloudWatch dashboard.



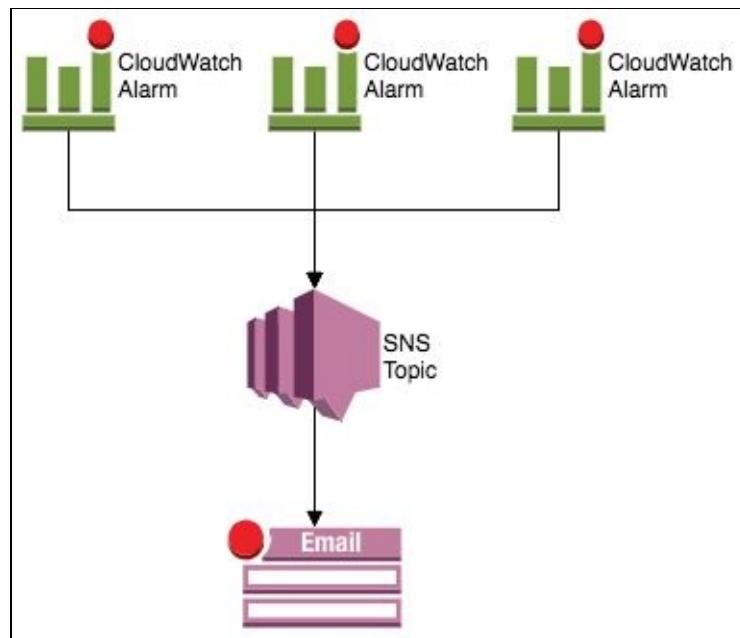
# **There's more...**

This recipe represents the simplest useful configuration of SNS topics and CW alarms, but there is a lot more depth available to you in this pattern.



# Existing topics

Instead of choosing New list in the wizard, you can use the Select list functionality. You then give the name of an existing SNS topic to use, rather than creating a new one.



This means you can set up a single topic to push multiple alarms to. In addition to being simpler it also means you only need to confirm the subscription *once*, instead of doing it for each alarm.



# Other subscriptions

An SNS topic that notifies an e-mail is the most common subscription, but not the only option. SNS topics can also send notifications to:

- HTTP(S) endpoints
- Amazon SQS
- AWS Lambda
- SMS



## See also

- The *Creating monitoring dashboards* recipe



# Publishing custom metrics in CloudWatch

Once you get used to using CloudWatch, it is highly likely that you will want to see more than just the built-in AWS metrics.

One of the most common metrics users ask for after starting to run servers in EC2 is memory usage; the built-in metrics for EC2 instances are CPU utilization, network in/out, disk reads/writes, and status —memory is not included by default!

This recipe will show you how to feed the amount of memory in use on your Linux instances to CloudWatch, so that you can see them alongside the other instance metrics.



*Knowing how utilized (or not) your instances are is a key component in choosing the right instance type to use for your workloads. Getting it wrong can cost you a lot of money!*



# Getting ready

You will need an EC2 instance running Linux, with the AWS CLI tool installed to perform this recipe. If you use an instance based on AWS Linux, you will have the AWS CLI tool installed for you.

The instance role or credentials you use to run the following commands must have permission to submit metrics to CloudWatch. This is the `cloudwatch:PutMetricData` permission.

```
<strong> aws cloudwatch put-metric-data \ </strong> <strong> --metric-name  
MemoryUsagePercent \</strong> <strong> --namespace CustomMetrics \</strong>  
<strong> --dimensions InstanceId=`curl -s \</strong> <strong>  
http://169.254.169.254/latest/meta-data/instance-id` \</strong> <strong> --unit Percent \</strong>
```

```
<strong> --value `free | grep Mem | awk '{print $3/$2 * 100.0}'`</strong>
```

2. Go to the CloudWatch console, and navigate to the Metrics dashboard. Your metric will appear under the namespace CustomMetrics, InstanceId, and the unique ID for the instance, with the metric name MemoryUsagePercent.

It can take *up to* 15 minutes for a custom metric to appear in the CloudWatch dashboard (although it usually takes less). Even for the built-in metrics, it may take a minute or two for the metric data to appear in the console.



# How it works...

In this recipe, we use the built-in `put-metric-data` AWS CLI command to send our metric to CloudWatch.

We start by defining the metric name and namespace that the values will appear under. This is important because it defines how we will see the metric in the console and dashboards. Names should identify and describe the metric. They do not need to be unique, as the dimension(s) we add will take care of that (we will discuss this later). Namespaces are used to group similar metrics together, like a category. The built-in metrics appear under the namespace `AWS/`. For example, EC2 metrics appear under the `AWS/EC2` namespace.

We then specify a dimension for the metric. A **dimension** is a way to uniquely identify similar metrics. In this case we are using the instance's ID to identify the metric, because the metric is unique to that instance, but we will likely have many instances of the `MemoryUsagePercent` metric (across many EC2 instances). We are obtaining the instance ID by querying (via the `curl` command) the instance metadata service, which is accessed over HTTP on the special IP address `169.254.169.254`.



*There's a lot of other useful information in the instance metadata. See the AWS documentation on instance metadata for more details <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html>.*

Next we specify a percent, because we know what kind of data we are dealing with. This argument can be left off if you don't know (or care), as CloudWatch attaches no significance to it (although some other applications may be able to use it, for example, for display).

Finally we specify the value to send. We work this value out dynamically from the output of the `free` command and use `awk` to convert it to a percentage of memory in use.

Once the metric is being sent to CloudWatch, we can view it in the console. The easiest way is to select your specific metric and view it in the Metrics section of the CloudWatch console.



# **There's more...**

This is a good real-world use-case to get started with your own custom metrics, but there's a lot more you can do with them.



# Cron

One-off metric values are rarely useful on their own. The real value comes when you can plot and see them over time; how they change, how fast they change, what their range is, and so on.

On Linux you can schedule a command easily with the `cron` command. By putting the AWS CLI commands in a script, and scheduling it with `cron` to run periodically, you can feed metrics consistently to CloudWatch, without the overhead of running a dedicated agent on your instances.



# Auto scaling

Instance-based metrics like memory usage become especially useful when collected from all the instances in an auto scaling group.

By collecting instance or even application-specific metrics (for example: number of threads used, internal request duration, and so on) you can make your auto scaling groups increase and decrease in size at the most appropriate times to your workload and performance profile.

To do this, make the auto scaling group name one of the dimensions (you can define multiple dimensions) sent along with your metric value.



# Backfilling

You can backfill metrics by running the same command and supplying an additional `--timestamp` argument. The timestamp argument accepts an ISO 8601 date and time stamp in UTC time for example: `2017-01-01T12:00:00.000Z`

Keep in mind that CloudWatch will only retain your metrics for a certain period, decided by the granularity of your metrics. The retention period is:

- Data points with a period of 60 seconds (1 minute) are available for 15 days
- Data points with a period of 300 seconds (5 minute) are available for 63 days
- Data points with a period of 3600 seconds (1 hour) are available for 455 days (15 months)



*While you can send metrics with millisecond precision, the minimum value CloudWatch will store is at the 1 minute level. Anything less than the 1 minute level and CloudWatch will aggregate the values. When aggregated, you can see some additional information about your metric; namely the sample size, minimum and maximum value, and the average of the values.*



## See also

- The *Creating monitoring dashboards* recipe
- The *Launching an Instance* recipe in [Chapter 4, Using AWS Compute](#).



# Creating monitoring dashboards

The real value of collecting metrics is the ability to spot trends and relationships (often unknown or unexpected) between disparate systems. With this kind of visibility, you are able to identify and troubleshoot issues before they become an incident.

In addition to providing a way to aggregate and view metrics from your systems, the CloudWatch service also makes it easy to create monitoring dashboards so that you can quickly and clearly view the most important metrics.

This recipe uses the AWS console because you cannot create dashboards via CloudFormation or the AWS CLI tool yet.



# Getting ready

You will need to have some metrics already present in CloudWatch in order to create a dashboard.

If you have been using AWS services (for example: EC2, RDS, DDB, and so on), then you should have plenty—almost all the AWS services populate metrics in CloudWatch by default.



# How to do it...

1. Navigate to the CloudWatch section of the AWS console:

The screenshot shows the AWS CloudWatch Management Console interface. The left sidebar menu is open, showing options like Dashboards, Alarms, Events, Rules, Logs, and Metrics (with a 'NEW' badge). The main content area has three main sections: 'Metric Summary', 'Alarm Summary', and 'Service Health'. The 'Metric Summary' section displays a message about monitoring operational and performance metrics for AWS resources. The 'Alarm Summary' section indicates no alarms have been created and provides instructions for setting up billing alarms. The 'Service Health' section shows the Amazon CloudWatch Service is operating normally. The top right corner shows user information (Rowan Udell, N. Virginia region) and navigation links (Additional Info, Getting Started Guide, Monitoring Scripts Guide, Overview and Features, Documentation, Forums, Report an Issue).

2. Go to the Dashboards section of the console via the link on the left-hand menu:

CloudWatch Management Console

Secure https://console.aws.amazon.com/cloudwatch/home?region=us-east-1#dashboards:

Guest

Services Resource Groups

CloudWatch Dashboards Alarms ALARM INSUFFICIENT OK Billing Events Rules Logs Metrics NEW

Dashboards Create dashboard

Name	Last updated (UTC)
You have no CloudWatch dashboards. Please create a dashboard.	

Additional Information

Getting Started Guide Documentation Forums Report an Issue

Feedback English

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

### 3. Click the Create Dashboard button:

CloudWatch Management Console

Secure https://console.aws.amazon.com/cloudwatch/home?region=us-east-1#dashboards:

Guest

Services Resource Groups

CloudWatch Dashboards Alarms ALARM INSUFFICIENT OK Billing Events Rules Logs Metrics NEW

Dashboards Create dashboard

Name	Last updated (UTC)
You have no CloudWatch dashboards. Please create a dashboard.	

Create new dashboard

Dashboard name: MyExampleDashboard

Cancel Create dashboard

Additional Information

Getting Started Guide Documentation Forums Report an Issue

Feedback English

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

### 4. Choose the type of widget you want to use to display your metric. In this example, we will choose the most versatile, Line:

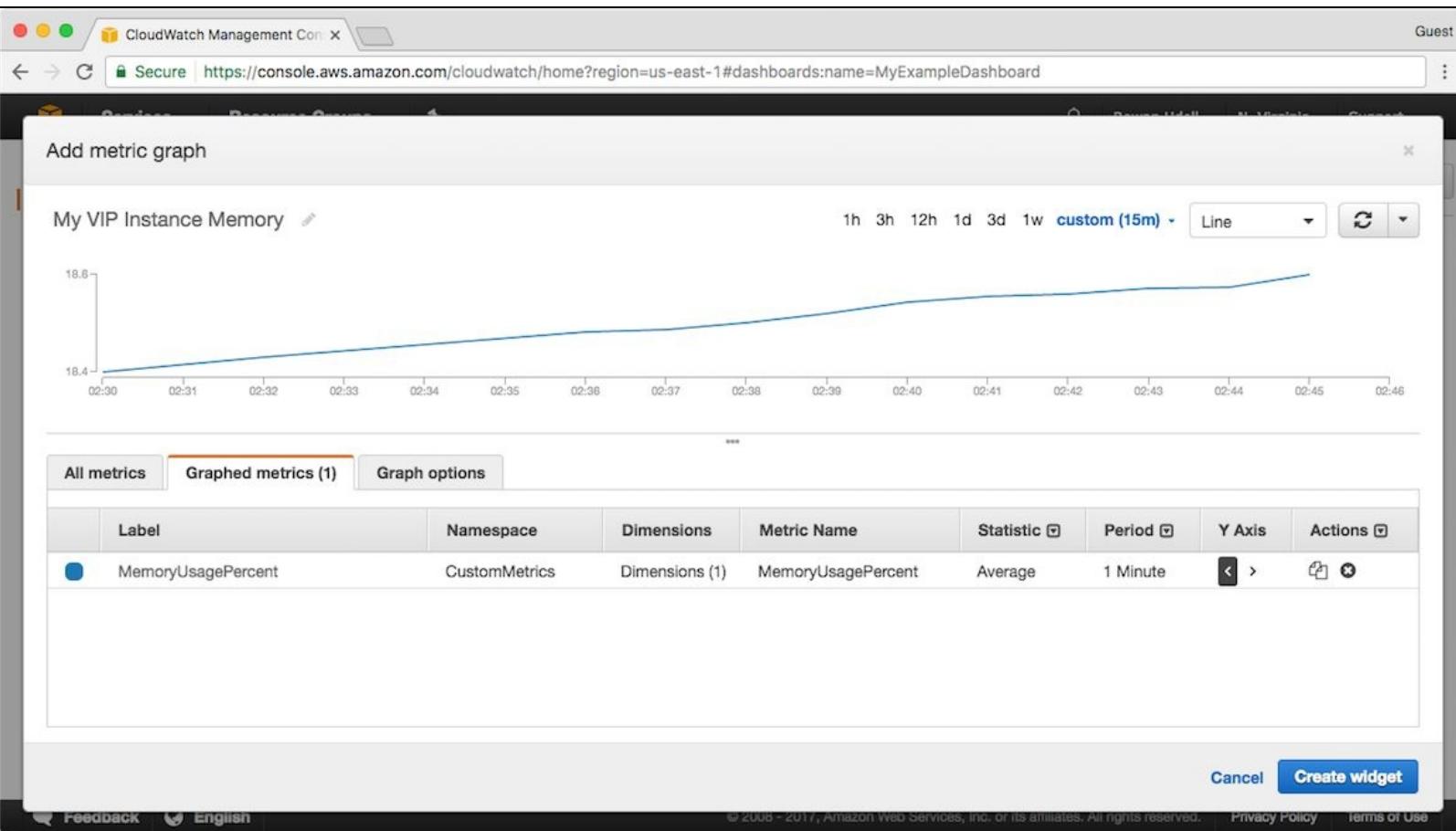
Screenshot of the CloudWatch Management Console showing the 'Add to this dashboard' dialog. The dialog lists four widget types: Line, Stacked area, Number, and Text. Each type has a preview image and a brief description. At the bottom are 'Cancel' and 'Configure' buttons.

5. Navigate the All metrics tab to find the metric(s) you want to include, selecting it by clicking the tick box on the left of the metric details. You will see a preview of the metric(s) and how they will look:

Screenshot of the CloudWatch Management Console showing the 'Add metric graph' dialog. It displays a line graph titled 'My VIP Instance Memory' showing memory usage over time. Below the graph, the 'Graphed metrics (1)' tab is selected, showing a table with one row. The table has columns for 'Instance Name (1)', 'Instanceld', and 'Metric Name'. The first row shows 'PutMetricData' under 'Instance Name (1)', 'i-06b65771be39a3913' under 'Instanceld', and 'MemoryUsagePercent' under 'Metric Name'. At the bottom are 'Cancel' and 'Create widget' buttons.

6. Once selected, you can modify how the metric is displayed via the settings on the Graphed metrics

tab. In this case we have given the widget a name, and changed the Period setting for our metric to 1 Minute to reflect the additional granularity available (You can see that the metric line appears *smoother* because of it).



- Once you click Create widget, you will see your widget on the dashboard. Once you click Save dashboard, it will appear under the Dashboards heading on the left-hand menu:

Screenshot of the CloudWatch Management Console showing the 'MyExampleDashboard' page. The dashboard contains a single line chart titled 'My VIP Instance Memory' showing 'MemoryUsagePercent' over time. The chart shows a steady increase from approximately 18.4% at 02:30 to about 18.7% at 02:45. The interface includes a sidebar with navigation links like CloudWatch, Dashboards, Alarms, Events, Rules, Logs, and Metrics. Top navigation bar shows 'Services', 'Resource Groups', and user information.

8. At a dashboard level, you can turn on Auto refresh and the refresh frequency interval:

Screenshot of the CloudWatch Management Console showing the 'MyExampleDashboard' page. A context menu is open on the right side of the dashboard header, displaying options for 'Auto refresh' and 'Refresh interval'. The 'Refresh interval' section is expanded, showing four options: '1 Minute' (selected), '2 Minutes', '5 Minutes', and '15 Minutes'. The dashboard itself displays the same 'My VIP Instance Memory' chart as the previous screenshot.

9. You can resize and rearrange your widgets by dragging them. Just remember to click Save dashboard to persist any changes:



Services

Resource Groups



Rowan Udell

N. Virginia

Support

MyExampleDashboard

Add widget

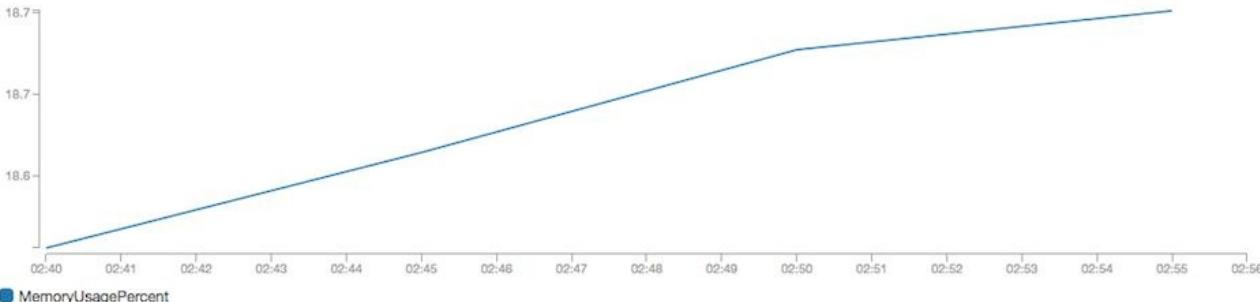
Actions

Save dashboard

1h 3h 12h 1d 3d 1w custom (15m)



## My VIP Instance Memory Usage



Feedback

English

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.

[Privacy Policy](#)[Terms of Use](#)



# There's more...

CloudWatch dashboard's value is the ease and simplicity that it allows you to publicize your most important metrics.



*As with any dashboard, make sure that the metrics you choose to display are relevant and actionable. There's no point in displaying a metric if there's no action required when it changes.*



# Widget types

Line graphs are not the only type of widget that can be displayed in a dashboard. There is also:

- Stacked area
- Number
- Text

Depending on the type of metrics you are collecting or are interested in, you should experiment with different types of widgets to display them. Not all metrics are suited to line graphs.



## See also

- The *Publishing custom metrics in CloudWatch* recipe.



# Creating a budget

One of the main attractions of using AWS, is its pay-as-you-go model. You only pay for what you use, no more and no less.

Unfortunately, this can sometimes result in what's known as **bill shock** at the end of the month. This happens when you do something that you might not know is a charged service, or you do not know how much is charged for it, and you don't find out until it's too late. Especially when getting started, users may not fully appreciate the cost of the activities they're undertaking.

There are also ways to optimize your costs on AWS, for example, by transferring at slower speeds, removing external access, and so on. All this means that you should be aware of your cost obligations, and manage them in real time. To this end, you can create budgets that help you be aware of your usage and spending.

While you can create budgets via the AWS CLI tool, it is useful to know how the Billing dashboard works for administration purposes, so we will use the AWS console for this recipe.



# Getting ready

By default, IAM Users do not have access to the billing section of the AWS console. You must perform these steps using the root login details for your account, or enable IAM access for other users, which is a one-off step.

While you should not generally use the root credentials for your AWS account when administering, creating budgets (which should happen only infrequently) is an exception.



*You should not be creating access keys for your root account under any circumstances, which is another reason why we use the console (and not the CLI) for this recipe.*



# How to do it...

1. Log in to the AWS console with your root credentials, and navigate to the My Billing Dashboard via the user menu accessed by clicking on your name in the top right:

The screenshot shows the AWS Management Console homepage. At the top, there's a navigation bar with links for 'Services', 'Resource Groups', and a user dropdown for 'Rowan Udell' in 'N. Virginia'. Below the navigation is a search bar and a 'Recently visited services' section which includes 'CloudWatch' and 'All services'. A large central area titled 'Build a solution' contains six quick-launch cards: 'Launch a virtual machine' (With EC2, ~1 minutes), 'Build a web app' (With Elastic Beanstalk, ~6 minutes), 'Deploy a serverless microservice' (With Lambda, API Gateway, ~2 minutes), 'Host a static website' (With S3, CloudFront, Route 53, ~5 minutes), 'Create a backend for your mobile app' (With Mobile Hub, ~5 minutes), and 'Register a domain' (With Route 53, ~3 minutes). To the right, there's a sidebar with 'My Account', 'My Billing Dashboard', 'My Security Credentials', and a 'Sign Out' link. Below that is a 'Get best practices' section with a link to AWS Trusted Advisor. Further down is an 'Announcements' section with links to 'Amazon Aurora - New Features' and 'Announcing PostgreSQL compatibility for Amazon Aurora'. At the bottom right of the sidebar is a 'See all' link.

2. The Billing dashboard displays your up-to-date usage for the month. Click on Budgets in the left-hand menu:

Billing Management Console X

Secure <https://console.aws.amazon.com/billing/home?region=us-east-1#/>

Services Resource Groups

Dashboard Bills Cost Explorer Budgets Reports Cost Allocation Tags Payment Methods Payment History Consolidated Billing Preferences Credits Tax Settings DevPay

## Billing & Cost Management Dashboard

What's New in AWS Billing and Cost Management?

- Easily upload your Cost and Usage Reports into Redshift and QuickSight
- Manage your spend with AWS Budgets
- Visualize your costs and usage with the newly-optimized Cost Explorer

Spend Summary Cost Explorer

Welcome to the AWS Account Billing console. Your last month and month-to-date costs appear below.

Current month-to-date balance for January 2017

**\$0.00**

No Amount Due \$0.00



- When you first arrive at the Budgets console, there will be no budgets to display. Click on the Create budget button to get started:

Billing Management Console X

Secure <https://console.aws.amazon.com/billing/home?region=us-east-1#/budgets>

Services Resource Groups

Dashboard Bills Cost Explorer Budgets Reports Cost Allocation Tags Payment Methods Payment History Consolidated Billing Preferences Credits Tax Settings DevPay

## AWS Budgets

AWS Budgets lets you quickly create custom budgets that will automatically alert you when your AWS costs or usage exceed, or are forecasted to exceed, the thresholds you set.

**Create budget**

### Getting started with AWS Budgets



**Create and manage budgets**  
Set custom cost and usage budgets to more easily manage your AWS spend. Monitor budget status from the Budgets Dashboard.



**Refine your budget using filters**  
Track your cost or usage across multiple dimensions by adding filters related to Service, Linked Account(s), Availability Zone, and more.



**Add notifications to a budget**  
Ensure that the right people know a threshold has been exceeded by sending notifications via email or publishing notifications to your SNS topic.

For more information about managing your AWS Budgets, refer to [Managing Your Costs With Budgets](#) in the Billing & Cost Management User Guide.

4. Start by filling out the budget details, such as `cost` for the measurement type, `Monthly` for the period, and the budget amount. Select the Start date (which defaults to the first of the current month), and optionally the End date. Leave the End date field blank to create a rolling budget that is reset each month:

The screenshot shows the AWS Billing Management Console with the URL <https://console.aws.amazon.com/billing/home?region=us-east-1#/createbudget>. The left sidebar lists various services like Dashboard, Bills, Cost Explorer, etc. The main area is titled "Create budget" with a sub-section "1 Budget details". It contains fields for Name (Monthly Budget), Select cost or usage (Cost), Period (Monthly), Start date (01/01/17), End date (blank), and Budgeted Amount (1000). To the right, a callout box titled "Create your AWS Budget" provides instructions: 1. Name your budget, specify the budgeted amount, and set the length of time the budget will be active. 2. Send notifications via email and SNS topic when a budget threshold is reached. Below the budget details, section 2 "Notifications (optional)" is shown with a note about creating a billing alarm for email alerts.

5. Next enter the notification details. This includes the threshold for notification, which we will set to be 80% (of our budget) in forecasted use. For e-mail notifications, simply enter the e-mail addresses you want to receive the notifications. Click Create when finished:

Billing Management Console Guest

[Secure https://console.aws.amazon.com/billing/home?region=us-east-1#/createbudget](https://console.aws.amazon.com/billing/home?region=us-east-1#/createbudget)

Budgeted Amount\* 1000

2 Notifications (optional)

You can create a billing alarm to receive e-mail alerts when your current or forecasted AWS charges meet the threshold you choose. Please provide at least one email contact or SNS topic ARN in order to receive notification.

Notify me when forecasted costs are greater than 80 % of budgeted amount

Email contacts rowan@rowanudell.com

SNS topic ARN Please fill in a valid SNS topic ARN Verify

SNS topic policy statement

+ Add new notification

\* Required Cancel Create

6. You will be returned to the Budgets section of the Billing dashboard, and you can see your newly created budget:

Billing Management Console Guest

[Secure https://console.aws.amazon.com/billing/home?region=us-east-1#/budgets](https://console.aws.amazon.com/billing/home?region=us-east-1#/budgets)

Services Resource Groups Global Support

Feedback English © 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Dashboard Bills Cost Explorer Reports Cost Allocation Tags Payment Methods Payment History Consolidated Billing Preferences Credits Tax Settings DevPay

AWS Budgets

Create budget Copy Edit Delete Download CSV

Filter by budget name

	Budget name	Current	Forecasted	Budgeted	Current vs. budg...	Forecasted vs. b...
<input type="checkbox"/>	Monthly Budget	\$0.00	Pending	\$1,000.00	0%	-

Viewing 1 to 1 of 1 budgets

*It may take up to 24 hours to receive updated data for your first budget(s)*

7. For each of the budgets you create, you can select it to view the full details:



Services

Resource Groups



rowan udell

Global

Support

Dashboard

Bills

Cost Explorer

**Budgets**

Reports

Cost Allocation Tags

Payment Methods

Payment History

Consolidated Billing

Preferences

Credits

Tax Settings

DevPay

## AWS Budgets

**Create budget**

Copy

Edit

Delete

Download CSV



Filter by budget name

		Budget name	Current	Forecasted	Budgeted	Current vs. budg...	Forecasted vs. b...
<input type="checkbox"/>	Monthly Budget	\$0.00	Pending	\$1,000.00	0%	<div style="width: 0%; height: 10px; background-color: #ccc;"></div>	--

**Budget details**

**Start date** 01/01/17

**End date** -

**Budget period** Monthly

**Variance analysis**

**Budget remaining** \$1,000.00

**% of budget remaining** 100%



Viewing 1 to 1 of 1 budgets

*It may take up to 24 hours to receive updated data for your first budget(s)*



# How it works...

The Billing dashboard is closely tied to the account itself, which is why it is not part of the regular services in the console. Accessing it via the user menu hints at the special access it requires. Generally, you would configure a budget when you first open a new AWS account, so you don't get any surprises in your bill at the end of the month.

If you get access denied messages in the Billing dashboard, it is most likely because you are using an IAM user and IAM access has not been enabled. You must use your root account credentials (such as that you used to create the account), or enable IAM access. IAM access can only be enabled by the root user.

When you first arrive at the billing section, you will see a high-level summary of your usage and expenses. As I performed this example in a new account, there's not much to see at this point. The Month-to-Date Spend by Service graph on the right can be particularly useful to find out what the most popular services you use are. This is a great place to start when trying to reduce or optimize your AWS spending.

We then navigate to the budgets section and create a new budget. Most of the details should be self-explanatory, and obvious for the purposes of budgeting. Your main choice is to decide if you want to alert on usage or costs. Cost budgets work against the dollar (or appropriate billing currency) amount you will be charged. Usage budgets work against a selected unit of usage, for example, instance hours or data transfer for EC2. A usage budget can only track one type of usage unit, so you will need to create multiple budgets to track the various units that you might be charged for. This is one reason why we prefer a cost budget, as it takes into account multiple forms of usage.

Specifying e-mail addresses to alert is the simplest way to send any alerts from the budget. For more advanced use cases, you can specify an SNS topic to receive notifications. An example might be if you wanted to receive budget alerts on your phone via an SMS message, or send the alert to a different system automatically (via HTTP/JSON).

Once finished, you can view all your budgets in the dashboard. You can repeat the process to create multiple budgets. This means you can create budgets for forecast usage and actual usage, as well as different time periods.



# Feeding log files into CloudWatch logs

**CloudWatch logs** is a managed, highly durable, log storage system in AWS. It's capable of ingesting logs from many sources. We're going to focus on what is probably the most common use case which is shipping logs off your EC2 instances into CloudWatch logs.

This capability is particularly important in highly dynamic auto scaling environments. Since the lifetime of your EC2 instances can be quite short, any logs which are written only to a local disk will be lost upon instance termination. You'll inevitably find yourself wishing you had access to server logs after an instance has disappeared.

The following pattern we're about to show you allows you to aggregate, search and filter log entries across a number of sources. You can then create custom metrics and trigger alarms based on log activity. Super handy!

In this recipe we're going to:

- Launch an EC2 instance
- Configure it to send logs to CloudWatch logs
- Create a filter based on SSH logins to the instance
- Send ourselves an e-mail alert on filter matches



*This might be something you'd consider doing on your bastion boxes since they will typically be the sole point of SSH access to your environments and it can be a good idea to make a lot of noise if people are logging in to production servers.*



# Getting ready

We're going to do all of this in `us-east-1` with the AWS Linux AMI. If you wish to do this in a different region you'll simply need to provide a different AMI ID to the template we're going to create.

Let's get in to it; you'll need the following:

- The VPC ID of your default VPC in `us-east-1`. You don't have to use the default VPC, you'll just need to make sure you choose a VPC which has a public subnet (which is configured to assign public IP addresses)
- The subnet ID of the public subnet
- An SSH key pair configured in `us-east-1`
- An e-mail address we can send alerts to

AmiId:

Type: AWS::EC2::Image::Id Description: AMI ID to launch instances from Default: ami-0b33d91d VpcId:

Type: AWS::EC2::VPC::Id Description: VPC where load balancer and instance will launch SubnetIDs:

Type: List<AWS::EC2::Subnet::Id> Description: Public subnet where the instance will launch (pick at least 1) KeyPair:

Type: AWS::EC2::KeyPair::KeyName Description: Key to launch EC2 instance with AlertEmail:

Type: String

Description: Email Address which alert emails will be sent to

ExampleRole:

Type: AWS::IAM::Role Properties:

AssumeRolePolicyDocument: Version: "2012-10-17"

Statement:

-

Effect: Allow

Principal:

Service:

- ec2.amazonaws.com Action:

- sts:AssumeRole

Path: /

Policies:

PolicyName: WriteToCloudWatchLogs PolicyDocument:

Version: "2012-10-17"

Statement:

Effect: Allow

Action:

- logs:CreateLogGroup - logs:CreateLogStream - logs:PutLogEvents -  
logs:DescribeLogStreams Resource: "\*"

ExampleInstanceProfile: Type: AWS::IAM::InstanceProfile Properties:

Roles:

- !Ref ExampleRole Path: /

ExampleEC2InstanceSecurityGroup:

Type: AWS::EC2::SecurityGroup Properties:

GroupDescription: Security Group for example Instance SecurityGroupIngress: -  
IpProtocol: tcp CidrIp: "0.0.0.0/0"

FromPort: 22

ToPort: 22

VpcId: !Ref VpcId

ExampleEC2Instance:

Type: AWS::EC2::Instance Properties:

IamInstanceProfile: !Ref ExampleInstanceProfile InstanceType: t2.nano KeyName:  
!Ref KeyPair UserData:

Fn::Base64:

Fn::Sub: |

```
#!/bin/bash -ex
```

```
yum update -y
```

```
yum install -y awslogs cat << EOF > /etc/awslogs/config/var-log-secure.conf  
[/var/log/secure]
```

```
datetime_format = %b %d %H:%M:%S
```

```
file = /var/log/secure buffer_duration = 5000
```

```
log_stream_name = {instance_id}
```

```
initial_position = start_of_file log_group_name = /var/log/secure
```

# EOF

```
    service awslogs start chkconfig awslogs on ImageId: !Ref AmiId SecurityGroupIds: -  
Fn::GetAtt: ExampleEC2InstanceSecurityGroup.GroupId SubnetId: !Select [ 0, Ref:  
SubnetIDs ]
```

ExampleSNSTopic:

Type: AWS::SNS::Topic Properties:

Subscription:

-

Endpoint: !Ref AlertEmail Protocol: email

ExampleLogGroup:

Type: AWS::Logs::LogGroup Properties:

LogGroupName: /var/log/secure RetentionInDays: 7

ExampleLogsMetricFilter: Type: AWS::Logs::MetricFilter Properties:

FilterPattern: "Accepted publickey for ec2-user from"

LogGroupName: /var/log/secure MetricTransformations: -

MetricValue: "1"

MetricNamespace: SSH/Logins MetricName: LoginCount DependsOn:

ExampleLogGroup

ExampleLoginAlarm:

Type: AWS::CloudWatch::Alarm Properties:

AlarmDescription: SSH Login Alarm AlarmActions:

- Ref: ExampleSNSTopic MetricName: LoginCount Namespace: SSH/Logins Statistic: Sum

Period: 60

EvaluationPeriods: 1

Threshold: 0

ComparisonOperator: GreaterThanThreshold

Outputs:

ExampleEC2InstancePublicIp: Value: !GetAtt [ ExampleEC2Instance, PublicIp ]

```
<strong> aws cloudformation create-stack \ </strong> <strong> --template-body \<br/> file://05-feed-log-files-in-to-cloudwatch-logs.yaml \ </strong> <strong> --stack-name example-cloudwatchlogs \ </strong> <strong> --capabilities CAPABILITY_IAM \<br/> </strong> <strong> --parameters \ </strong> <strong> ParameterKey=VpcId,ParameterValue=<your-vpc-id> \ </strong> <strong> ParameterKey=SubnetIDs,ParameterValue='<your-subnet-id>' \ </strong> <strong> ParameterKey=KeyPair,ParameterValue=<your-ssh-key-name> \ </strong> <strong> ParameterKey=AlertEmail,ParameterValue=<your-email-address></strong>
```

10. Before proceeding you'll need to check your e-mail and confirm your subscription to the SNS topic. If you don't do this you won't receive any alerts from CloudWatch:



You have chosen to subscribe to the topic:

**arn:aws:sns:us-east-1:067180688831:example-cloudwatchlogs-ExampleSNSTopic-VMKGCL96G3NE**

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):

[Confirm subscription](#)

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)

In the following screenshot, an example of confirmed subscription is illustrated:

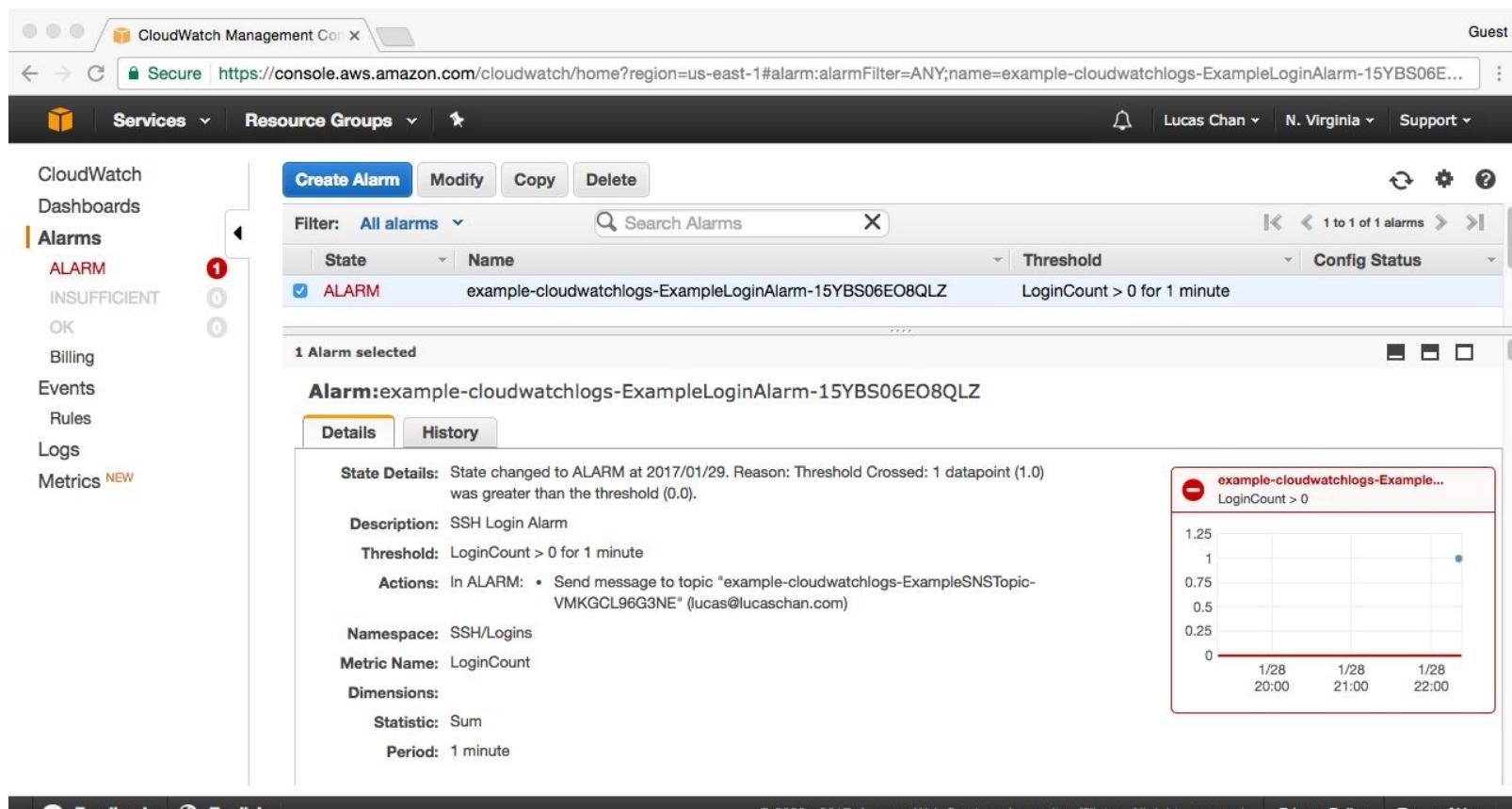
## Subscription confirmed!

You have subscribed lucas@lucaschan.com to the topic:  
**example-cloudwatchlogs-ExampleSNSTopic-VMKGCL96G3NE**.

Your subscription's id is:  
arn:aws:sns:us-east-1:067180688831:example-cloudwatchlogs-  
ExampleSNSTopic-VMKGCL96G3NE:e14f5c97-6189-4403-a9d1-fb0c18b7alf0

If it was not your intention to subscribe, [click here to unsubscribe](#).

11. Go ahead and SSH to your instance. If your login is successful, you'll see your alarm triggered in the CloudWatch web console:



The screenshot shows the AWS CloudWatch Management Console interface. On the left, a sidebar menu is open with 'Alarms' selected. Under 'Alarms', there is a red '1' badge next to 'ALARM'. The main content area displays a table of alarms. One alarm is selected, highlighted with a yellow border. The alarm details are as follows:

- Name:** example-cloudwatchlogs-ExampleLoginAlarm-15YBS06E08QLZ
- Description:** SSH Login Alarm
- Threshold:** LoginCount > 0 for 1 minute
- Actions:** In ALARM: Send message to topic "example-cloudwatchlogs-ExampleSNSTopic-VMKGCL96G3NE" (lucas@lucaschan.com)
- Namespace:** SSH/Logins
- Metric Name:** LoginCount
- Dimensions:**
- Statistic:** Sum
- Period:** 1 minute

A chart on the right shows a single data point at 1.0 on the y-axis at 22:00 on 1/28. The x-axis shows 20:00, 21:00, and 22:00.

An e-mail will land in your inbox as shown in the following screenshot:



AWS Notifications <no-reply@sns.amazonaws.com>  
to lucas Show details

9:22 AM (less than a minute ago)

Inbox

Reply

More ▾

You are receiving this email because your Amazon CloudWatch Alarm "example-cloudwatchlogs-ExampleLoginAlarm-15YBS06EO8QLZ" in the US East - N. Virginia region has entered the ALARM state, because "Threshold Crossed: 1 datapoint (1.0) was greater than the threshold (0.0)." at "Saturday 28 January, 2017 22:22:29 UTC".

View this alarm in the AWS Management Console:

<https://console.aws.amazon.com/cloudwatch/home?region=us-east-1#s=Alarms&alarm=example-cloudwatchlogs-ExampleLoginAlarm-15YBS06EO8QLZ>

Alarm Details:

- Name: example-cloudwatchlogs-ExampleLoginAlarm-15YBS06EO8QLZ
- Description: SSH Login Alarm
- State Change: INSUFFICIENT\_DATA -> ALARM
- Reason for State Change: Threshold Crossed: 1 datapoint (1.0) was greater than the threshold (0.0).
- Timestamp: Saturday 28 January, 2017 22:22:29 UTC
- AWS Account: 067180688831

Threshold:

- The alarm is in the ALARM state when the metric is GreaterThanThreshold 0.0 for 60 seconds.

Monitored Metric:

- MetricNamespace: SSH/Logins
- MetricName: LoginCount
- Dimensions:
- Period: 60 seconds
- Statistic: Sum
- Unit: not specified



# How it works...

It's important that you understand the difference between log streams and log groups.

**Log streams** are log sequences which come from a single source. This could be an EC2 instance, an application process, or another source within AWS. In our case the name of our log stream is the ID of our EC2 instance. In fact, the CloudWatch logs agent will set the `log_stream_name` to the instance ID by default.

**Log groups** are collections of log streams with the same properties. In our previous example, the log groups will correspond to `/var/log/secure`. So, we end up with a configuration which looks like:  
`log_group_name = /var/log/secure` `log_stream_name = {instance_id}`

When you install the CloudWatch logs agent, it actually sets up `/var/log/messages` in exactly the same manner as we've just described:

```
| log_group_name = /var/log/messages  
|   log_stream_name = {instance_id}
```

Once the agent has started, it will ship new log entries off the box to CloudWatch logs approximately every 5 seconds.



# There's more...

- CloudWatch logs supports ingestion of traditional text-based log entries as well as JSON formatted logs.
- Logs can be ingested from other sources including CloudTrail, IAM, Kinesis Streams and Lambda.
- By default, logs are stored indefinitely. You can customize this time period to suit your needs however.
- Metric filters, like the one we created previously, can be used to graph and chart in the CloudWatch console. Add them to your dashboards as well as your alerting system.
- The CloudWatch web console allows you to test metric filters before you add them. Using this feature will save you a lot of trial and error with CloudFormation. Don't rely on the web console completely however: you should move these metric filters to CloudFormation as soon as you get them right.
- There is a one-one relationship between a log stream and a log source. For example, you can't have multiple instances sending `/var/log/secure` to the same log stream.
- The non-alarm state for the alarm we've created, will be `INSUFFICIENT_DATA`. This is because our metric filter outputs a value only if a login is detected.



# Database Services

In this chapter, we will cover:

- Creating a database with automatic failover
- Creating a NAT gateway
- Creating a database read-replica
- Promoting a read-replica to master
- Creating a one-time database backup
- Restoring a database from a snapshot
- Migrating a database
- Calculating DynamoDB performance



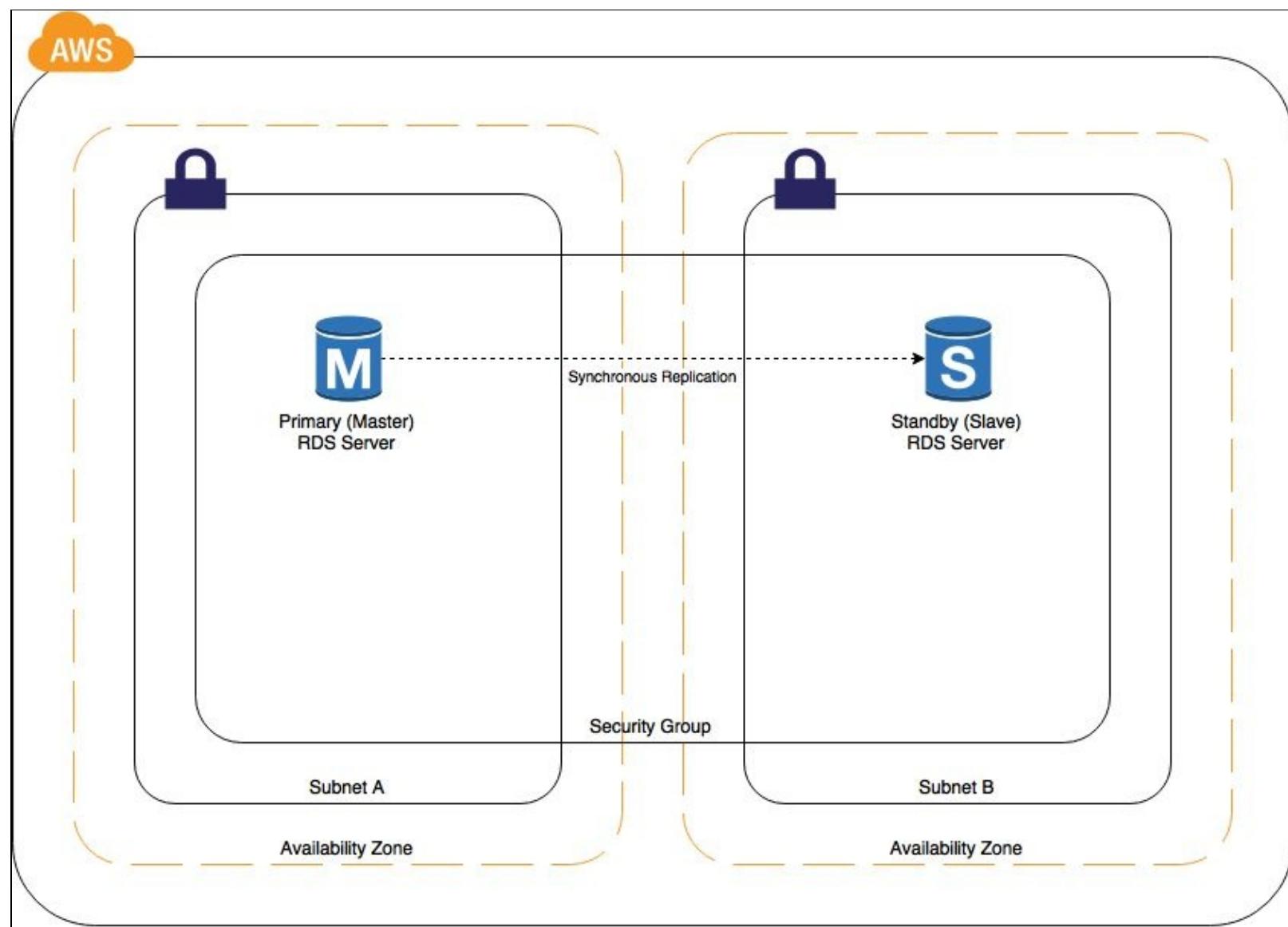
# Introduction

Having a persistent storage service is a key component of effectively using the AWS cloud for your systems. By ensuring that you have a highly available, fault-tolerant location to store your application state in, you can stop depending on individual servers for your data.



# Creating a database with automatic failover

In this recipe, we're going to create a MySQL RDS database instance configured in multi-AZ mode to facilitate automatic failover.



Database with automatic failover



# Getting ready

The default VPC will work fine for this example. Once you are comfortable with creating databases, you may want to consider a VPC containing private subnets that you can use to segment your database away from the Internet and other resources (in the style of a three tier application). Either way, you'll need to note down the following:

- The ID of the VPC
- The CIDR range of the VPC
- The IDs of at least two subnets in your VPC. These subnets need to be in different Availability Zones, for example, `us-east-1a` and `us-east-1b`

VPCId:

Type: AWS::EC2::VPC::Id Description: VPC where DB will launch SubnetIds:

Type: List<AWS::EC2::Subnet::Id> Description: Subnets where the DB will launch (pick at least 2) SecurityGroupAllowCidr: Type: String

Description: Allow this CIDR block to access the DB

Default: "172.30.0.0/16"

DBUsername:

Type: String

Description: Username to access the database MinLength: 1

AllowedPattern: "[a-zA-Z][a-zA-Z0-9]\*"

ConstraintDescription: must start with a letter, must be alphanumeric

DBPassword:

Type: String

Description: Password to access the database MinLength: 1

AllowedPattern: "[a-zA-Z0-9]\*"

NoEcho: true

ConstraintDescription: must be alphanumeric

DBInstanceClass:

Type: String

Description: The instance type to use for this database Default: db.t2.micro

DBStorageAmount:

Type: Number

Description: Amount of storage to allocate (in GB) <strong> </strong> Default: 10

DBStorageType:

Type: String

Description: Type of storage volume to use (standard [magnetic] or gp2) Default: standard AllowedValues:

- standard

- gp2

DBEngineVersion:

Type: String

Description: DB engine version Default: "5.7.11"

DBPort:

Type: Number

Description: Port number to allocate Default: 3306

MinValue: 1150

MaxValue: 65535

DBMultiAZ:

Type: String

Description: Should this DB be deployed in Multi-AZ configuration?

Default: true

AllowedValues:

- true

- false

DBBackupRetentionPeriod: Type: Number

Description: How many days to keep backups (0 disables backups) Default: 1

MinValue: 0

MaxValue: 35

ExampleDBSecurityGroup:

Type: AWS::EC2::SecurityGroup Properties:

GroupDescription: Example security group for inbound access to DB

SecurityGroupIngress: - IpProtocol: tcp CidrIp: !Ref SecurityGroupAllowCidr  
FromPort: !Ref DBPort ToPort: !Ref DBPort VpcId: !Ref VPCId

ExampleDBSubnetGroup:

Type: AWS::RDS::DBSubnetGroup Properties:

DBSubnetGroupDescription: Example subnet group for example DB

SubnetIds:

- Fn::Select: [ 0, Ref: SubnetIds ]
- Fn::Select: [ 1, Ref: SubnetIds ]

ExampleDBInstance:

Type: AWS::RDS::DBInstance Properties:

AllocatedStorage: !Ref DBStorageAmount BackupRetentionPeriod: !Ref DBBackupRetentionPeriod DBInstanceClass: !Ref DBInstanceClass  
DBSubnetGroupName: !Ref ExampleDBSubnetGroup Engine: mysql

EngineVersion: !Ref DBEngineVersion MasterUsername: !Ref DBUsername  
MasterUserPassword: !Ref DBPassword MultiAZ: !Ref DBMultiAZ

StorageType: !Ref DBStorageType VPCSecurityGroups: - !GetAtt  
ExampleDBSecurityGroup.GroupId

Outputs:

ExampleDbHostname: Value: !GetAtt ExampleDBInstance.Endpoint.Address

<strong>aws cloudformation create-stack \

--stack-name rds1 \

--template-body \

file://06-create-database-with-automatic-failover.yaml \

--parameters \

ParameterKey=DBUsername,ParameterValue=<username> \

ParameterKey=DBPassword,ParameterValue=<password> \

ParameterKey=SubnetIds,"ParameterValue='<subnet-id-a>, \

<subnet-id-b>'" \

ParameterKey=VPCId,ParameterValue=<vpc-id></strong>



# How it works...

In a multi-AZ configuration, AWS will provision a standby MySQL instance in a separate Availability Zone. Changes to your database will be replicated to the standby DB instance in a synchronous fashion. If there is a problem with your primary DB instance AWS will automatically failover to the standby, promote it to be the primary DB, and provision a new standby.

You don't have access to query standby databases directly. So you can't use it to handle all of your read queries, for example. If you wish to use additional database instances to increase read capacity, you'll need to provision a *read-replica*. We'll cover those in a separate recipe.

Backups will always be taken from the standby instance, which means there is no interruption to your DB availability. This is not the case if you opted against deploying your DB in multi-AZ mode.

When you deploy this example it will take roughly 20 minutes or more for the stack to report completion. This is because the RDS service needs to go through the following process in order to provision a fully working multi-AZ database:

- Provision the primary database
- Back up the primary database
- Provision the standby database using the backup from the primary
- Configure both databases for synchronous replication

## WARNING



*Be careful about making changes to your RDS configuration after you've started writing data to it, especially when using CloudFormation updates. Some RDS configuration changes require the database to be re-provisioned, which can result in data loss. We'd recommend using CloudFormation change sets, which will give you an opportunity to see which changes are about to cause destructive behavior. The CloudFormation RDS docs also provide some information on this.*



# There's more...

- You can define a maintenance window for your RDS instance. This is the time period when AWS will perform maintenance tasks such as security patches or minor version upgrades. If you don't specify a maintenance window (which we don't in this example), one is chosen for you.



# Creating a NAT gateway

Unless required, your instances should not be publicly exposed to the Internet. When your instances are on the Internet, you have to assume that they will be attacked at some stage.

This means most of your workloads should run on instances in private subnets. Private subnets are those that are not connected directly to the Internet.

In order to give your private instances access to the Internet you use **network address translation (NAT)**. A NAT gateway allows your instances to initiate a connection to the Internet, without allowing connections from the Internet.



# Getting ready

For this recipe, you must have the following resources:

- A VPC with an **Internet gateway (IGW)**
- A public subnet
- A private subnet route table

You will need the IDs for the public subnet and private subnet route table. Both of these resources should be in the same AZ.

AWSTemplateFormatVersion: "2010-09-09"

Description: Create NAT Gateway and associated route.

Parameters:

PublicSubnetId: Description: Public Subnet ID to add the NAT Gateway to Type: AWS::EC2::Subnet::Id RouteTableId: Description: The private subnet route table to add the NAT

Gateway route to Type: String

Resources:

EIP:

Type: AWS::EC2::EIP

Properties:

Domain: vpc

NatGateway:

Type: AWS::EC2::NatGateway Properties:

AllocationId: !GetAtt EIP.AllocationId SubnetId: !Ref PublicSubnetId

Route:

Type: AWS::EC2::Route Properties:

RouteTableId: !Ref RouteTableId DestinationCidrBlock: 0.0.0.0/0

NatGatewayId: !Ref NatGateway



# How it works...

The parameters required for this recipe are as follows:

- A public subnet ID
- A private subnet route table ID

The public subnet ID is needed to host the NAT gateway, as it must have Internet access. The private subnet route table will be updated with a route to the NAT gateway.

Using the AWS NAT gateway service means that AWS takes care of hosting and securing the service for you. The service will be hosted redundantly in a single AZ.



*You can use the recipe multiple times to deploy NAT gateways in each of your private subnets. Just make sure the public subnet and the private subnet are in the same AZ.*

To cater for the unlikely event of an AZ outage (unlikely, but possible) you should deploy a NAT gateway per subnet. This means if one NAT gateway goes offline, instances in the other AZ can continue to access the Internet as normal. You *are* deploying your application in multiple AZs, aren't you?

This recipe will only work if you have created your own private subnets, as the default subnets in a new AWS account are all *public*. Instances in a public subnet have direct access to the Internet (via an IGW), so they do not need a NAT gateway.



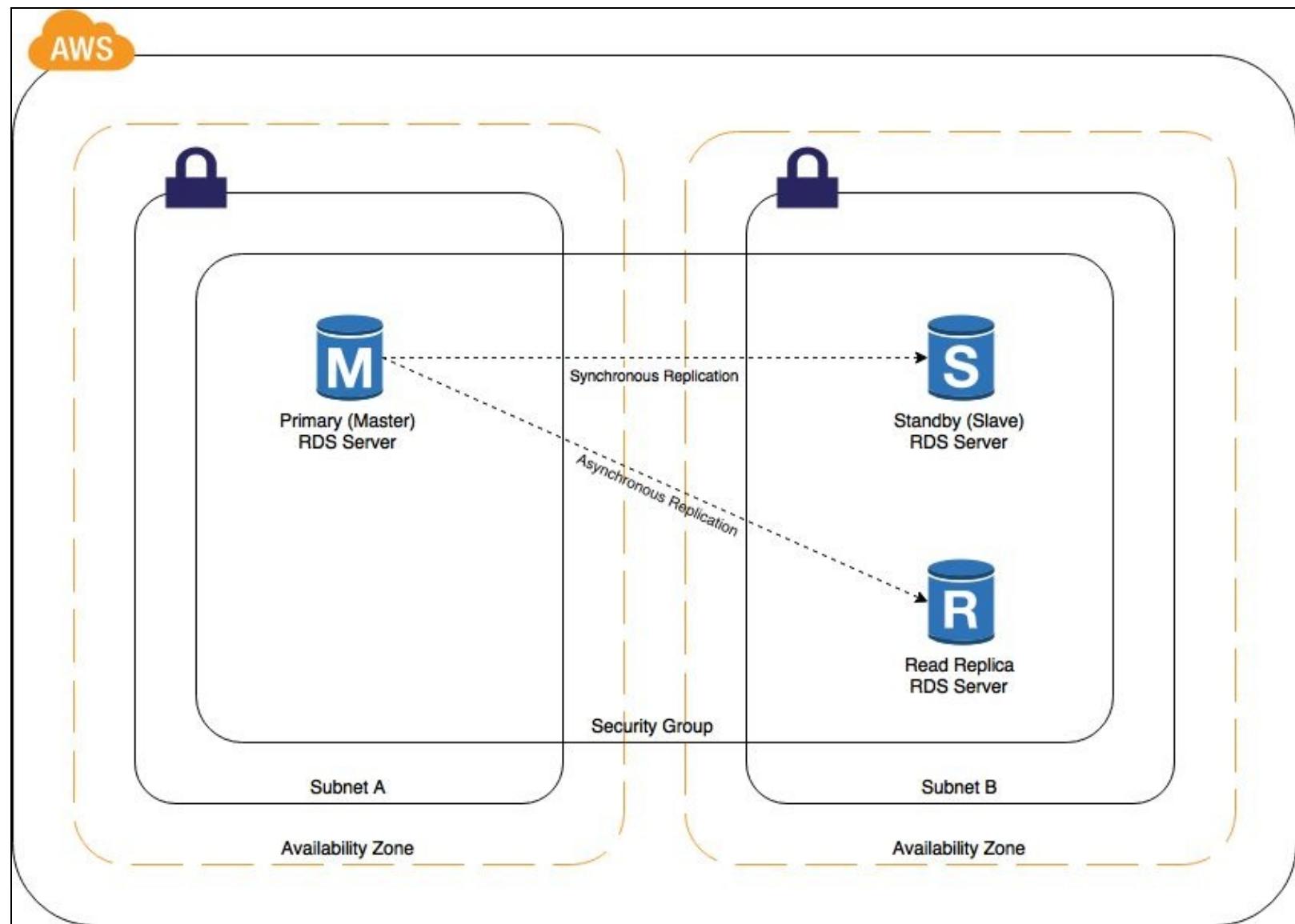
## See also

- The *Building a secure network* recipe in [Chapter 7, Networking](#)



# Creating a database read-replica

This recipe will show you how to create an RDS read-replica. You can use read-relicas in order to increase the performance of your application by off-loading database reads to a separate database instance. You can provision up to five read-relicas per source DB.





# Getting ready

You will need an RDS DB deployed with backup retention enabled. We are going to build upon the DB deployed in the previous *Creating a database with automatic failover* recipe.

You're going to need the following values:

- The identifier for your source RDS instance, for example, `eexocwv5k5kv5z`
- A unique identifier for the read-replicate we're going to create, for example, `read-replica-1`



# How to do it...

In the AWS CLI, type this command:

```
aws rds create-db-instance-read-replica \
--source-db-instance-identifier <source-db-identifier> \
--db-instance-identifier <unique-identifier-for-replica>
```



# How it works...

RDS will now go ahead and create a new read-replica for you.

Some parameters are inherited from the source instance and can't be defined at the time of creation:

- Storage engine
- Storage size
- Security group

The CLI command accepts some parameters that we could have defined, but didn't to keep things simple. They will instead be inherited from the source database. The main two are as follows:

- `--db-instance-class`: The same class as the source instance is used
- `--db-subnet-group-name`: The source instance's subnet group will be used and a subnet is chosen at random (hence, an Availability Zone is chosen at random)



# There's more...

- Read-replicas are deployed in a single Availability Zone; there is no standby read-replica.
- It's not possible to enable backups on read-replicas during time of creation. This must be configured afterwards.
- The default storage type is `standard` (`magnetic`). You can increase performance by choosing `gp2` or using provisioned IOPS.
- It's possible to add MySQL indexes directly to a read-replica to further increase read performance. These indexes are not required to be present on the primary DB.
- Using read-replicas for availability purposes is more of a complimentary DR strategy and shouldn't be used in place of multi-AZ RDS. A multi-AZ configuration gives you the benefit of failure detection and automatic failover.
- It is possible to deploy a read-replica in an entirely different region.
- Unlike the replication between a primary and standby DB (which is synchronous), replication to a read-replica is asynchronous. This means that it's possible for a read-replica to fall behind the primary. Keep this in mind when sending time sensitive read queries to your read-replicas.



# Promoting a read-replica to master

We're going to show you how to promote an RDS read-replica to be a primary instance. There are a few reasons you might like to do this:

- To handle a table migration that would typically cause a large amount of downtime, especially when messing with columns or indexes
- Because you need to implement sharding
- Recovery from failure, should you choose not to deploy your existing primary in multi-AZ mode (not recommended)



# Getting ready

You're going to need the unique ID, which has been assigned to an RDS read-replica. If you followed the previous *Creating a database with automatic failover*, and *Creating a database read-replica* recipes, then you'll be all set.

It's also a good idea to have backups enabled on this read-replica prior to promoting it. This shortens the promotion process because you won't need to wait for a backup to be taken. You'll want to set the backup retention period to a value between 1 and 8.



*Enabling backups on your read-replica will cause it to reboot!*



*You can drop the `--apply-immediately` parameter if you prefer to wait for the reboot to happen during the configured maintenance window. But you'll still want to wait until after the reboot happens before you continue with the promotion process.*

*To ensure that you have the most up-to-date data before promotion you'll want to stop all write traffic to the current source primary DB before going ahead. It's also a good idea to make sure that the replication lag on your read-replica is 0 (you can check this in CloudWatch).*

```
<strong> aws rds promote-read-replica </strong> <strong> --db-instance-identifier  
<identifier-for-read-replica></strong>  
  
<strong> aws rds modify-db-instance </strong> <strong> --db-instance-identifier  
<identifier-for-new-primary> </strong> <strong> --multi-az </strong>  
  
<strong> --apply-immediately</strong>
```



# **Creating a one-time database backup**

We're now going to show you how to make a one-off snapshot of your database. You might opt to do this if you have a specific requirement around keeping a point in time backup of your DB. You might also want to take a snapshot for the purpose of creating a new working copy of your dataset.



# Getting ready

In order to proceed you're going to need the following:

- The identifier for the RDS instance you wish to back up
- A unique identifier that you'd like to assign to this snapshot

The snapshot identifier has some constraints:

- It needs to start with a letter
- It must not be longer than 255 characters



*If your primary database isn't running in a multi-AZ configuration then be aware that creating a snapshot will cause an outage. In a multi-AZ configuration the snapshot is taken on the standby instance so no outage occurs.*



# How to do it...

Type the following AWS CLI command to initiate the creation of a snapshot. You'll need to wait for a few minutes for the snapshot to complete before you can use it:

```
aws rds create-db-snapshot \
--db-instance-identifier <primary-rds-id> \
--db-snapshot-identifier <unique-id-for-snapshot>
```



# **Restoring a database from a snapshot**

We'll now talk through how to restore a database from a snapshot. This process creates a new database that will retain a majority of the configuration of the database that the snapshot was taken from.



# Getting ready

You'll need the following pieces of information:

- The ID of the snapshot you wish to restore from
- A name or identifier that you wish to give to the database we're about to create



*AWS does not allow RDS services in your account to share the same identifier. If the source database is still online you'll need to make sure to choose a different identifier (or rename the source database).*

```
<strong> aws rds restore-db-instance-from-db-snapshot </strong> <strong> --db-snapshot-identifier <name-of-snapshot-to-restore> </strong> <strong> --db-instance-identifier <name-for-new-db> </strong> <strong> --db-subnet-group-name <your-db-subnet-group> </strong> <strong> --multi-az</strong>

<strong> aws rds modify-db-instance </strong> <strong> --db-instance-identifier <name-of-newly-restored-db> </strong> <strong> --vpc-security-group-ids <id-of-security-group></strong>
```

The `modify-db-instance` command will return an error unless the state of the target database is available.

Also, security group names aren't valid with this command; you'll need to use a security group ID instead, for example, `sg-7603d50a`.



## **There's more...**

The previous command includes the parameter for enabling multi-AZ on the new DB. If you'd like the new DB to be running in single-AZ mode only then can you simply remove this flag.



# Migrating a database

In this recipe, we will use **Database Migration Service (DMS)** to move an external database into **Relational Database Service (RDS)**.

Unlike many of the other recipes, this will be performed manually through the web console.

Most database migrations are one-off, and there are many steps involved. We suggest that you first perform the process manually via the console before automating it, if required (which you can do with the AWS CLI tool or SDKs).



# Getting ready

For this recipe you will need the following:

- An external database
- An RDS database instance

The source database in this example is called **employees**, so substitute your own database name as required.

Both databases must be accessible from the replication instance that will be created as part of the recipe. The simplest way to do this is to allow access to the databases from the Internet, but obviously this has security implications.



# How to do it...

## 1. Navigate to the DMS console:

The screenshot shows the AWS DMS console homepage. The left sidebar has a 'Dashboard' section with links for 'Get started', 'Tasks', 'Endpoints', 'Certificates', 'Replication instances', and 'Subnet groups'. The main content area has a 'What's new' section with three bullet points: 'AWS DMS expands to Seoul, Mumbai, and São Paulo regions.', 'DMS officially supports ongoing replication, enables SSL, and adds SAP ASE support.', and 'SCT supports conversions from Oracle DW and Teradata to Amazon Redshift.' Below that is an 'Active tasks' section stating 'No active tasks'. Under 'Get started', there is a note about needing a Replication Instance and a 'Create migration' button. To the right, there's an 'Additional info' section with links to 'Getting started', 'Overview and features', 'Documentation', 'API reference', 'Pricing', 'Forums', 'AWS Java SDK', and 'AWS CLI'. There's also a 'Related services' section linking to 'Amazon RDS' and 'Data Pipeline'. At the bottom, there are 'Feedback', 'English', and language selection buttons, along with copyright and legal links.

## 2. Click on Create Migration to start the migration wizard:

Database Migration Service Guest

Secure https://console.aws.amazon.com/dms/home?region=us-east-1#create-first-migration:

Services Resource Groups Rowan Udell N. Virginia Support

**Step 1: Welcome**

- Step 2: Replication instance
- Step 3: Database endpoints
- Step 4: Task

## Welcome to AWS Database Migration Service

AWS Database Migration Service tasks require at least a source, a target, and a replication instance. Your source is the database you wish to move data from and the target is the database you're moving data to. The replication instance processes the migration tasks and requires access to your source and target endpoints inside your VPC. Replication instances come in different sizes depending on your performance needs. If you're migrating to a different database engine, AWS Schema Conversion Tool can generate the new schema for you. [Download AWS Schema Conversion Tool](#)

The diagram illustrates the data replication process. It shows three main components: 'Source Database' (represented by a server icon with an orange arrow pointing up), 'Task on replication instance' (represented by a gear icon with a checkmark), and 'Target Database' (represented by a cloud icon with a database icon inside). Arrows indicate the flow from the source to the task and then to the target.

Data is replicated from a **source** database to a **target** database by a **task** running on a **replication instance**.

Cancel Previous Next

Feedback English © 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

3. Specify the details for your replication instance. Unless you have a specific VPC configuration, the defaults will be fine:

Database Migration Service Guest

Secure https://console.aws.amazon.com/dms/home?region=us-east-1#create-first-migration:

Services Resource Groups Rowan Udell N. Virginia Support

## Setting up your first replication task

**Step 1: Welcome**

- Step 2: Replication instance**
- Step 3: Database endpoints
- Step 4: Task

### Create replication instance

A replication instance initiates the connection between the source and target databases, transfers the data, and caches any changes that occur on the source database during the initial data load. Use the fields below to configure the parameters of your new replication instance including network and security information, encryption details, and performance characteristics.

Name*	<input type="text" value="replication-instance-1"/>	<span style="font-size: small;">i</span>
Description*	<input type="text" value="Replication instance 1"/>	<span style="font-size: small;">i</span>
Instance class*	<input type="text" value="dms.t2.small"/>	<span style="font-size: small;">i</span>
VPC*	<input type="text" value="vpc-a3ea49c5"/>	<span style="font-size: small;">i</span>
Multi-AZ	<input type="text" value="No"/>	<span style="font-size: small;">i</span>
<input checked="" type="checkbox"/> Publicly accessible <span style="font-size: small;">i</span>		

Feedback English © 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

4. While waiting for the replication instance to be ready, fill out the source and target endpoint information, including server hostname and port, and the username and password to use when connecting:

Setting up your first replication task

Step 1: Welcome

Step 2: Replication instance

**Step 3: Database endpoints**

Step 4: Task

Connect source and target database endpoints

Your replication instance is being created. You can start entering your endpoint connection details now. Once created, you can test your endpoint connections and move on to task creation.

Your database endpoint can be on-premise, in EC2, RDS or in the cloud. Define the connection details below. It is recommended that you test your endpoint connections here to avoid errors later.

Source database connection details

Endpoint identifier\*

Source engine\*

Server name\*

Port\*

Target database connection details

Endpoint identifier\*

Target engine\*

We recommend Aurora. Amazon Aurora is a high-performance, MySQL-compatible, enterprise-class database at a tenth the cost of commercial databases. [Learn more](#)

Server name\*

Feedback English © 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

5. Once the instance is ready, the interface will update and you can proceed:

Database Migration Service

Secure https://console.aws.amazon.com/dms/home?region=us-east-1#create-first-migration:

Guest

Services Resource Groups

Rowan U dell N. Virginia Support

## Setting up your first replication task

Step 1: Welcome

Step 2: Replication instance

Step 3: Database endpoints

Step 4: Task

### Connect source and target database endpoints

Replication instance created successfully.

Your database endpoint can be on-premise, in EC2, RDS or in the cloud. Define the connection details below. It is recommended that you test your endpoint connections here to avoid errors later.

**Source database connection details**

Endpoint identifier\* SourceEndpoint

Source engine\* mysql

Server name\* 138.197.89.66

Port\* 3306

**Target database connection details**

Endpoint identifier\* TargetEndpoint

Target engine\* mysql

We recommend Aurora. Amazon Aurora is a high-performance, MySQL-compatible, enterprise-class database at a tenth the cost of commercial databases. [Learn more](#)

Server name\* mysql-target.cieytwghp7ok.us-eas

Feedback English

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

6. In order to confirm and create the source and target endpoints, click on the Run test button for each of your databases:

Database Migration Service

Secure https://console.aws.amazon.com/dms/home?region=us-east-1#create-first-migration:

Guest

Services Resource Groups

Rowan U dell N. Virginia Support

Port\* 3306 Server name\* mysql-target.cieytwghp7ok.us-eas

SSL mode\* none Port\* 3306

User name\* dms SSL mode\* none

Password\* ..... User name\* root

Advanced Advanced

Run test Run test

Connection tested successfully Connection tested successfully

Cancel Previous Next

Feedback English

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

7. After the endpoints have been successfully tested and created, define your task. In this recipe, we will simply migrate the data (without ongoing replication):

The screenshot shows the 'Create task' page in the AWS DMS console. The left sidebar lists steps: Step 1: Welcome, Step 2: Replication instance, Step 3: Database endpoints, and Step 4: Task (which is selected). The main area is titled 'Create task' and contains the following fields:

- Task name\***: ProdEndpoint-TestEndpoint
- Task description\***: Migrate external source to RDS target
- Source endpoint**: soureendpoint
- Target endpoint**: targetendpoint
- Replication instance**: replication-instance-1
- Migration type\***: Migrate existing data
- Start task on create**:

At the bottom left, there is a 'Task Settings' dropdown. The footer includes links for Feedback, English, Copyright (2008-2017), Privacy Policy, and Terms of Use.

8. For simplicity, drop the tables in the target database (which should be empty) to ensure parity between the databases:

The screenshot shows the AWS Database Migration Service (DMS) console. The user is in the 'Create first migration' wizard, specifically on the 'Task Settings' step. The 'Target table preparation mode' is set to 'Drop tables on target'. The 'Include LOB columns in replication' option is set to 'Limited LOB mode'. The 'Max LOB size (kb)' is set to 32. The 'Enable logging' checkbox is unchecked. Below this, there is a link to 'Advanced Settings'. The 'Table mappings' step is shown below, with a large empty area for defining mappings. At the bottom right, there are 'Cancel', 'Previous', and 'Create task' buttons. The footer includes links for 'Feedback', 'English', 'Privacy Policy', and 'Terms of Use'.

9. Finally, define the mappings between the two databases. In this case, we will migrate all the tables (by using the wildcard %) in the employees database on the source:

Database Migration Service

Secure https://console.aws.amazon.com/dms/home?region=us-east-1#create-first-migration:

Guest

Services Resource Groups

Table mappings

Guided JSON

Selection rules ⓘ

At least one selection rule with an include action is required. Once you have one or more selection rules, you can add transformation rules.

Where ⓘ

Schema name is employees

Table name is like %

Use % as a wildcard.

Action ⓘ Include

Filter ⓘ

Add column filter

Add selection rule

Feedback English

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

10. Once you click Add selection rule you will see your rule in the selection rules list:

MAX LCU SIZE (LCU)

Enable logging

Advanced Settings

Table mappings

Guided JSON

Selection rules ⓘ

where schema name is like 'employees' and table name is like '%', include

+ add selection rule

Transformation rules ⓘ

+ add transformation rule

Feedback English

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

11. Once the task is defined you have finished the wizard. You will then see the task being created:

The screenshot shows the AWS DMS console with the URL <https://console.aws.amazon.com/dms/home?region=us-east-1#tasks>. The left sidebar has a 'Tasks' section selected. The main area shows a table with one row of data:

ID	Status	Source	Target	Type	Complete %	Elapsed time	Tables
prodendpoint-testendpoint	Creating	soureendpoint	targetendpoint	Full Load	0	0	0

At the bottom, there are links for Feedback, English, Privacy Policy, and Terms of Use.

12. Once the status of the task is Ready you can select it and click on the Start/Resume button:

The screenshot shows the AWS DMS console. On the left, a sidebar menu is open under the 'Tasks' section, listing 'Dashboard', 'Endpoints', 'Certificates', 'Replication instances', and 'Subnet groups'. The main area displays a table of tasks. A single row is selected, showing the task ID 'prodendpoint-testendpoint' with a status of 'Starting'. Below the table, a detailed view of the task 'prodendpoint-testendpoint' is shown. The 'Overview' tab is selected, displaying various configuration details such as Task name, Task ARN, Status, Migration type, Replication instance, Source endpoint, Target endpoint, Mapping method, and Task settings. At the bottom of the page, there are links for 'Feedback', 'English', 'Privacy Policy', and 'Terms of Use'.

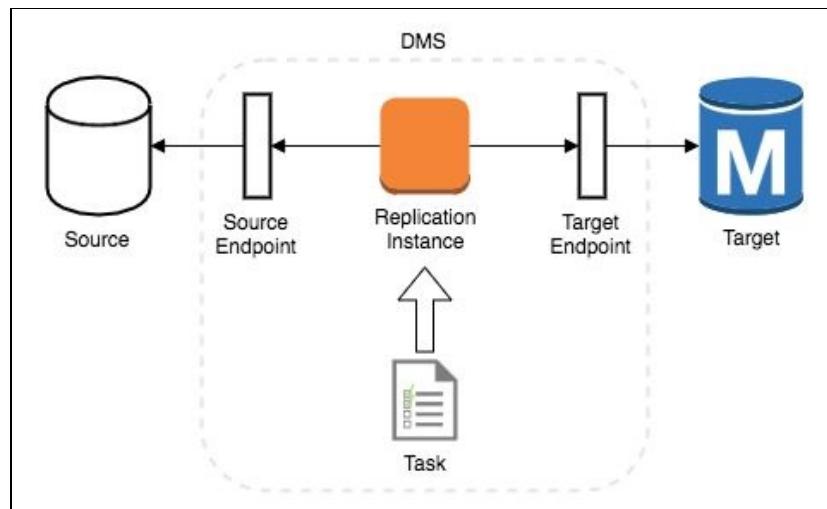
13. When complete, you will see the task's details updated in the console:

The screenshot shows the AWS DMS console after the task has completed. The task table now shows the task 'prodendpoint-testendpoint' with a status of 'Load complete'. The 'Overview' tab of the task details page is still selected, showing the same configuration details as before, but the status has changed to 'stopped'. The progress bar for 'Complete %' is now at 100%, and the 'Elapsed time' is 0m. The 'Tables' column shows a value of 6, indicating the number of tables processed.



# How it works...

At a high level, this is what the DMS architecture looks like:



Both the **Source** and **Target** databases are external to **DMS**. They are represented internally by endpoint resources that are references to the databases. Endpoints can be reused between different tasks if needed.

This recipe starts by defining the replication instance details. Keep in mind that the DMS migration process works best when the migration/transform between the two databases is kept *in memory*. This means that for larger jobs you should allocate a more powerful instance. If the process needs to temporarily write data to disk (such as swap) then the performance and throughput will be much lower. This can have flow-on effects, particularly for tasks that include ongoing replication.

Next, the two endpoints are defined. It is very important to verify your endpoint configuration by using the built-in testing feature so that your tasks do not fail later in the process. Generally, if the connectivity test fails, it is one of two main issues:

- Network connectivity issues between the replication instance and the database. This is particularly an issue for on-premise databases, which are usually specifically restricted from being accessed externally.
- User permissions issues: For example, in the case of MySQL, the root user cannot be used to connect to the database externally, so this default user cannot be used.

Defining the task involves defining your migration type. The recipe uses the simplest type; migrate tables. This means that the data will be copied between the two databases, and will be complete when the data is propagated. We also get to define the behavior on the target database. For simplicity, we have configured the task to drop the tables in the target database ensuring that the two databases look as similar as possible, even if the tables are renamed, or the table mappings change. For the task table mappings we use the wildcard symbol % to match all tables in the source database. Obviously, you could be more selective if you only wanted to match a subset of your data.

Once the replication instance, endpoints, and task are defined the wizard ends and you are returned to the DMS console. After the task is finished creating it can be started.

As it is a *migrate existing data-type* task, it will complete once all the data has been propagated to the target database.



# **There's more...**

This is obviously a simple example of what DMS can do. There are other features and performance aspects that you should consider in more advanced scenarios.



# Database engines

While this example uses two MySQL databases, it is possible to migrate from one database engine to a complete database engine, for example, Oracle to MySQL. Unfortunately, this can be a complex process, and while this functionality is very useful it is beyond the scope of this recipe. Due to the differences in the various engines, there are some limitations on what you can migrate and transform.



*See the AWS Schema Conversion Tool documentation for more details on what can be migrated between different database engines.*



# Ongoing replication

There are also some limits around the ongoing propagation of data—only table data can be migrated. Things such as indexes, users, and permissions cannot be replicated continually.



# Multi-AZ

For ongoing replication tasks, you may want to create a multi-AZ replication instance so that the impact of any interruptions of services are minimized. Obviously you will need to have a similarly configured (such as multi-AZ) RDS instance as your target to get the full benefit!



*For best performance, when setting up your replication instance you should make sure it is in the same AZ as your target RDS instance.*



# Calculating DyanmoDB performance

**DynamoDB (DDB)** is the managed NoSQL database service from AWS.

As DDB pricing is based on the amount of read and write capacity units provisioned, it is important to be able to calculate the requirements for your use case.

This recipe uses a written formula to estimate the required **read capacity units (RCU)** and **write capacity units (WCUs)** that should be allocated to your DDB table.

It is also crucial to remember that while new partitions will be automatically added to a DDB table, they cannot be automatically taken away. This means that excessive partitioning can cause long-term impacts to your performance, so you should be aware of them.



# Getting ready

All of these calculations assume that you have chosen a good partition key for your data. A good partition key ensures the following:

- Data is evenly spread across all the available partitions
- Read and write activity is spread evenly in time

Unfortunately, choosing a good partition key is very data-specific, and beyond the scope of this recipe.

All reads are assumed to be strongly consistent.



# How to do it...

1. Start with the size of the items, in **kilobytes (KB)**:

*ItemSize = Size of the items (rows) in KB*

2. Work out the required number of RCUs required by dividing the number by 4, and rounding up:

*RCU Per Item = ItemSize / 4 (rounded up)*

3. Define the expected number of read operations per second. This is one of the numbers you will use to provision your table with:

*Required RCU = Expected Number of Reads \* RCU Per Item*

4. Divide the number by 3,000 to calculate the number of DDB partitions required to reach the capacity:

*Read Partitions = Required RCU / 3,000*

5. Next, work out the write capacity required by dividing the item size by 1, and rounding up:

*WCU Per Item = ItemSize / 1 (rounded up)*

6. Define the expected number of write operations per second. This is one of the numbers you will use to provision your table with:

*Required WCU = Expected Number of Writes \* WCU Per Item*

7. Divide the number by 1,000 to calculate the number of DDB partitions required to reach the capacity:

*Write Partitions = Required WCU / 1,000*

8. Add these two values to get the capacity partitions required (rounding up to a whole number):

*Capacity Partitions = Read Partitions + Write Partitions (rounded up)*

9. Work out the minimum number of partitions required by the amount of data you plan to store:

*Size Partitions = Total Size in GB / 10 (rounded up)*

10. Once you have the partition requirements for your use case, take the maximum of your previous calculations:

*Required Partitions = Maximum value between Capacity Partitions and Size Partitions*

11. Since your allocated capacity is spread evenly across partitions, divide the RCU and WCU values to get the per-partition performance of your table:

*Partition Read Throughput = Required RCU / Required Partitions*

*Partition Write Throughput = Required WCU / Required Partitions*



# How it works...

Behind the scenes, DDB throughput is controlled by the number of partitions that are allocated to your table. It is important to consider how your data will be spread across these partitions to ensure you get the performance you expect *and have paid for*.

We start this recipe by calculating the size of the items in your database, for throughput purposes. DDB has a minimum size it will consider, and even if an operation uses less than this size, it is rounded up in terms of allocated throughput used. The minimum size depends on the type of operation:

- Read operations are calculated in 4-K blocks
- Write operations are calculated in 1-K blocks

We then work out what the required RCU and WCU is, based on the expected number of operations. These values are what can then be used to provision the DDB table, as they represent the minimum required throughput (in optimal conditions).

Once you have these values, you can use them to provision your table.

Next, we calculate the throughput per partition key. These calculations rely on knowing what the performance of each partition is expected to be. The numbers 3,000 (for RCUs) and 1,000 (for WCUs) represent the capacity of a single DDB partition. By expressing the capacity in terms of partition performance (reads and writes) and adding them together we get the minimum number of partitions required from a capacity point of view.

We then do the same calculation for total data size. Each DDB partition can handle up to 10 GB of data. Any more than that will need to be split between multiple partitions.



*The specific values for partition capacity (for reads, writes, and size) have been stable for a while, but may change in the future. Double-check that the current values are the same as used here for complete accuracy.*

Once we have the minimum partitions for both capacity and size, we take the highest value and work with that. This ensures we meet both the capacity and size requirements.

Finally, we take the provisioned capacity and divide it by the number of partitions. This gives us the throughput performance for each partition key, which we can then use to confirm against our use case.



# **There's more...**

There are many nuances to using DDB efficiently and effectively. Here are some of the more important/impactful things to note.



# Burst capacity

There is a burst capacity available to tables that go over their allocated capacity. Unused read and write capacity can be retained for up to five minutes (such as 300 seconds, for calculation purposes). Relying on this capacity is not good practice, and it will undoubtedly cause issues at some stage in the future.



# Metrics

DDB tables automatically send data to CloudWatch metrics. This is the quickest and easiest way to confirm that your calculations and provision capacity are meeting your needs. It also helps you keep an eye on your usage to track your throughput needs over time. All metrics appear in the *AWS/DynamoDB* namespace. Some of the most interesting metrics for throughput calculations are as follows:

- ConsumedReadCapacityUnits
- ConsumedWriteCapacityUnits
- ReadThrottleEvents
- WriteThrottleEvents

There are other metrics available; see the *Amazon DynamoDB Metrics and Dimensions* documentation for more details.



# Eventually consistent reads

Using eventually consistent reads (as opposed to strongly consistent reads) *halves* the RCU requirements for calculation purposes. In this recipe, we have used strongly consistent reads because it works with all workloads, but you should confirm that your use case actually requires it. Use eventually consistent reads if it does not.



*By reducing the required provisioned capacity for reads, you effectively reduce your cost for using DDB.*



# Networking

In this chapter, we will cover:

- Building a secure network
- Creating a NAT gateway
- Canary deployment via DNS
- Hosting a domain
- Routing based on location with failover
- Network logging and troubleshooting



# Introduction

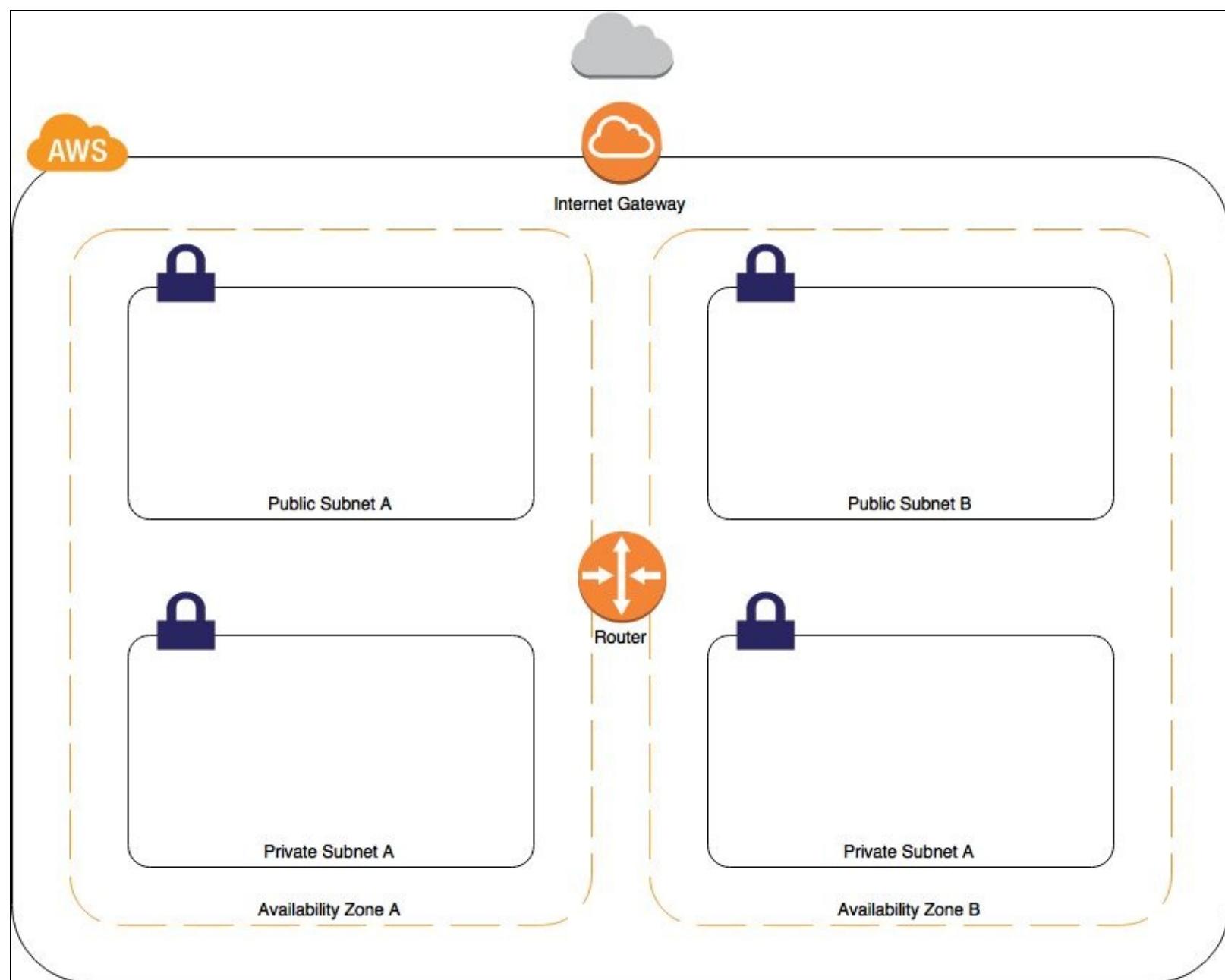
Networking is a foundational component of using other AWS services such as EC2, RDS, and others. Using constructs such as VPCs and NAT gateways gives you the capability and confidence to secure your resources at a networking level. At a DNS level, Route 53 provides connectivity to your users in a responsive and fault-tolerant way that ensures the best performance in a variety of scenarios.



# Building a secure network

In this recipe, we're going to build a secure network (VPC) in AWS. This network will consist of two public and private subnets split across two Availability Zones. It will also allow inbound connections to the public subnets for the following:

- SSH (port 22)
- HTTP (port 80)
- HTTPS (port 443)







# Getting ready

Before we proceed, you're going to need to know the names of at least two Availability Zones in the region we're deploying to. The recipes in this book will typically deploy to `us-east-`, so to get things moving you can just use the following:

- `us-east-1a`
- `us-east-1b`



*When you create an AWS account, your zones are randomly allocated. This means that `us-east-1a` in your account isn't necessarily the same data center as `us-east-1a` in my account.*

Parameters:

AvailabilityZone1:

Description: Availability zone 1 name (e.g. us-east-1a) Type:  
AWS::EC2::AvailabilityZone::Name AvailabilityZone2:

Description: Availability zone 2 name (e.g. us-east-1b) Type:  
AWS::EC2::AvailabilityZone::Name

Resources:

# VPC & subnets

ExampleVPC:

Type: AWS::EC2::VPC

Properties:

CidrBlock: !Ref VPCCIDR

EnableDnsSupport: true EnableDnsHostnames: true Tags:

- { Key: Name, Value: Example VPC }

PublicSubnetA:

Type: AWS::EC2::Subnet Properties:

AvailabilityZone: !Ref AvailabilityZone1

CidrBlock: !Ref PublicSubnetACIDR

MapPublicIpOnLaunch: true VpcId: !Ref ExampleVPC

Tags:

- { Key: Name, Value: Public Subnet A }

PublicSubnetB:

Type: AWS::EC2::Subnet Properties:

AvailabilityZone: !Ref AvailabilityZone2

CidrBlock: !Ref PublicSubnetBCIDR

MapPublicIpOnLaunch: true VpcId: !Ref ExampleVPC

Tags:

- { Key: Name, Value: Public Subnet B }

PrivateSubnetA:

Type: AWS::EC2::Subnet Properties:

AvailabilityZone: !Ref AvailabilityZone1

CidrBlock: !Ref PrivateSubnetACIDR

VpcId: !Ref ExampleVPC

Tags:

- { Key: Name, Value: Private Subnet A }

PrivateSubnetB:

Type: AWS::EC2::Subnet Properties:

AvailabilityZone: !Ref AvailabilityZone2

CidrBlock: !Ref PrivateSubnetBCIDR

VpcId: !Ref ExampleVPC

Tags:

- { Key: Name, Value: Private Subnet B }

VPCCIDR:

Description: CIDR block for VPC

Type: String

Default: "172.31.0.0/21" # 2048 IP addresses PublicSubnetACIDR:

Description: CIDR block for public subnet A Type: String

Default: "172.31.0.0/23" # 512 IP address PublicSubnetBCIDR:

Description: CIDR block for public subnet B

Type: String

Default: "172.31.2.0/23" # 512 IP address PrivateSubnetACIDR:

Description: CIDR block for private subnet A Type: String

Default: "172.31.4.0/23" # 512 IP address PrivateSubnetBCIDR:

Description: CIDR block for private subnet B

Type: String

Default: "172.31.6.0/23" # 512 IP address

# Internet Gateway

ExampleIGW:

Type: AWS::EC2::InternetGateway Properties:

Tags:

- { Key: Name, Value: Example Internet Gateway }

IGWAttachment:

Type: AWS::EC2::VPCGatewayAttachment DependsOn: ExampleIGW

Properties:

VpcId: !Ref ExampleVPC

InternetGatewayId: !Ref ExampleIGW

```
# Public Route Table

# Add a route for Internet bound traffic pointing to our IGW

# A route for VPC bound traffic will automatically be added PublicRouteTable:

Type: AWS::EC2::RouteTable Properties:

VpcId: !Ref ExampleVPC

Tags:
- { Key: Name, Value: Public Route Table }

PublicInternetRoute:
Type: AWS::EC2::Route

DependsOn: IGWAttachment Properties:

RouteTableId: !Ref PublicRouteTable GatewayId: !Ref ExampleIGW

DestinationCidrBlock: "0.0.0.0/0"

RouteAssociationPublicA: Type: AWS::EC2::SubnetRouteTableAssociation Properties:
  RouteTableId: !Ref PublicRouteTable SubnetId: !Ref PublicSubnetA
RouteAssociationPublicB: Type: AWS::EC2::SubnetRouteTableAssociation Properties:
  RouteTableId: !Ref PublicRouteTable SubnetId: !Ref PublicSubnetB

# Private Route Table

# We don't add any entries to this route table because there is no NAT gateway

# However a route for VPC bound traffic will automatically be added

PrivateRouteTable:
Type: AWS::EC2::RouteTable Properties:

VpcId: !Ref ExampleVPC

Tags:
```

- { Key: Name, Value: Private Route Table }

PrivateSubnetAssociationA: Type: AWS::EC2::SubnetRouteTableAssociation

Properties:

RouteTableId: !Ref PrivateRouteTable SubnetId: !Ref PrivateSubnetA

PrivateSubnetAssociationB: Type: AWS::EC2::SubnetRouteTableAssociation Properties:

RouteTableId: !Ref PrivateRouteTable SubnetId: !Ref PrivateSubnetB

# Public NACL

PublicNACL:

Type: AWS::EC2::NetworkAcl Properties:

VpcId: !Ref ExampleVPC

Tags:

- { Key: Name, Value: Example Public NACL }

# Allow outbound to everywhere NACLRulePublicEgressAllowAll: Type:

AWS::EC2::NetworkAclEntry Properties:

CidrBlock: "0.0.0.0/0"

Egress: true

Protocol: 6

PortRange: { From: 1, To: 65535 }

RuleAction: allow

RuleNumber: 100

NetworkAclId: !Ref PublicNACL

# Allow outbound to VPC on all protocols NACLRulePublicEgressAllowAllToVPC:

Type: AWS::EC2::NetworkAclEntry Properties:

CidrBlock: !Ref VPCCIDR

Egress: true

Protocol: -1

RuleAction: allow

RuleNumber: 200

NetworkAclId: !Ref PublicNACL

# Allow inbound from everywhere to ephemeral ports (above 1024)

NACLRulePublicIngressAllowEphemeral: Type: AWS::EC2::NetworkAclEntry

Properties:

CidrBlock: "0.0.0.0/0"

Protocol: 6

PortRange: { From: 1024, To: 65535 }

RuleAction: allow

RuleNumber: 100

NetworkAclId: !Ref PublicNACL

# Allow inbound from everywhere on port 22 for SSH

NACLRulePublicIngressAllowSSH: Type: AWS::EC2::NetworkAclEntry Properties:

CidrBlock: "0.0.0.0/0"

Protocol: 6

PortRange: { From: 22, To: 22 }

RuleAction: allow

RuleNumber: 200

NetworkAclId: !Ref PublicNACL

# Allow inbound from everywhere on port 443 for HTTPS

NACLRulePublicIngressAllowHTTPS: Type: AWS::EC2::NetworkAclEntry Properties:

CidrBlock: "0.0.0.0/0"

Protocol: 6

PortRange: { From: 443, To: 443 }

RuleAction: allow

RuleNumber: 300

NetworkAclId: !Ref PublicNACL

# Allow inbound from everywhere on port 80 for HTTP

NACLRulePublicIngressAllowHTTP: Type: AWS::EC2::NetworkAclEntry Properties:

CidrBlock: "0.0.0.0/0"

Protocol: 6

PortRange: { From: 80, To: 80 }

RuleAction: allow

RuleNumber: 400

NetworkAclId: !Ref PublicNACL

# Allow inbound from VPC on all protocols NACLRulePublicIngressAllowFromVPC:  
Type: AWS::EC2::NetworkAclEntry Properties:

CidrBlock: !Ref VPCCIDR

Protocol: -1

RuleAction: allow

RuleNumber: 500

NetworkAclId: !Ref PublicNACL

NACLAssociationPublicSubnetA: Type: AWS::EC2::SubnetNetworkAclAssociation  
Properties:

NetworkAclId: !Ref PublicNACL

SubnetId: !Ref PublicSubnetA NACLAssociationPublicSubnetB: Type:  
AWS::EC2::SubnetNetworkAclAssociation Properties:

NetworkAclId: !Ref PublicNACL

SubnetId: !Ref PublicSubnetB

# Private NACL

PrivateNACL:

Type: AWS::EC2::NetworkAcl Properties:

VpcId: !Ref ExampleVPC

Tags:

- { Key: Name, Value: Example Private NACL }

# Allow all protocols from VPC range NACLRulePrivateIngressAllowVPC: Type:  
AWS::EC2::NetworkAclEntry Properties:

CidrBlock: !Ref VPCCIDR

Protocol: -1

RuleAction: allow

RuleNumber: 100

NetworkAclId: !Ref PrivateNACL

# Allow TCP responses from everywhere NACLRulePrivateIngressAllowEphemeral:  
Type: AWS::EC2::NetworkAclEntry Properties:

CidrBlock: "0.0.0.0/0"

Protocol: 6

PortRange: { From: 1024, To: 65535 }

RuleAction: allow

RuleNumber: 200

NetworkAclId: !Ref PrivateNACL

# Allow outbound traffic to everywhere, all protocols

NACLRulePrivateEgressAllowVPC: Type: AWS::EC2::NetworkAclEntry Properties:

CidrBlock: "0.0.0.0/0"

Egress: true

Protocol: -1

RuleAction: allow

RuleNumber: 100

NetworkAclId: !Ref PrivateNACL

NACLAffiliationPrivateSubnetA: Type: AWS::EC2::SubnetNetworkAclAssociation

Properties:

NetworkAclId: !Ref PrivateNACL

SubnetId: !Ref PrivateSubnetA NACLAffiliationPrivateSubnetB: Type:  
AWS::EC2::SubnetNetworkAclAssociation Properties:

NetworkAclId: !Ref PrivateNACL

SubnetId: !Ref PrivateSubnetB

Outputs:

ExampleVPC:

Value: !Ref ExampleVPC

PublicSubnetA:

Value: !Ref PublicSubnetA PublicSubnetB:

Value: !Ref PublicSubnetB

PrivateRouteTable:

Value: !Ref PrivateRouteTable PublicRouteTable:

Value: !Ref PublicRouteTable PrivateSubnetA:

Value: !Ref PrivateSubnetA PrivateSubnetB:

Value: !Ref PrivateSubnetB

<strong> aws cloudformation create-stack </strong> <strong> --stack-name secure-vpc </strong> <strong> --template-body file://07-building-a-secure-network.yaml </strong> <strong> --parameters </strong>

<strong> ParameterKey=AvailabilityZone1,ParameterValue=<az-1> </strong> <strong> ParameterKey=AvailabilityZone2,ParameterValue=<az-2></strong>



# How it works...

When you run this template, AWS will go ahead and create an isolated, secure network just for you. While it contains a number of resources and concepts which will be familiar to network administrators, it's essentially an empty shell, which you can now go ahead and populate.

For example, each VPC contains a virtual router. You can't see it and you can't log into it to perform any special configuration, but you can customize its behavior by modifying the route tables in this template.

The NACLs we've deployed are not stateful and should *not* be considered a substitution for security groups. NACLs are *complementary* to security groups, which are stateful and frankly much easier to change and manage than NACLs. While the NACLs in our recipe allow everywhere (`0.0.0.0/0`) to make inbound connections to port `22`, for example, you'll want to use security groups to lock this down to a specific IP range (your corporate data center, for example).



# There's more...

Actually, there's a *lot* more. Despite the amount of code in this recipe, we've really only covered the basics of what's possible with VPCs and networking in AWS. Here are some of the main VPC topics you'll encounter as you progress with your VPC usage:

- **Direct Connect:** This is a method of connecting your DC to your VPC using a private, dedicated pipe. Doing this often provides better network performance, and may also be cheaper than a VPN connection over the Internet.
- **Virtual Private Gateway (VPN):** You can configure your VPC to connect to your corporate DC over the Internet via VPN. This requires that you run supported VPN hardware in your DC.
- IPv6 support was added recently. We've left it out to keep things simple.
- **VPC endpoints:** This feature exposes AWS endpoints inside your VPC so that you don't have to route traffic over public Internet to consume them. Only S3 is supported at the time of writing.
- **VPC peering:** You can peer a VPC to one or more VPCs so that (unencrypted) traffic can flow between them. The IP ranges must not clash and, while the peering is free, you will still need to pay for traffic between VPCs. Transitive peering isn't supported, so if you need traffic to traverse VPCs you'll require a VPN/routing appliance of some kind. Cross-account VPC peering is supported (we use this feature quite often), but cross-region peering isn't yet available.
- **VPC sizing:**
  - IPv4: You can deploy networks between sizes /28 and /16.
  - IPv6: Your VPCs will be fixed in size at /56.
  - Once your VPC has been deployed you can't change its size. If you run out of IP space, your only option is to deploy a larger VPC and migrate everything (ouch!), or you can perhaps mitigate your problem with VPC peering.
- **VPC flow-logs:** You will want to enable VPC flow-logs in order to monitor traffic and do any kind of network debugging.
- Multicast traffic isn't supported.
- Subnets must reside in a single availability zone; they can't span Availability Zones.
- **Elastic Load Balancers (ELBs)** can scale out to use a lot of private IP addresses if you are sending a large amount of traffic through them. Keep this in mind when you're sizing your subnets.
- The number of VPCs you can deploy is limited to five per region, per account. You can request to increase this limit if necessary. Internet gateways have the same limit, and increasing one limit increases the other.
- The *default* VPC:
  - First and foremost, the default VPC is created automatically for you when you create your account. It has some different properties and behaviors to the VPCs you create for yourself.
  - If you try to launch an EC2 instance without specifying a subnet ID, AWS will attempt to launch it in your default VPC.
  - It consists of only public subnets. These subnets are configured to provide a public IP address

to all instances by default.

- It's possible to delete the default VPC in a region. If you do this by mistake, or have simply decided that you'd like to undo this action, you'll need to log a support ticket with AWS to have them create a new one for you.



## See also...

- The *Creating a NAT gateway* recipe



# Creating a NAT gateway

Unless required, your instances should not be publicly exposed to the Internet. When your instances are on the Internet, you have to assume they will be attacked at some stage.

This means most of your workloads should run on instances in private subnets. Private subnets are those that are not connected directly to the Internet.

In order to give your private instances access to the Internet, you use **network address translation (NAT)**. A NAT gateway allows your instances to initiate a connection *to* the Internet, without allowing connections *from* the Internet.



# Getting ready

For this recipe, you must have the following existing resources:

- A VPC with an **Internet gateway (IGW)**
- A public subnet
- A private subnet route table

You will need the IDs for the public subnet and private subnet route table. Both of these resources should be in the same AZ.

AWSTemplateFormatVersion: "2010-09-09"

Description: Create NAT Gateway and associated route.

Parameters:

PublicSubnetId: Description: Public Subnet ID to add the NAT Gateway to Type: AWS::EC2::Subnet::Id RouteTableId:

Description: The private subnet route table to add the NAT

Gateway route to Type: String

Resources:

EIP:

Type: AWS::EC2::EIP

Properties:

Domain: vpc

NatGateway:

Type: AWS::EC2::NatGateway Properties:

AllocationId: !GetAtt EIP.AllocationId SubnetId: !Ref PublicSubnetId

Route:

Type: AWS::EC2::Route Properties:

RouteTableId: !Ref RouteTableId DestinationCidrBlock: 0.0.0.0/0

NatGatewayId: !Ref NatGateway

```
<strong> aws cloudformation create-stack </strong> <strong> --stack-name nat-gateway </strong> <strong> --template-body file://07-nat-gateway.yaml </strong> <strong> --parameters </strong> <strong> ParameterKey=RouteTableId,ParameterValue=<route-table-id> </strong> <strong> ParameterKey=PublicSubnetId,ParameterValue=<public-subnet-id></strong>
```



# How it works...

The parameters required for this recipe are as follows:

- A public subnet ID
- A private subnet route table ID

The public subnet ID is needed to host the NAT gateway, which must have Internet access. The private subnet route table will be updated with a route to the NAT gateway.

Using the AWS NAT gateway service means that AWS takes care of hosting and securing the service for you. The service will be hosted redundantly in a single AZ.



*You can use this recipe multiple times to deploy NAT gateways in each of your private subnets. Just make sure the public subnet and the private subnet are in the same AZ.*

In the unlikely (but possible) event of an AZ outage, you should deploy a NAT gateway per subnet. This means that if one NAT gateway goes offline, instances in the other AZ can continue to access the Internet as normal. You are deploying your application in multiple subnets, aren't you?

This recipe will only work if you have created your own private subnets, as the default subnets in a new AWS account are all *public*. Instances in a public subnet have direct access to the Internet (via an IGW), so they do not need a NAT gateway.



## See also

- The *Building a secure network* recipe



# Canary deployment via DNS

Canary deployment is a popular deployment method in the cloud. It allows you to deploy new versions of your resources alongside your old resources, gradually and selectively directing parts of your traffic to the new resource.

By directing a small portion of your traffic to your new resources, you can get valuable real-world data and metrics. This means you don't need to engage in a *big bang* deployment—where you switch over all of your traffic at once.

It also gives you more flexibility in terms of troubleshooting and monitoring; if you see errors for your new resources, you can redirect the traffic back to your old resources while you investigate.

In this recipe, we will create the resources necessary to do a DNS-based canary deployment, and cut traffic from one resource to another (that is, old to new).



# Getting ready

This recipe requires a few things to be in place:

- A Route 53 hosted zone for your domain suffix
- Existing DNS records for your *old* and *new* resources/endpoints

AWSTemplateFormatVersion: "2010-09-09"

Description: Create a weighted DNS setup for canary deployments.

Parameters:

HostedZoneName:

Type: String

Description: The hosted zone to create records in

DomainName:

Type: String

Description: The domain name to create in the hosted zone

OldResource:

Type: String

Description: The older resource domain name

NewResource:

Type: String

Description: The newer resource domain name

OldWeight:

Type: Number

Default: 1

Description: The ratio of requests to send to the older endpoint

NewWeight:

Type: Number

Default: 0

Description: The ratio of requests to send to the newer endpoint

Resources:

RecordSetGroup:

Type: AWS::Route53::RecordSetGroup Properties:

HostedZoneName: !Ref HostedZoneName Comment: Canary deployment record set group RecordSets:

- Name: !Join [ ".", [ Ref: DomainName, Ref: HostedZoneName ] ]

Type: CNAME

TTL: "300"

SetIdentifier: Old

Weight: !Ref OldWeight ResourceRecords:

- !Ref OldResource

- Name: !Join [ ".", [ Ref: DomainName, Ref: HostedZoneName ] ]

Type: CNAME

TTL: "300"

SetIdentifier: New

Weight: !Ref NewWeight ResourceRecords:

- !Ref NewResource

```
<strong> aws cloudformation create-stack </strong> <strong> --stack-name canary </strong>
<strong> --template-body file://07-canary-deployments.yaml </strong>
<strong> --parameters </strong> <strong>
ParameterKey=DomainName,ParameterValue=<your-domain-name> </strong> <strong>
ParameterKey=OldResource,ParameterValue=<old-resource-dns> </strong> <strong>
ParameterKey>NewResource,ParameterValue=<new-resource-dns> </strong> <strong>
```

ParameterKey=HostedZoneName,ParameterValue=<your-hosted-zone> </strong>

<strong> aws cloudformation update-stack </strong> <strong> --stack-name canary </strong> <strong> --parameters </strong> <strong> ParameterKey=HostedZoneName,UsePreviousValue=true </strong> <strong> ParameterKey=DomainName,UsePreviousValue=true </strong> <strong> ParameterKey=OldResource,UsePreviousValue=true </strong> <strong> ParameterKey>NewResource,UsePreviousValue=true </strong> <strong> ParameterKey=OldWeight,ParameterValue=0 </strong> <strong> ParameterKey>NewWeight,ParameterValue=1 </strong> <strong> --use-previous-template</strong>



# How it works...

This template focuses on utilizing the features of a Route 53 record set group, and the most useful properties have been parameterized.

The value for your `DomainName` parameter will be created as multiple `CNAME` records in your hosted zone (as set in `HostedZoneName`), one for each of your resources, old and new.

The `oldResource` and `NewResource` parameters represent the target domain names that the incoming requests will be shared between.

Once the stack is deployed, you will be able to go to your domain name and see your *old* resource. By default, this template will send all traffic to the old resource endpoint.

Once you've verified the setup is working correctly, you can start to deploy by updating the stack to send *some* of your requests to the new resource.

Changing the resource record set's weightings via the CLI is quite involved, as it requires passing a complex JSON object as an argument. It is much simpler and safer to simply update the existing CloudFormation stack you deployed, changing just the weighting parameters that are already present.

With the `update-stack` command, the new weightings will be propagated to your record set group members (without interruption) and the new distribution of traffic will start taking effect.

For the parameters without default values, you must explicitly tell CloudFormation to use the previous values, as well as the template body supplied previously.

Remember that the distribution will be determined by the target's weight divided by the total weight value of all targets. This means you can easily *turn off* a target by setting its weight to `0`, regardless of the other weight values. In this recipe, we have used `0` and `1` as simple values to illustrate the impact, but you can (and should) use more fine-grained parameters.



# Hosting a domain

In this recipe, we'll show you how to host a domain in Route 53 and add some records to it:

The screenshot shows the AWS Route 53 Management Console interface. On the left, there's a sidebar with links like Dashboard, Hosted zones (which is selected and highlighted in orange), Health checks, Traffic flow, Traffic policies, Policy records, Domains, Registered domains, and Pending requests. The main content area has tabs at the top: Create Hosted Zone, Go to Record Sets, and Delete Hosted Zone. A search bar shows the ID 'Z2VQFEU5NWJPO0'. Below the search bar, a table displays one hosted zone entry: 'gitrepository.com.' (Domain Name), 'Public' (Type), '5' (Record Set Count), and 'Z2VQFEU5NWJPO0' (Hosted Zone ID). To the right of the table, under 'Hosted Zone Details', it shows the domain name 'gitrepository.com.', type 'Public Hosted Zone', and Hosted Zone ID 'Z2VQFEU5NWJPO0'. It also lists the record set count as 5 and provides a comment field. Below this, it lists the 'Name Servers' which include ns-1355.awsdns-41.org, ns-732.awsdns-27.net, ns-1914.awsdns-47.co.uk, and ns-412.awsdns-51.com. A note below the servers states: 'Before the Domain Name System will start to route queries for this domain to Route 53 name servers, you must update the name server records either with the current DNS service or with the registrar for the domain, as applicable. For more information, click the ? icon above.' At the bottom right of the details panel, there's a 'Tags' section with a link to 'View and manage tags for your hosted zones using Tag Editor'. The footer of the page includes links for Feedback, English, and various AWS terms and policies.

Hosting a domain



# Getting ready

You technically don't need to have registered a domain name in order to proceed with this recipe, but it sure helps if you have a real domain that you can use.

Parameters:

  DomainName: Description: Your domain name (example.org) Type: String

Resources:

  DNSHostedZone: Type: AWS::Route53::HostedZone Properties: Name: !Ref DomainName

<strong> aws cloudformation create-stack </strong> <strong> --stack-name example-hosted-zone </strong> <strong> --template-body file://07-hosting-a-domain.yaml </strong> <strong> --parameters </strong> <strong> ParameterKey=DomainName,ParameterValue=<your-domain-name></strong>



# How it works...

This will create a hosted zone in Route 53. Once the stack has finished creating, go and find it in the web console. You'll see that there are a number of name servers associated with it. These are the name servers to use if you wish to proceed with delegating your domain name to AWS's Route 53 servers using your domain name registrar's control panel.

## DNSRecords:

Type: AWS::Route53::RecordSetGroup Properties: HostedZoneId: Ref: DNSHostedZone RecordSets: - Name: !Ref DomainName Type: A

TTL: 60

ResourceRecords: - "127.0.0.1"

- Name: !Ref DomainName Type: MX

TTL: 60

ResourceRecords: - "10 smtp.example.org"

- "20 smtp.example.org"

- Name: !Ref DomainName Type: TXT

TTL: 60

ResourceRecords: - "'v=spf1 include:spf.example.org ?all'"

## Some items of note:

For the priority in MX records, add the number at the start of the record followed by a space.

For TXT records such as SPF entries, which are typically required to be quoted, you can surround double quotes with single quotes.

Here's how they look in the Route 53 web console:

Secure <https://console.aws.amazon.com/route53/home?region=us-east-1#resource-record-sets:Z2VQFEU5NWJPOO>

Services ▾ Resource Groups ▾

Back to Hosted Zones Create Record Set Import Zone File Delete Record Set Test Record Set

Record Set Name: gitrepository.com. Any Type: Aliases Only Weighted

Only

Displaying 1 to 5 out of 5 Record Sets

Name	Type	Value
gitrepository.com.	A	127.0.0.1
gitrepository.com.	MX	20 smtp.example.org 10 smtp.example.org
gitrepository.com.	NS	ns-1355.awsdns-41.org. ns-732.awsdns-27.net. ns-1914.awsdns-47.co.uk. ns-412.awsdns-51.com.
gitrepository.com.	SOA	ns-1355.awsdns-41.org. awsdns-hostmaster.amazon.com.
gitrepository.com.	TXT	"v=spf1 include:spf.example.org ?all"

Edit Record Set

Name: gitrepository.com.

Type: A – IPv4 address

Alias:  Yes  No

TTL (Seconds): 60 1m 5m 1h 1d

Value: 127.0.0.1

IPv4 address. Enter multiple addresses on separate lines.  
Example:  
192.0.2.235  
198.51.100.234

Routing Policy: Simple

Route 53 responds to queries based only on the values in this record. [Learn More](#)

Save Record Set

# Hosting a domain



## See also...

- The *Hosting a static website* recipe in [Chapter 3, Storage and Content Delivery](#)



# Routing based on location with failover

In this recipe, we're going to show you two Route 53 routing policies:

- Geolocation routing
- Failover routing

In fact, we're actually going to combine these two policies together. A perusal of the AWS documentation might lead you to believe that this isn't particularly common practice, but understand that by combining routing policies, you can do great things for your performance and availability.



# Getting ready

Given that we're demonstrating a failover task, you'll want to set up two ELBs before we proceed. We're going to assume you're doing this in different regions, but this isn't strictly necessary. These ELBs will need to accept HTTP connections (on port 80 of course) and have at least one instance attached to them (which is passing its health check and serving content).



*The Creating security groups recipe in Chapter 4, Using AWS Compute deployed in two different regions, should fit the bill nicely.*

You'll also need a domain name that you'd like to create as a new hosted zone in Route 53. You technically don't need to delegate this domain to Route 53 from your registrar, so you can complete this recipe with any domain you choose. Just remember that using a real domain you can delegate to Route 53, which will save you messing with your localhost's file or DNS setup in order to test this recipe.

In summary, you'll need the following:

- The DNS names for both ELBs
- The hosted zone IDs for both ELBs
- A domain name of your choosing

Parameters:

  DomainName:

    Description: Your domain name (example.org) Type: String

    LoadBalancerDNSNameRegionA: Description: The DNS name of your ELB in region A Type: String

    LoadBalancerHostedZoneRegionA: Description: The Hosted Zone ID of your ELB in region A Type: String

    LoadBalancerDNSNameRegionB: Description: The DNS name of your ELB in region B

    Type: String

    LoadBalancerHostedZoneRegionB: Description: The Hosted Zone ID of your ELB in region B

    Type: String

Resources:

  DNSHostedZone:

    Type: AWS::Route53::HostedZone Properties:

      Name: !Ref DomainName

  RegionAHealthCheck:

    Type: AWS::Route53::HealthCheck Properties:

      HealthCheckConfig: FailureThreshold: 3

      FullyQualifiedDomainName: !Ref LoadBalancerDNSNameRegionA Port: 80

      RequestInterval: 30

      ResourcePath: "/"

      Type: HTTP

HealthCheckTags:

- { Key: Name, Value: Region A Health Check }

RegionBHealthCheck: Type: AWS::Route53::HealthCheck Properties:

HealthCheckConfig: FailureThreshold: 3

FullyQualifiedDomainName: !Ref LoadBalancerDNSNameRegionB

Port: 80

RequestInterval: 30

ResourcePath: "/"

Type: HTTP

HealthCheckTags:

- { Key: Name, Value: Region B Health Check }

RegionAPrimary:

Type: AWS::Route53::RecordSet Properties:

Name: !Join [ ., [ a, Ref: DomainName ] ]

Type: A

HostedZoneId: !Ref DNSHostedZone AliasTarget:

HostedZoneId: !Ref LoadBalancerHostedZoneRegionA DNSName: !Ref LoadBalancerDNSNameRegionA Failover: PRIMARY

SetIdentifier: primary-region-a HealthCheckId: !Ref RegionAHealthCheck  
RegionBPrimary:

Type: AWS::Route53::RecordSet Properties:

Name: !Join [ ., [ b, Ref: DomainName ] ]

Type: A

HostedZoneId: !Ref DNSHostedZone AliasTarget:

HostedZoneId: !Ref LoadBalancerHostedZoneRegionB

DNSName: !Ref LoadBalancerDNSNameRegionB

Failover: PRIMARY

SetIdentifier: primary-region-b HealthCheckId: !Ref RegionBHealthCheck

RegionAFailover:

Type: AWS::Route53::RecordSet Properties:

Name: !Join [ ., [ a, Ref: DomainName ] ]

Type: A

HostedZoneId: !Ref DNSHostedZone AliasTarget:

HostedZoneId: !Ref LoadBalancerHostedZoneRegionB

DNSName: !Ref LoadBalancerDNSNameRegionB

Failover: SECONDARY

SetIdentifier: secondary-region-a RegionBFailover:

Type: AWS::Route53::RecordSet Properties:

Name: !Join [ ., [ b, Ref: DomainName ] ]

Type: A

HostedZoneId: !Ref DNSHostedZone AliasTarget:

HostedZoneId: !Ref LoadBalancerHostedZoneRegionA DNSName: !Ref LoadBalancerDNSNameRegionA Failover: SECONDARY

SetIdentifier: secondary-region-b

NorthAmericaGeolocation:

Type: AWS::Route53::RecordSet Properties:

Name: !Ref DomainName Type: A

HostedZoneId: !Ref DNSHostedZone AliasTarget:

HostedZoneId: !Ref DNSHostedZone DNSName: !Join [ ., [ a, Ref: DomainName ] ]

GeoLocation:

ContinentCode: NA # North America SetIdentifier: geolocation-region-a

RestOfWorldGeolocation: Type: AWS::Route53::RecordSet Properties:

Name: !Ref DomainName Type: A

HostedZoneId: !Ref DNSHostedZone AliasTarget:

HostedZoneId: !Ref DNSHostedZone DNSName: !Join [ ., [ b, Ref: DomainName ] ]

GeoLocation:

CountryCode: "\*" # Rest of world SetIdentifier: geolocation-region-b

<strong> aws cloudformation create-stack </strong> <strong> --stack-name geolocation-failover </strong> <strong> --template-body file://07-routing-based-on-location.yaml </strong> <strong> --parameters </strong> <strong>

ParameterKey=DomainName,ParameterValue=gitrepository.com </strong> <strong>

ParameterKey=LoadBalancerDNSNameRegionA,ParameterValue=<elb-a> </strong>

<strong> ParameterKey=LoadBalancerHostedZoneRegionA, </strong>

ParameterValue=<elb-zoneid-a> </strong> <strong>

ParameterKey=LoadBalancerDNSNameRegionB,ParameterValue=<elb-b> </strong>

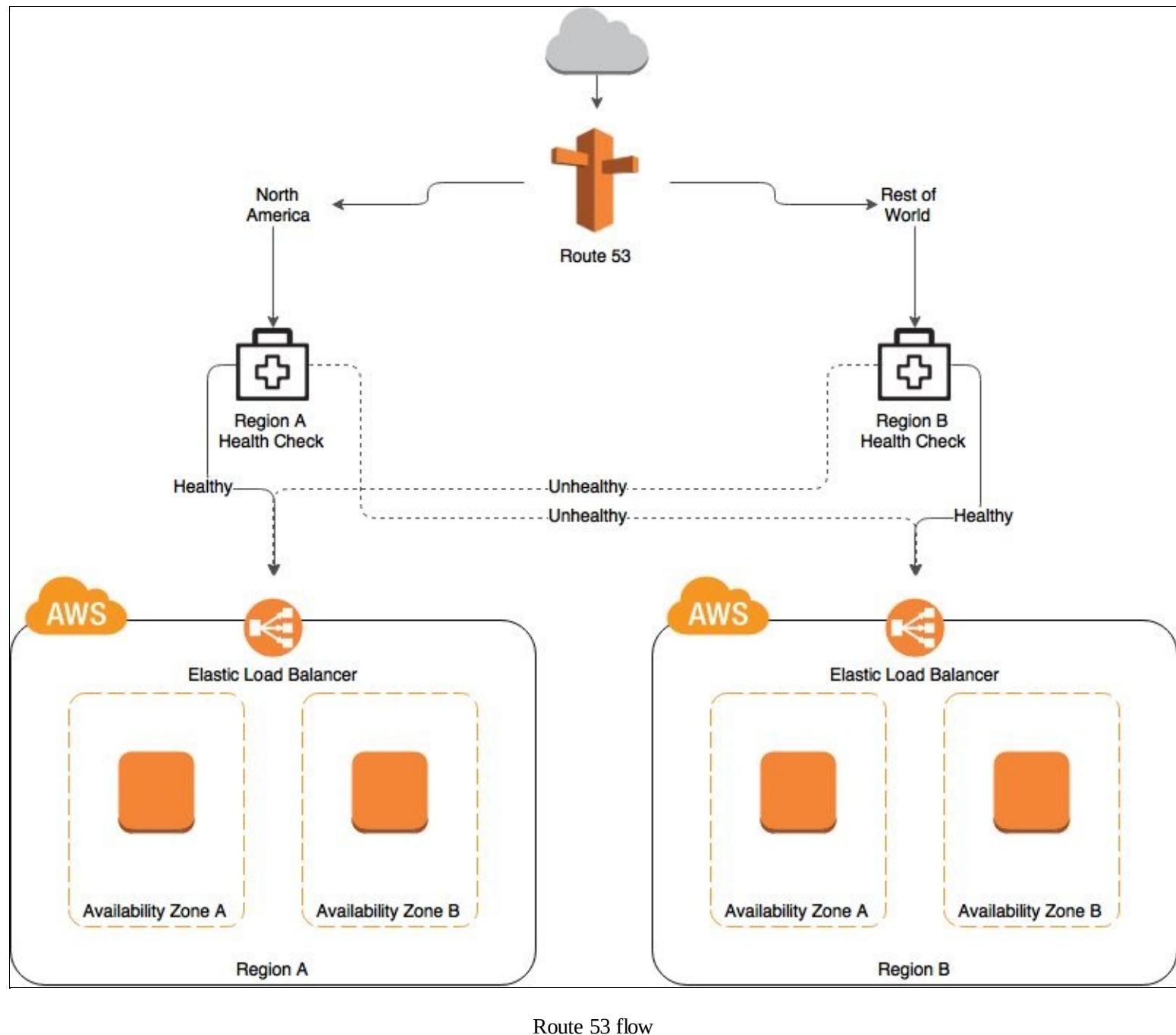
<strong> ParameterKey=LoadBalancerHostedZoneRegionB, </strong>

ParameterValue=<elb-zoneid-b></strong>



# How it works...

We've effectively constructed a small decision tree, as follows:

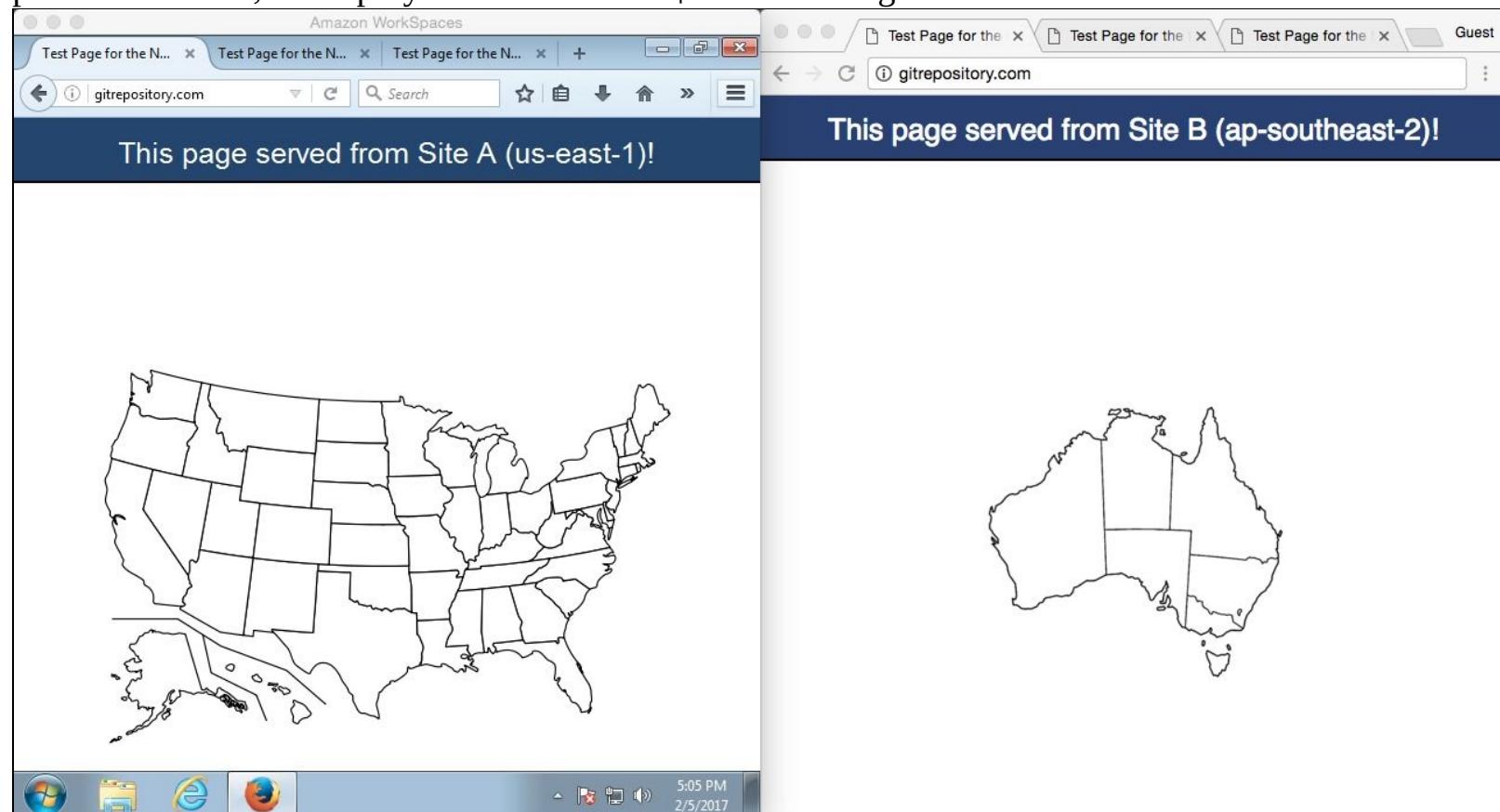


In order to test this for yourself, you'll need to have some way of performing DNS responses from other regions. In the following screenshots, we have provisioned a machine using AWS workspaces in North America (left), while our actual location is in Australia (right).



# Normal operation (geolocation routing)

Under normal operation, our North American user (left) will connect to region A, which, for practical reasons, we've deployed in `us-east-1`, although it could be in any region. Our Australian user (right) will connect to region B, which is the region we've designated as being for the *rest of the world*. Again, for practical reasons, we deployed this site to the `ap-southeast-2` region:



Left: Region A served to North American user. Right: Region B served to Australian user.



# Region A failure

To simulate a failure of region A, we'll simply stop the web server, which is attached to the ELB as follows:

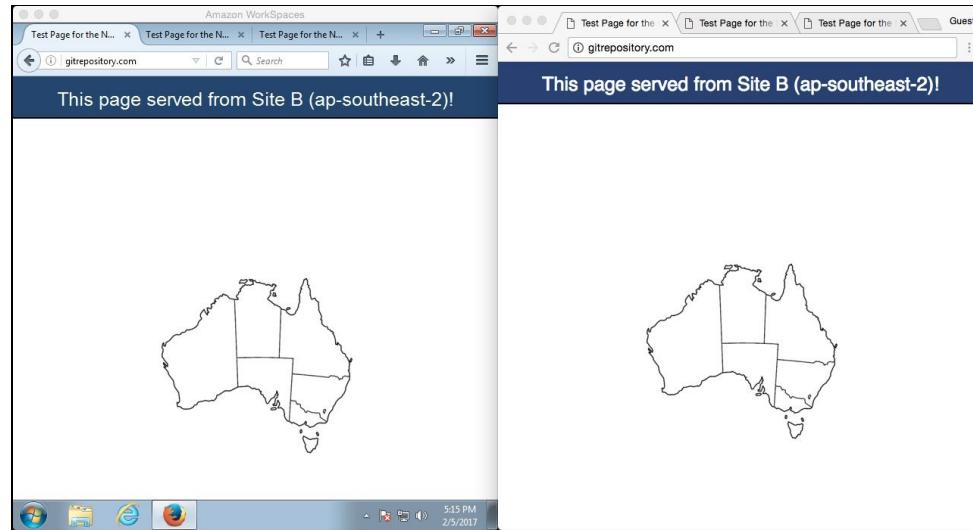
```
[root@ip-172-30-0-153 ec2-user]# service nginx stop  
Stopping nginx: [ OK ]
```

After a short period, the web console will show that the health check for region A is failing:

The screenshot shows the AWS Route 53 Management Console. On the left, there's a sidebar with options like Dashboard, Hosted zones, Health checks (which is selected and highlighted in orange), Traffic flow, Traffic policies, Policy records, Domains, Registered domains, and Pending requests. The main area has tabs for Create health check, Delete health check, and Edit health check. Below that is a table titled 'Filter by keyword' with columns for Name, Status, Description, and Alarms. It lists two entries: 'Region B Health Check' (Status: Healthy) and 'Region A Health Check' (Status: Unhealthy). The status bar at the bottom indicates 'No health check selected.'

Region A failing health check

Our North American user (left) now sees region B instead:



Left: Region B served to North American users due to failover. Right: Region B served to Australian users as normal.



# Region B failure

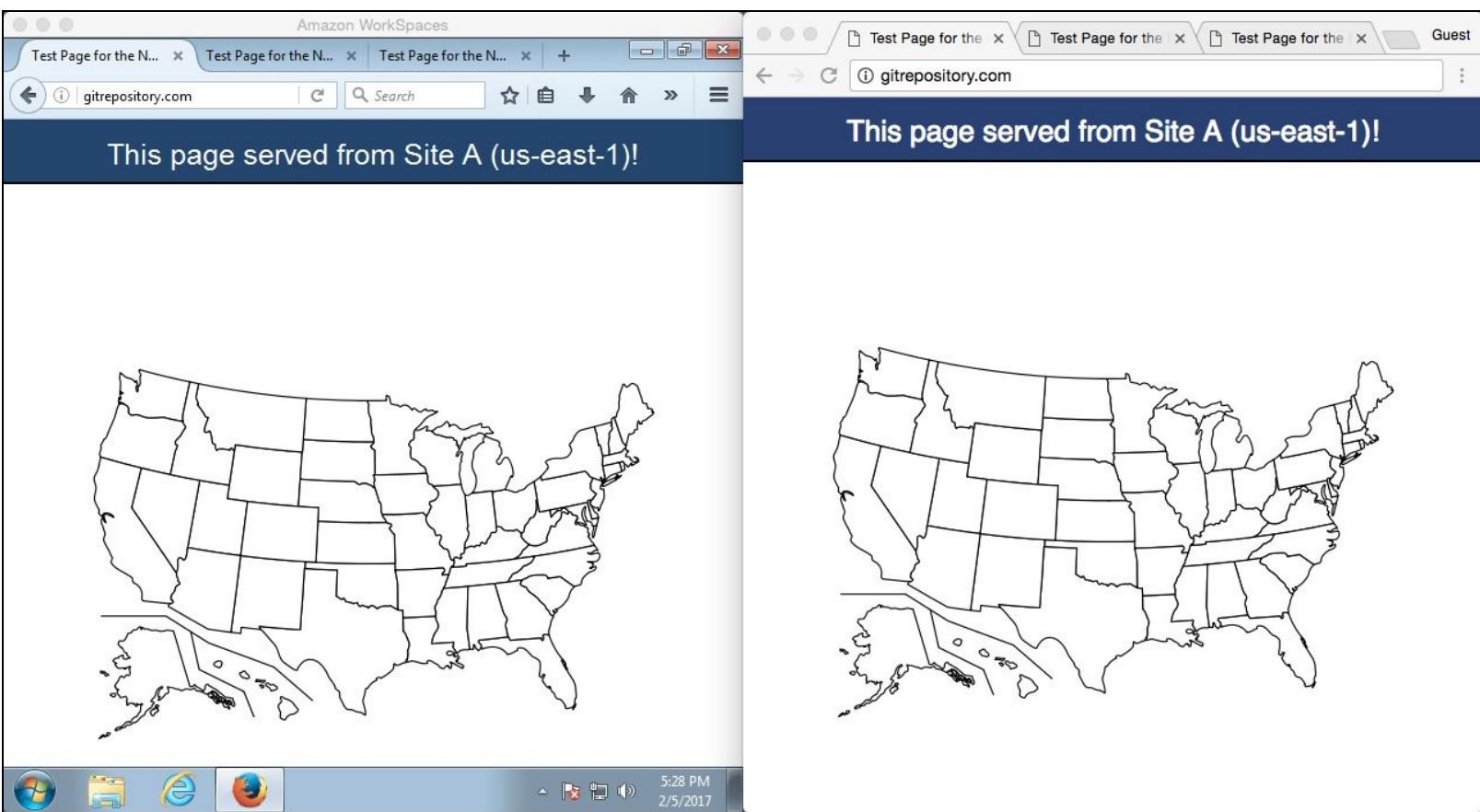
We'll now flip the script and simulate the same scenario in region B. This time, the web server in this region is stopped, but the server in region A is healthy:

The screenshot shows the AWS Route 53 Management Console interface. On the left, there's a sidebar with options like Dashboard, Hosted zones, Health checks (which is selected and highlighted in orange), Traffic flow, Traffic policies, Policy records, Domains, Registered domains, and Pending requests. The main content area has a title bar with 'Create health check', 'Delete health check', and 'Edit health check' buttons, along with a 'Filter by keyword' search bar. Below this is a table with columns for Name, Status, Description, and Alarms. There are two entries: 'Region B Health Check' (Status: Unhealthy, last checked 3 hours ago) and 'Region A Health Check' (Status: Healthy, last checked 3 hours ago). At the bottom of the main content area, there are tabs for Info, Monitoring, Alarms, Tags, Health checkers, and Latency, with 'Info' being the active tab. A message 'No health check selected.' is displayed below the tabs. The footer of the page includes links for Feedback, English, Copyright information (© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.), Privacy Policy, and Terms of Use.

	Name	Status	Description	Alarms
<input type="checkbox"/>	Region B Health Check	<div style="width: 20px; height: 10px; background-color: green;"></div> <div style="width: 10px; height: 10px; background-color: red;"></div> now	Unhealthy http://nginx-Examp-ZVTF5IQHNUW3-63...	No alarms configured.
<input type="checkbox"/>	Region A Health Check	<div style="width: 10px; height: 10px; background-color: green;"></div> <div style="width: 20px; height: 10px; background-color: green;"></div> now	Healthy http://nginx-Examp-1XBNWQ8WL88RM...	No alarms configured.

Region B failing health check

Region A content will now be shown to both North American users and those designated as the *rest of the world* (including Australia):



Left: Region A served to North American users as normal. Right: Region A served to Australian users due to failover.



# There's more...

Route 53 offers a couple of other useful routing policies, so you should have a think about which best suits you:

- **Latency-based routing:** This policy makes the Route 53 DNS servers respond to you with IP addresses that provide the lowest latency. This will not necessarily be the endpoint geographically closest to you.
- **Weighted routing:** This allows you to divvy up your traffic between endpoints based on a weighting system. You might have a 50/50 split between two regions, or you may elect to have a 90/10 ratio instead.



## See also...

- The *Hosting a static website* recipe in [Chapter 3, Storage and Content Delivery](#)
- The *Creating security groups* recipe in [Chapter 4, Using AWS Compute](#)



# Network logging and troubleshooting

One of the benefits of using virtualized infrastructure is that you can get a level of introspection that is difficult or costly with physical hardware. Being able to quickly switch on logging at a network-device level is an extremely useful feature, especially when getting used to the interactions between VPCs, subnets, NACLs, routing, and security groups.

In this recipe, we will turn on logging for our network resources. You could do this all the time, to give yourself another layer for monitoring and auditing, or you could selectively enable it during troubleshooting, saving yourself any additional datastorage charges.



# Getting ready

For this recipe, you must have a VPC to log activity on.

AWSTemplateFormatVersion: "2010-09-09"

Description: Flow logs for networking resources

Parameters:

VpcId:

Type: String

Description: The VPC to create flow logs for

Resources:

LogGroup:

Type: AWS::Logs::LogGroup DeletionPolicy: Delete Properties:

LogGroupName: LogGroup

IamRole:

Type: AWS::IAM::Role Properties:

AssumeRolePolicyDocument: Version: "2012-10-17"

Statement:

-

Effect: Allow

Principal:

Service: vpc-flow-logs.amazonaws.com Action: sts:AssumeRole Policies:

-

PolicyName: CloudWatchLogsAccess PolicyDocument: Version: "2012-10-17"

Statement:

Action:

- logs:CreateLogGroup - logs:CreateLogStream - logs:PutLogEvents -  
logs:DescribeLogGroups - logs:DescribeLogStreams Effect: Allow

Resource: "\*"

FlowLog:

Type: AWS::EC2::FlowLog DependsOn: LogGroup Properties:

    DeliverLogsPermissionArn: !GetAtt IamRole.Arn LogGroupName: LogGroup  
    ResourceId: !Ref VpcId ResourceType: VPC

TrafficType: ALL

```
<strong> aws cloudformation create-stack </strong> <strong> --stack-name  
VpcFlowLogs </strong> <strong> --template-body file://07-flow-logs.yml </strong>  
<strong> --capabilities CAPABILITY_IAM </strong> <strong> --parameters  
ParameterKey=VpcId,ParameterValue=<your-vpc-id></strong>
```

8. Once launched (and assuming you have network activity), you will be able to see your flow-log in the CloudWatch logs console.



# How it works...

The only parameter required for this template is the VPC ID to target. We specifically target a VPC to turn on flow-logging for, because it gives us the most *bang for buck*. While you can enable flow-logs for subnets and **Elastic Network Interfaces (ENIs)** individually, if you enable them on a VPC you get flow-logs for all the networking resources contained in that VPC—which includes subnets and ENIs.

In the resources section, we start by explicitly defining the log group to *hold* the flow-logs. If you don't create the log group yourself (and specify it in your flow-log resource configuration), a log group will be created for you. This means that you will still be able to use flow-logs, but the log group won't be managed by CloudFormation and will have to be maintained (for example, deleted) manually. We have also set a **deletion policy** of *delete* for our log group. This means it will be deleted if the CloudFormation stack is deleted, which is fine for a demonstration such as this. If using in a *real* environment (such as production), remove the `DeletionPolicy` property and its value.



*By default, CloudWatch log groups are not deleted when the stack that created them is deleted. This lets you retain any important logs, but it can incur an ongoing cost.*

Next we define the IAM role to use. Via the `AssumeRolePropertyDocument` value, we give the AWS flow-logs service permission to assume this role. Without this access, the flow-logs service cannot access the account. In the `Policies` property, we give the role permission to create and update log groups and streams.

Finally, now that we have created the dependent resources, we define the flow-log resource itself. You don't need to define the resources in order of dependencies, but it is usually easier to read if you do. In the resource, we also define a `DependsOn` relationship to the log group we defined earlier, so that the log group is ready to receive the flow-logs when it is created.

The final step is to launch the template you have created, passing the VPC ID as parameter. As this template creates an IAM role to allow the VPC service to send logs to CloudWatch logs, the command to create the stack must be given the `CAPABILITY_IAM` flag to signify that you are aware of the potential impact of launching this template.



# **There's more...**

Turning on logging is just the start of the troubleshooting process. There are a few other things you should be aware of when using flow-logs.



# Log format

Once logging is enabled, you can view the logs in the CloudWatch logs console. Here is a summary of the type of information you will see in the flow-log (in order):

- The VPC flow-logs version
- The AWS account ID
- The ID of the network interface
- The source IPv4 or IPv6 address
- The destination IPv4 or IPv6 address
- The source port of the traffic
- The destination port of the traffic
- The IANA protocol number of the traffic
- The number of packets transferred
- The number of bytes transferred
- The start time of the capture window (in Unix seconds)
- The end time of the capture window (in Unix seconds)
- The action associated with the traffic; for example, ACCEPT OR REJECT
- The logging status of the flow-log; for example, OK, NODATA, or SKIPDATA



*To identify the protocol, check the protocol number field against the IANA protocol numbers list at <http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>.*



# Updates

You cannot update the configuration of an existing flow-log; you must delete it and recreate it if you want to change any settings associated. This is another reason why it is good to explicitly create and manage the associated log group.



# Omissions

Some traffic is not captured by the flow-logs service, as follows:

- Traffic to the Amazon DNS server ( $x.x.x.2$  in your allocated range)
- Traffic for Amazon Windows license activation (obviously only applicable to Windows instances)
- Traffic to and from the instance metadata service (that is, IP address  $169.254.169.254$ )
- DHCP traffic
- Traffic to the reserved VPC IP address for the default VPC router ( $x.x.x.1$  in your allocated range)



## See also

- The *Building a secure network* recipe



# Security and Identity

In this chapter, we will cover:

- Federating with your AWS account
- Creating SSL certificates
- Active Directory as a service
- Creating users
- Creating instance roles
- Cross-account user roles
- Storing secrets



# Introduction

Security is one of the most critical areas of using the cloud. It's important to get it right because good security practices reinforce themselves, leading to a virtuous cycle of capabilities and control.

There are many tools and AWS services to ensure that your cloud-based infrastructure is as secure—if not more secure—than your own resources.

AWS IAM is the backbone of security in AWS. It provides incredibly granular levels of permissions to allow (and deny) specific users access to your resources.

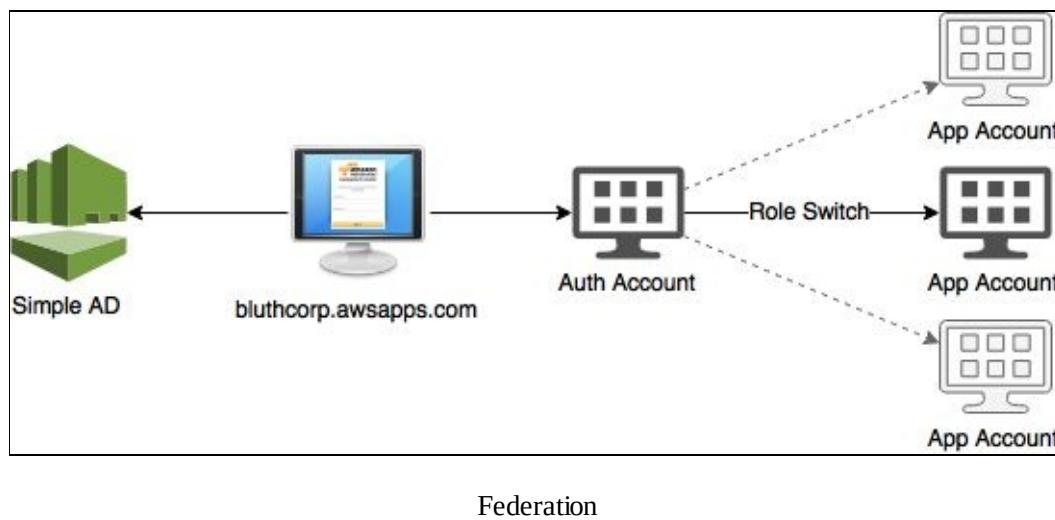


# Federating with your AWS account

This recipe will show you how to federate identities from your Active Directory and use AD groups and IAM roles to provide different levels of access to multiple AWS accounts.

At a high level, we're going to have an AWS account that is designated as an **Auth Account**. Users will log in to this account and be assigned a role. This role will have next to no privileges because we don't want them doing anything in the **Auth Account**. However, they will be able to use role switching to access another AWS account; we'll call this the **App Account**.

This is a reasonably common pattern whereby users will have access to a number of AWS accounts and use role switching to jump between them—all using credentials that are verified against an AD backend and a level of access that is derived from AD groups.





# Getting ready

You'll need the following before we can proceed:

- An instance of Simple AD. Refer to the *Active Directory as a service* recipe.
- The name of an access URL, which your users will use to log in (that is, <https://bluthcorp.awsapps.com>).
- Two AWS accounts. One of these will be your *Auth Account*, the other will be your *App Account*.
- A Windows server in your VPC, joined to your Simple AD domain, with Remote Server Admin Tools installed so we can manage groups and users.



*If you launch the Windows server using the launch wizard, it will give you the option of joining the domain at boot time. Note that the server will need to be running with an instance role that will have the following two AWS Managed Policies: AmazonEC2RoleForSSM and AmazonSSMReadOnlyAccess.*



# How to do it...

This recipe is split up in to five parts:

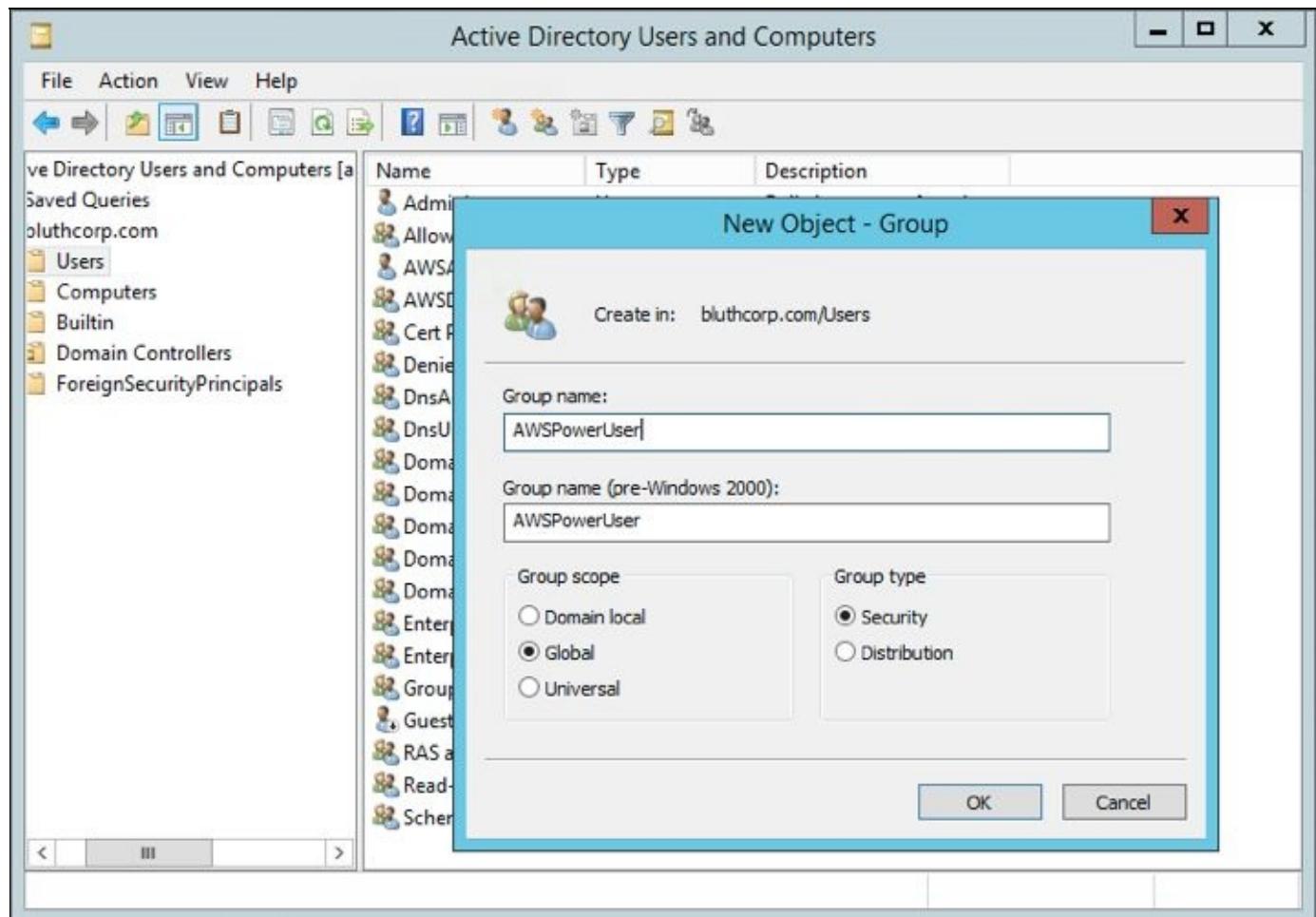
- Active Directory configuration
- Auth Account policy configuration
- Auth Account role configuration
- Simple AD Directory configuration
- App Account role configuration



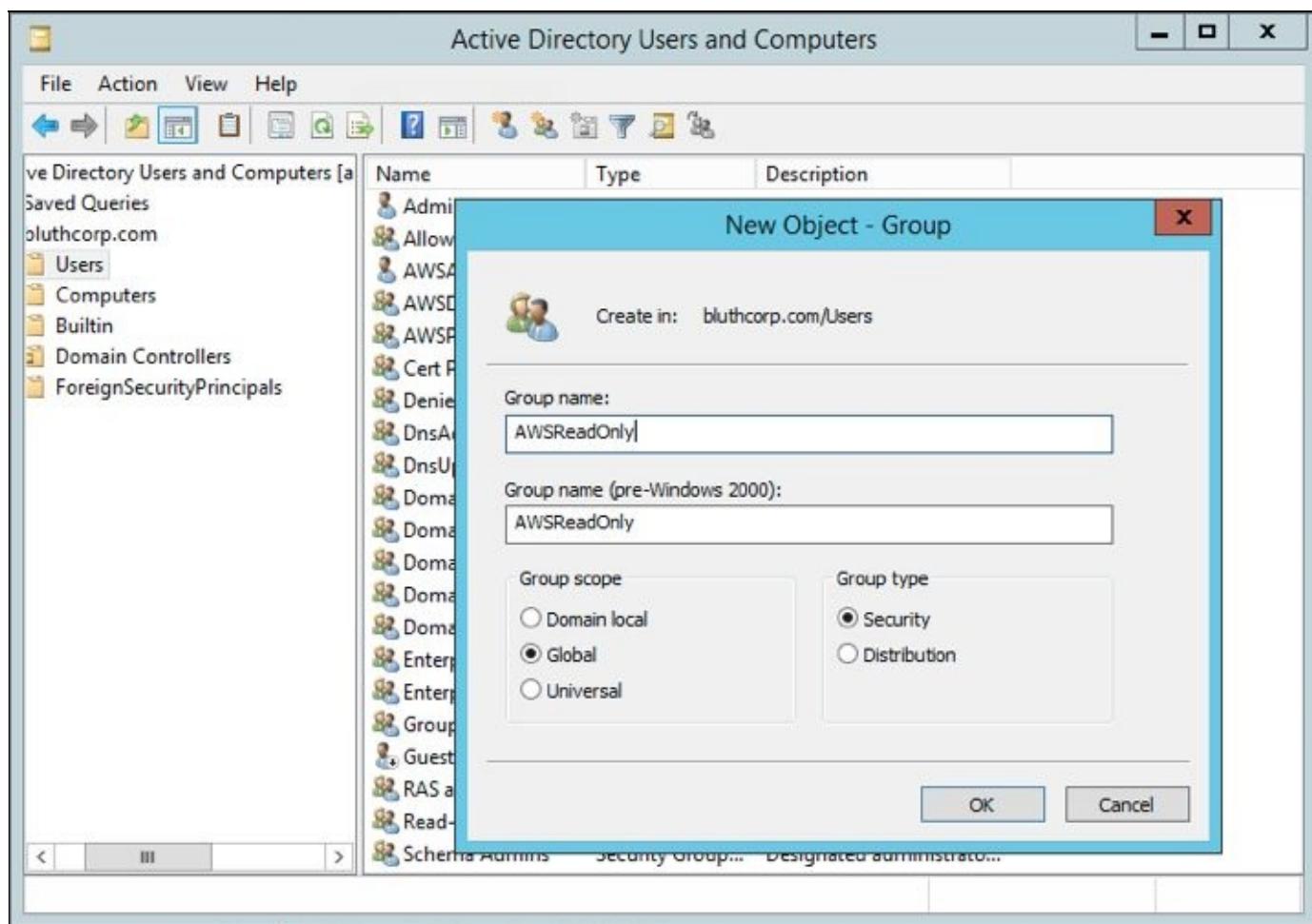
# Active Directory configuration

Our first task will be to create the necessary groups in Active Directory:

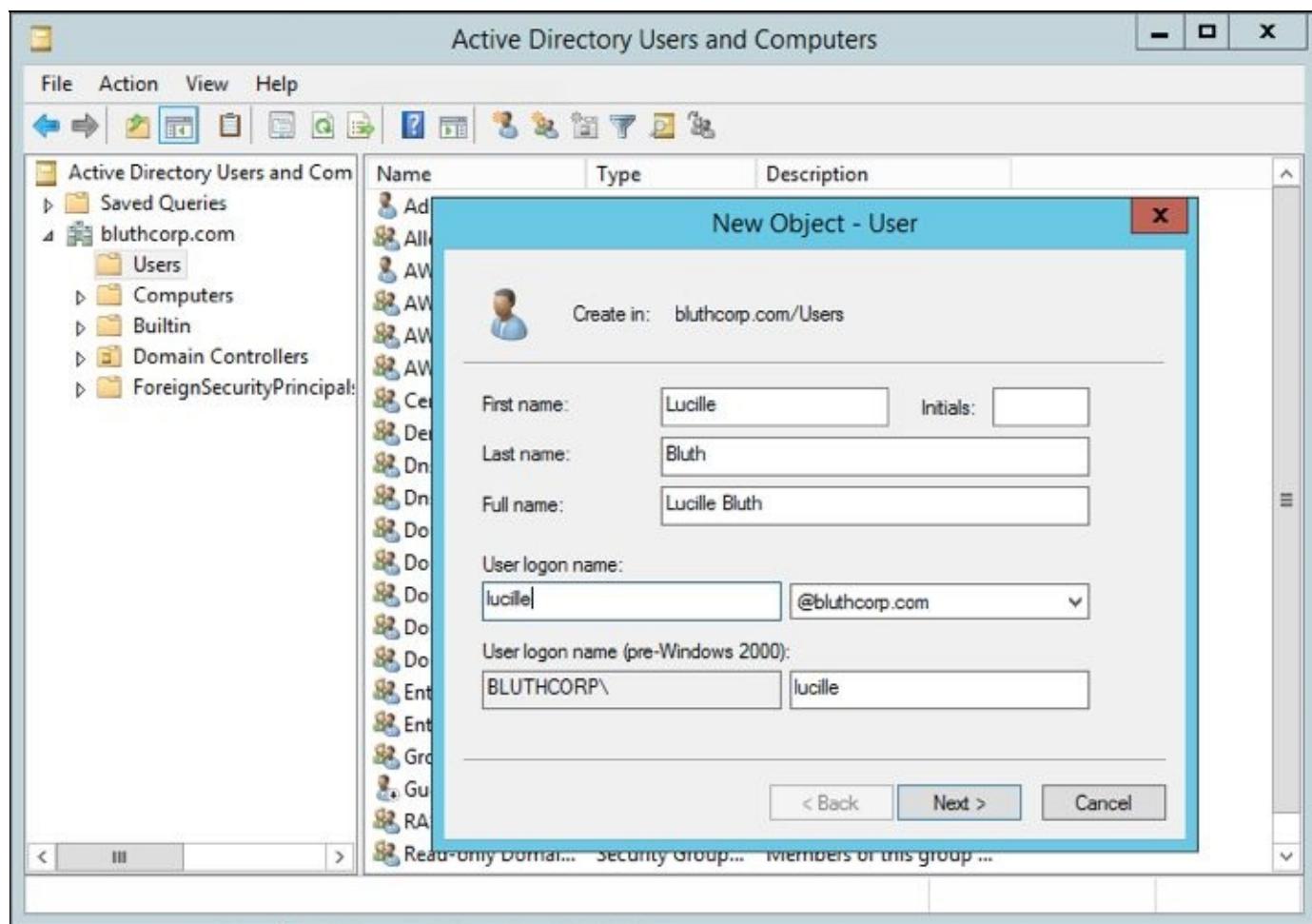
1. Go ahead and create a group called `AWSPowerUser`, as shown in the following screenshot:



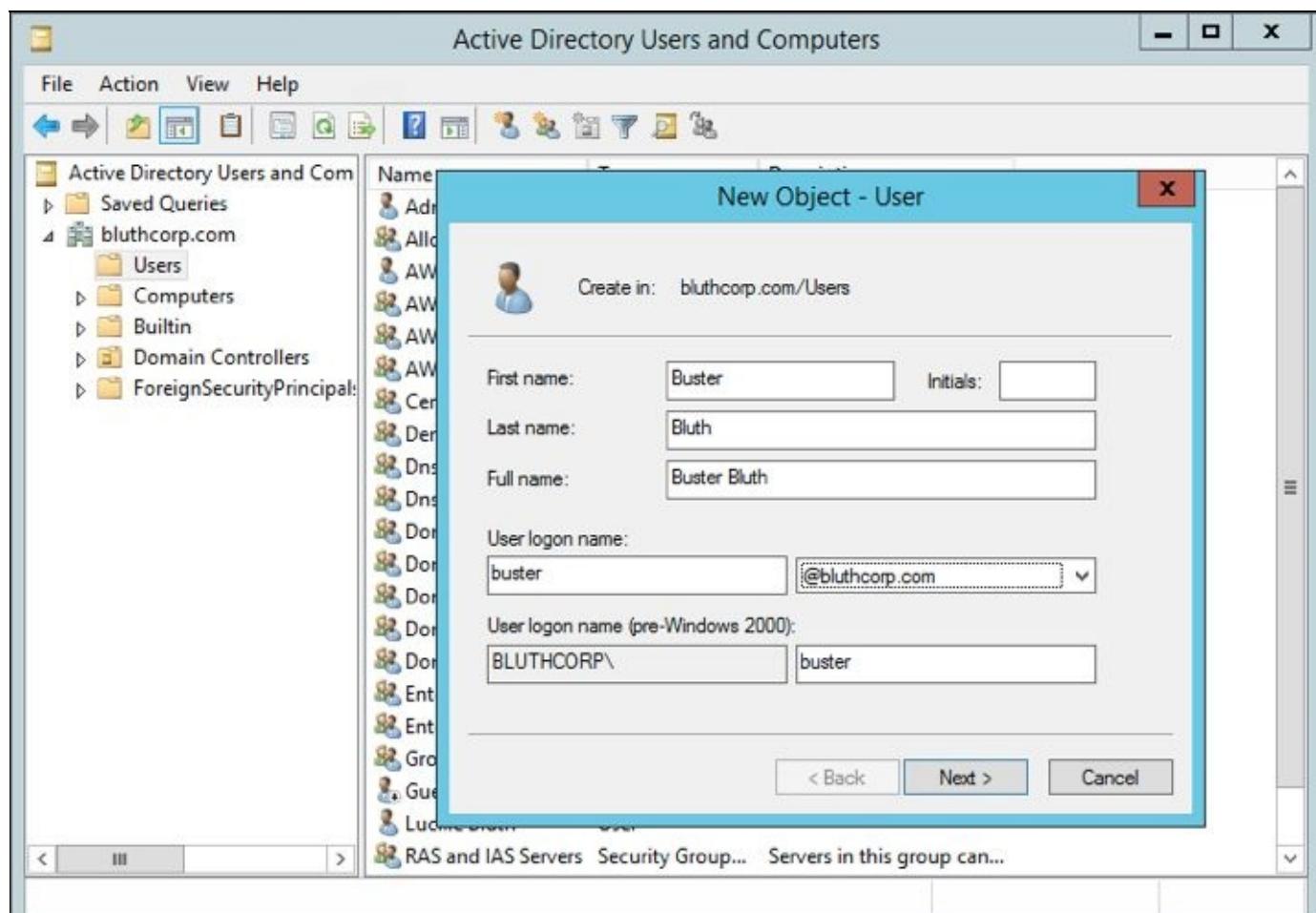
2. Do the same for the `AWSReadOnly` group:



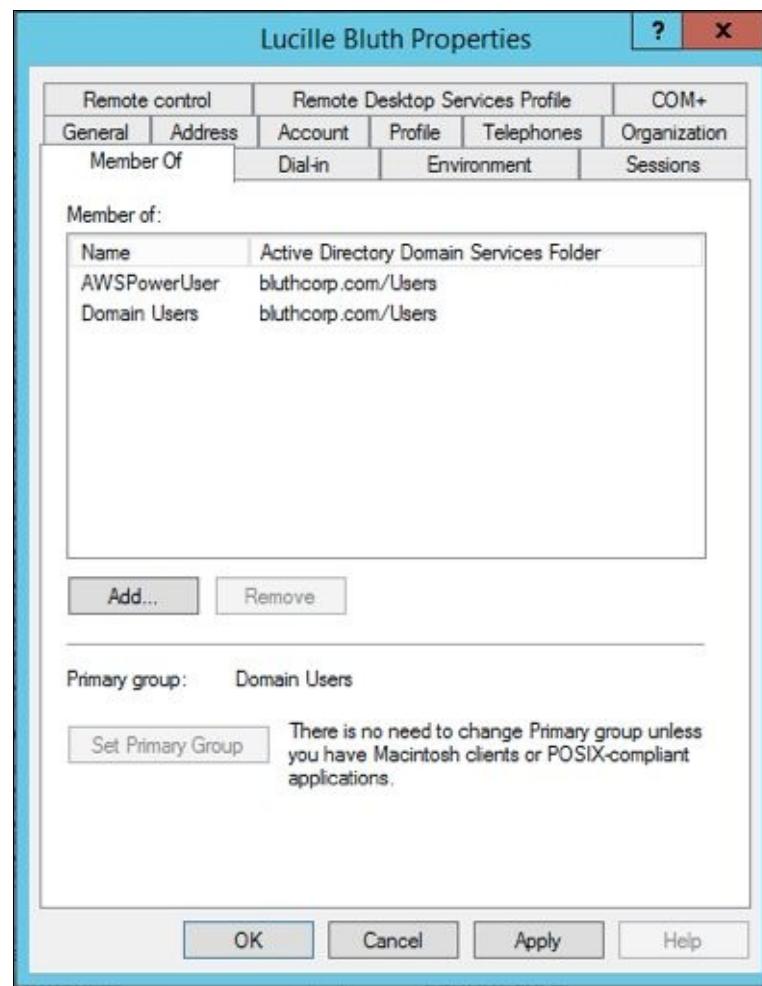
3. We're now going to create a couple of users. The first one is `Lucille`, as shown in the following screenshot:



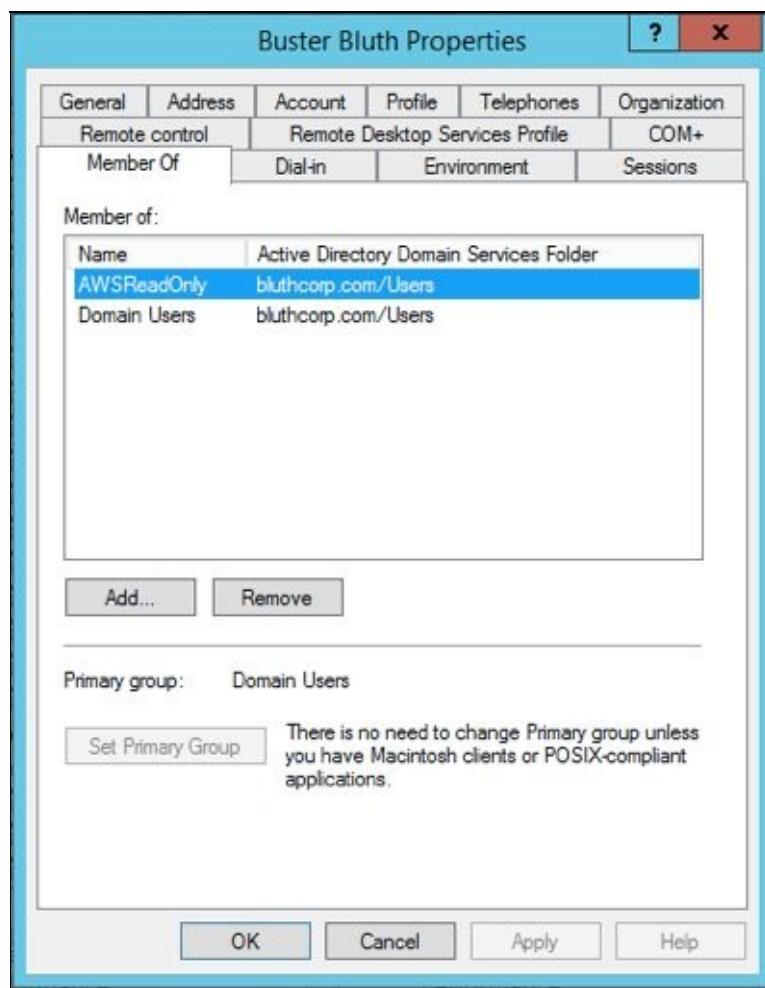
4. The next user will be **Buster**. Let's add him now:



5. Lucille is going to be our power user, so we'll add her to the AWSPowerUser group:



6. We don't really trust **Buster** at all. True to his name, he's prone to breaking things. Let's add him to the AWSReadOnly group:



{

"Version": "2012-10-17", "Statement": [

{

"Sid": "Stmt1487396837000", "Effect": "Allow", "Action": [

"sts:AssumeRole"

],

"Resource": [

"arn:aws:iam::<app-acct-number>:role/\*"

]

}

]

}

AWS Directory Service Console      EC2 Management Console      IAM Management Console

Services    Resource Groups    Lucas Chan    Global    Support

Create Policy

Step 1 : Create Policy

Step 2 : Set Permissions

Step 3 : Review Policy

Review Policy

Customize permissions by editing the following policy document. For more information about the access policy language, see [Overview of Policies](#) in the [Using IAM](#) guide. To test the effects of this policy before applying your changes, use the [IAM Policy Simulator](#).

**Policy Name**

AllowAssumeRole

**Description**

This policy allows a user to attempt to assume a role in our application account. All other actions in the Authentication account are disallowed.

**Policy Document**

```
8         "STS:AssumeRole"
9
10        ],
11        "Resource": [
12           "arn:aws:iam::333561987669:role/*"
13        ]
14        }
15 }
```

Use autoformatting for policy editing

Cancel    Validate Policy    Previous    **Create Policy**

# Auth Account policy config



# Auth Account role configuration

Now we're going to create two roles. These roles will correspond to the groups we defined in Active Directory:

- **AWSPowerUser**: CanAssumePowerUser
- **AWSReadOnly**: CanAssumeReadOnly

1. Start by creating the `CanAssumePowerUser` role first:

The screenshot shows the AWS IAM Management Console with the URL `console.aws.amazon.com/iam/home?#/roles`. The top navigation bar includes links for AWS Directory Service Console, EC2 Management Console, IAM Management Console, Services (selected), Resource Groups, Lucas Chan (user), Global, and Support.

The main content area is titled "Create Role" and "Step 1 : Set Role Name". It displays a form with a "Role Name" input field containing "CanAssumePowerUser". Below the input field is a note: "Enter a role name. You cannot edit the role name after the role is created." A small note also states: "Maximum 64 characters. Use alphanumeric and '+,-,@,\_' characters". At the bottom right of the form are "Cancel" and "Next Step" buttons.

2. We want this role to be an AWS Directory Service role, so be sure to select it before proceeding:

Create Role

Step 1 : Set Role Name

**Step 2 : Select Role Type**

Step 3 : Establish Trust

Step 4 : Attach Policy

Step 5 : Review

Select Role Type

AWS Service Roles

Amazon EC2  
Allows EC2 instances to call AWS services on your behalf. Select

AWS Directory Service  
Allows AWS Directory Service to manage access for existing directory users and groups to AWS services. Select →

AWS Lambda  
Allows Lambda Function to call AWS services on your behalf. Select

Amazon Redshift  
Allows Amazon Redshift Clusters to call AWS services on your behalf Select

Amazon API Gateway  
Allows API Gateway to call AWS resources on your behalf Select

Role for Cross-Account Access

Role for Identity Provider Access

Cancel Previous Next Step

3. Attach the `AllowAssumeRole` policy we have already created to this role:

*Hint: You can filter the roles using the search box to make finding them easier.*



Attach Policy

Select one or more policies to attach. Each role can have up to 10 policies attached.

	Policy Name	Attached Entities	Creation Time	Edited Time
<input checked="" type="checkbox"/>	AllowAssumeRole	0	2017-02-18 18:59 UTC+1100	2017-02-18 18:59 UTC+...

Showing 1 results

Cancel Previous Next Step

4. Click Create Role to confirm:

Review

Review the following role information. To edit the role, click an edit link, or click **Create Role** to finish.

Role Name	CanAssumePowerUser	Edit Role Name
Role ARN	arn:aws:iam::067180688831:role/CanAssumePowerUser	
Trusted Entities	The identity provider(s) ds.amazonaws.com	
Policies	arn:aws:iam::067180688831:policy/AllowAssumeRole	Change Policies

Cancel Previous Create Role

5. Now go ahead and do exactly the same for the CanAssumeReadOnly role. Again, attach the AllowAssumeRole policy we created earlier:

AWS Directory Service Console      EC2 Management Console      IAM Management Console

Services | Resource Groups | [Create New Role](#) | [Role Actions](#)

Search IAM

Dashboard    Groups    Users    **Roles**    Policies    Identity providers    Account settings    Credential report

Encryption keys

Filter

Showing 3 results

<input type="checkbox"/>	Role Name	Creation Time
<input type="checkbox"/>	CanAssumePowerUser	2017-02-18 19:06 UTC+1100
<input type="checkbox"/>	CanAssumeReadOnly	2017-02-18 19:07 UTC+1100
<input type="checkbox"/>	WindowsDomainJoin	2017-02-18 17:03 UTC+1100

Feedback    English

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. | Privacy Policy | Terms of Use



# Simple AD configuration

We now need to go through the process of enabling user accounts in the directory to log in to the AWS management console.

1. Point your browser to the AWS Directory Service Console and edit the configuration of your Simple AD directory. Enter the access URL you've chosen:

The screenshot shows the AWS Directory Service Console interface. At the top, there are three tabs: AWS Directory Service Console, EC2 Management Console, and IAM Management Console. Below the tabs, the directory configuration is displayed:

Directory name	bluthcorp.com	Launch time	Sat Feb 18 17:04:43 GMT+1100 2017
NetBIOS name	BLUTHCORP	Availability zones	us-east-1b, us-east-1e
Description	(edit)	VPC	vpc-d76e53b3
DNS Address	172.30.1.103, 172.30.3.143	Subnets	subnet-5c1e852a, subnet-d33143ee
Directory size	Small		

Below the configuration details, there are three tabs: Apps & services (which is selected), Snapshots, and Monitoring. The Apps & services section contains the following information:

**Access URL**  
Create a public endpoint to access your AWS applications and the AWS Management Console. To provide access to users in your directory to AWS applications and the AWS Management Console, you must create an Access URL. Once you create an Access URL for this directory, it cannot be changed. [Learn more](#).

A text input field contains the URL `https://bluthcorp.awsapps.com`, which is highlighted with a large blue arrow pointing to it. To the right of the input field is a blue button labeled "Create Access URL".

**AWS apps & services**  
AWS applications and services enabled for this directory. User accounts in this directory are permitted to authenticate to the selected applications. Application-level access policies control what the users can do. Go to the associated application to turn directory access on or off. [Learn more](#).

Amazon WorkSpaces	Not Enabled
Amazon WorkSpaces Application Manager	Not Enabled
Amazon WorkDocs	Not Enabled
Amazon WorkMail	Not Enabled
AWS Management Console	Not Enabled

2. We now want to enable the AWS Management Console for this service. Click on it to proceed to the next step:

The screenshot shows the AWS Directory Service Console with the 'AWS apps & services' section selected. It lists several services with their status: Amazon WorkSpaces (Not Enabled), Amazon WorkSpaces Application Manager (Not Enabled), Amazon WorkDocs (Not Enabled), Amazon WorkMail (Not Enabled), and AWS Management Console (Not Enabled). A blue arrow points from the text 'So select Use Existing Role, as shown in the following screenshot:' to the 'AWS Management Console' row.

AWS Directory Service Console

DNS Address 172.30.1.103, 172.30.3.143

Directory size Small

Apps & services Snapshots Monitoring

Access URL

Current access URL bluthcorp.awsapps.com

WorkDocs single sign-on Not enabled ⓘ

Enable

AWS apps & services

AWS applications and services enabled for this directory. User accounts in this directory are permitted to authenticate to the selected applications. Application-level access policies control what the users can do. Go to the associated application to turn directory access on or off. [Learn more](#).

Amazon WorkSpaces	Not Enabled	
Amazon WorkSpaces Application Manager	Not Enabled	
Amazon WorkDocs	Not Enabled	– bluthcorp.awsapps.com/workdocs
Amazon WorkMail	Not Enabled	– bluthcorp.awsapps.com/workmail
AWS Management Console	Not Enabled	– bluthcorp.awsapps.com/console

Feedback English © 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

3. We've already created roles and assigned a policy to them. So select Use Existing Role, as shown in the following screenshot:

The screenshot shows the 'Add Role' wizard in the AWS IAM console, specifically Step 1: Select Type. It includes sections for 'Select Role Type', 'Assign Users and Groups to an Existing Role', and 'Create New Role from Template and Assign Users and Groups'. A blue arrow points from the text 'So select Use Existing Role, as shown in the following screenshot:' to the 'Use Existing Role' button in the 'Assign Users and Groups to an Existing Role' section.

Add Role

Step 1: Select Type

Step 2: Select Role

Step 3: Select Users

Step 4: Review

Select Role Type

To allow your directory users to access the AWS Management Console, you can assign users or groups to an existing IAM role or create a new IAM role from an existing template. Assigning a group to an IAM role will allow access to all direct members of that group. (Note: nested group membership is currently not supported.)

Selected Directory bluthcorp.com (d-9067255b71)

Assign Users and Groups to an Existing Role

Provide access using an existing IAM role to users from the selected directory. The selected role must have a valid trust policy for AWS Directory Service.

Use Existing Role

Create New Role

Create New Role from Template and Assign Users and Groups

Create a new IAM role from one of several predefined templates, and provide access using this role to users from the selected directory. If you need a custom access policy, you may visit the IAM Console page to create additional roles.

4. Start with the CanAssumePowerUser role. We need to map it to the AWSPowerUser group we created in AD (the

one Lucille resides in):

The screenshot shows the AWS Directory Service Console interface. On the left, a sidebar lists steps: 'Add Role', 'Step 1: Select Type', 'Step 2: Select Role', 'Step 3: Select Users', and 'Step 4: Review'. The main area is titled 'Select Existing Role' and shows a table of roles for the selected directory 'bluthcorp.com (d-9067255b71)'. The table has columns for 'Role', 'Creation Time', and 'Policy'. Three roles are listed: 'CanAssumePowerUser' (selected), 'CanAssumeReadOnly', and 'WindowsDomainJoin'. Each role has a 'View Role in IAM' link. At the bottom right are buttons for 'Cancel', 'Previous', and 'Next Step'.

5. Search for `AWSPowerUser` and then proceed to the next step:

The screenshot shows the AWS Directory Service Console interface. The sidebar now includes 'Services' and 'Resource Groups'. The main area is titled 'Select Users/Groups' and shows a search interface for users or groups. A search bar contains 'AWSP' with a dropdown showing 'AWSAdministratorAccess' and 'AWSPowerUser' (selected). Below the search bar is a table header for 'Users and Groups to Add' with columns 'Name', 'Id', and 'Type'. A message box at the bottom says 'No users or groups selected.' and 'To add a user or group, search for users or groups in the above text box.' At the bottom right are buttons for 'Cancel', 'Previous', and 'Next Step'.

6. You now need to repeat these steps for the `CanAssumeReadOnly` role. Map it to the `AWSReadOnly` role we created in AD:

The screenshot shows the AWS Directory Service Console interface. The top navigation bar includes links for AWS Directory Service Console, EC2 Management Console, and IAM Management Console. The user is signed in as Lucas Chan from the N. Virginia region.

In the left sidebar, the 'Directories' section is selected, showing a list of schemas.

The main content area displays a success message: "Role Assignments Created" followed by the subtext "Users and groups have been assigned to CanAssumeReadOnly." Below this, the breadcrumb navigation shows "Directories > bluthcorp.com (d-9067255b71) > AWS Management Console Access".

Two buttons are present: "New Role" (highlighted in blue) and "Remove Role". To the right of these buttons are refresh and help icons.

The main list shows two roles under the "Role" tab:

Role	Action
CanAssumePowerUser	<a href="#">View Role in IAM</a>
CanAssumeReadOnly	<a href="#">View Role in IAM</a>

At the bottom of the page, there are links for Feedback, English language selection, copyright information (© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.), Privacy Policy, and Terms of Use.



# App Account role configuration

It's now time to configure our application account. In it, we need to create some new roles and then set up a trust relationship between those new roles and the roles we created in our Auth Account:

1. Start by going to the IAM console in the Auth Account and creating a new role. This role will be

PowerUserRole:

The screenshot shows the 'Create Role' wizard in the IAM Management Console. The current step is 'Step 1 : Set Role Name'. The 'Role Name' field contains 'PowerUserRole'. The instructions say: 'Enter a role name. You cannot edit the role name after the role is created.' Below the field, it says: 'Maximum 64 characters. Use alphanumeric and '+,-,@,\_' characters'. At the bottom right are 'Cancel' and 'Next Step' buttons.

2. This role will be of the Role for Cross-Account Access kind. Make sure to select this type:

IAM Management Console

Secure https://console.aws.amazon.com/iam/home?region=us-west-2#/roles

Guest

Services Resource Groups

Create Role

Step 1 : Set Role Name

**Step 2 : Select Role Type**

Step 3 : Establish Trust

Step 4 : Attach Policy

Step 5 : Review

Select Role Type

AWS Service Roles

Role for Cross-Account Access

Provide access between AWS accounts you own  
Allows IAM users from one of your other AWS accounts to access this account.

Provide access between your AWS account and a 3rd party AWS account  
Allows IAM users from a 3rd party AWS account to access this account and enforces use of External ID.

Role for Identity Provider Access

Select

Cancel Previous Next Step

3. You'll be prompted to enter an AWS Account ID. This is the account ID of the Auth Account:

IAM Management Console

Secure https://console.aws.amazon.com/iam/home?region=us-west-2#/roles

Guest

Services Resource Groups

Create Role

Step 1 : Set Role Name

Step 2 : Select Role Type

**Step 3 : Establish Trust**

Step 4 : Attach Policy

Step 5 : Review

Enter the ID of the AWS account whose IAM users will be able to access this account.

Account ID: 067180688831

Require MFA:

Cancel Previous Next Step

4. For this role, we are going to use the AWS Managed Policy for PowerUserAccess, so go ahead and attach

this policy now:

The screenshot shows the IAM Management Console with the URL <https://console.aws.amazon.com/iam/home?region=us-west-2#/roles>. The left sidebar shows steps: Step 1 : Set Role Name, Step 2 : Select Role Type, Step 3 : Establish Trust, **Step 4 : Attach Policy**, and Step 5 : Review. The main area is titled "Attach Policy" with the sub-instruction "Select one or more policies to attach. Each role can have up to 10 policies attached." A table lists policies filtered by "Policy Type" set to "PowerUser". One policy, "PowerUserAccess", has a checked checkbox. The table columns are: Policy Name, Attached Entities, Creation Time, and Edited Time. The table shows six results. At the bottom right are buttons: "Cancel", "Previous", and "Next Step".

5. Click Create Role on the confirmation page and we're ready for the next step:

The screenshot shows the IAM Management Console with the URL <https://console.aws.amazon.com/iam/home?region=us-west-2#/roles>. The left sidebar shows steps: Step 1 : Set Role Name, Step 2 : Select Role Type, Step 3 : Establish Trust, Step 4 : Attach Policy, and **Step 5 : Review**. The main area is titled "Review" with the instruction "Review the following role information. To edit the role, click an edit link, or click **Create Role** to finish." It displays the following details:

- Role Name:** PowerUserRole [Edit Role Name](#)
- Role ARN:** arn:aws:iam::333561987669:role/PowerUserRole
- Trusted Entities:** The account 067180688831
- Policies:** arn:aws:iam::aws:policy/PowerUserAccess [Change Policies](#)
- Give this link to users who can switch roles in the console:** <https://signin.aws.amazon.com/switchrole?account=333561987669&roleName=PowerUserRole> [Copy Link](#)

At the bottom right are buttons: "Cancel", "Previous", and "Create Role".

6. AWS will automatically create a trust relationship on our behalf. Unfortunately, it's not quite right, so we need to edit it:

The screenshot shows the IAM Management Console interface. The left sidebar has a 'Roles' section selected. The main area displays the 'PowerUserRole' summary, including its ARN, creation time, and a link to switch roles. Below this, there are tabs for 'Permissions', 'Trust Relationships' (which is highlighted), 'Access Advisor', and 'Revoke Sessions'. A blue arrow points to the 'Edit Trust Relationship' button. The 'Trusted Entities' section shows one entity: 'The account 067180688831'. The 'Conditions' section indicates there are no conditions associated with this role.

7. We want anyone who has the `CanAssumePowerUser` role in our Auth Account to be able to assume `PowerUserRole` in our App Account. So we need to make a small change to the trust relationship like so (remember to replace the account IDs with your own):

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::<auth-account-number>:  
                      role/CanAssumePowerUser"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

IAM Management Console Guest

Secure | https://console.aws.amazon.com/iam/home?region=us-west-2#/roles/PowerUserRole

Services Resource Groups Lucas Chan Global Support

Edit Trust Relationship

## Edit Trust Relationship

You can customize trust relationships by editing the following access control policy document.

**Policy Document**

```

1  {
2    "Version": "2012-10-17",
3    "Statement": [
4      {
5        "Effect": "Allow",
6        "Principal": {
7          "AWS": "arn:aws:iam::067180688831:role/CanAssumePowerUser"
8        },
9        "Action": "sts:AssumeRole"
10      }
11    ]
12 }
```

Cancel Update Trust Policy

8. Repeat these steps by creating a role called `ReadOnlyRole` and attach the AWS Managed `ReadOnlyAccess` policy to it:

IAM Management Console Guest

Secure | https://console.aws.amazon.com/iam/home?region=us-west-2#/roles

Services Resource Groups Lucas Chan Global Support

Create Role

Step 1 : Set Role Name  
Step 2 : Select Role Type  
Step 3 : Establish Trust  
**Step 4 : Attach Policy**  
Step 5 : Review

## Attach Policy

Select one or more policies to attach. Each role can have up to 10 policies attached.

**Filter:** Policy Type  Showing 54 results

	Policy Name	Attached Entities	Creation Time	Edited Time
<input type="checkbox"/>	CloudSearchReadOnlyAccess	0	2015-02-07 05:39 UTC+1100	2015-02-07 05:39 UTC+1..
<input type="checkbox"/>	CloudWatchEventsReadOnl...	0	2016-01-15 05:27 UTC+1100	2016-01-15 05:27 UTC+1..
<input type="checkbox"/>	CloudWatchLogsReadOnlyA...	0	2015-02-07 05:40 UTC+1100	2015-12-04 06:59 UTC+1..
<input type="checkbox"/>	CloudWatchReadOnlyAccess	0	2015-02-07 05:40 UTC+1100	2015-02-07 05:40 UTC+1..
<input type="checkbox"/>	IAMReadOnlyAccess	0	2015-02-07 05:40 UTC+1100	2016-09-07 03:06 UTC+1..
<input checked="" type="checkbox"/>	ReadOnlyAccess	0	2015-02-07 05:39 UTC+1100	2017-02-14 07:51 UTC+1..
<input type="checkbox"/>	ResourceGroupsandTagEdit...	0	2015-02-07 05:39 UTC+1100	2015-02-07 05:39 UTC+1..
<input type="checkbox"/>	ServiceCatalogAdminReadO...	0	2015-09-30 04:40 UTC+1100	2016-11-10 06:21 UTC+1..

Cancel Previous Next Step

9. Again, we want to update the trust policy. Here we're going to allow both `canAssumePowerUser` and

canAssumeReadOnly to switch to the ReadOnlyRole. This will be useful for administrators who would want to avoid accidents while clicking around the console:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::<auth-account-number>:  
                    role/CanAssumeReadOnly"  
            },  
            "Action": "sts:AssumeRole"  
        },  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::<auth-account-number>:  
                    role/CanAssumePowerUser"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

The screenshot shows the AWS IAM Management Console with the URL <https://console.aws.amazon.com/iam/home?region=us-west-2#/roles/ReadOnlyRole>. The user is logged in as 'Lucas Chan'. The 'Edit Trust Relationship' dialog is open, displaying a policy document. The policy document is identical to the one shown in the code block above, defining two statements allowing assume role for specific AWS accounts.

```
1 [ {  
2     "Version": "2012-10-17",  
3     "Statement": [  
4         {  
5             "Effect": "Allow",  
6             "Principal": {  
7                 "AWS": "arn:aws:iam::067180688831:role/CanAssumeReadOnly"  
8             },  
9             "Action": "sts:AssumeRole"  
10        },  
11        {  
12            "Effect": "Allow",  
13            "Principal": {  
14                "AWS": "arn:aws:iam::067180688831:role/CanAssumePowerUser"  
15            },  
16            "Action": "sts:AssumeRole"  
17        }  
18    ]  
}
```

Buttons at the bottom right of the dialog are 'Cancel' and 'Update Trust Policy'.

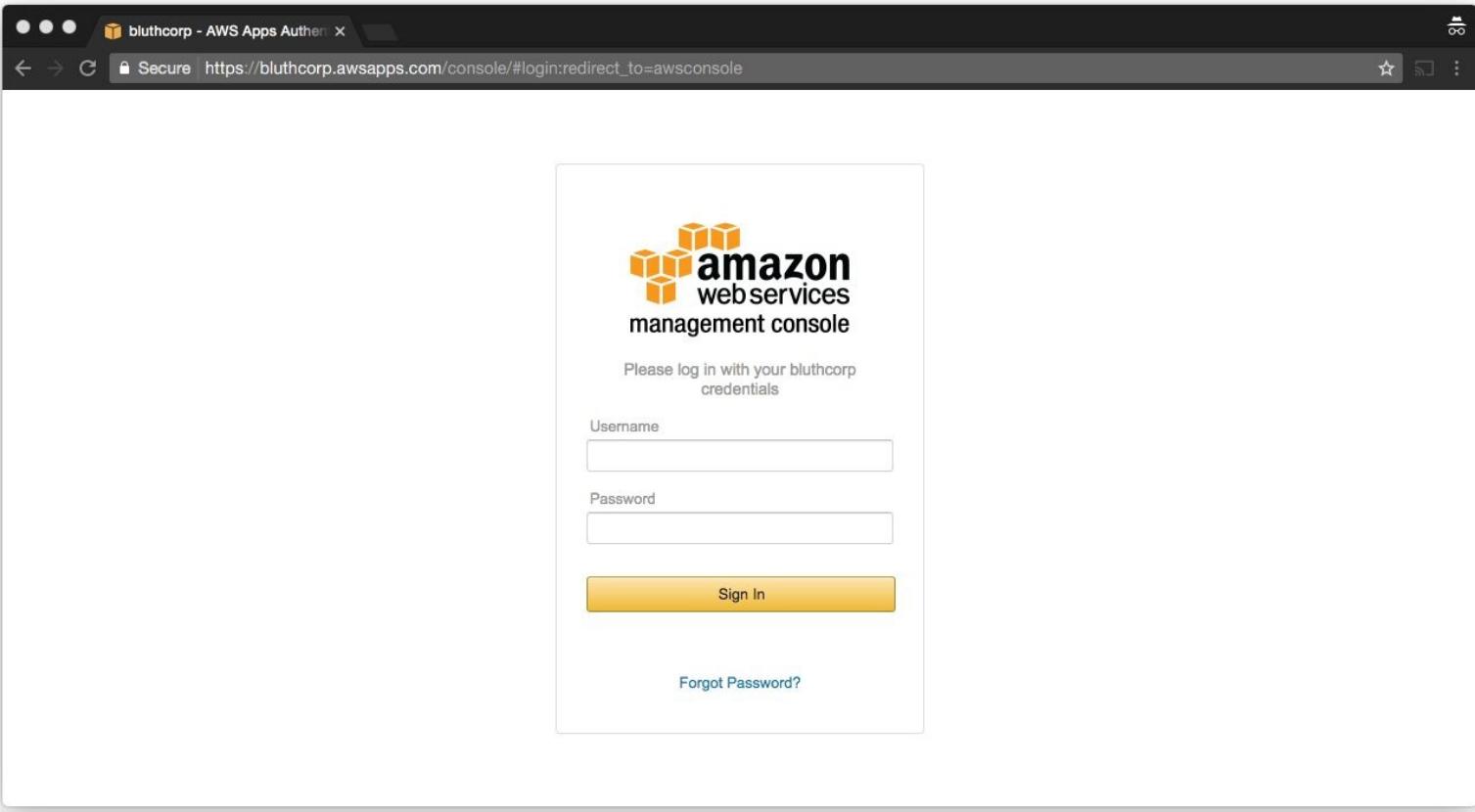
### App Account role config

That was our final step. It's now time to test it out.



# How it works...

1. Visit the access URL you assigned to your Simple AD directory (for example, <https://bluthcorp.awsapps.com/console>). Log in with the credentials of the user Lucille so we can test out our PowerUserRole:



2. If you click around the AWS console, you'll notice you don't really have access to do anything at all. This is because you're currently bound by a policy that only allows you to assume a role (in the application account). So, let's try doing that. Click on your account name in the top-right corner and choose Switch Role:

The screenshot shows the AWS EC2 Management Console interface. In the top right corner, there is a dropdown menu for the user 'CanAssumePowerUser/lucille'. A blue arrow points from the text 'Clicking Switch Role here will log you in to the application account under PowerUserRole:' to the 'Switch Role' link in this menu. The menu also includes links for 'My Account', 'My Billing Dashboard', 'Sign Out', and external links like 'Forums', 'Pricing', and 'Contact Us'.

3. On the next page, you want to enter the account ID of the application account and the role you wish to assume: **PowerUserRole**. Clicking **Switch Role** here will log you in to the application account under **PowerUserRole**:

The screenshot shows the 'Amazon Web Services Switch' page at the URL <https://signin.aws.amazon.com/switchrole?account=333561987669&roleName=PowerUserRole>. The page has a heading 'Switch Role' and a sub-instruction: 'Allows management of resources across AWS accounts using a single user ID and password. You can switch roles after an AWS administrator has configured a role and given you the account and role details. [Learn more](#)'. It contains four input fields: 'Account\*' with value '333561987669', 'Role\*' with value 'PowerUserRole', 'Display Name' with value 'PowerUserRole @ 333561', and 'Color' with a color palette. Below the fields are buttons for '\*Required', 'Cancel', and 'Switch Role'. At the bottom, there are links for 'English' and 'Terms of Use Privacy Policy © 1996-2017, Amazon Web Services, Inc. or its affiliates.'

4. You should now have an active session under `PowerUserRole` in the application account. You'll recall that we assigned a `PowerUserAccess` policy to this role. So you should be free to do almost anything in this account using the profile of the user, `Lucille` (notable exceptions being IAM and organizations management). If you click on your name again, you'll see details about which role was assigned to you when you logged in and which role is currently active:

The screenshot shows the AWS EC2 Management Console interface. At the top, the URL is `https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#`. The top navigation bar includes 'Services' (selected), 'Resource Groups', a bell icon, and 'PowerUserRole @ 333561987669'. A dropdown menu for 'PowerUserRole' is open, showing 'Currently active as: PowerUserRole' and 'Account: 3335-6198-7669'. Below this, there are sections for 'Attributes' (with 'PowerUserAccess' listed), 'Information' (with links to 'All EC2 Resources', 'Forums', 'Pricing', and 'Contact Us'), and 'AWS Marketplace' (with a note about finding software trial products). On the left sidebar, categories like 'EC2 Dashboard', 'INSTANCES', 'IMAGES', 'ELASTIC BLOCK STORE', and 'NETWORK & SECURITY' are visible. The main content area shows 'Resources' (0 Running Instances, 0 Dedicated Hosts, etc.) and a 'Create Instance' section with a 'Launch Instance' button.

5. Try switching to the `ReadOnlyRole`. Verify that you aren't able to create any resources, perhaps by trying to create a new EC2 key pair or by creating an empty security group:

EC2 Management Console https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#

Services Resource Groups

EC2 Dashboard Resources

You are using the following Amazon EC2 resources in the US East (N. Virginia) region:

- 0 Running Instances
- 0 Dedicated Hosts
- 0 Volumes
- 0 Key Pairs
- 0 Placement Groups

Just need a simple virtual private server? Get everything you need to jumpstart your project - compute, storage, and networking – for a low, predictable price. Try Amazon Lightsail for free.

Logged in as: CanAssumePowerUser

Currently active as: ReadOnlyRole

Account: 0671-8068-8831

Account: 3335-6198-7669

Role History:  
ReadOnlyRole @ 333561987669  
PowerUserRole @ 333561987669

My Account  
My Billing Dashboard  
Back to CanAssumePowerUser

Switch Role  
Sign Out

Attributes

Instances  
AMIs  
Bundle Tasks  
Elastic Block Store  
Volumes  
Snapshots  
Network & Security  
Security Groups  
Elastic IPs

Create Instance

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

Launch Instance

Note: Your instances will launch in the US East (N. Virginia) region

Service Health Scheduled Events

Service Status: US East (N. Virginia):

Forums  
Pricing  
Contact Us

AWS Marketplace

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

Feedback English

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

6. Log out and go back to the access URL for Simple AD. Sign in with the credentials of the user, Buster. Again, you'll see you don't have access to do much in the Auth Account:

EC2 Management Console https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#

Services Resource Groups

EC2 Dashboard Resources

You are using the following Amazon EC2 resources in the US East (N. Virginia) region:

You are not authorized to describe Running Instances

You are not authorized to describe Dedicated Hosts

You are not authorized to describe Volumes

You are not authorized to describe Key Pairs

You are not authorized to describe Placement Groups

Just need a simple virtual private server? Get everything you need to jumpstart your project - compute, storage, and networking – for a low, predictable price. Try Amazon Lightsail for free.

Logged in as: CanAssumeReadOnly/buster

Currently active as: ReadOnlyRole

Account: 0671-8068-8831

My Account  
My Billing Dashboard

Role History:  
ReadOnlyRole @ 333561987669  
PowerUserRole @ 333561987669

Switch Role  
Sign Out

Attributes

Instances  
AMIs  
Bundle Tasks  
Elastic Block Store  
Volumes  
Snapshots  
Network & Security  
Security Groups  
Elastic IPs

Create Instance

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

AWS Marketplace

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

Cisco Cloud Services Router (CSR) 1000V - Direct Connect Multi-Gig

Feedback English

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

7. You should be able to switch to the `ReadOnlyRole` in the application account. Try it now to make sure it works. You can use the Role History shortcut to avoid typing in the account number and role name again:

AWS Management Console - https://console.aws.amazon.com/console/home?region=us-east-1#

Services | Resource Groups | AWS services | N. Virginia | Support

ReadOnlyRole @ 333561987669

Logged in as: CanAssumeReadOnly

Currently active as: ReadOnlyRole

Account: 0671-8068-8831 Account: 3335-6198-7669

Role History:

- ReadOnlyRole @ 333561987669
- PowerUserRole @ 333561987669

Switch Role | Sign Out

**AWS services**

Find a service by name (for example, EC2, S3, Elastic Beanstalk).

Recently visited services

- EC2

All services

**Build a solution**

Get started with simple wizards and automated workflows.

- Launch a virtual machine With EC2 ~1 minute
- Build a web app With Elastic Beanstalk ~6 minutes
- Deploy a serverless microservice With Lambda, API Gateway ~2 minutes
- Host a static website With S3, CloudFront, Route 53 ~5 minutes
- Create a backend for your mobile app With Mobile Hub ~5 minutes
- Register a domain With Route 53 ~3 minutes

**What's new?**

Announcing AWS Batch  
Now generally available, AWS Batch enables developers, scientists, and engineers to process large-scale batch jobs with ease. [Learn more](#)

Announcing Amazon Lightsail  
See how this new service allows you to launch and manage your VPS with AWS for a low, predictable price. [Learn more](#)

8. Finally, try switching `Buster` to `PowerUserRole` in the application account. `Buster` definitely shouldn't have access to it and you should see an error page that looks like this:

Amazon Web Services Switch X

Secure https://signin.aws.amazon.com/switchrole

 amazon web services™

## Switch Role

Allows management of resources across AWS accounts using a single user ID and password. You can switch roles after an AWS administrator has configured a role and given you the account and role details. [Learn more](#)

Could not switch roles using the provided information. Please check your settings and try again. If you continue to have problems, contact your administrator.

**Account\***  ⓘ

**Role\***  ⓘ

**Display Name**  ⓘ

**Color**

---

\*Required

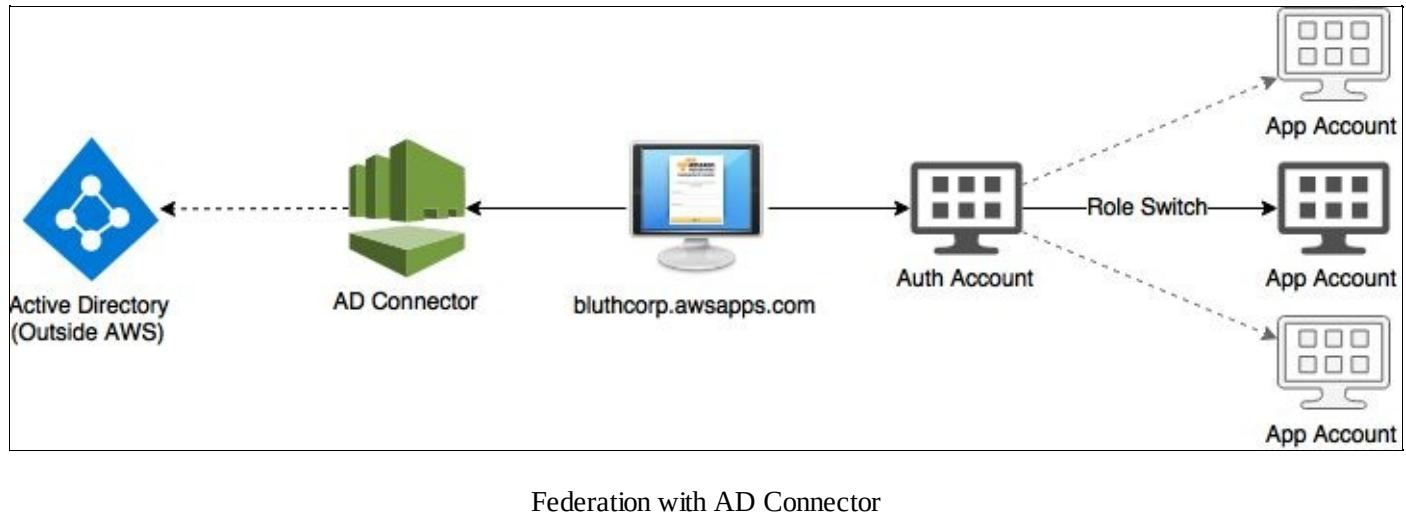
Cancel Switch Role

English 



# There's more...

- Exactly the same setup can be achieved with your existing Active Directory installation, even if it resides outside AWS in your data center. You will need to swap out Simple AD for AD Connector:



Federation with AD Connector

- You can also use ADFS and SAML 2.0 to enable federation to AWS from your existing AD installation. This would negate the need for users to log in to the console using a \*.awsapps.com domain and would also negate the need for an Auth account.



## See also

- The *Active Directory as a service* recipe



# Creating SSL certificates

SSL-based communications are now becoming the de facto standard—unsafe methods are no longer *good enough*.

AWS provides the **AWS Certificate Manager (ACM)** service to provision AWS-backed SSL certificates that you can use with your AWS resources, such as **Elastic Load Balancers (ELBs)** and CloudFront.



*ACM is free to use! There's nothing to pay for the certificates themselves. You pay for the underlying resources you use with them as normal.*

```
<strong> aws acm request-certificate --domain-name <your-domain></strong>
```

2. You can now see the request in the ACM console, but note the request is pending:

Screenshot of the AWS Certificate Manager console showing a pending validation certificate for rowanudell.com.

**Status**

**Validation not complete**  
The status of this certificate request is "Pending validation". Further action is needed to validate and approve the certificate. Learn more.

**Status** Pending validation  
**Detailed status** Email to validate the request was sent at 2017-02-08T09:07:20UTC but we have not yet received approval to issue the certificate for the following domains:  
rowanudell.com

**Details**

Type	Amazon Issued	Requested at	2017-02-08T09:07:20UTC
In use?	No	Public key info	RSA 2048-bit
Domain name	rowanudell.com	Signature algorithm	SHA256WITHRSA
Number of additional names	0	ARN	arn:aws:acm:us-east-1:365407189188:certificate/59510e19-a410-42cf-8901-b845b0acf4a
Identifier	59510e19-a410-42cf-8901-b845b0acf4a	Serial number	N/A

**Tags**

Edit

Name

« < Viewing 1 to 1 of 1 certificates > »

3. Check your domain administration e-mail(s). You will receive a confirmation of the request that will look like the following message:

# Certificate approval for rowanudell.com



**Amazon Certificates** <no-reply@certificates.amazon.com>  
to postmaster Show details

Inbox

8:07 PM (4 minutes ago)

Reply to All

More ▾



Greetings from Amazon Web Services,

We received a request to issue an SSL/TLS certificate for **rowanudell.com**.

Verify that the following domain, AWS account ID, and certificate identifier correspond to a request from you or someone in your organization.

Domain: **rowanudell.com**

AWS account ID: **3654-0718-9188**

AWS Region name: **us-east-1**

Certificate identifier: **59510e19-a410-42cf-8901-b845b0acfe4a**

To approve this request, go to [Amazon Certificate Approvals](#) (

<https://certificates.amazon.com/approvals?code=0a3c56fd-841a-47c6-b3ec-62bfc3a463d7&context=b35dca7b-8115-439f-bbe5-96e44fd97edf-75732d656173742d31> )

and follow the instructions on the page.

If you choose not to approve this request, you do not need to do anything.

*This email is intended solely for authorized individuals for rowanudell.com. To express any concerns about this email or if this email has reached you in error, forward it along with a brief explanation of your concern to [validation-questions@amazon.com](mailto:validation-questions@amazon.com).*

Sincerely,

Amazon Web Services

Amazon Web Services, Inc. is a subsidiary of Amazon.com, Inc. Amazon.com is a registered trademark of Amazon.com, Inc.

This message produced and distributed by Amazon Web Services, Inc., 410 Terry Ave. North, Seattle, WA 98109-5210.

©2015, Amazon Web Services, Inc. or its affiliates. All rights reserved. View our privacy policy.

- Once you approve the request, you will be given a confirmation message:



Amazon Web Services (AWS) has received a request to issue an SSL certificate for rowanudell.com. You are listed as one of the authorized representatives for this domain name. Your authorization is required prior to issuing this certificate.

Verify that the domain name, AWS account ID, and certificate identifier below correspond to a request from you or a person authorized to request certificates for this domain name.

<b>Domain name</b>	rowanudell.com
<b>AWS account number</b>	3654-0718-9188
<b>AWS Region</b>	us-east-1
<b>Certificate identifier</b>	59510e19-a410-42cf-8901-b845b0acfe4a

Review the information presented above and click **I Approve** only if you recognize the request and the account requesting it. By clicking **I Approve**, you authorize Amazon to request a certificate for the above domain name.

If you choose not to approve this request, close this page.

If you have concerns about the validity of this request, forward the email you received with a brief explanation of your concern to: [validation-questions@amazon.com](mailto:validation-questions@amazon.com)

## 5. You can now see that the certificate is ready to use in the AWS ACM console:



Success!

You have approved an SSL/TLS certificate for the domain name rowanudell.com

<b>Domain name</b>	rowanudell.com
<b>AWS account number</b>	3654-0718-9188
<b>AWS Region</b>	us-east-1
<b>Arn</b>	arn:aws:acm:us-east-1:365407189188:certificate/59510e19-a410-42cf-8901-b845b0acfe4a

Once all the domain names in the certificate request are approved, the authorized AWS account holder can review the certificate via the [AWS Management Console](#), CLI, or API, or provision the certificate for use with integrated services, such as Amazon CloudFront or Elastic Load Balancing. For more information refer to the [AWS Certificate Manager User Guide](#).

## 6. Use the Identifier value to apply the certificate to your resources:

AWS Certificate Manager

Secure https://console.aws.amazon.com/acm/home?region=us-east-1#/?id=59510e19-a410-42cf-8901-b845b0acf4a

Services Resource Groups Actions

Request a certificate Import a certificate Actions

< Viewing 1 to 1 of 1 certificates >

	Name	Domain name	Additional names	Status	Type	In use?
<input type="checkbox"/>	rowanudell.com			Issued	Amazon Issued	No

**Status**

Status Issued  
Detailed status The certificate was issued at 2017-02-08T09:14:28UTC

**Details**

Type	Amazon Issued	Requested at	2017-02-08T09:07:20UTC
In use?	No	Issued at	2017-02-08T09:14:28UTC
Domain name	rowanudell.com	Not before	2017-02-08T00:00:00UTC
Number of additional names	0	Not after	2018-03-08T12:00:00UTC
Identifier	59510e19-a410-42cf-8901-b845b0acf4a	Public key info	RSA 2048-bit
Serial number	07:a8:7e:18:92:c6:e4:6d:03:3e:63:43:77:2b:f7:a7	Signature algorithm	SHA256WITHRSA
		ARN	arn:aws:acm:us-east-1:365407189188:certificate/59510e19-a410-42cf-8901-b845b0acf4a

**Tags**

Edit

Name

< Viewing 1 to 1 of 1 certificates >

Feedback English

© 2006 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use



# How it works...

Using the CLI tool is the quickest and easiest way to create a certificate request. Create the certificate in the region you plan to use it in; that is where your ELB(s) are located.



*If you plan to use your certificate with CloudFront, you must create it in the us-east-1 region.*

After the request has been created, AWS will confirm the request is valid by sending approval e-mails to various standard e-mail addresses, based on conventions and the WHOIS information for your domain. Approval e-mails will be sent to the following:

- The domain registrant
- The technical contact
- The administrative contact
- And the following addresses:
  - administrator@<your-domain>
  - hostmaster@<your-domain>
  - postmaster@<your-domain>
  - webmaster@<your-domain>
  - admin@<your-domain>

You must accept at least one of the approvals before you can use your certificate.

Once approved, you can use the Identifier value in the configuration of your other resources, such as EC2 ELB(s) and CloudFront distributions.



# **There's more...**

While ACM makes getting SSL certificates for your application trivial, there are a few limitations to be aware of.



# EC2 instances

You'll notice in the documentation that only ELBs and CloudFront are supported by ACM certificates. You cannot put an ACM certificate directly on an EC2 instance.

While this is a limitation, in practice it's not a big issue. Generally, you wouldn't want to expose your instances to the Internet directly—they should be behind an ELB/ALB for security, performance, and management reasons. If you are serving static assets, CloudFront is going to be much more secure, and performs better at a lower cost.



# Importing certificates

You can import your own certificates in to ACM, so that they can be used with your ELB(s) and CloudFront. This might be done because you have already purchased a certificate from a third-party provider, or require a particular signing authority.

Resources:

MyCertificate: Type: "AWS::CertificateManager::Certificate"

Properties: DomainName: <your-domain>



# Active Directory as a service

This recipe will show you how to deploy an AWS **Simple Active Directory (Simple AD)** service.

Simple AD is powered by Samba 4 and is a Microsoft Active Directory compatible managed service. It will work with many applications that require Active Directory support and provides a large range of the commonly used Active Directory features, including the following:

- User accounts
- Single sign-on (Kerberos)
- Group memberships
- Domain joining

It also integrates with other services provided by AWS, such as the following:

- AWS Management Console
- WorkMail
- WorkDocs
- WorkSpaces and WorkSpaces Application Manager

AWS manages backup and restoration of the directory for you in the form of daily snapshots and the ability to perform point-in-time recovery.

Features that aren't supported include the following:

- Trust relationships with other AD domains
- DNS dynamic updates
- Schema extensions
- MFA
- LDAPS
- PowerShell AD cmdlets
- Transfer of FSMO roles

The ideal scenario for Simple AD usage is when you don't require advanced AD features and you're supporting less than 5,000 users. If either of these isn't true, you will want to look at AWS' fully fledged Microsoft Active Directory service. Brace yourself for some added complexity and much higher cost if you choose this path, however.



# Getting ready

Before going ahead, we'll need the following pieces of info:

- The FQDN for your directory (for example, <http://megacorp.com/>).
- A password for administering your directory. This password corresponds to the `Administrator` user that will be created on your behalf. Note that the password needs to be between 8-64 characters and will also need to contain one character from three of the following four groups:
  - Lowercase letters
  - Uppercase letters
  - Numbers
  - Non-alphanumeric characters
- The ID of the VPC we're deploying to.
- The IDs of two subnets in this VPC. These subnets need to be in different Availability Zones.
- The size of the directory you'd like to deploy. You can choose between *Small* and *Large*.

A domain controller is going to be deployed in each of the two subnets you've chosen. They'll be communicating between each other on a fairly large number of ports. Ideally, these subnets would exist in the same *tier* in your VPC and by extension would not have any NACLs which would stop the controllers from talking with each other.



*If, for some reason, you're restricting traffic using NACLs within your VPC tiers, you will want to refer to the AWS docs for a list of which ports to allow.*

*For more details, visit [http://docs.aws.amazon.com/directoryservice/latest/admin-guide/prereq\\_simple.html](http://docs.aws.amazon.com/directoryservice/latest/admin-guide/prereq_simple.html).*

AWSTemplateFormatVersion: '2010-09-09'

Parameters:

FullyQualifiedName:

Description: The fully qualified name for the directory (e.g. megacorp.com)

Type: String

AllowedPattern: '^([a-zA-Z0-9]+[\\.-])+([a-zA-Z0-9])+\$'

Password:

Description: The password for the directory Administrator Type: String

NoEcho: true

VpcId:

Description: The ID of the VPC to deploy to Type: AWS::EC2::VPC::Id SubnetIds:

Description: Subnets where the directory will be deployed to (pick at least 2)

Type: List<AWS::EC2::Subnet::Id> DirectorySize:

Description: The size of the directory to deploy Type: String

AllowedValues:

- Small

- Large

Resources:

ExampleDirectory:

Type: AWS::DirectoryService::SimpleAD

Properties:

Name: !Ref FullyQualifiedName Password: !Ref Password Size: !Ref DirectorySize

VpcSettings:

SubnetIds:

- !Select [ 0, Ref: SubnetIds ]

- !Select [ 1, Ref: SubnetIds ]

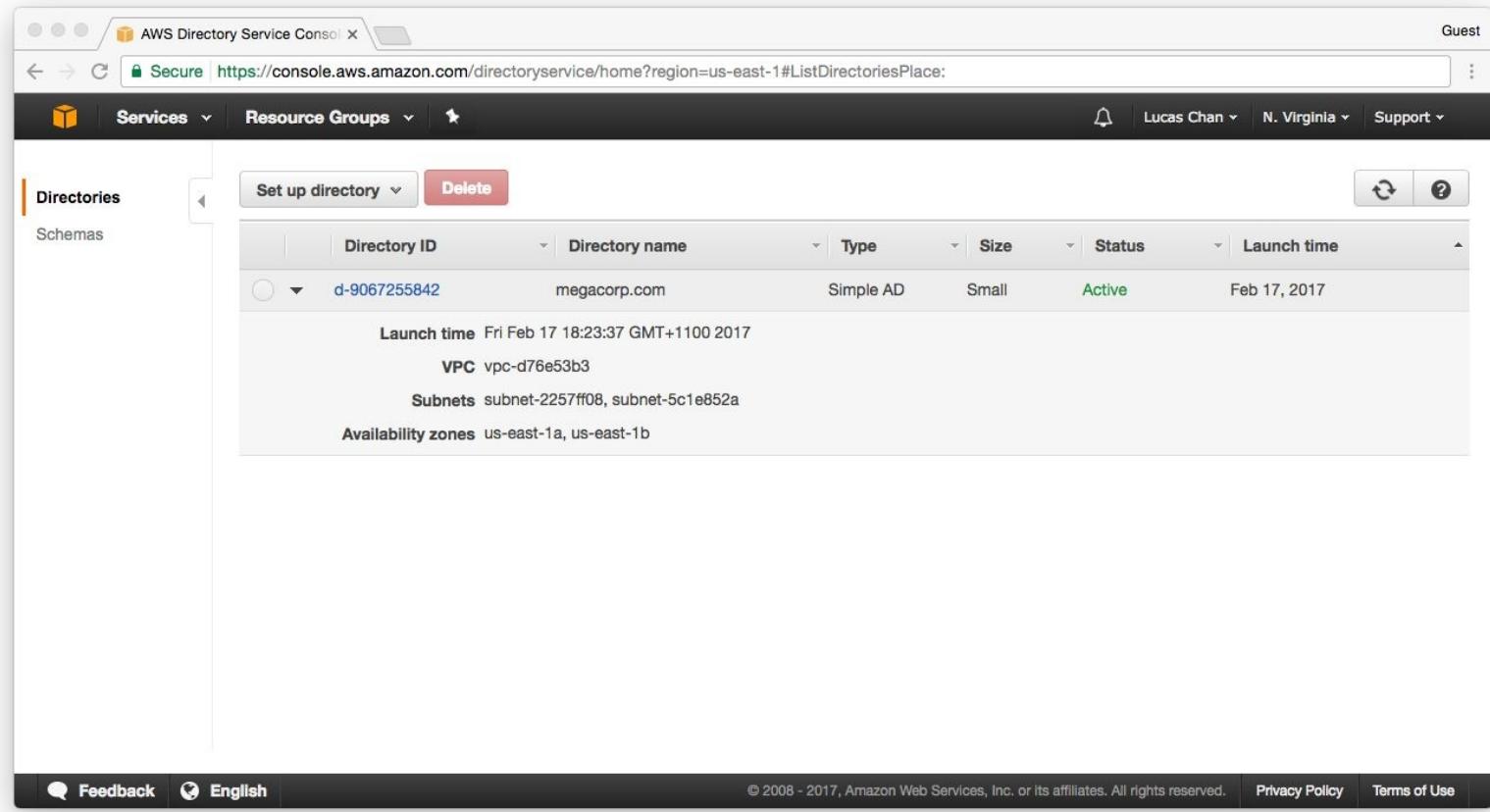
VpcId: !Ref VpcId

```
<strong> aws cloudformation create-stack \ </strong> <strong> --stack-name example-directory \ </strong> <strong> --template-body file://08-active-directory-as-a-service.yaml \ </strong> <strong> --parameters \ </strong> <strong> ParameterKey=FullyQualifiedName,ParameterValue=<fqdn> \ </strong> <strong> ParameterKey=Password,ParameterValue=<password> \ </strong> <strong> ParameterKey=VpcId,ParameterValue=<vpd-id> \ </strong> <strong> "ParameterKey=SubnetIds,ParameterValue='<subnet-1>,<subnet-2>'" \ </strong> <strong> ParameterKey=DirectorySize,ParameterValue=<Small/Large></strong>
```



# How it works...

It will take several minutes to create the directory. Once the Status becomes Active, you may proceed with further setup and integration tasks. Your directory listing page will eventually show a directory listing that looks similar to this:



The screenshot shows the AWS Directory Service Console interface. At the top, there's a navigation bar with tabs for Services (selected), Resource Groups, and a search bar. On the left, a sidebar has 'Directories' selected. The main area displays a table of directories. One row is highlighted for 'megacorp.com'. Below the table, detailed information for this directory is shown:

Directory ID	Directory name	Type	Size	Status	Launch time
d-9067255842	megacorp.com	Simple AD	Small	Active	Feb 17, 2017

Below the table, specific details are listed:

- Launch time: Fri Feb 17 18:23:37 GMT+1100 2017
- VPC: vpc-d76e53b3
- Subnets: subnet-2257ff08, subnet-5c1e852a
- Availability zones: us-east-1a, us-east-1b

At the bottom of the page, there are links for Feedback, English, Copyright notice (© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.), Privacy Policy, and Terms of Use.

Clicking on the directory ID will reveal more detailed information about your directory, like so:

AWS Directory Service Console Guest

Secure https://console.aws.amazon.com/directoryservice/home?region=us-east-1#DirectoryDetailsPlace:d-9067255842

Services Resource Groups

Directories > megacorp.com (d-9067255842)

Details

Directory type	Simple AD	Status	Active
Directory ID	d-9067255842	Status last updated	Fri Feb 17 18:27:04 GMT+1100 2017
Directory name	megacorp.com	Launch time	Fri Feb 17 18:23:37 GMT+1100 2017
NetBIOS name	megacorp	Availability zones	us-east-1a, us-east-1b
Description	(edit)	VPC	vpc-d76e53b3
DNS Address	172.30.0.18, 172.30.1.17	Subnets	subnet-2257ff08, subnet-5c1e852a
Directory size	Small		

Apps & services Snapshots Monitoring

Access URL

Create a public endpoint to access your AWS applications and the AWS Management Console. To provide access to users in your directory to AWS applications and the AWS Management Console, you must create an Access URL. Once you create an Access URL for this directory, it cannot be changed. [Learn more](#).



# There's more...

- The password for the `Administrator` account can't be retrieved or reset. Be sure to keep this password somewhere safe.
- You may notice an additional security group appear in your EC2 console. This group is necessary for the directory controllers (although you won't see these appear as EC2 instances in your console).
- The directory will contain an account with the prefix `AWSAdminD-`. This account is necessary for AWS to perform maintenance tasks such as backup and FSMO role transfers. Removing this account or changing its password is almost certainly a bad idea.



## See also

- The *Building a secure network* recipe in [Chapter 7, Networking](#).



# Creating users

Before we introduce this recipe, we need to talk briefly about **Identity and Access Management (IAM)**. It's free and is enabled on every account. It allows you to create groups and users and allows you to control exactly what they can and can't do using policy assignment.

By default, groups and users will have no permissions until you assign them either an *AWS Managed Policy* or a *Customer Managed Policy* (one which you manage). You'll want to use AWS Managed Policies as much as possible to avoid having to create and maintain your own.



*There's a third kind of policy called an **Inline Policy**. Use this sparingly. In fact, the only time we typically see it is in CloudFormation templates.*

You pretty much never want to assign a policy directly to a user. If you go down this path, you'll create a lot of work for yourself in the future. Instead, you want to apply policies to groups and then assign users to those groups. Fortunately, it's a pretty easy process and we're about to walk you through it.

The IAM dashboard provides a URL that your IAM users can use to log in to the web console (if you've assigned them a password and given them access to do so). You can also customize this *IAM sign-in link* if necessary. Don't forget to give this URL to any IAM users you create so they know where to go to sign in.

It will look something like this until you customize it:

`https://<account-id>.signin.aws.amazon.com/console`

Now, jump right in. There's no excuse for not using IAM. Start today!



# Getting ready

All you need to proceed is the CLI tools installed with a profile which can call the AWS IAM API. If you don't have this, you can follow along with the recipe steps using the AWS web console instead as the process is the same.

```
<strong> aws iam create-group --group-name <group-name></strong>
{
    "Group": {
        "Path": "/",
        "GroupId": "AGPAIHM2XJ2ELQTNYBFQQ", "Arn": "arn:aws:iam::067180688831:group/PowerUsers", "GroupName": "PowerUsers"
    }
}

<strong> aws iam attach-group-policy </strong> <strong> --group-name <group-name> </strong> <strong> --policy-arn <policy-arn></strong>
<strong> aws iam list-policies</strong>
<strong> arn:aws:iam::aws:policy/PowerUserAccess</strong>
<strong> aws iam create-user --user-name <new-username></strong>
{
    "User": {
        "UserName": "lucille.bluth", "Path": "/",
        "CreateDate": "2017-02-19T06:16:50.558Z", "UserId": "AIDAIU5P6ESCGYTVGACFE", "Arn": "arn:aws:iam::07180688831:user/lucille.bluth"
    }
}

<strong> aws iam create-login-profile --user-name <username> </strong> <strong> --password <password> </strong> <strong> --password-reset-required</strong>
{
```

"LoginProfile": {

    "UserName": "lucille.bluth", "CreateDate": "2017-02-19T06:29:06.244Z",

    "PasswordResetRequired": true }

}

<strong> aws iam create-access-key --user-name <username></strong>

{

    "AccessKey": {

        "UserName": "lucille.bluth", "Status": "Active",

        "CreateDate": "2017-02-19T06:59:45.273Z", "SecretAccessKey":

        "abcdefghijklmnopqrstuvwxyz", "AccessKeyId": "AAAAAAAAAAAAAAA"AAA

}

}

<strong> aws iam add-user-to-group \</strong> <strong> --group-name <group-name>\</strong> <strong> --user-name <username></strong>

<strong> aws iam list-groups-for-user --user-name <username></strong>

{

    "Groups": [

{

        "Path": "/",

        "CreateDate": "2017-02-19T07:24:46Z", "GroupId": "AGPAIHM2XJ2ELQTNYBFQQ",

        "Arn": "arn:aws:iam::067180688831:group/PowerUsers", "GroupName": "PowerUsers"

}

]

}



# There's more...

This pretty much covers the basics of how to create IAM groups and users and assign policies to them. Here are some of the IAM tips and gotchas we've run into over the years:

- Users can exist in more than one group. Use this to your advantage.
- Groups, however, cannot exist within other groups.
- Users can have more than one set of API keys. This is necessary when they need to perform key rotation.
- You can (and should) define a strong password policy for your IAM users.
- The `PowerUserAccess` policy is good but does not allow IAM access. At first this might not seem to be a problem; however, if you are bound by this policy you will encounter issues when running CloudFormation stacks that create IAM roles for EC2 instances, for example.
- IAM is a global service, meaning that users and groups are global, not region-specific. By default, a user can use AWS services in any region.
- EC2 key pairs are region-specific and not specific to an IAM user. In other words, IAM users don't have SSH keys associated with them.
- Your IAM username and password (and access keys) won't provide you with SSH or RDP access to running instances. Credentials for these services are managed separately.
- You can assign up to 10 policies to a group or user.
- You should also consider enabling MFA on IAM user accounts for added security. This is used primarily for accessing the web console but you can also configure your policies so that MFA will be required for API calls too. You can choose between hardware and software tokens. A good rule of thumb is to use software tokens for IAM users and hardware tokens for root logins. MFA via SMS is due to arrive soon and is currently in public preview.



## See also

- The *Federating with your AWS account* recipe
- The *Cross-account user roles* recipe



# Creating instance roles

This recipe is reasonably short but it contains a really important concept to anyone who is new to the AWS platform. Understanding and utilizing IAM roles for EC2 will significantly reduce your exposure to lost credentials and probably help you sleep a little better at night too. In a nutshell, instance roles help you get AWS credentials off your servers and out of your code base(s).

Roles contain one or more policies. We're going to create a role that has some AWS Managed Policies as well as an Inline Policy. As the name would suggest, an AWS Managed Policy is a policy that is created and fully controlled by AWS. The Inline Policy is going to be created by us and will be embedded in our role definition.

The AWS Managed Policies we'll use will allow read-only access to the S3 and EC2 APIs. The Inline Policy we'll create will allow write access to CloudWatch logs. We'll talk through why you would or wouldn't choose a Managed Policy later in this recipe.

AWSTemplateFormatVersion: '2010-09-09'

Resources:

ExampleRole:

Type: AWS::IAM::Role Properties:

AssumeRolePolicyDocument: Version: "2012-10-17"

Statement:

-

Effect: Allow

Principal:

Service:

- ec2.amazonaws.com Action:

- sts:AssumeRole ManagedPolicyArns: -

arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess -  
arn:aws:iam::aws:policy/AmazonEC2ReadOnlyAccess Path: /

Policies:

-

PolicyName: WriteToCloudWatchLogs PolicyDocument: Version: "2012-10-17"

Statement:

-

Effect: Allow

Action:

- logs>CreateLogGroup - logs>CreateLogStream - logs>PutLogEvents -  
logs>DescribeLogStreams Resource: "\*"

ExampleInstanceProfile:

Type: AWS::IAM::InstanceProfile Properties:

Roles:

- !Ref ExampleRole Path: /

Outputs:

ExampleInstanceProfile: Value: !Ref ExampleInstanceProfile

ExampleInstanceProfileArn: Value: !GetAtt ExampleInstanceProfile.Arn

<strong> aws cloudformation create-stack </strong> <strong> --stack-name example-instance-profile </strong> <strong> --template-body file://08-creating-instance-roles.yaml </strong> <strong> --capabilities CAPABILITY\_IAM</strong>

This role can now be assigned to your EC2 instances. The *Feeding log files in to CloudWatch logs* recipe in [Chapter 5, Management Tools](#), shows how you can define a role and assign it to an EC2 instance at launch using CloudFormation.



# How it works...

How on earth does this solve the problem of hardcoded AWS API keys? Well, something really interesting happens when you assign a role to an EC2 instance. The metadata for that instance will return a set of short-lived API keys. You can retrieve these keys by sending an HTTP request to the metadata URL (this is a service EC2 instances can use to fetch information about themselves):

```
http://169.254.169.254/latest/meta-data/iam/security-credentials/<role name>
```

The output of a curl request to this URL will look something like this:

```
{  
    "Code" : "Success",  
    "LastUpdated" : "2017-02-17T11:14:23Z",  
    "Type" : "AWS-HMAC", "AccessKeyId" : "AAAAAAAAAAAAAAA",  
    "SecretAccessKey" : "zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz",  
    "Token" : "token",  
    "Expiration" : "2017-02-17T12:14:23Z"  
}
```

If you take the `AccessKeyId` and `SecretAccessKey` returned in the response, you can use them to query the AWS API. The policies applied to the instance based on the role assigned to it will determine exactly what API actions the instance is able to perform using these keys.

The really fun part is that you don't have to worry too much about handling these keys at all (although it's really useful to know how all this works under the hood). For example, the AWS CLI tools will automatically fetch these keys for you prior to running any CLI commands. The same goes for the AWS SDKs.

Take a scenario where your developers are building an application that needs to fetch files from S3. As long as they are using the AWS SDK to do this and the application is running on an EC2 instance that has been assigned a role containing a policy that allows files to be fetched from S3, then no credentials are required by the application whatsoever! The SDK will take care of the queries to the metadata service for you.

The AWS SDKs are available for almost every widely used language, so there's no excuse for keeping hardcoded AWS credentials in config files or source code.

You will see your instances roles listed in the IAM console under the `Roles` section:

Screenshot of the IAM Management Console showing the Roles page.

The left sidebar shows navigation links: Dashboard, Groups, Users, **Roles**, Policies, Identity providers, Account settings, Credential report, and Encryption keys. The main content area shows a table with one result:

Role Name	Creation Time
example-instance-profile-ExampleRole-1J7JS5XR8LAW	2017-02-17 22:41 UTC+1100

At the bottom, there are Feedback, English, Privacy Policy, and Terms of Use links.

Clicking on the role will reveal further details, such as the policies that have been assigned to it:

Screenshot of the IAM Management Console showing the Permissions tab for the selected role.

The left sidebar shows the same navigation links as the previous screenshot. The main content area has tabs: Permissions (selected), Trust Relationships, Access Advisor, and Revoke Sessions. The Permissions tab displays two sections: Managed Policies and Inline Policies.

**Managed Policies:** Shows two policies attached: AmazonEC2ReadOnlyAccess and AmazonS3ReadOnlyAccess. Each policy has Show Policy, Detach Policy, and Simulate Policy options.

Policy Name	Actions
AmazonEC2ReadOnlyAccess	Show Policy   Detach Policy   Simulate Policy
AmazonS3ReadOnlyAccess	Show Policy   Detach Policy   Simulate Policy

**Inline Policies:** Shows one inline policy: WriteToCloudWatchLogs. It has Show Policy, Edit Policy, Remove Policy, and Simulate Policy options.

Policy Name	Actions
WriteToCloudWatchLogs	Show Policy   Edit Policy   Remove Policy   Simulate Policy

At the bottom, there are Feedback, English, Privacy Policy, and Terms of Use links.



# There's more...

- IAM is a global service. This means that the roles and policies you create will be available in every region.
- You'll find all the available AWS Managed Policies in the AWS web console. There's quite a few of them so don't be afraid to use the search bar.
- There's a third kind of policy called a Customer Managed Policy. These are policies which are managed by you and will appear in the AWS console amongst the AWS Managed Policies.
- As of February 2017, it is possible to attach an IAM role to an existing/running EC2 instance. This previously wasn't the case and the role could only be assigned at the time the instance launched.
- AWS automatically and periodically rotates the credentials returned by the metadata service.
- It's not always appropriate to use an AWS Managed Policy. For example, if a server needs to write to CloudWatch logs, it may be tempting to assign it the AWS Managed Policy that provides full access. If you do this, however, you'll also be giving the server access to delete log groups and streams. This is almost certainly undesirable. You'll want to inspect the policies before you apply them and defer to an Inline or Customer Managed Policy where appropriate. The principle of least privilege applies here.



## See also

- The *Feeding log files in to CloudWatch logs* recipe in [Chapter 5, Management Tools](#)



# Cross-account user roles

Using multiple accounts to provision your resources (for example, development and production environments) provides a form of *blast radius* protection—even in a worst-case scenario, any issues or damages are limited to the account they occur in, not your entire AWS presence.

Creating and assuming roles across accounts is the best way to manage access to multiple accounts. Specific roles provide a clear and explicit declaration of permissions that can be easily reviewed, and revoked if needed.

This recipe provides a way to scale your access across many accounts, without compromising your security.



# Getting ready

This recipe assumes you already have two AWS accounts created and ready to go.

In one account (the **source** account, referred to as *Account A*) you will need an IAM user.



*While you will need to use your account's root credentials to set up the first role in an account, do not use them on a day-to-day basis. The root account has permissions to do anything in your account, and should only be used when necessary.*

AWSTemplateFormatVersion: "2010-09-09"

Description: This template creates a role that can be assumed from another account.

Parameters:

SourceAccountNumber: Type: String

Description: The AWS account number to grant access to assume the role.

AllowedPattern: "[0-9]+"

MaxLength: "12"

MinLength: "12"

Resources:

CrossAccountRole: Type: "AWS::IAM::Role"

Properties:

Path: "/"

AssumeRolePolicyDocument: Version: "2012-10-17"

Statement:

- Sid: ""

Action: "sts:AssumeRole"

Effect: Allow Principal:

AWS:

!Sub "arn:aws:iam::\${SourceAccountNumber}:root"

Policies:

- PolicyName: DoEverything PolicyDocument: Version: "2012-10-17"

Statement:

- Action:

- "\*"

Effect: Allow Resource: "\*"

Sid: DoEverything

Outputs:

RoleARN:

Description: The Role ARN that can be assumed by the other account.

Value: !GetAtt CrossAccountRole.Arn

<strong> aws cloudformation create-stack \

--stack-name CrossAccountRole </strong> <strong> --template-body file://src/08-target-account-role.yaml </strong> <strong> --parameters \

ParameterKey=SourceAccountNumber, \

ParameterValue=<your-source-a</strong><strong>ccount-number> </strong><strong> --capabilities CAPABILITY\_IAM</strong>

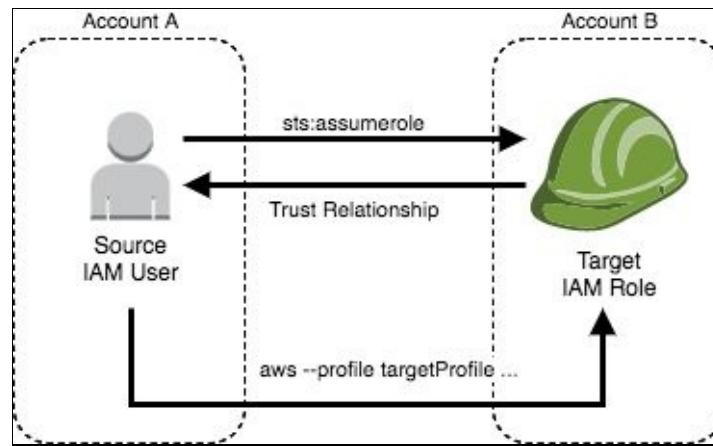
<strong> aws cloudformation describe-stacks </strong> <strong> --stack-name CrossAccountRole </strong> <strong> --query 'Stacks[0].Outputs[0].OutputValue' </strong> <strong> --output text</strong>

<strong> aws sts assume-role </strong> <strong> --role-arn <your-target-role-arn> </strong> <strong> --role-session-name CrossAccountRole</strong>



# How it works...

While cross-account roles are extremely useful for administering multiple AWS accounts, they're not the most intuitive thing to configure. Here's a diagram that illustrates the resources and their interactions:



The first few steps of this recipe are simply creating the **Target IAM Role** in a clear and repeatable way using CloudFormation.

You must explicitly call out the AWS account number that will be allowed to assume this role. If you want to allow multiple accounts to assume the role, simply add more statements to the `AssumeRolePolicyDocument` property of the role.

The sample policy created in this template gives full access to the target account (because the `Action` and `Resource` are both set to `*`). You should adjust this as appropriate for your needs.

Defining an output value that returns the IAM role's ARN will make it easier to get the generated ID later in the recipe.

We then launch the template in the target account. As this template creates IAM resources, you must supply the `--capabilities CAPABILITY_IAM` argument. If you don't have any existing IAM users that can launch it, use the AWS web console (after logging in with your root credentials). This means you don't need to bother creating IAM users in the target account.

Once you have deployed the template, you will no longer need to log in to the account manually—you can just assume the newly created role from the trusted (source) account. Using an IAM role in the target account means that your day-to-day access does not require multiple passwords, which takes work to manage and store securely. You only need to have one password—the password of your source IAM user.

After the stack has finished creating (which shouldn't take long, as it's only creating one resource), you can quickly extract the target role's ARN with a `describe-stacks` call, combined with a specifically-crafted `--query` argument. The JMESPath query `stacks[0].outputs[0].outputValue` gets the `OutputValue` property of the first output in the first stack returned, which we know will be the target role ARN because there is only one output in the template.

Finally, the sample `assume-role` command will return the credentials for the target role (that is, `ACCESS_KEY_ID`

and `SECRET_ACCESS_KEY`). You can then use this in an API call, via the CLI tool or one of the SDKs. Keep in mind that these tokens will be short-lived.

See the next section for a more convenient way to use the credentials with the CLI tool by creating profiles.



# **There's more...**

Just as there are multiple ways to use roles, there are multiple ways to utilize cross-account roles.



# AWS CLI profiles

One of the easiest ways to use a cross-account role is configuring it as a profile for the AWS CLI tool to use. This means you can quickly and easily switch accounts just by changing the profile you use when executing your commands.

To do this, you must define the target role in the CLI configuration file. With this configuration, it is assumed that your `default` profile is in the source account (that is, Account A).

Add the following snippet to the `~/.aws/config` file on Linux and Mac computers, and `c:\users\[USERNAME]\.aws\config` file on Windows: [profile accountb] role\_arn = <your-target-account-role-arn> source\_profile = default

To use switch roles, all you need to do is pass the `--profile` argument along with your command:

```
| aws --profile accountb ...
```



## See also

- The *Creating users* recipe.



# Storing secrets

A common mistake new administrators make when getting started with Infrastructure-as-Code is committing secrets (passwords, access keys, and so on) in their repositories. While this makes their infrastructure repeatable, it also makes it much more likely their credentials will be compromised. Once something is in version control, it's hard and annoying to remove it (that's the point of version control!). Even if you do remove it, it's almost impossible to know if it has already been viewed/copied by someone unintended.

In this recipe, we will introduce and use the open source tool, **Unicreds**.



*Unicreds is a Golang port of the Python tool, Credstash: <https://github.com/fugue/credstash>. While the functionality is very similar, Unicreds has the benefit of being cross-platform and dependency-free!*

Since this pattern is completely backed by AWS services, it removes the need to manage (and worry about) password vaults, shared passwords, and committing sensitive information to SCM.

You might even use Unicreds to store non-secret information, because it provides a convenient way to store and share settings without the need to run or maintain any servers!



# Getting ready

You must have Unicreds present on your target system.

As it is written in Golang, it is easily distributed as a standalone binary application—no installer or dependencies are required.



*Releases for all platforms are available at <https://github.com/Versent/unicreds/releases>.*

These commands assume your default profile has the permission to create KMS keys and DynamoDB tables. You can override the profile used by passing the `--profile` argument with all of the commands in the recipe. You must also have your AWS region setting configured.

<strong> aws kms create-key --query 'KeyMetadata.KeyId' --output text</strong>

<strong> aws kms create-alias --alias-name 'alias/credstash' \  
--target-key-id "<your-key-id>"</strong>

<strong> unicreds setup</strong>

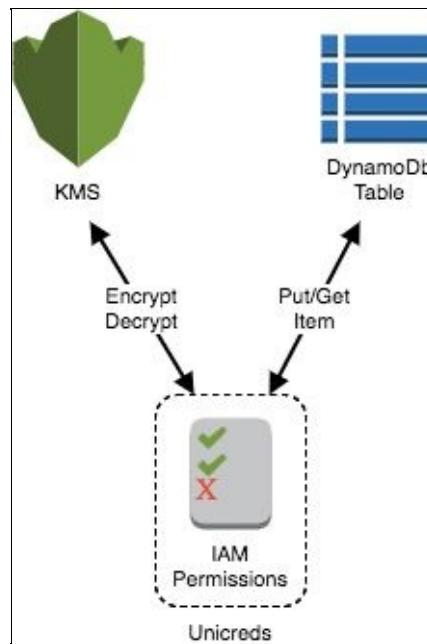
<strong> unicreds put foo bar</strong>

<strong> unicreds get foo</strong>



# How it works...

Here is a high-level diagram that illustrates the components involved in these Unicreds commands:



We start this recipe by creating the key that will be used to encrypt the secrets in KMS. Note that we never get to see this key—it only exists in KMS. All you can do is request that KMS encrypts or decrypts data with it for you.

It is possible to import your own key in to KMS (so that you could decrypt the secrets outside of AWS if you needed to), but this is not required for Unicreds to work. The `create-key` command returns the GUID for the key, which will be used in the following steps.

Aliases make it much easier to deal with KMS keys. You can use them in most commands in place of the full key ARN. More importantly, it makes it obvious which key you are dealing with so that you can quickly, easily, and confidently assign access permissions.

The default alias for the key used with Unicreds is `alias/credstash`. While this might seem a bit confusing at first, it means that Unicreds is backward-compatible with Credstash. You can choose your own alias; you will simply need to override it when you give your other commands (such as `setup`, `put`, and so on).

The `setup` command creates the required resources in your AWS account. This effectively means creating a DynamoDB table to store the secrets in.

Once everything is set up, you can start storing secrets using Unicreds. In this example, the secret is stored with the (highly original) key `foo` and the value `bar`.

At this stage, you can go to DynamoDB in the AWS console and see the stored value in the `credential-store` table. You can also change the name of the DDB table used when you run the `credstash setup` command, if you want to.

Once there's a secret stored, you can retrieve it with the `get` command. It's important to remember that there's no need to do this from the same machine you stored it from. As long as the AWS user/role has sufficient permission to use the KMS service and access the DDB table, they will be able to retrieve the secret.



# **There's more...**

Unicreds leverages the built-in functionality of AWS, so you get an enterprise-grade solution without the overhead of needing to run your own servers. Here are some other useful things you can do to make your secrets even more secure.



# Key aliases

Creating multiple KMS keys—and referring to them with unique aliases—is a great way to limit the access to put/get secrets to specific applications or teams.

Instead of using the default `alias/credstash` alias, you could give a team their own alias and be confident that they aren't going to see or write to anyone else's secrets.

```
{  
  "Version": "2012-10-17", "Statement": [  
    {  
      "Action": [  
        "kms:Decrypt"  
      ],  
      "Effect": "Allow", "Resource": "arn:aws:kms:us-east-1:<your-account-id>: key/<your-key-id>"  
    },  
    {  
      "Action": [  
        "dynamodb:GetItem", "dynamodb:Query", "dynamodb:Scan"  
      ],  
      "Effect": "Allow", "Resource": "arn:aws:dynamodb:us-east-1:<your-account-id>: table/credential-store"  
    }  
  ]  
}
```

```
{  
  "Version": "2012-10-17", "Statement": [  
    {  
      "Action": [  
        "kms:GenerateDataKey"  
      ],  
      "Effect": "Allow", "Resource": "arn:aws:kms:us-east-1:<your-account-id>: key/<your-key-id>"  
    },  
    {  
      "Action": [  
        "dynamodb:PutItem"  
      ],  
      "Effect": "Allow", "Resource": "arn:aws:dynamodb:us-east-1:<your-account-id>: table/credential-store"  
    }  
  ]  
}
```



# The put-file command

You can put entire files in to storage with Unicreds. Just use the `put-file` command:

```
| unicreds put-file foo bar.txt
```



# Versioning

While storing your secrets securely is a great start, it is still good practice to change/rotate your passwords, keys, and other secrets regularly.

Unicreds has built-in support for versioning, which means you can update your secrets while still keeping records of previous versions.

When you put to the same secret name multiple times, Unicreds will automatically create new versions for the values. You can get a specific version of secret by providing a version argument with a `get` or `put` command: **unicreds get foo 1**



## See also

- The *Creating users* recipe



# Estimating Costs

In this chapter, we will cover:

- Calculating costs
- Estimating CloudFormation template costs
- Purchasing reserved instances
- Estimating total cost of ownership



# Introduction

One of the hardest things to get used to when starting with AWS is that you pay for almost everything that you use. One of the biggest benefits of AWS is that you only pay for what you use. This makes it hard to quickly answer the *how much is it going to cost?* question that often arises when people first start using AWS; they don't know exactly what they currently use!

In a *traditional* infrastructure or data center setup, many costs are paid during the initial outlay or in annual contracts. As AWS has no upfront fees, and few long-term commitments, so the usual thought process around costs is turned on its head.

There are a number of helpful tools to get a better estimate of your AWS usage costs. Don't forget that every AWS service page has a pricing section. While some pricing models can be a bit confusing at the start, it quickly makes sense.



# Calculating costs

AWS Simple Monthly Calculator is a website application provided to help you estimate and forecast your AWS costs. By listing the resources you expect to consume you can calculate your pay-as-you-go costs, which is how AWS bills you—there's no upfront costs involved.



# Getting ready

In order to use the AWS Simple Monthly Calculator effectively, you need to already know the specific services and resources that you will use on a monthly basis.

You also need to know specifics about things such as monthly data transfer and the amount of data you will need to store. In AWS, you get charged for data in and out of AWS (for example, visitors to your website), but not between AWS services (for example, EC2 instances to RDS databases).



# How to do it...

1. Go to the calculator website, <http://calculator.s3.amazonaws.com/index.html>:

The screenshot shows the AWS Simple Monthly Calculator interface. On the left, a sidebar lists services: Amazon EC2, Amazon S3, Amazon Route 53, Amazon CloudFront, Amazon RDS, Amazon DynamoDB, Amazon ElastiCache, Amazon CloudWatch, Amazon SES, Amazon SNS, Amazon Elastic Transcoder, Amazon WorkSpaces, and Amazon WorkDocs. The main area has tabs for 'Services' and 'Estimate of your Monthly Bill (\$ 0.00)'. A 'Choose region:' dropdown is set to 'US-East / US Standard (Virginia)'. Below it, a note states: 'Inbound Data Transfer is Free and Outbound Data Transfer is 1 GB free per region per month'. A note about Amazon EC2 instances follows. There are three tables for Compute: Amazon EC2 Instances, Compute: Amazon EC2 Dedicated Hosts, and Storage: Amazon EBS Volumes. A sidebar on the right titled 'Common Customer Samples' includes: Free Website on AWS, AWS Elastic Beanstalk Default, Marketing Web Site, Large Web Application (All On-Demand), Media Application, European Web Application, and Disaster Recovery and Backup.

2. Select/deselect the free usage tier option as relevant for your account—if the account is less than 12 months old, you are eligible for the free usage tier.

The screenshot shows the same AWS Simple Monthly Calculator interface as the previous one, but with the 'FREE USAGE TIER' section highlighted by a red box. This section contains the text: 'Get Started with AWS: Learn more about our Free Tier or Sign Up for an AWS Account ». Below it is a 'Reset All' button and a 'Services' tab. The 'Estimate of your Monthly Bill (\$ 0.00)' tab is active. The 'Choose region:' dropdown is still set to 'US-East / US Standard (Virginia)'. The sidebar and other sections remain the same as in the first screenshot.

3. Make sure you have the correct region selected before adding resources, as they can differ in price from region to region:

Amazon Web Services Simple Guest

calculator.s3.amazonaws.com/index.html Language: English

**amazon webservices SIMPLE MONTHLY CALCULATOR**

Get Started with AWS: [Learn more about our Free Tier](#) or [Sign Up for an AWS Account](#)

FREE USAGE TIER: New Customers get free usage tier for first 12 months

**Services** Estimate of your Monthly Bill (\$ 0.00)

**Choose region:** US-East / US Standard (Virginia) Inbound Data Transfer is Free and Outbound Data Transfer is 1 GB free per region per month

Amazon EC2 Amazon S3 Amazon Route 53 Amazon CloudFront Amazon RDS Amazon DynamoDB Amazon ElastiCache Amazon CloudWatch Amazon SES Amazon SNS Amazon Elastic Transcoder Amazon WorkSpaces Amazon WorkDocs

Common Customer Samples

Free Website on AWS  
AWS Elastic Beanstalk Default  
Marketing Web Site  
Large Web Application (All On-Demand)  
Media Application  
European Web Application  
Disaster Recovery and Backup

Compute: Amazon EC2 Instances:

Description	Instances	Usage	Type	Billing Option	Monthly Cost
(+)	Add New Row				

Compute: Amazon EC2 Dedicated Hosts:

Description	Number of Hosts	Usage	Type	Billing Option
(+)	Add New Row			

Storage: Amazon EBS Volumes:

Description	Volumes	Volume Type	Storage	IOPS	Baseline Throughput	Snapshot Storage
(+)	Add New Row					

Elastic IP:

Number of Additional Elastic IPs: 0

4. Add your resources by selecting the relevant service from the left-hand menu, and filling in your details:

Amazon Web Services Simple Guest

calculator.s3.amazonaws.com/index.html Language: English

**amazon webservices SIMPLE MONTHLY CALCULATOR**

Get Started with AWS: [Learn more about our Free Tier](#) or [Sign Up for an AWS Account](#)

FREE USAGE TIER: New Customers get free usage tier for first 12 months

**Services** Estimate of your Monthly Bill (\$ 720.66)

**Choose region:** US-East / US Standard (Virginia) Inbound Data Transfer is Free and Outbound Data Transfer is 1 GB free per region per month

Amazon EC2 Amazon S3 Amazon Route 53 Amazon CloudFront Amazon RDS Amazon DynamoDB Amazon ElastiCache Amazon CloudWatch Amazon SES Amazon SNS Amazon Elastic Transcoder Amazon WorkSpaces Amazon WorkDocs

Common Customer Samples

Free Website on AWS  
AWS Elastic Beanstalk Default  
Marketing Web Site  
Large Web Application (All On-Demand)  
Media Application  
European Web Application  
Disaster Recovery and Backup

Amazon RDS is a web service that makes it easier to set up, operate, and scale a relational database in the cloud. Cost calculation for Amazon Aurora is coming soon. Please check the pricing page for [Amazon Aurora pricing](#) details.

Clear Form

**Amazon RDS On-Demand DB Instances:**

Description	DB Instances	Usage	DB Engine and License	Class and Deployment	Storage	I/O
(-)	Store	1	100 % Utilized/H	MySQL	db.t2.small	General P Provisioned IOPS: 0
(+)	Add New Row					

**Additional Backup Storage (Free backup storage up to 100% of provisioned Storage):**

Backup Type	Backup Storage		
(+)	Add New Row		

**Amazon RDS Reserved DB Instances:**

Description	DB Instances	Usage	DB Engine and License	Class and Deployment	Offering and Term	Storage	I/O
(+)	Add New Row						

Data Transfer:

5. Continue to add resources as necessary:

Amazon Web Services Simple Guest

calculator.s3.amazonaws.com/index.html

**amazon webservices SIMPLE MONTHLY CALCULATOR**

Need Help? [Watch the Videos](#) or [Read How AWS Pricing Works](#)

Get Started with AWS: [Learn more about our Free Tier](#) or [Sign Up for an AWS Account](#)

FREE USAGE TIER: New Customers get free usage tier for first 12 months

**Services** Estimate of your Monthly Bill (\$ 720.66)

Choose region: US-East / US Standard (Virginia) Inbound Data Transfer is Free and Outbound Data Transfer is 1 GB free per region per month

Amazon EC2 Amazon S3 Amazon Route 53 Amazon CloudFront Amazon RDS Amazon DynamoDB Amazon ElastiCache Amazon CloudWatch Amazon SES Amazon SNS Amazon Elastic Transcoder Amazon WorkSpaces Amazon WorkDocs

Common Customer Samples

Free Website on AWS  
AWS Elastic Beanstalk Default  
Marketing Web Site  
Large Web Application (All On-Demand)  
Media Application  
European Web Application  
Disaster Recovery and Backup

Reset All

**Compute: Amazon EC2 Instances:**

Description	Instances	Usage	Type	Billing Option	Monthly Cost
Web server	2	50 % Utilized/t	Linux on t2.large	On-Demand (No Cor.	\$ 68.82
<a href="#">Add New Row</a>					

**Compute: Amazon EC2 Dedicated Hosts:**

Description	Number of Hosts	Usage	Type	Billing Option
<a href="#">Add New Row</a>				

**Storage: Amazon EBS Volumes:**

Description	Volumes	Volume Type	Storage	IOPS	Baseline Throughput	Snapshot Storage
Data	1	General Purpose SSD (gp	5 GB	100	128 MBs/sec	0 GB-month of Storage
<a href="#">Add New Row</a>						

**Elastic IP:**

Number of Additional Elastic IPs: 0

6. Once you've added all your resources, view the estimated monthly bill on the tab:

Amazon Web Services Simple Guest

calculator.s3.amazonaws.com/index.html

**amazon webservices SIMPLE MONTHLY CALCULATOR**

Need Help? [Watch the Videos](#) or [Read How AWS Pricing Works](#)

Get Started with AWS: [Learn more about our Free Tier](#) or [Sign Up for an AWS Account](#)

FREE USAGE TIER: New Customers get free usage tier for first 12 months

**Services** Estimate of your Monthly Bill (\$ 720.66)

**Estimate of Your Monthly Bill**

Show First Month's Bill (include all one-time fees, if any)

Below you will see an estimate of your monthly bill. Expand each line item to see cost breakout of each service. To save this bill and input values, click on 'Save and Share' button. To remove the service from the estimate, jump back to the service and clear the specific service's form.

[Save and Share](#)

Amazon EC2 Service (US-East) \$ 69.32  
 Amazon S3 Service (US-East) \$ 0.05  
 Amazon Route 53 Service \$ 0.90  
 Amazon CloudFront Service \$ 600.26  
 Amazon RDS Service (US-East) \$ 50.94  
 AWS Support (Basic) \$ 0.00  
 Free Tier Discount: \$ -0.81  
 Total Monthly Payment: \$ 720.66

Common Customer Samples

Free Website on AWS  
AWS Elastic Beanstalk Default  
Marketing Web Site  
Large Web Application (All On-Demand)  
Media Application  
European Web Application  
Disaster Recovery and Backup

7. After confirming the estimate's detail, click on the Save and Share button to add some additional metadata about your report. All the fields are optional:

Amazon Web Services Simple Guest

calculator.s3.amazonaws.com/index.html

**amazon web services SIMPLE MONTHLY CALCULATOR**

Get Started with AWS: [Learn more about our Free Tier or Sign Up for an AWS Account »](#)

FREE USAGE TIER: New Customers get free usage tier for first 12 months

Services

Estimate of your Monthly Bill (\$ 720.66)

**Save and Share this Configuration**

Name: Website Infrastructure  
Includes: Highly-available web servers and databases. Hosted DNS. CDN with static asset hosting.  
Description: Infrastructure to run a small web application in a highly-available configuration.

All fields are Optional.

Save your bill and get a URL to share your calculations with others, OK?

Cancel OK

Common Customer Samples

Free Website on AWS

AWS Elastic Beanstalk Default

	\$	69.32
Marketing Web Site	\$	0.05
Large Web Application (All On-Demand)	\$	0.90
Media Application	\$	600.26
European Web Application	\$	50.94
Disaster Recovery and Backup	\$	0.00
	\$	-0.81
	\$	720.66

es, click on 'Save and Share'

Save and Share

Total Monthly Payment:

## 8. A specific, one-time URL will be generated for your report that you can then share with others:

Amazon Web Services Simple Guest

calculator.s3.amazonaws.com/index.html

**amazon web services SIMPLE MONTHLY CALCULATOR**

Get Started with AWS: [Learn more about our Free Tier or Sign Up for an AWS Account »](#)

FREE USAGE TIER: New Customers get free usage tier for first 12 months

Services

**Save and Share**

Your calculations have been saved.

You can use the following URL to retrieve your calculations or share it with the world.

<http://calculator.s3.amazonaws.com/index.html#r=IAD&key=calc-861445C0-D6C8-42A7-9BA3-B1F5E47E7A17>

Thank you for using the AWS Simple Monthly Calculator

OK

Common Customer Samples

Free Website on AWS

AWS Elastic Beanstalk Default

	\$	69.32
Marketing Web Site	\$	0.05
Large Web Application (All On-Demand)	\$	0.90
Media Application	\$	600.26
European Web Application	\$	50.94
Disaster Recovery and Backup	\$	0.00
	\$	-0.81
	\$	720.66

click on 'Save and Share'

Save and Share

Total Monthly Payment:



# How it works...

The accuracy of the calculator is completely dependent on your ability to forecast your requirements and usage—not an easy thing to do when you first start using AWS!

Unfortunately, not all AWS services are present in the calculator (a notable exception is AWS Lambda). For those services, you will have to do your own calculations based on the service-specific pricing pages.



*The cost of services and resources can vary from region to region. In general, the us-east-1 region is the cheapest, and also has the most services (not all services are available in all regions), so use that if you want to know the lowest-cost option. Other regions' prices vary due to supply and demand, cost of operations, and undoubtedly many other reasons that AWS doesn't go in to.*

Some services (for example, DynamoDB, Lambda, and so on) have a free tier that applies even if your account does not qualify for the *standard* free tier (that is, the account is more than 12 months old). These services will have a note on their specific calculator page detailing the inclusions:

The screenshot shows a web browser window for 'calculator.s3.amazonaws.com/index.html'. The title bar says 'Amazon Web Services Simple Monthly Calculator'. The main content area features the 'amazon web services' logo and the title 'SIMPLE MONTHLY CALCULATOR'. A sidebar on the left lists services: 'Amazon EC2', 'Amazon S3', and 'Amazon Route 53'. The main form has a green header 'Services' and 'Estimate of your Monthly Bill (\$ 720.66)'. A dropdown menu 'Choose region:' is set to 'US-East / US Standard (Virginia)'. Below it, a paragraph describes Amazon DynamoDB. A red box highlights a yellow box containing text about the 'FREE TIER' for DynamoDB users. The URL in the address bar is 'calculator.s3.amazonaws.com/index.html'.

Once completed, you can generate a specific URL for your estimation report that you can share with anyone. There's no authentication to access this URL, so don't put any sensitive information in your reports. The only protection is that the URL is unlikely to be guessed (given it's just the calculator site with a GUID parameter).



## See also

- The *Estimating CloudFormation template costs* recipe



# **Estimating CloudFormation template costs**

Most of the recipes in this book have been managed and launched using CloudFormation, the AWS Infrastructure as Code service.



# Getting ready

For this recipe, you will need an existing CloudFormation template. The template does not need to be deployed as a stack; just the file is required.

In this example, we will use the template from [Chapter 4, Using AWS Compute](#), to securely access private instances: `06-create-database-with-automatic-failover.yaml`.

```

<strong> aws cloudformation estimate-template-cost </strong> <strong> --template-body
\

    file://06-create-database-with-automatic-failover.yaml </strong> <strong> --
parameters ParameterKey=VPCId,ParameterValue=test </strong> <strong>
ParameterKey=SubnetIds,ParameterValue="test,test" </strong> <strong>
ParameterKey=DBUsername,ParameterValue=test </strong> <strong>
ParameterKey=DBPassword,ParameterValue=test </strong> <strong> --query Url \
</strong> <strong> --output text</strong>

```

- Click or copy and paste the URL into a browser to see the report:

The screenshot shows a web browser window titled "Amazon Web Services Simple Monthly Calculator". The URL in the address bar is "calculator.s3.amazonaws.com/index.html?key=cloudformation/9df3069b-9ef7-46f0-8053-f29ff7b2e771". The page displays usage information for the Amazon EC2 service, including a table for Compute: Amazon EC2 Instances and another for Compute: Amazon EC2 Dedicated Hosts. A sidebar on the left lists various AWS services, and a sidebar on the right shows common customer samples.

Description	Instances	Usage	Type	Billing Option	Monthly Cost
(empty)	(empty)	(empty)	(empty)	(empty)	(empty)
<a href="#">Add New Row</a>					

Description	Number of Hosts	Usage	Type	Billing Option
(empty)	(empty)	(empty)	(empty)	(empty)
<a href="#">Add New Row</a>				

Description	Volumes	Volume Type	Storage	IOPS	Baseline Throughput	Snapshot Storage
(empty)	(empty)	(empty)	(empty)	(empty)	(empty)	(empty)
<a href="#">Add New Row</a>						

**Elastic IP:**  
Number of Additional Elastic IPs:

- Click on Amazon RDS in the left-hand menu to see the individual service page details:

Amazon Web Services Simple X Guest

calculator.s3.amazonaws.com/index.html?key=cloudformation/9df3069b-9ef7-46f0-8053-f29ff7b2e771 Language: English

**amazon web services SIMPLE MONTHLY CALCULATOR**

Need Help? [Watch the Videos](#) or [Read How AWS Pricing Works](#)

Get Started with AWS: [Learn more about our Free Tier](#) or [Sign Up for an AWS Account](#) »

FREE USAGE TIER: New Customers get free usage tier for first 12 months

**Services** Estimate of your Monthly Bill (\$ 26.89)

Choose region: US-East / US Standard (Virginia) Inbound Data Transfer is Free and Outbound Data Transfer is 1 GB free per region per month

Amazon RDS is a web service that makes it easier to set up, operate, and scale a relational database in the cloud. Cost calculation for Amazon Aurora is coming soon. Please check the pricing page for [Amazon Aurora pricing](#) details. Clear Form

**Amazon RDS On-Demand DB Instances:**

Description	DB Instances	Usage	DB Engine and License	Class and Deployment	Storage	I/O
ExampleDBInst:	1	24 Hours/Day	MySQL	db.t2.micro	Magnetic	Provisioned IOPS: 0
<a href="#">Add New Row</a>						

**Additional Backup Storage (Free backup storage up to 100% of provisioned Storage):**

Backup Type	Backup Storage
<a href="#">Add New Row</a>	

**Amazon RDS Reserved DB Instances:**

Description	DB Instances	Usage	DB Engine and License	Class and Deployment	Offering and Term	Storage	I/O
<a href="#">Add New Row</a>							

**Common Customer Samples**

- Free Website on AWS
- AWS Elastic Beanstalk Default
- Marketing Web Site
- Large Web Application (All On-Demand)
- Media Application
- European Web Application
- Disaster Recovery and Backup

4. Click on Estimate of your Monthly Bill to see a total summary of the template resources:

Amazon Web Services Simple X Guest

calculator.s3.amazonaws.com/index.html?key=cloudformation/9df3069b-9ef7-46f0-8053-f29ff7b2e771 Language: English

**amazon web services SIMPLE MONTHLY CALCULATOR**

Need Help? [Watch the Videos](#) or [Read How AWS Pricing Works](#)

Get Started with AWS: [Learn more about our Free Tier](#) or [Sign Up for an AWS Account](#) »

FREE USAGE TIER: New Customers get free usage tier for first 12 months

**Services** Estimate of your Monthly Bill (\$ 26.89)

**Estimate of Your Monthly Bill**

Show First Month's Bill (include all one-time fees, if any)

Below you will see an estimate of your monthly bill. Expand each line item to see cost breakout of each service. To save this bill and input values, click on 'Save and Share' button. To remove the service from the estimate, jump back to the service and clear the specific service's form.

**Save and Share**

	\$	26.89
Amazon RDS Service (US-East)	\$	24.89
DB instances:	\$	2.00
AWS Support (Basic)	\$	0.00
<b>Total Monthly Payment:</b>	<b>\$</b>	<b>26.89</b>

**Common Customer Samples**

- Free Website on AWS
- AWS Elastic Beanstalk Default
- Marketing Web Site
- Large Web Application (All On-Demand)
- Media Application
- European Web Application
- Disaster Recovery and Backup



# How it works...

The `estimate-template-cost` command requires all the parameters of your template. As you can see in the first step, the actual values aren't important because the template won't actually be launched. You simply need to make sure the type of value you give matches the required type for that parameter (for example, the `SubnetIds` value must be a list of values in this template).



*The region you specify is important! Some services (but not all) can cost different amounts depending on the region they are in. Generally, the `us-east-1` region is the cheapest.*

At the end of the command, we limit the output to just the report URL via the `--query` argument.



*You can share the URL generated with others, but you will not be able to retrieve earlier reports unless you keep track of the URL yourself.*

On the calculator website, the template's resources will be pre-populated, even if you can't immediately see them. The report always defaults to the Amazon EC2 service page, so you will have to go to the relevant service page via the left-hand menu (in this case, Amazon RDS).

Finally, you can see a complete report of your template's monthly cost on the Estimate of your Monthly Bill tab. If your template contains many different types of resources/services, you will see them summarized here.



## See also

- The *Create a database with automatic failover* recipe in [Chapter 6, Database Services](#)
- The *Calculating costs* recipe



# Purchasing reserved instances

Reserved instances can be the cause of some confusion and are often misunderstood. Here are a few pointers to get you going down the right path:

- Reserved instances have no distinguishing technical features compared to regular on-demand instances.
- Reserved instances are not a specific type or class of instance.
- Put simply, purchasing a reserved instance entitles you to a discounted hourly rate on an on-demand instance that matches the properties of the reserved instance.
- The discounted hourly rate will be of a varying size depending on how much you pay upfront. As a general rule, the more you pay upfront, the higher the discount.

When you purchase a reserved instance, you're required to specify the following properties:

- Platform (Linux/Windows)
- Scope (Region or Availability Zone)
- Instance Type (for example, m3.large)
- Tenancy (shared or dedicated)
- Offering Class (standard or convertible)
- Term (1-12 months or 1-3 years)
- Payment Option (no upfront, partial upfront, all upfront)

We'll explore the ins and outs a little more later in this section. For now, let's dive in and see how to make a purchase.



# Getting ready

You'll need an AWS account and some idea of which instance types you wish to reserve and for how long. Refer to the reserved instance properties mentioned previously for the exact information you'll need to proceed.

The Payment Option you choose will dramatically affect the price you pay when purchasing the reservation:

- No Upfront: This means you pay nothing now but you will be charged the discounted hourly rate for the entire term whether or not you have an instance that matches the reservation. Also note that choosing this option limits you to 1 year for standard reservations and 3 years for convertible reservations (we'll discuss these later in this section).
- Partial Upfront: These reservations mean that you pay a smaller upfront fee and then you are charged a discounted hourly rate only for the instance hours you use.
- All Upfront: As the name suggests, you'll be required to pay the full cost of the instance for the entire term. An effective 100% discount is applied to the hourly rate of your matching instances for that term.

Once you know all the properties of the instance reservation, you can go ahead and make a purchase.



# **How to do it...**

1. Go to the EC2 web console, select Reserved Instances, then Purchase Reserved Instances:

EC2 Management Console | Guest

Secure | https://ap-southeast-2.console.aws.amazon.com/ec2/v2/home?region=ap-southeast-2#ReservedInstances:sort=reservedInstancesId

Services | Resource Groups | Actions | Filter by attributes or search by keyword | Lucas Chan | Sydney | Support | None found

EC2 Dashboard | Events | Tags | Reports | Limits | Instances | Instances | Spot Requests | Reserved Instances | Reserved Instances | Dedicated Hosts | Images | AMIs | Bundle Tasks | Elastic Block Store | Volumes | Snapshots | Network & Security | Security Groups | Elastic IPs | Placement Groups | Feedback | English | © 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. | Privacy Policy | Terms of Use

2. We now need to perform a search for the instance type you wish to purchase. In this example, we're going to choose the following:

- Platform: Linux/UNIX
- Tenancy: Default
- Offering Class: Standard
- Instance Type: t2.micro
- Term: 1 months-12 months
- Payment Option: All Upfront

3. Obviously, choose the options that best match your workload. You almost certainly want to choose Default as Tenancy here. Dedicated tenancy/instances are run on hardware that will be occupied by only one customer (you) and are a lot more expensive:

## Purchase Reserved Instances

Only show offerings that reserve capacity

Platform **Linux/UNIX** ▾

Tenancy **Default** ▾

Offering Class **Standard** ▾

Instance Type **t2.micro** ▾

Term **1 months - 1...** ▾

Payment Option **All Upfront** ▾

Search

To find a Reserved Instance offering, complete the following steps:

1. Specify the offering details and click **Search**.
2. Select the Reserved Instances and specify the quantity that you want, and **Add** them to your cart.
3. Click **View Cart** to view your cart before purchasing the Reserved Instances.
4. Click **Purchase** to complete your purchase.

Note: Additional taxes may apply.

Reserved Instances sold through the Reserved Instance Marketplace are identical to those sold by Amazon Web Services, except they may have different prices and terms. For more information about the Reserved Instance Marketplace, go to the [Reserved Instance Marketplace](#) web page.

4. The console will return a price for the instance reservation. Note that because we didn't select Only show offerings that reserve capacity, what we are seeing is a single result, that is a reservation which applies to the region we're currently viewing in the console. Think of this as a *region level* reservation:

EC2 Management Console Guest

Secure https://ap-southeast-2.console.aws.amazon.com/ec2/v2/home?region=ap-southeast-2#ReservedInstances:sort=reservedInstancesId

## Purchase Reserved Instances

Only show offerings that reserve capacity

Platform		Linux/UNIX	Tenancy		Default	Offering Class		Standard		
Instance Type		t2.micro	Term		1 months - 1...	Payment Option		All Upfront		
Seller	Term	Effective Rate	Upfront Price	Hourly Rate	Payment Option	Offering Class	Quantity Available	Desired Quantity		
AWS	12 months	\$0.012	\$103.00	\$0.000	All Upfront	standard	Unlimited	1	<input type="button" value="Add to Cart"/>	

- Now try selecting Only show offerings that reserve capacity and notice that all Availability Zones are showing for the current region. You can think of these as *AZ level* reservations. Choosing one of these options obviously locks you in to a specific Availability Zone; however, you also get a *capacity reservation* (discussed next):

EC2 Management Console Guest

Secure https://ap-southeast-2.console.aws.amazon.com/ec2/v2/home?region=ap-southeast-2#ReservedInstances:sort=reservedInstancesId

## Purchase Reserved Instances

Only show offerings that reserve capacity

Platform		Linux/UNIX	Availability Zone		Any	Tenancy		Default	Offering Class		Standard
Instance Type		t2.micro	Term		1 months - 1...	Payment Option		All Upfront			
Seller	Term	Effective Rate	Upfront Price	Hourly Rate	Availability Zone	Payment Option	Offering Class	Quantity Available	Desired Quantity		
AWS	12 months	\$0.012	\$103.00	\$0.000	ap-southeast-2c	All Upfront	standard	Unlimited	1	<input type="button" value="Add to Cart"/>	
AWS	12 months	\$0.012	\$103.00	\$0.000	ap-southeast-2a	All Upfront	standard	Unlimited	1	<input type="button" value="Add to Cart"/>	
AWS	12 months	\$0.012	\$103.00	\$0.000	ap-southeast-2b	All Upfront	standard	Unlimited	1	<input type="button" value="Add to Cart"/>	

- Choose the reservation that looks right for you, and then click Add to Cart and then View Cart.
- The next page shows a summary of your imminent purchase. Click Purchase to proceed. Note that this is the point of no return. Reserved instances can't be canceled. Choose wisely!

## Shopping Cart

## Items to buy now

**All Upfront Linux/UNIX t2.micro in ap-southeast-2c for 12 months**

Payment Terms:Upfront + Every Hour

Sold By:Amazon

[Delete](#)

	Upfront Price	Hourly Price	Quantity
	\$103.00	\$0.000	1

Your cart: 1 Reserved Instances, Total Due Now: **\$103.00**

Additional taxes may apply.

[Add More To Cart](#)[Purchase](#)



# How it works...

After you've completed your purchase, your reservation will be marked as Payment Pending and then soon after Active (there's a third possible status, which is Retired).

Once your reservation is Active, the discount will automatically apply to matching instances. AWS refers to this hourly discount as a *billing benefit*.

Choosing a Convertible reservation class immediately rules out anything but a 3 year term. In return, you get a little more flexibility than the Standard reservations because if you decide the reservation no longer meets your needs, you can *convert* it to a reservation that is of equal or higher value, paying the difference of course.

If you made a reservation for a specific Availability Zone, AWS also provides you with a *capacity reservation*, which will give you some guarantees around the availability of instances in that zone. This is something you might want to consider if your workload needs to maintain a certain amount of capacity in the event of an entire Availability Zone outage, for example. An event such as this tends to cause a rush of new instance requests in the unaffected zones; however, customers without a capacity reservation may find their new instance requests can't be fulfilled because of a lack of capacity (this is not unheard of), causing them to miss out or forcing them to issue new instance requests for a different zone and/or instance type while at the same time crossing their fingers.



*Unlike the billing benefit (hourly discount), which is applied immediately after purchase, a capacity reservation is used by the first instance you launch in the zone matching the properties of your reservation.*



# There's more...

- Services that launch instances on your behalf (auto scaling, Elastic Beanstalk, and so on) are also eligible to have hourly discounts applied to them.
- Standard reservations can be made for either 1 or 3 years. As mentioned before, Convertible reservations are fixed at 3 years.
- Under a consolidated billing model, reserved instances discounts are applied across all your sub accounts. For example, if you purchase a reserved instance intended for account A but there is no server matching its properties, the reservation will automatically apply to matching instances in account B. This only applies to the billing benefit and not the capacity benefit.
- Reserved instances can be sold in the AWS marketplace. This is useful if the reservation no longer suits your needs. Note that you will need a US bank account for this.
- If reserved instances don't seem to match your type of workload, you might consider a scheduled instance instead.



# **Estimating total cost of ownership**

The AWS TCO Calculator is designed to provide you with a ballpark view of how much it will cost you to run equivalent infrastructure on AWS in comparison to your co-located or on-premise data center.

The calculator has been audited by an independent third-party, but you should of course check its output against your own calculations before you make any purchasing decisions.



# Getting ready

In this example, we're going to describe a typical three tier Rails image processing application running with a modest amount of hardware. You can use our example configuration or follow along with your own hardware requirements.



# How to do it...

1. Navigate to <https://awstcoccalculator.com/>.
2. Choose your currency, location, AWS region, and workload type. In our case we're going to choose the following:
  - Australian dollar
  - Colocation
  - Asia Pacific (Sydney)
  - General

TCO Calculator Guest

Secure <https://awstcocalculator.com>

Contact Sales

**AWS Total Cost of Ownership (TCO) Calculator**

Basic ▼

Use this calculator to compare the cost of running your applications in an on-premises or colocation environment to AWS. Describe your on-premises or colocation configuration to produce a detailed cost comparison with AWS. You can switch between the basic and advanced views to provide additional configuration details.

Select Currency	Australian dollar <span style="float: right;">▼</span>						
What type of environment are you comparing against?	<input checked="" type="radio"/> On-Premises <input type="radio"/> Colocation						
Which AWS region is ideal for your geo requirements?	Asia Pacific (Sydney) <span style="float: right;">▼</span>						
Choose workload type:	General <span style="float: right;">▼</span>						
<b>Servers</b>							
Are you comparing physical servers or virtual machines?							
<input checked="" type="radio"/> Physical Servers <input type="radio"/> Virtual Machines							
Provide your configuration details:							
Server Type <i>i</i>	App. Name <i>i</i>	# of Processors/ Server <i>i</i>	# of Cores/ Processor <i>i</i>	# of Servers <i>i</i>	Memory (GB) <i>i</i>	DB Engine <i>i</i>	

#### TCO Calculator—workload

3. Now we need to describe our server requirements. We're going to specify that our app is running on physical servers with tiers that look like this:

- App Name: `nginx`
  - Server Type: Non DB
  - # of Processors/Server: 2
  - # of Cores/Processor: 2
  - # of Servers: 2
  - Memory (GB): 16
- App Name: `rails`
  - Server Type: Non DB
  - # of Processors/Server: 2
  - # of Cores/Processor: 4
  - # of Servers: 4
  - Memory (GB): 32
- App Name: `mysql`
  - Server Type: DB
  - # of Processors/Server: 2
  - # of Cores/Processor: 8
  - # of Servers: 2
  - Memory (GB): 64
  - DB Engine: MySQL

TCO Calculator Guest

Secure <https://awstcocalculator.com>

Contact Sales

**What type of environment are you comparing against?**  On-Premises  Colocation

**Which AWS region is ideal for your geo requirements?** Asia Pacific (Sydney)

**Choose workload type:** General

**Servers**

Are you comparing physical servers or virtual machines?  Physical Servers  Virtual Machines

Provide your configuration details:

Server Type <i>i</i>	App. Name <i>i</i>	# of Processors/ Server <i>i</i>	# of Cores/ Processor <i>i</i>	# of Servers <i>i</i>	Memory (GB) <i>i</i>	DB Engine <i>i</i>		
<input style="border: 1px solid #ccc; width: 100px; height: 25px; font-size: 10px; margin-right: 10px;" type="button" value="Non DB"/>	<input style="width: 150px; height: 25px; border: 1px solid #ccc;" type="text" value="nginx"/>	<input style="border: 1px solid #ccc; width: 50px; height: 25px; font-size: 10px; margin-right: 10px;" type="button" value="2"/>	<input style="border: 1px solid #ccc; width: 50px; height: 25px; font-size: 10px; margin-right: 10px;" type="button" value="2"/>	<input style="border: 1px solid #ccc; width: 50px; height: 25px; font-size: 10px; margin-right: 10px;" type="button" value="2"/>	<input style="border: 1px solid #ccc; width: 50px; height: 25px; font-size: 10px; margin-right: 10px;" type="button" value="16"/>			
<input style="border: 1px solid #ccc; width: 100px; height: 25px; font-size: 10px; margin-right: 10px;" type="button" value="Non DB"/>	<input style="width: 150px; height: 25px; border: 1px solid #ccc;" type="text" value="rails"/>	<input style="border: 1px solid #ccc; width: 50px; height: 25px; font-size: 10px; margin-right: 10px;" type="button" value="2"/>	<input style="border: 1px solid #ccc; width: 50px; height: 25px; font-size: 10px; margin-right: 10px;" type="button" value="4"/>	<input style="border: 1px solid #ccc; width: 50px; height: 25px; font-size: 10px; margin-right: 10px;" type="button" value="4"/>	<input style="border: 1px solid #ccc; width: 50px; height: 25px; font-size: 10px; margin-right: 10px;" type="button" value="32"/>			
<input style="border: 1px solid #ccc; width: 100px; height: 25px; font-size: 10px; margin-right: 10px;" type="button" value="DB"/>	<input style="width: 150px; height: 25px; border: 1px solid #ccc;" type="text" value="mysql"/>	<input style="border: 1px solid #ccc; width: 50px; height: 25px; font-size: 10px; margin-right: 10px;" type="button" value="2"/>	<input style="border: 1px solid #ccc; width: 50px; height: 25px; font-size: 10px; margin-right: 10px;" type="button" value="8"/>	<input style="border: 1px solid #ccc; width: 50px; height: 25px; font-size: 10px; margin-right: 10px;" type="button" value="2"/>	<input style="border: 1px solid #ccc; width: 50px; height: 25px; font-size: 10px; margin-right: 10px;" type="button" value="64"/>	<input style="border: 1px solid #ccc; width: 50px; height: 25px; font-size: 10px; margin-right: 10px;" type="button" value="MySQL"/>		

Total no.of Physical Servers: 8 + Add Row

### TCO Calculator—servers

4. Lastly, we need to input our storage requirements. For our example, the `rails` application, we need the following:

- Storage Type: Object
- Raw Storage Capacity: 2TB
- % Accessed Infrequently: 90



Contact Sales

## Storage

Provide your storage footprint details

Storage Type	i	Raw Storage Capacity	i	% Accessed Infrequently	i
Object	2	TB	90		

[+ Add Row](#)

**Calculate TCO**

CERTIFIED BY   
F R O S T & S U L L I V A N

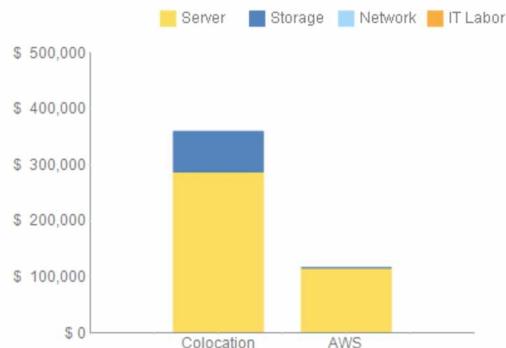
TCO Calculator—storage

5. Go ahead and click Calculate TCO.
6. The 3 year cost breakdown graphs provide a high-level view of your potential cost savings. You can see that, in our example, AWS estimates we'll save 68% on our infrastructure costs over the next 3 years. That's pretty impressive!



Contact Sales

Download Report

**Colocation vs. AWS Summary**You could save **68%** a year by moving your infrastructure to AWS.Your three year total savings would be **\$ 242,820**.**3 Years Cost Breakdown**

3 Yr. Total Cost of Ownership		
	Colocation	AWS
Server	\$ 287,287	\$ 115,787
Storage	\$ 72,038	\$ 717
Network	\$ -	\$ -
IT-Labor	\$ -	\$ -
<b>Total</b>	<b>\$ 359,325</b>	<b>\$ 116,505</b>

AWS cost includes business level support

**TCO Calculator—summary**

7. Scroll further through the report to see cost breakdowns categorized by resource type:



Contact Sales

Download Report

## Your Colocation Cost Breakdown

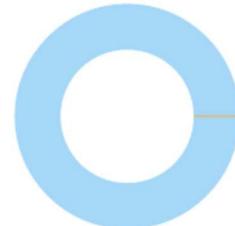
## Server



■ Hardware : \$ 101,527 [35%]  
■ Software : \$ 0 [0%]  
■ Overhead: \$ 185,760 [65%]

## Your AWS Cost Breakdown

## Compute EC2



■ 3 Yr Partial Upfront RI : \$ 106,141 [100%]  
■ On Demand : \$ 0 [0%]

## Storage



## Storage



TCO Calculator—graphs



# How it works...

The calculator will take your server requirements and map them to EC2 instances of an appropriate size. Since we've been specific that we need an object store for our storage, it will calculate our storage costs based on the price for S3 storage in our region.



# There's more...

Let's take a look under the hood and see how we're able to save so much money on AWS:

- The prices for our EC2 instances are based on a 3 year reserved instance price with a partial upfront payment. Is this a fair comparison? Yes and no. You would probably be locked in to a fixed hardware contract with your on-premise or co-located solution, so it makes sense to apply similar contract terms to your AWS pricing model. In reality, you'd probably want to think about purchasing reserved instances *after* you've moved to AWS and performed some fine tuning around which instance types to use. On the flip side, the AWS costs could be reduced even further if your servers ran under All Upfront instance reservations.
- The comparison of object storage systems may or may not be fair depending on the feature set of your on-premise or co-located solution. For example, S3 has the ability to apply an *infrequently accessed* storage class on stored objects, which reduces their cost but also (theoretically) slightly reduces their availability. You'd probably not have this feature in your on-premise or co-located storage.
- The 3 year cost for storage in our on-premise/co-located facility is AU \$69,660, of which a whopping 97% of that is the *monthly cost to operate a rack*. This includes rental of space, cooling, power, and so on.
- While the cost calculator is taking a purely infrastructure view, it also does factor in support costs. If you are new to AWS, you will probably be leaning on AWS support a little bit to get up and running.
- You'll also want to factor in some costs around training and potentially hiring staff who are skilled in deploying and migrating systems to AWS. Your developers are also going to start thinking differently about how to build and deploy their applications. Make sure to factor this in too.
- If you aren't totally happy with the on-premise or co-location estimates, you can go ahead and change the figures used in the calculation. Scroll to the top of the page and click Modify Assumptions to input your own hardware prices:

TCO Calculator Guest

Secure <https://awstcoccalculator.com/#>

**AWS To Colocation**

Are you satisfied with your current colocation provider? [Learn more](#)

Colocation Options

3 Years Cost

\$ 500,000  
\$ 400,000  
\$ 300,000

**Modify Assumptions**

**Colocation Assumption**

**Server Hardware**

Physical host type	List Price	Discounted Price	* Revised Price
2 procs and 2 cores	\$ 5,120	\$ 3,840	\$ 1200 - 55000
2 procs and 4 cores	\$ 5,888	\$ 4,416	\$ 1200 - 55000
2 procs and 8 cores	\$ 7,424	\$ 5,568	\$ 1200 - 55000

\* a default discount of 25% will be applied to the revised price you enter above

**Storage**

Storage Type	Price/GB	Discounted Price	* Revised Price/GB
Object storage (Price per GB)	\$ 2	\$ 0.90	\$ 1 - 3

\* a default discount of 50% will be applied to the revised price you enter above

**Infrequent Parameter**

	Current value	Changed value
% Downloaded Per Month	i 10%	0 - 100

**Download Report**

[« Change Input](#)

TCO calculator? [Click here](#)

TCO Calculator—modify assumptions



## See also

- The *Purchasing reserved instances* recipe