O'REILLY®

AWS Cookbook

Recipes for Success on AWS

Early Release

RAW & UNEDITED

John Culkin, Mike Zazon
& James Ferguson

AWS Cookbook

Building Practical Solutions with AWS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

John Culkin and Mike Zazon

AWS Cookbook

by John Culkin and Mike Zazon

Copyright © 2021 Culkins Coffee Shop LLC and Mike Zazon. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (http://oreilly.com). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

- Acquisitions Editor: Jennifer Pollock
- Development Editor: Virginia Wilson
- Production Editor: Christopher Faucher
- Interior Designer: David Futato
- Cover Designer: Karen Montgomery
- Illustrator: O'Reilly Media, Inc.
- December 2021: First Edition

Revision History for the Early Release

- 2020-12-11: First Release
- 2021-02-19: Second Release
- 2021-04-14: Third Release

See http://oreilly.com/catalog/errata.csp?isbn=9781492092605 for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *AWS Cookbook*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the authors, and do not represent the publisher's views. While the publisher and the authors have used good faith efforts to ensure that the information and instructions

contained in this work are accurate, the publisher and the authors disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-492-09253-7

Preface

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the Preface of the final book. If you have feedback or content suggestions for the authors, please email *awscookbook@gmail.com*.

The vast majority of workloads will go to the cloud.

We're just at the beginning—there's so much more to happen.

Andy Jassy[1]

Cloud usage has been gaining traction with enterprises and small businesses over the last decade and continues to accelerate. Gartner said the Worldwide IaaS Public Cloud Services Market grew 37.3% in 2019.[2] The rapid growth of cloud has led to a skills demand that numerous organizations are trying to satisfy.[3] Many IT professionals understand the basic concepts of the cloud, but want to become more comfortable working in the cloud. A skills shortage in a fast growing area presents a significant opportunity for individuals to attain high paying positions.[4] We wrote this book to share some of our knowledge and enable you to quickly acquire useful skills for working in the cloud. We hope that you will find yourself using this book as reference material for many years to come.

Amazon Web Services (AWS) is the recognized leader in Cloud Infrastructure and Platform services.[5] Through our years of experience we have had the benefit of working on AWS projects in many different roles. We have learned that developers are often looking for guidance on how and when to use AWS services. We would now like to share some of the learnings with you and give you a leg up.

What You Will Learn

In addition to enriching your pocketbook, being able to harness the power of AWS will give you the ability to create powerful systems and applications that solve many interesting and demanding problems in our world today. The on-demand consumption model, vast capacity, advanced capabilities, and global footprint of the cloud create new possibilities that need to be explored. Would you like to handle 60,000 cyber threats per second using AWS Machine Learning like Siemens?[6] Or reduce your organization's on premises footprint and expand its use of microservices like Capital One?[7] If so, the practical examples in this book will help expedite your learning by providing tangible examples showing how you can fit the building blocks of AWS together to form practical solutions that address common scenarios.

Who This Book is For

This book is for developers, engineers, and architects of all levels, from beginner to expert. The recipes in this book aim to bridge the gap between "Hello World" proofs of concept and enterprise grade applications by using applied examples with guided walk-throughs of common scenarios that you can directly apply to your current or future work. These skillful and experience-building tasks will immediately deliver value regardless of your AWS experience level.

The Recipes

We break the book up into chapters which focus on general functional areas of IT (e.g: networking, databases, etc). The recipes contained within the chapters are bite-sized, self-contained, and quickly consumable. Each recipe has a Problem Statement, Solution, and Discussion. Problem statements are tightly defined to avoid confusion. Solution Steps walk you through the work needed to accomplish the goal. We include code (https://github.com/awscookbook) to follow along with and reference later when you need it. Finally, we end each recipe with a short discussion to help you understand the process, ways to utilize in practice, and suggestions to extend the solution.

Some recipes will be "built from scratch" and others will allow you to interact with common scenarios seen in the real world. If needed, foundational resources will be "pre-baked" before you start the recipe. When preparation for a recipe is needed, you will use the AWS Cloud Development Kit which is a fantastic tool for intelligently defining and declaring infrastructure.

> **NOTE**
>
> There are many ways to achieve similar outcomes on AWS, this will not be an exhaustive list. Many factors will dictate what overall solution will have the best fit for your use case.

You'll find recipes for things like:

- Organizing multiple accounts for enterprise deployments
- Creating a chatbot that can pull answers from a knowledge repository
- Automating security group rule monitoring, looking for rogue traffic flows

Also with recipes, we'll also provide one and two liners that will quickly accomplish valuable and routine tasks.

What You'll Need

Here are the requirements to get started and some tips on where to find assistance:

- Personal Computer/Laptop
- Software
  - — Web Browser
    - Edge, Chrome or Firefox
  - — Terminal with Bash
  - — Git
    - https://github.com/git-guides/install-git
  - — Homebrew
    - https://docs.brew.sh/Installation
  - — AWS account
    - https://aws.amazon.com/premiumsupport/knowledge-center/create-and-activate-aws-account/
  - — Code editor
    - E.g.: Visual Studio Code or AWS Cloud9
  - — aws-cli/2.1.1 Python/3.9.0 or later (Can be installed with Homebrew)
    - https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html

— Python 3.7 (and pip) or later (Can be installed with Homebrew)

— AWS Cloud Development Kit (Can be installed with Homebrew)

- https://docs.aws.amazon.com/cdk/latest/guide/getting_started.html
- Version 1.74.0 or later

> **NOTE**
>
> Please ensure that you are using the latest version of AWS CLI **Version 2**

Put on your apron and let's get cooking with AWS!

> **NOTE**
>
> There is a free tier to AWS but implementing recipes in this book could incur costs. We will provide clean up instructions but you are responsible for any costs in your account. We recommend checking out the Well Architected Labs (https://www.wellarchitectedlabs.com/) developed by AWS on expenditure awareness - available at wellarchitectedlabs.com and leveraring AWS Budgets Actions to control costs.
>
> Although we work for AWS, the opinions expressed in this book are our own.

---

1 https://www.forbes.com/sites/siliconangle/2015/01/28/andy-jassy-aws-trillion-dollar-cloud-ambition/.

2 https://www.gartner.com/en/newsroom/press-releases/2020-08-10-gartner-says-worldwide-iaas-public-cloud-services-market-grew-37-point-3-percent-in-2019.

3 https://www.gartner.com/en/newsroom/press-releases/2019-01-17-gartner-survey-shows-global-talent-shortage-is-now-the-top-emerging-risk-facing-organizations.

4 https://www.crn.com/news/global-it-salaries-hit-new-high-2019-it-skills-and-salary-report.

5 https://www.gartner.com/doc/reprints?id=1-242R58F3&ct=200902&st=sb.

6 https://aws.amazon.com/solutions/case-studies/siemens/.

7 https://aws.amazon.com/solutions/case-studies/capital-one-enterprise/.

Chapter 1. Networking

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 2nd chapter of the final book. If you have feedback or content suggestions for the authors, please email *awscookbook@gmail.com*.

1.0 Introduction

In today's world of exciting topics like computer vision, IoT devices, and AI enabled chat bots, traditional core technologies are sometimes ignored. While it's great to have many new capabilities at your fingertips, these technologies would not be possible without a strong foundation of reliable and secure connectivity. Data processing is only useful if the results are reliably delivered and accessible over a network. Containers are a fantastic application deployment method on their own, but they are even more effective and efficient when they are networked together.

Networking is an area that was essential to get right during the birth of the cloud when people began extending their data centers into the cloud with a hybrid approach. Today, networking continues to be at the forefront of the cloud world but doesn't always get the fanfare it deserves. Fundamental technologies, like networking, can be an area of innovation as well. Networking in AWS is a rapidly changing area, as made evident by the popular annual "One to Many: Evolving VPC Design" re:Invent session

Suggested viewing: A great AWS re:Invent networking talk is Eric Brandwine's "Another Day, Another Billion Packets" from 2015.

In this chapter, you will learn about essential cloud networking services and features. We will only focus on recipes that are realistic for you to

accomplish in your personal account. Some advanced operations (e.g. AWS Direct Connect setup) are too dependent on external factors so we felt they should be left out in order to focus on more easily accessible recipes and outcomes.

Gaining a better understanding of networking will allow you to have a better grasp of the cloud and therefore be more comfortable using it.

Table 1-1. Summary of AWS Services

Use Case	Functionality	AWS Service
Build a cloud network	Define and provision a logically isolated network for your AWS resources	Amazon VPC
	Connect VPCs and on-premises networks through a central hub	AWS Transit Gateway
	Provide private connectivity between VPCs, services, and on-premises applications	AWS PrivateLink
	Route users to Internet applications with a managed DNS service	Elastic Load Balancing
Scale your network design	Automatically distribute traffic across a pool of resources, such as instances, containers, IP addresses, and Lambda functions	Elastic Load Balancing
	Direct traffic through the AWS Global network to improve global application performance	AWS Global Accelerator
Secure your network traffic	Safeguard applications running on AWS against DDoS attacks	AWS Shield
	Protect your web applications from common web exploits	AWS WAF
	Centrally configure and manage firewall rules	Amazon Firewall Manager
Build a hybrid IT network	Connect your users to AWS or on-premises resources using a Virtual Private Network	AWS Virtual Private Network (VPN)—Client
	Create an encrypted connection between your network and your Amazon VPCs or AWS Transit Gateways	AWS Virtual Private Network (VPN)—Site to Site
	Establish a private, dedicated connection between AWS and your datacenter, office, or colocation environment	AWS Direct Connect

Use Case	Functionality	AWS Service
Content delivery networks	Securely deliver data, videos, applications, and APIs to customers globally with low latency, and high transfer speeds	Amazon CloudFront
Build a network for microservices architectures	Provide application-level networking for containers and microservices	AWS App Mesh
	Create, maintain, and secure APIs at any scale	Amazon API Gateway
	Discover AWS services connected to your applications	AWS Cloud Map

Source: https://aws.amazon.com/products/networking/

Workstation Configuration

You will need a few things installed to be ready for the recipes in this chapter:

General Setup

Set and export your default region in your terminal

```
AWS_REGION=us-east-1
```

Validate AWS Command Line Interface (AWS CLI) setup and access

```
aws sts get-caller-identity
```

Set your AWS ACCOUNT ID by parsing output from the `aws sts get-caller-identity` operation.

```
AWS_ACCOUNT_ID=$(aws sts get-caller-identity \
        --query Account --output text)
```

> **NOTE**
>
> The aws sts get-caller-identity operation "returns details about the IAM user or role whose credentials are used to call the operation." From: *https://awscli.amazonaws.com/v2/documentation/api/latest/reference/sts/get-caller-identity.html*

Checkout this Chapter's repo

```
git clone https://github.com/AWSCookbook/Networking
```

1.1 Defining Your Private Virtual Network in the Cloud by Creating a VPC

Problem

You need a network foundation to host cloud resources within a region.

Solution

You will create an Amazon Virtual Private Cloud (Amazon VPC) and configure a CIDR block for it.

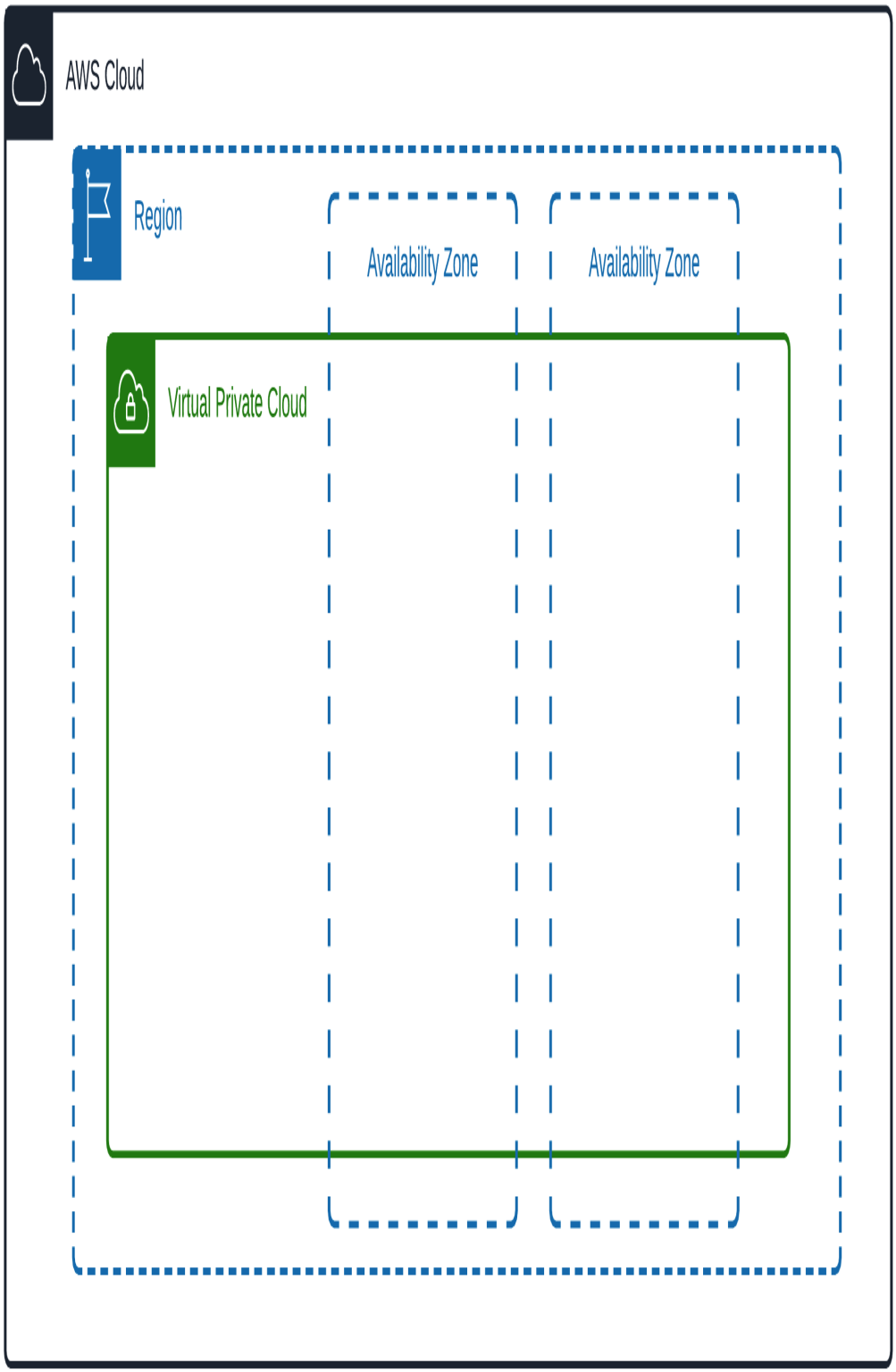

AWS Cloud
Region
Availability Zone
Availability Zone
Virtual Private Cloud

Figure 1-1. VPC deployed in a region

Steps

Create a VPC with an IPv4 CIDR Block. We will use “10.10.0.0/16” as the address range but you can modify based on your needs.

```
VPC_ID=$(aws ec2 create-vpc --cidr-block 10.10.0.0/16 \
        --tag-specifications 'ResourceType=vpc,Tags=
[{Key=Name,Value=AWSCookbook201}]' \
        --output text --query Vpc.VpcId)
```

> **NOTE**
>
> TIP The Name tag is displayed in the console for some resources (e.g. VPCs) that you create. It is helpful to assign a Name tag for these resources to help you easily identify them.

> **NOTE**
>
> When you are creating a VPC, the documentation states that the largest block size for VPC IPv4 CIDRs is a /16 netmask (65,536 IP addresses). The smallest is a /28 netmask (16 IP addresses)

Use the following command to check when the state reaches “associated” for the additional CIDR block

```
aws ec2 describe-vpcs --vpc-ids $VPC_ID \
        --query Vpcs[0].CidrBlockAssociationSet
```

> **NOTE**
>
> Per the VPC user guide, the initial quota of IPv4 CIDR blocks per VPC is 5. This can be raised to 50. The allowed number of IPv6 CIDR blocks per VPC is 1.

Validation steps

As a smoke test, describe the VPC you created using the AWS CLI

```
aws ec2 describe-vpcs --vpc-ids $VPC_ID
```

Challenge

Associate an additional IPv4 CIDR Block to your VPC

```
aws ec2 associate-vpc-cidr-block \
        --cidr-block 10.11.0.0/16 \
        --vpc-id $VPC_ID
```

Clean Up

Delete the VPC you created

```
aws ec2 delete-vpc --vpc-id $VPC_ID
```

Discussion

You created an Amazon Virtual Private Cloud (Amazon VPC) to define a logically isolated virtual network on AWS. You specified an IPv4 (https://en.wikipedia.org/wiki/IPv4) CIDR block which defines the address range available for the subnets you can provision in your VPC (see the next recipe).

WARNING

Two important reasons for carefully selecting CIDR block(s) for your VPC are:

- Once a CIDR Block is associated with a VPC, it can't be modified. If you wish to change a CIDR block, it (and all resources within it) will need to be deleted and recreated.
- If a VPC is connected to other networks by peering (see Recipe 2.11) or gateways (e.g. Transit and VPN), you can not have overlapping IPs ranges.

Also, you added additional IPv4 space to the VPC by using the aws ec2 associate-vpc-cidr-block command to specify the additional IPv4 space. When IP space is scarce, it's good to know that you don't need to dedicate a large block to a VPC, especially if you aren't sure if it all will be utilized

In addition to IPv4, VPC also supports IPv6 (https://en.wikipedia.org/wiki/IPv6). You can configure an amazon-provided IPv6 CIDR block by specifying the --amazon-provided-ipv6-cidr-block option.

Example: Create a VPC with an IPv6 CIDR Block

```
aws ec2 create-vpc --cidr-block 10.10.0.0/16 \
        --amazon-provided-ipv6-cidr-block \
        --tag-specifications 'ResourceType=vpc,Tags=
[{Key=Name,Value=AWSCookbook201-IPv6}]'
```

A VPC is a regional construct in AWS. Regions span all Availability Zones (AZs), which are groups of isolated physical data centers. The number of Availability Zones per region varies, but all regions have at least 3. VPCs can also be extended to AWS Local Zones, AWS Wavelength Zones, and AWS Outposts. For the most up to date information about AWS regions and AZs, see this link: *https://aws.amazon.com/about-aws/global-infrastructure/regions_az/*

Once you have a VPC created, you can begin to define resources within it. See the next recipe (2.2) to begin working with subnets and route tables.

1.2 Creating a Network Tier with Subnets and a Route Table in a VPC

Problem

You have a VPC and need to create a network layout consisting of individual IP spaces for segmentation and redundancy.

Solution

Create a route table within your VPC. Create two subnets in separate Availability Zones in a VPC. Associate the route table with the subnets.

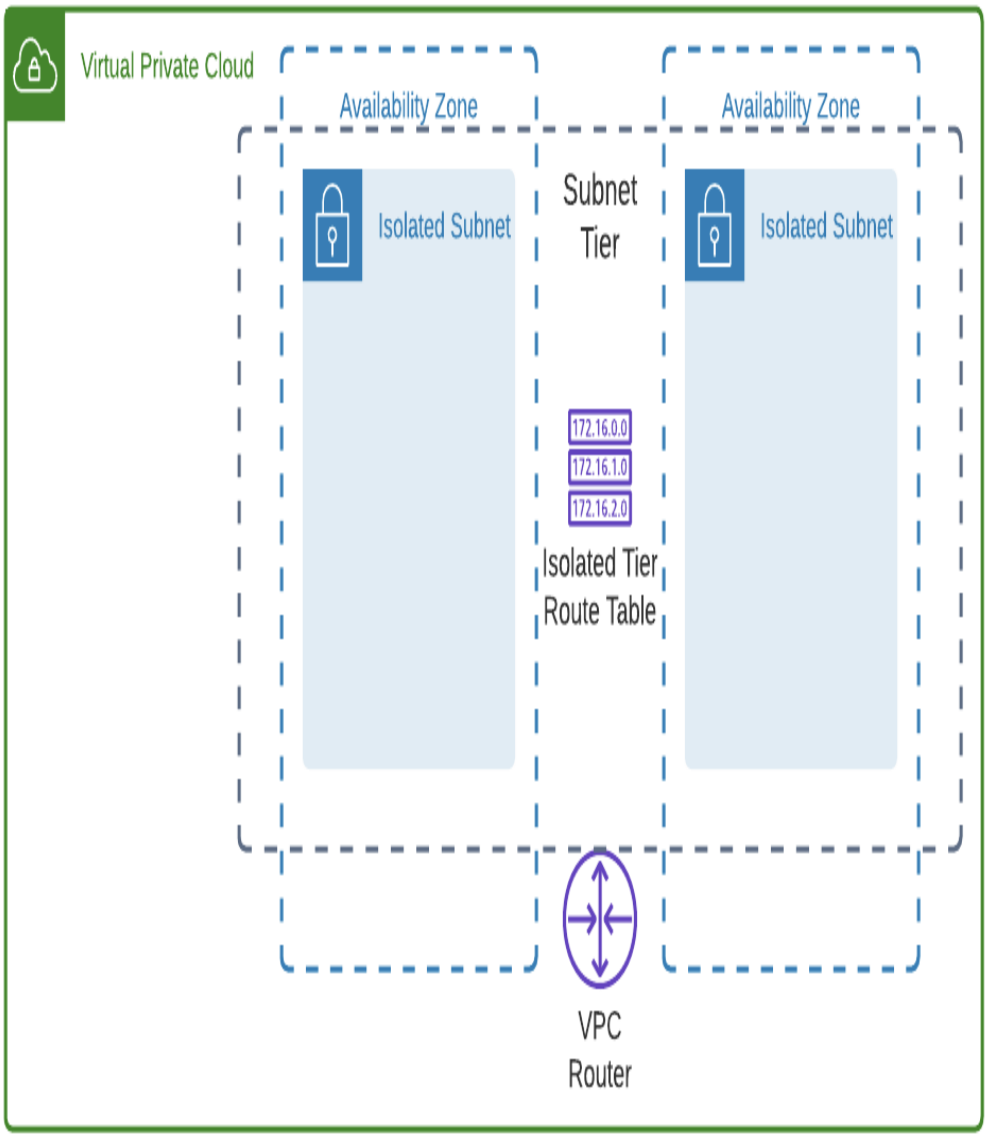


Figure 1-2. Isolated Subnet Tier and Route Table

Prerequisites

- A VPC

Preparation

```
VPC_ID=$(aws ec2 create-vpc --cidr-block 10.10.0.0/23 \
        --tag-specifications \
'ResourceType=vpc,Tags=[{Key=Name,Value=AWSCookbook202}]' \
        --output text --query Vpc.VpcId)
```

Steps

Create a route table. This will allow you to create customized traffic routes for subnets associated with it.

```
ROUTE_TABLE_ID=$(aws ec2 create-route-table --vpc-id $VPC_ID \
        --tag-specifications \
'ResourceType=route-table,Tags=[{Key=Name,Value=AWSCookbook202}]'
\
        --output text --query RouteTable.RouteTableId)
```

Create two subnets, one in each AZ. This will define the address space for you to create resources of your VPC.

```
SUBNET_ID_1=$(aws ec2 create-subnet --vpc-id $VPC_ID \
        --cidr-block 10.10.0.0/24 --availability-zone
${AWS_REGION}a \
        --tag-specifications \
'ResourceType=subnet,Tags=[{Key=Name,Value=AWSCookbook202a}]' \
        --output text --query Subnet.SubnetId)
SUBNET_ID_2=$(aws ec2 create-subnet --vpc-id $VPC_ID \
        --cidr-block 10.10.1.0/24 --availability-zone
${AWS_REGION}b \
        --tag-specifications \
'ResourceType=subnet,Tags=[{Key=Name,Value=AWSCookbook202b}]' \
        --output text --query Subnet.SubnetId)
```

NOTE

In the above commands, the --availability-zone parameter uses an environment variable for your region appended with lowercase a or b characters to indicate which logical availability zone (e.g. us-east-1a) to provision each subnet. AWS states here that these names are randomized per account to balance resources across AZs.

To find Availability Zone (AZ) IDs for a region that are consistent across accounts run:

```
aws ec2 describe-availability-zones --region $AWS_REGION
```

Associate the route table with the two subnets.

```
aws ec2 associate-route-table \
        --route-table-id $ROUTE_TABLE_ID --subnet-id $SUBNET_ID_1
aws ec2 associate-route-table \
        --route-table-id $ROUTE_TABLE_ID --subnet-id $SUBNET_ID_2
```

Validation Steps

Retrieve the configuration of the subnets that you created using the AWS CLI

```
aws ec2 describe-subnets --subnet-ids $SUBNET_ID_1
aws ec2 describe-subnets --subnet-ids $SUBNET_ID_2
```

Challenge

Create a 2nd route table and associate it with $SUBNET_ID_2. Configuring route tables for every AZ is a common pattern. This allows the configuration to ensure that network traffic stays local to the zone when desired. We'll see more about this concept in the next recipes.

Clean Up

Delete your subnets

```
aws ec2 delete-subnet --subnet-id $SUBNET_ID_1
aws ec2 delete-subnet --subnet-id $SUBNET_ID_2
```

Delete your route table:

```
aws ec2 delete-route-table --route-table-id $ROUTE_TABLE_ID
```

Delete your VPC:

```
aws ec2 delete-vpc --vpc-id $VPC_ID
```

Unset your manually created environment variables

```
unset VPC_ID
unset ROUTE_TABLE_ID
```

```
unset SUBNET_ID_1
unset SUBNET_ID_2
```

Discussion

First you created a route table in your VPC. This allows routes to be configured for the network tier so that traffic is sent to the desired destination. You created subnets in two Availability Zones within a VPC. You allocated /24 sized CIDR blocks for the subnets. When designing a subnet strategy, you should choose subnet sizes that fit your current needs and account for your application's future growth. Subnets are used for Elastic Network Interface (ENI) placement for AWS resources which require a connection to your logical network within your VPC. This means that a particular ENI lives within a single Availability Zone.

> **NOTE**
>
> TIP You may run into a case where routes overlap. AWS provides information on how priority is determined here:
>
> *https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Route_Tables.html#route-tables-priority*

AWS reserves the first 4 and last IP address of every subnet's CIDR block for features and functionality when you create a subnet. These are not available for your use. Per the documentation, the reserved addresses in the case of your example are:

- .0: Network address.
- .1: Reserved by AWS for the VPC router.
- .2: Reserved by AWS for the IP address of the DNS server. This is always set to the VPC network range plus two.
- .3: Reserved by AWS for future use.
- .255: Network broadcast address. Broadcast in a VPC is not supported.

A subnet has one route table associated with it. Route tables can be associated with one or more subnets and direct traffic to a destination of your choosing (more on this with the NAT Gateway, Internet Gateway, and Transit Gateway recipes later). Entries within Route tables are called Routes and are defined as pairs of Destinations and Targets. When you created the route table, a default local route that handles intra-VPC traffic was automatically added for you. Subnet CIDR locations for the destination traffic use the default CIDR notation while targets are defined with the logical resource name of where to send the traffic. You have the ability to create custom Routes that fit your needs. For a complete list of targets available to use within route tables, see this support document: *https://docs.aws.amazon.com/vpc/latest/userguide/route-table-options.html*

> **NOTE**
>
> Elastic Network Interfaces (ENIs) receive an IP address from a virtual DHCP server within your VPC. The DHCP options set is automatically configured with defaults for assigning addresses within the subnets you define. This also provides DNS information to your ENIs. For more information about DHCP option sets, and how to create your own DHCP option sets, see this support document: *https://docs.aws.amazon.com/vpc/latest/userguide/VPC_DHCP_Options.html*

When creating a VPC in a region, it is best practice to have at least 1 subnet per Availability zone in that network tier. The number of availability zones differ per region but most have at least 3. An example of of this in practice would be: if you had a public tier and an isolated tier spread over 2 AZs, you would create a total of 4 subnets. 2 tiers x 2 subnets per tier (1 per Availability Zone).

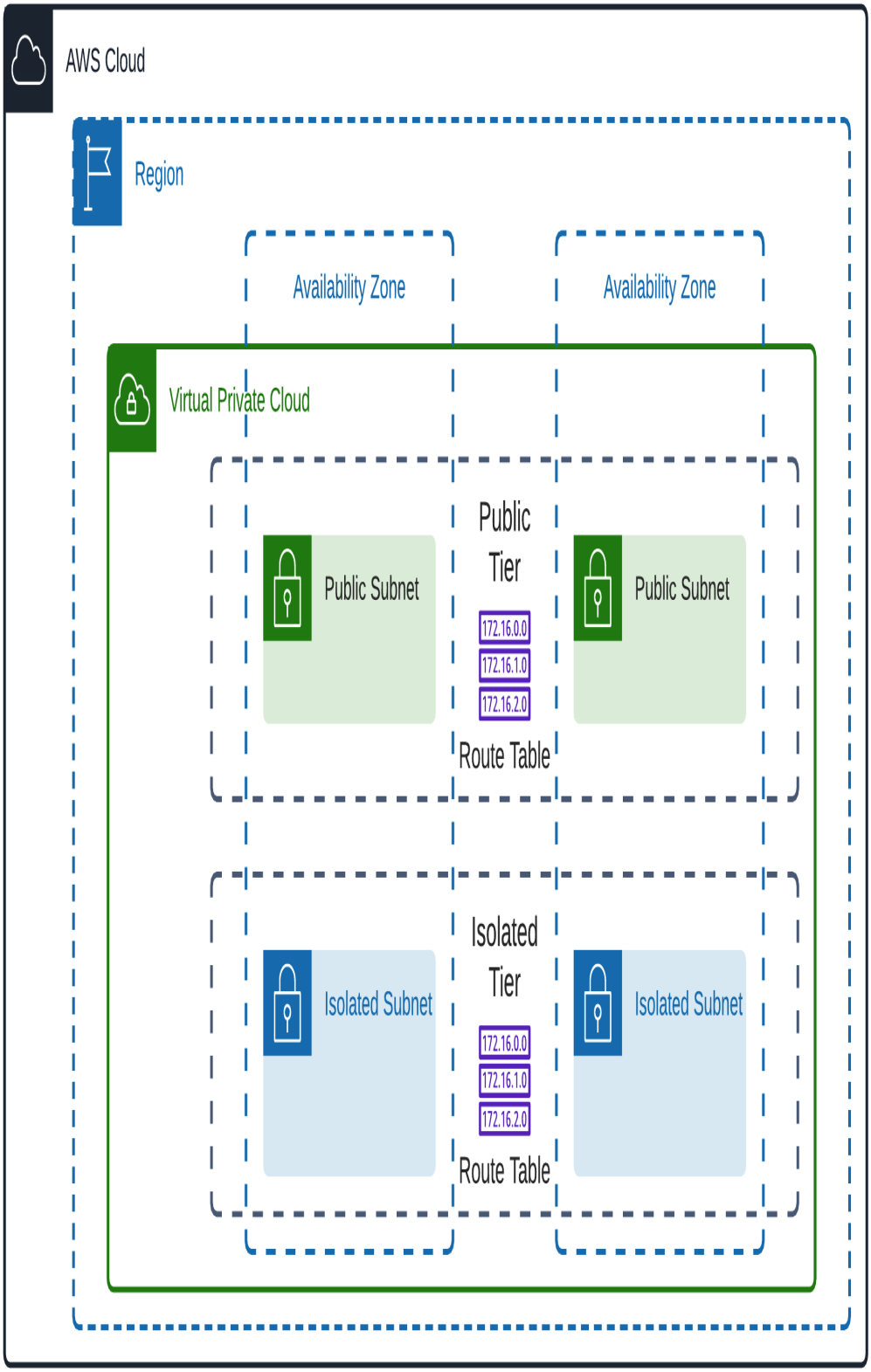

AWS Cloud
Region
Availability Zone
Availability Zone
Virtual Private Cloud
Public Subnet
Public Tier
172.16.0.0
172.16.1.0
172.16.2.0
Route Table
Public Subnet
Isolated Subnet
Isolated Tier
172.16.0.0
172.16.1.0
172.16.2.0
Route Table
Isolated Subnet

Figure 1-3. Isolated and Public Subnet Tiers and Route Tables

Since we have just mentioned “Public Subnets”, let’s move onto the next recipe and grant your VPC access to the Internet.

1.3 Connecting your VPC to the Internet using an Internet Gateway

Problem

You have an existing EC2 instance in a subnet of a VPC. You need to provide the ability for the instance to reach the Internet.

Solution

You will create an Internet Gateway and attach it to your VPC. Next you will modify the route table associated with the subnet. You will add a default route that sends traffic from the subnets to the Internet Gateway.

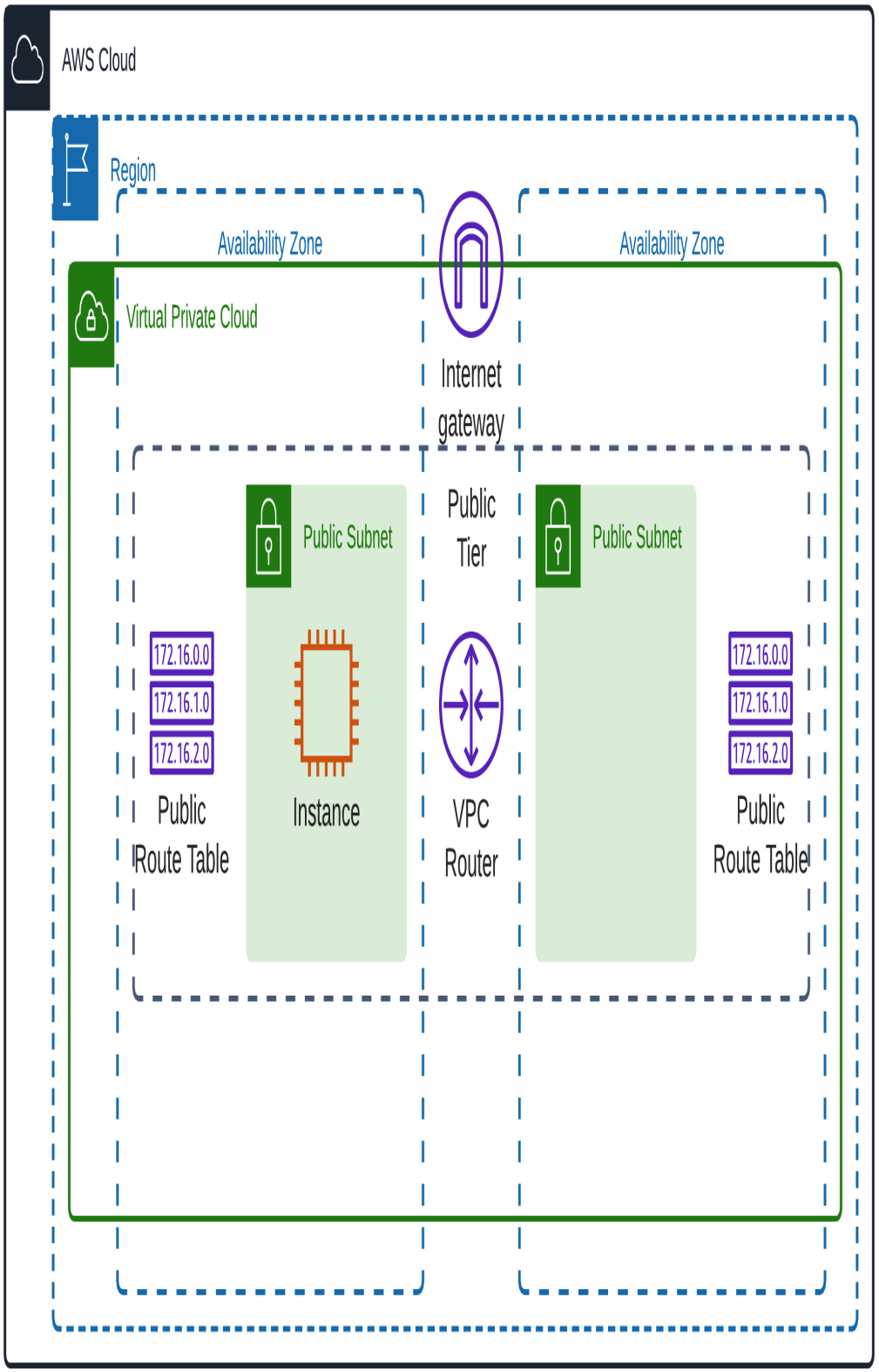

AWS Cloud
Region
Availability Zone
Availability Zone
Virtual Private Cloud
Internet gateway
Public Subnet
Public Tier
Public Subnet
172.16.0.0
172.16.1.0
172.16.2.0
Public Route Table
Instance
VPC Router
172.16.0.0
172.16.1.0
172.16.2.0
Public Route Table

Figure 1-4. Public Subnet Tier, Internet Gateway, and Route Table

Prerequisites

- VPC and subnets created in 2 AZs and associated route tables
- EC2 instance deployed. You will need the ability to connect to this for testing.

Preparation

This recipe requires some "prep work" which deploys resources that you'll build the solution on. You will use the AWS CDK to deploy these resources

In the root of this Chapter's repo **cd** to the "203-Utilizing-Internet-Gateways/cdk-AWS-Cookbook-203" directory

```
cd 203-Utilizing-Internet-Gateways/cdk-AWS-Cookbook-203/
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt --no-dependencies
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

Steps

Create an Internet Gateway.

```
INET_GATEWAY_ID=$(aws ec2 create-internet-gateway \
--tag-specifications \
'ResourceType=internet-gateway,Tags=
[{Key=Name,Value=AWSCookbook202}]' \
        --output text --query InternetGateway.InternetGatewayId)
```

Attach the Internet Gateway to the existing VPC

```
aws ec2 attach-internet-gateway \
        --internet-gateway-id $INET_GATEWAY_ID --vpc-id $VPC_ID
```

In each route table, create a route which sets the default route destination to the Internet Gateway

```
aws ec2 create-route --route-table-id $ROUTE_TABLE_ID_1 \
        --destination-cidr-block 0.0.0.0/0 --gateway-id
$INET_GATEWAY_ID
aws ec2 create-route --route-table-id $ROUTE_TABLE_ID_2 \
        --destination-cidr-block 0.0.0.0/0 --gateway-id
$INET_GATEWAY_ID
```

Create an Elastic IP (EIP)

```
ALLOCATION_ID=$(aws ec2 allocate-address --domain vpc \
        --output text --query AllocationId)
```

> **NOTE**
>
> AWS defines an Elastic IP address (EIP) as “a static IPv4 address designed for dynamic cloud computing. An Elastic IP address is allocated to your AWS account, and is yours until you release it.”

Associate the EIP with the existing EC2 instance

```
aws ec2 associate-address \
        --instance-id $INSTANCE_ID --allocation-id $ALLOCATION_ID
```

Validation Steps

Ensure your EC2 instance has registered with SSM. Use this command to check the status. This command should return the instance ID

```
aws ssm describe-instance-information \
        --filters Key=ResourceType,Values=EC2Instance \
        --query "InstanceInformationList[].InstanceId" --output
text
```

Connect to the EC2 instance using SSM Session Manager

```
aws ssm start-session --target $INSTANCE_ID
```

Ping a host on the internet to test internet connectivity

```
ping -c 4 homestarrunner.com
```

Output:

```
sh-4.2$ ping -c 4 homestarrunner.com
PING homestarrunner.com (72.10.33.178) 56(84) bytes of data.
64 bytes from homestarrunner.com (72.10.33.178): icmp_seq=1 ttl=49
time=2.12 ms
64 bytes from homestarrunner.com (72.10.33.178): icmp_seq=2 ttl=49
time=2.04 ms
64 bytes from homestarrunner.com (72.10.33.178): icmp_seq=3 ttl=49
time=2.05 ms
64 bytes from homestarrunner.com (72.10.33.178): icmp_seq=4 ttl=49
time=2.08 ms
--- homestarrunner.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 2.045/2.078/2.127/0.045 ms
sh-4.2$
```

NOTE

TIP Note that no modifications to the OS configuration needed to occur. If you want to retrieve the public IP from instance's metadata, you can use this command:

```
curl http://169.254.169.254/latest/meta-data/public-ipv4
```

Exit the Session Manager Session

```
exit
```

Challenge

Install a web server on the EC2 instance, modify the security group, and connect to the instance from your workstation. This is not a best practice for production but will help you learn. See recipe 2.7 for an example of how to configure internet access for instances in private subnets using a load balancer.

Clean Up

Disassociate the EIP from the EC2 Instance

```
aws ec2 disassociate-address --association-id \
        $(aws ec2 describe-addresses \
```

```
        --allocation-ids $ALLOCATION_ID \
        --output text --query Addresses[0].AssociationId)
```

Deallocate the Elastic IP address that you created:

```
aws ec2 release-address --allocation-id $ALLOCATION_ID
```

Detach the IGW

```
aws ec2 detach-internet-gateway \
        --internet-gateway-id $INET_GATEWAY_ID --vpc-id $VPC_ID
```

Delete the IGW

```
aws ec2 delete-internet-gateway \
        --internet-gateway-id $INET_GATEWAY_ID
```

To clean up the environment variables, run the `helper.py` script in this recipe's `cdk` directory with the `--unset` flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Unset the environment variable that you created manually

```
unset INET_GATEWAY_ID
unset ALLOCATION_ID
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && rm -r .venv/ && cd ../..
```

Discussion

You created an Internet Gateway (IGW) and attached it to your VPC. Next you configured a route table entry with a destination CIDR of 0.0.0.0/0 to target the IGW. This route table entry sends all non-local traffic to the IGW which provides your VPC internet connectivity. Because this was a running instance that you were working with, you created an Elastic IP and associated it with the instance. These steps enabled internet communication for the instance. There is an option to enable auto-assignment of public IPv4 addresses for newly launched instances in a subnet. However if you utilize

auto assignment, the public IPs will change after each instance reboot. EIPs associated with an instance will not change after reboots.

The security group associated with your instance does not allow inbound access. If you would like to allow inbound internet access to an instance in a public subnet, you will have to configure a security group ingress rule for this.

A subnet that has a route of 0.0.0.0/0 associated with an IGW is considered a public subnet. It is considered a security best practice to only place instances in this type of tier which require inbound access from the public internet. Load balancers are commonly placed in public subnets. A public subnet would not be an ideal choice for an application server or a database. In these cases, you can create a private tier or an isolated tier to fit your needs with the appropriate routing and use a NAT gateway to direct that subnet traffic to the Internet Gateway only when outbound internet access is required. Let's look at the next recipe to learn about granting outbound internet access for instance in private subnets.

1.4 Using a NAT Gateway for Outbound Internet Access from Private Subnets

Problem

You already have public subnets in your VPC that have a route to an Internet Gateway. You want to leverage this setup to provide outbound internet access for an instance in your private subnets.

Solution

Create a NAT gateway in one of the public subnets. Then create an Elastic IP and associate it with the NAT gateway. In the route table associated with the private subnets, add a route for internet bound traffic which targets the NAT gateway.

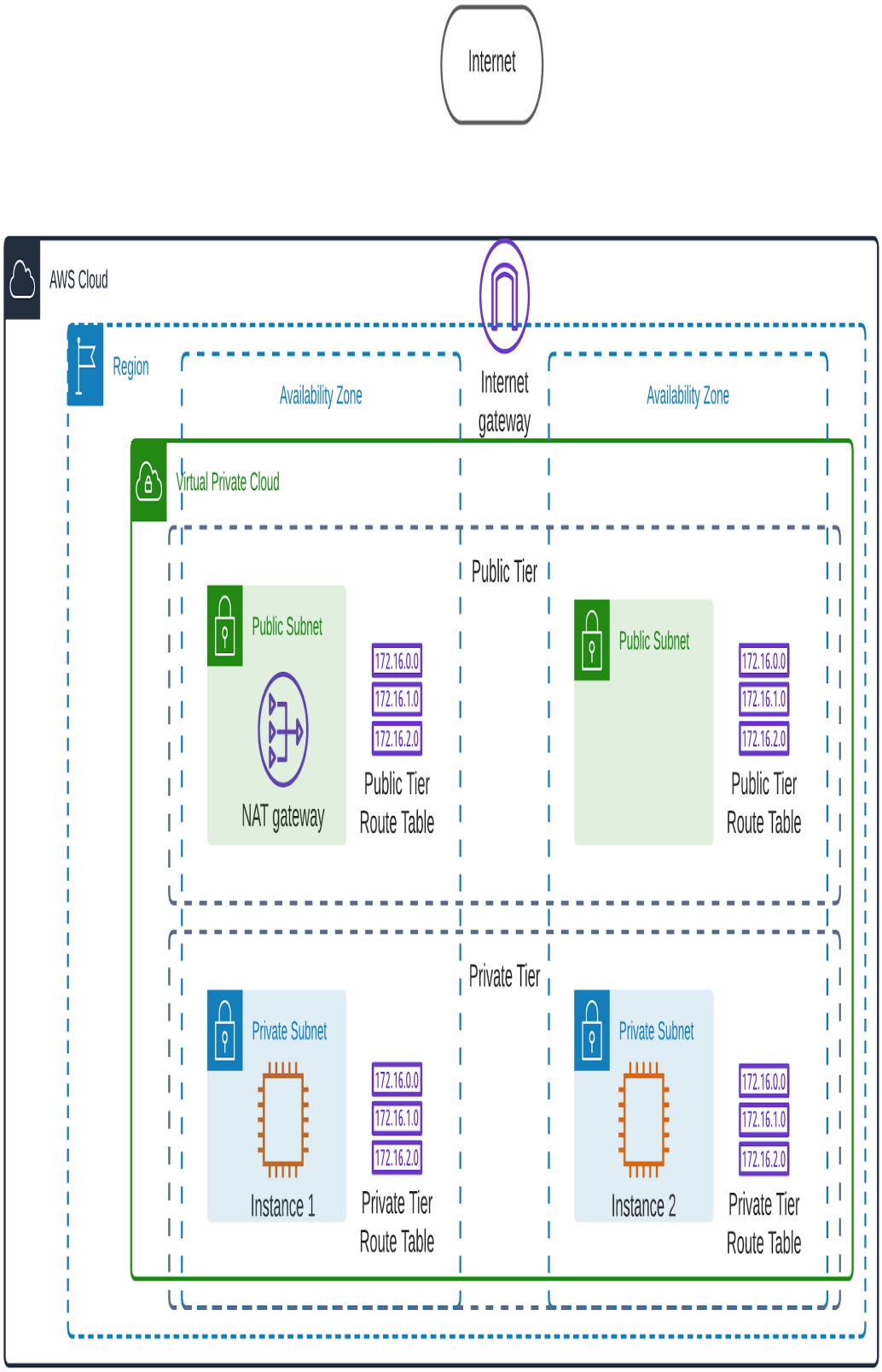

Internet
AWS Cloud
Region
Availability Zone
Internet gateway
Availability Zone
Virtual Private Cloud
Public Tier
Public Subnet
NAT gateway
172.16.0.0
172.16.1.0
172.16.2.0
Public Tier Route Table
Public Subnet
172.16.0.0
172.16.1.0
172.16.2.0
Public Tier Route Table
Private Tier
Private Subnet
Instance 1
172.16.0.0
172.16.1.0
172.16.2.0
Private Tier Route Table
Private Subnet
Instance 2
172.16.0.0
172.16.1.0
172.16.2.0
Private Tier Route Table

Figure 1-5. Internet access for private subnets provided by NAT gateways

Prerequisites

- VPC with public subnets in 2 AZs and associated route tables
- Isolated subnets created in 2 AZs (we will turn these into the private subnets) and associated route tables
- Two EC2 instances deployed in the isolated subnets. You will need the ability to connect to these for testing.

Preparation

This recipe requires some "prep work" which deploys resources that you'll build the solution on. You will use the AWS CDK to deploy these resources

In the root of this Chapter's repo **cd** to the *204-Using-A-Nat-Gateway/cdk-AWS-Cookbook-204* directory and follow the subsequent steps:

```
cd 204-Using-A-Nat-Gateway/cdk-AWS-Cookbook-204/
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt --no-dependencies
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

Steps

Create an Elastic IP to be used with the NAT gateway

```
ALLOCATION_ID=$(aws ec2 allocate-address --domain vpc \
        --output text --query AllocationId)
```

Create a NAT gateway within the public subnet of AZ1

```
NAT_GATEWAY_ID=$(aws ec2 create-nat-gateway \
        --subnet-id $VPC_PUBLIC_SUBNET_1 \
        --allocation-id $ALLOCATION_ID \
        --output text --query NatGateway.NatGatewayId)
```

This will take a few moments for the state to become “available”, check the status with:

```
aws ec2 describe-nat-gateways \
        --nat-gateway-ids $NAT_GATEWAY_ID \
        --output text --query NatGateways[0].State
```

Add a default route for 0.0.0.0/0 with a destination of the NAT gateway to both of the private tier’s route tables. A default route sends all traffic not matching a specific route to the destination specified.

```
aws ec2 create-route --route-table-id $PRIVATE_RT_ID_1 \
        --destination-cidr-block 0.0.0.0/0 \
        --nat-gateway-id $NAT_GATEWAY_ID
aws ec2 create-route --route-table-id $PRIVATE_RT_ID_2 \
        --destination-cidr-block 0.0.0.0/0 \
        --nat-gateway-id $NAT_GATEWAY_ID
```

Validation Steps

Ensure your EC2 instance #1 has registered with SSM. Use this command to check the status. This command should return the instance ID

```
aws ssm describe-instance-information \
        --filters Key=ResourceType,Values=EC2Instance \
        --query "InstanceInformationList[].InstanceId" --output
text
```

Connect to your EC2 instance using SSM Session Manager

```
aws ssm start-session --target $INSTANCE_ID_1
```

Test internet access

```
ping -c 4 aws.amazon.com
Example Output:
sh-4.2$ ping -c 4 aws.amazon.com
PING dr49lng3n1n2s.cloudfront.net (99.84.179.73) 56(84) bytes of
data.
64 bytes from server-99-84-179-73.iad89.r.cloudfront.net
(99.84.179.73): icmp_seq=1 ttl=242 time=1.59 ms
64 bytes from server-99-84-179-73.iad89.r.cloudfront.net
(99.84.179.73): icmp_seq=2 ttl=242 time=1.08 ms
64 bytes from server-99-84-179-73.iad89.r.cloudfront.net
```

```
(99.84.179.73): icmp_seq=3 ttl=242 time=1.13 ms
64 bytes from server-99-84-179-73.iad89.r.cloudfront.net
(99.84.179.73): icmp_seq=4 ttl=242 time=1.10 ms
--- dr49lng3n1n2s.cloudfront.net ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.081/1.227/1.590/0.214 ms
sh-4.2$
```

Exit the Session Manager Session

```
exit
```

(Optional - Repeat the Validation Steps for Instance2)

Challenge

Create a 2nd NAT gateway in the public subnet in AZ2. Then modify the default route in the route table associated with the private subnet in AZ2. Change the destination to the newly created NAT gateway.

Clean Up

Delete the NAT gateway that you created (this may take up to 1 minute to delete):

```
aws ec2 delete-nat-gateway --nat-gateway-id $NAT_GATEWAY_ID
```

Wait until the NAT gateway has reached the “deleted” state.

```
aws ec2 describe-nat-gateways --nat-gateway-id $NAT_GATEWAY_ID \
        --output text --query NatGateways[0].State
```

Release the Elastic IP address that you created:

```
aws ec2 release-address --allocation-id $ALLOCATION_ID
```

To clean up the environment variables, run the helper.py script in this recipe’s cdk- directory with the --unset flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Unset the environment variable that you created manually

```
unset ALLOCATION_ID
unset NAT_GATEWAY_ID
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && rm -r .venv/ && cd ../..
```

Discussion

You created a NAT gateway in a public subnet and configured the route tables for private subnets to send traffic destined for the 0.0.0.0/0 (public internet) to the NAT gateway ID. This allows you to have a subnet tier that allows outbound access, but does not permit direct inbound internet access to resources within. One way to allow internet resources inbound access to services running on resources in private subnets is to use a load balancer in the public subnets. We'll look more at that type of configuration in recipe 2.7.

You also created an Elastic IP address to associate with the NAT gateway. This EIP becomes the external IP address for all communication that goes through the NAT gateway. For example, if a vendor needed to configure a firewall rule to allow instances within your private subnet to communicate with it, the NAT gateway EIP would be the "source" ip address provided to the vendor. Your EIP will remain the same as long as you keep it provisioned within your account.

> **NOTE**
>
> TIP If you created a VPC with IPv6 capability, you can also create an egress-only internet gateway to allow outbound internet access for private subnets. You can read more about this here: *https://docs.aws.amazon.com/vpc/latest/userguide/egress-only-internet-gateway.html*

This NAT gateway was provisioned within one Availability Zone in your VPC. While this is a cost-effective way to achieve outbound internet access for your private subnets, for production and mission-critical applications you should consider provisioning NAT gateways in each AZ to provide resiliency and reduce the amount of cross-AZ traffic. This would also

require creating route tables for each of your private subnets so that you can direct the 0.0.0.0/0 traffic to the NAT gateway in that particular subnet's AZ.

> **NOTE**
>
> If you have custom requirements or would like more granular control of your outbound routing for your NAT implementation you can use a NAT instance. For a comparison of NAT gateway and NAT instance, see this support document.

1.5 Granting Dynamic Access by Referencing Security Groups

Problem

You have a group of two instances and need to allow SSH access between them. This needs to be configured in a way to easily add additional instances to the group in the future that will have the same access.

Solution

Create a security group and associate each to your EC2 instances. Create an ingress authorization allowing the security group to reach itself on TCP port 22.

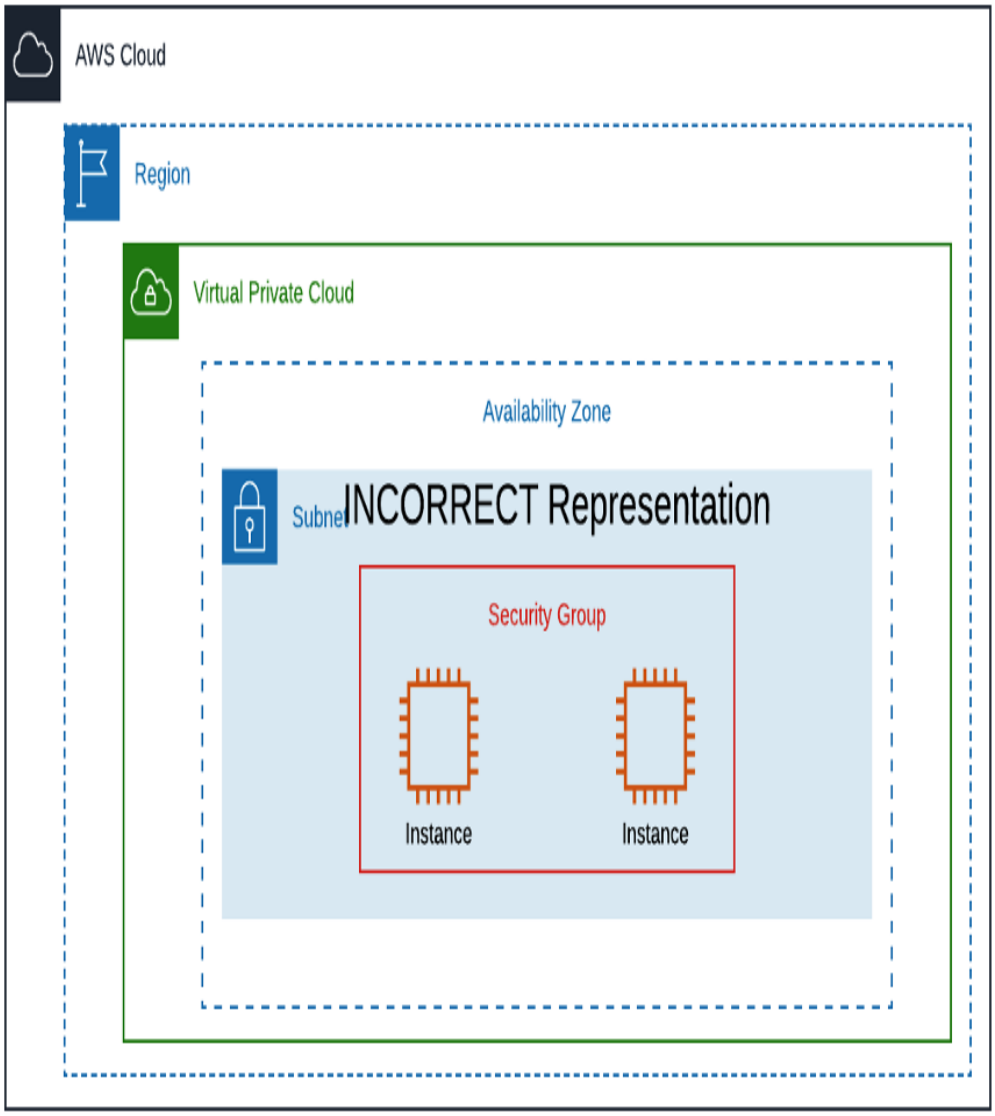


Figure 1-6. Incorrect representation of two instances using the same security group

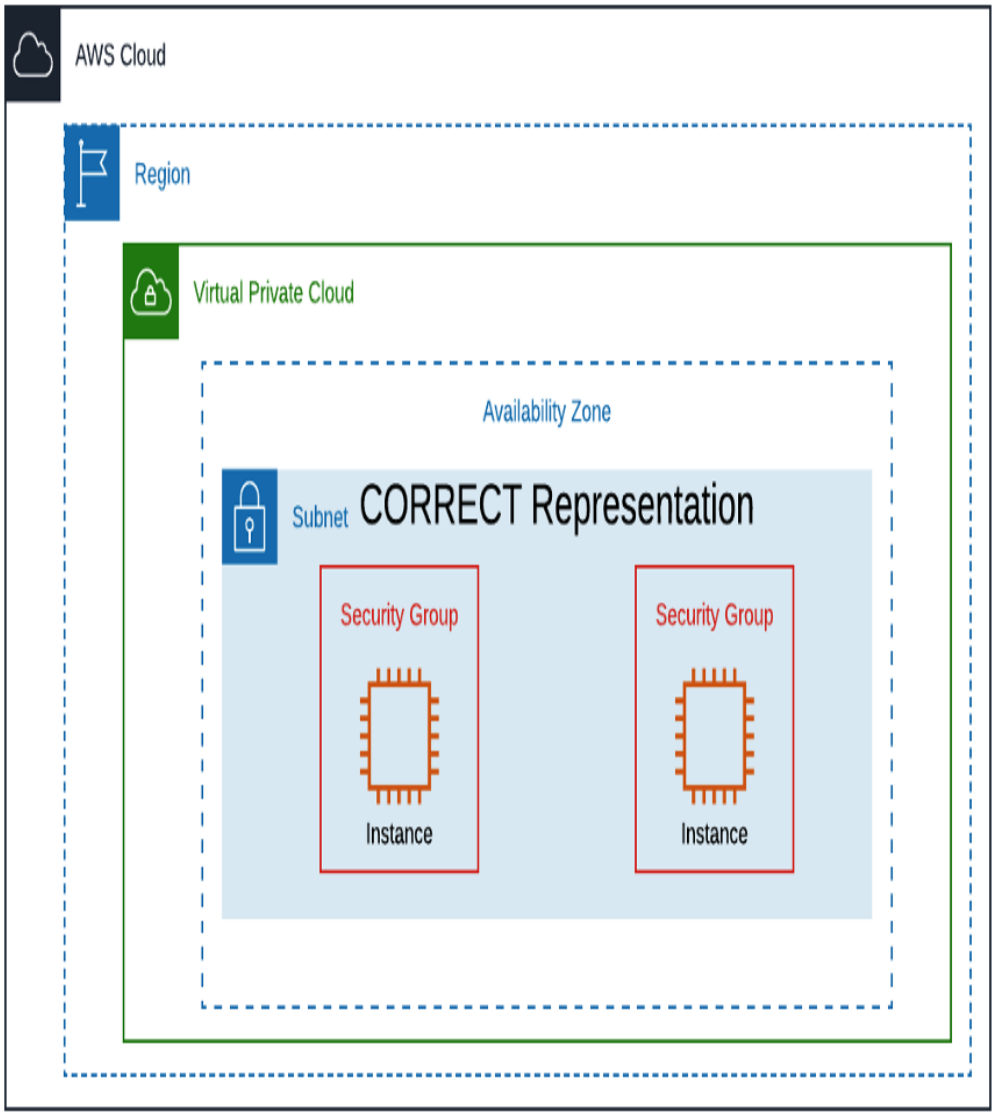


Figure 1-7. Correct visualization of two instances using the same security group

Prerequisites

- VPC with a subnet and associated route table
- Two EC2 instances deployed in the subnet. You will need the ability to connect to these for testing.

Preparation

This recipe requires some “prep work” which deploys resources that you’ll build the solution on. You will use the AWS CDK to deploy these resources

In the root of this Chapter’s repo **cd** to the “205-Using-Security-Group-References/cdk-AWS-Cookbook-205” directory and follow the subsequent steps:

```
cd 205-Using-Security-Group-References/cdk-AWS-Cookbook-205/
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt --no-dependencies
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

Steps

Create a new security group for the EC2 instances:

```
SG_ID=$(aws ec2 create-security-group \
        --group-name AWSCookbook205Sg \
        --description "Instance Security Group" --vpc-id $VPC_ID \
        --output text --query GroupId)
```

Attach the security group to Instance 1:

```
aws ec2 modify-instance-attribute --instance-id $INSTANCE_ID_1 \
        --groups $SG_ID
```

Attach security group to Instance 2:

```
aws ec2 modify-instance-attribute --instance-id $INSTANCE_ID_2 \
        --groups $SG_ID
```

NOTE

You used the modify-instance-attribute command to attach a new security group to your EC2 instances. You can have multiple security groups associated with your EC2 instances. To list the security groups associated with an EC2 instance, you can view them in the EC2 console under the “Security” tab of the instance details or use this command (replacing $INSTANCE_ID_1 with your own instance ID):

```
aws ec2 describe-security-groups --group-ids \
$(aws ec2 describe-instances --instance-id
$INSTANCE_ID_1 \
--query
"Reservations[].Instances[].SecurityGroups[].GroupId[]"
\
--output text) --output text
```

Add an ingress rule to security group that allows access on TCP port 22 from itself:

```
aws ec2 authorize-security-group-ingress \
        --protocol tcp --port 22 \
                --source-group $SG_ID \
--group-id $SG_ID
```

NOTE

TIP This type of security group rule is called a “self-referencing” rule which permits access to members of the same security group.

Validation Steps

Create and populate a SSM parameter to store values so that you can retrieve them from your EC2 instance

```
aws ssm put-parameter \
    --name "Cookbook205Instance2Ip" \
    --type "String" \
    --value $(aws ec2 describe-instances --instance-ids
$INSTANCE_ID_2 --output text --query
Reservations[0].Instances[0].PrivateIpAddress)
```

Ensure your EC2 instance #1 has registered with SSM. Use this command to check the status. This command should return the instance ID

```
aws ssm describe-instance-information \
        --filters Key=ResourceType,Values=EC2Instance \
        --query "InstanceInformationList[].InstanceId" --output
text
```

Connect to your EC2 instance using SSM Session Manager

```
aws ssm start-session --target $INSTANCE_ID_1
```

Install the Ncat utility

```
sudo yum -y install nc
```

Set the region by grabbing the value from the instance's metadata

```
export AWS_DEFAULT_REGION=$(curl --silent
http://169.254.169.254/latest/dynamic/instance-identity/document \
| awk -F'"' ' /region/ {print $4}')
```

Retrieve the IP for Instance 2

```
INSTANCE_IP_2=$(aws ssm get-parameters \
    --names "Cookbook205Instance2Ip" \
                --query "Parameters[*].Value" --output text)
```

Test ssh connectivity to the other instance

```
nc -vz $INSTANCE_IP_2 22
```

Example Output:

```
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Connected to 10.10.0.48:22.
Ncat: 0 bytes sent, 0 bytes received in 0.01 seconds.
sh-4.2$
```

Exit the Session Manager Session

```
exit
```

(Optional - Repeat the Validation Steps from Instance 2 to Instance 1)

Challenge 1

Create a 3rd EC2 instance, use the same security group. Test access to/from it.

```
INSTANCE_ID_3=$(aws ec2 run-instances \
        --image-id $AMZNLINUXAMI --count 1 \
        --instance-type t3.nano --security-group-ids $SG_ID \
        --subnet-id $VPC_ISOLATED_SUBNET_1 \
        --output text --query Instances[0].InstanceId)
```

Retrieve the IAM Instance Profile Arn for Instance2 so that you can associate it with your new instance. This will allow the instance to register with SSM.

```
INSTANCE_PROFILE=$(aws ec2 describe-iam-instance-profile-
associations \
        --filter "Name=instance-id,Values=$INSTANCE_ID_2" \
        --output text --query
IamInstanceProfileAssociations[0].IamInstanceProfile.Arn)
```

Associate the IAM Instance Profile with Instance3

```
aws ec2 associate-iam-instance-profile \
    --instance-id $INSTANCE_ID_3 \
    --iam-instance-profile Arn=$INSTANCE_PROFILE
```

Reboot the instance to have it register with SSM

```
aws ec2 reboot-instances --instance-ids $INSTANCE_ID_3
```

Once that is complete you can connect to it using SSM Session Manager

```
aws ssm start-session --target $INSTANCE_ID_3
```

Challenge 2

Use the steps in Recipe 2.6 to test the connectivity with the VPC Reachability Analyzer

Clean Up

Terminate Instance 3 if you created it

```
aws ec2 terminate-instances --instance-ids $INSTANCE_ID_3
```

Delete the SSM Parameters that you created

```
aws ssm delete-parameter --name "Cookbook205Instance2Ip"
```

Detach the security groups you created from each instance and attach the VPC default security group (so that you can delete the security groups in the next step):

```
aws ec2 modify-instance-attribute --instance-id \
        $INSTANCE_ID_1 --groups $DEFAULT_VPC_SECURITY_GROUP
aws ec2 modify-instance-attribute --instance-id \
        $INSTANCE_ID_2 --groups $DEFAULT_VPC_SECURITY_GROUP
```

Delete the security group that you created

```
aws ec2 delete-security-group --group-id $SG_ID
```

To clean up the environment variables, run the helper.py script in this recipe's `cdk` directory with the `--unset` flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Unset the environment variable that you created manually

```
unset SG_ID
unset INSTANCE_ID_3
unset INSTANCE_PROFILE
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && rm -r .venv/ && cd ../..
```

Discussion

You created a security group and associated it to two EC2 instances. You then added an ingress rule which allowed ssh (TCP port 22) access to the security group from itself. It's important to note that the "source' of the security group was a security group, not a list of IPs.

The on demand nature of the cloud (e.g. Auto Scaling) presents an opportunity for elasticity. Network security mechanisms available like security group references lend well to that. Traditionally, network architects might authorize CIDR ranges within firewall configurations. This type of authorization is generally referred to as static references. This legacy practice doesn't scale dynamically as you may add or remove instances from your workloads.

AWS resources that you provision requiring an Elastic Network Interface (ENI) are associated with subnets within a VPC. For example, if you

provision an EC2 instance, you must choose a VPC and subnet to associate with the ENI. A security group acts as a stateful virtual firewall for ENIs. The default behavior for security groups is to block all ingress while allowing all egress. You can associate multiple security groups with an ENI. There is an initial quota of 5 security groups per ENI and 60 rules (inbound or outbound) per security group.

A common misconception is that by merely associating the same security group to multiple EC2 instances, it will allow communication between the instances. This belief is associated with the incorrect configuration visualized in Figure 2.5.1 above. The correct representation is shown in Figure 2.5.2.

NOTE

TIP You can (and should) create "descriptions" for your security group rules to indicate the intended functionality of the authorization. You can also specify CIDR notation for authorizations. E.g. for an authorization intended to allow RDP access from your New York branch office, you would use the following:

```
aws ec2 authorize-security-group-ingress \
--group-id sg-1234567890abcdef0 \
--ip-permissions
IpProtocol=tcp,FromPort=3389,ToPort=3389,IpRanges='[{Cid
rIp=XXX.XXX.XXX.XXX/24,Description="RDP access from NY
office"}]'
```

WARNING

Remember that security groups can not be deleted if the following conditions are present:

- They are currently attached to an ENI
- They are referenced by other security groups (including themselves)

1.6 Using VPC Reachability Analyzer to Verify and Troubleshoot Network Paths

Problem

You have two EC2 instances deployed in isolated subnets. You need to troubleshoot SSH connectivity between them.

Solution

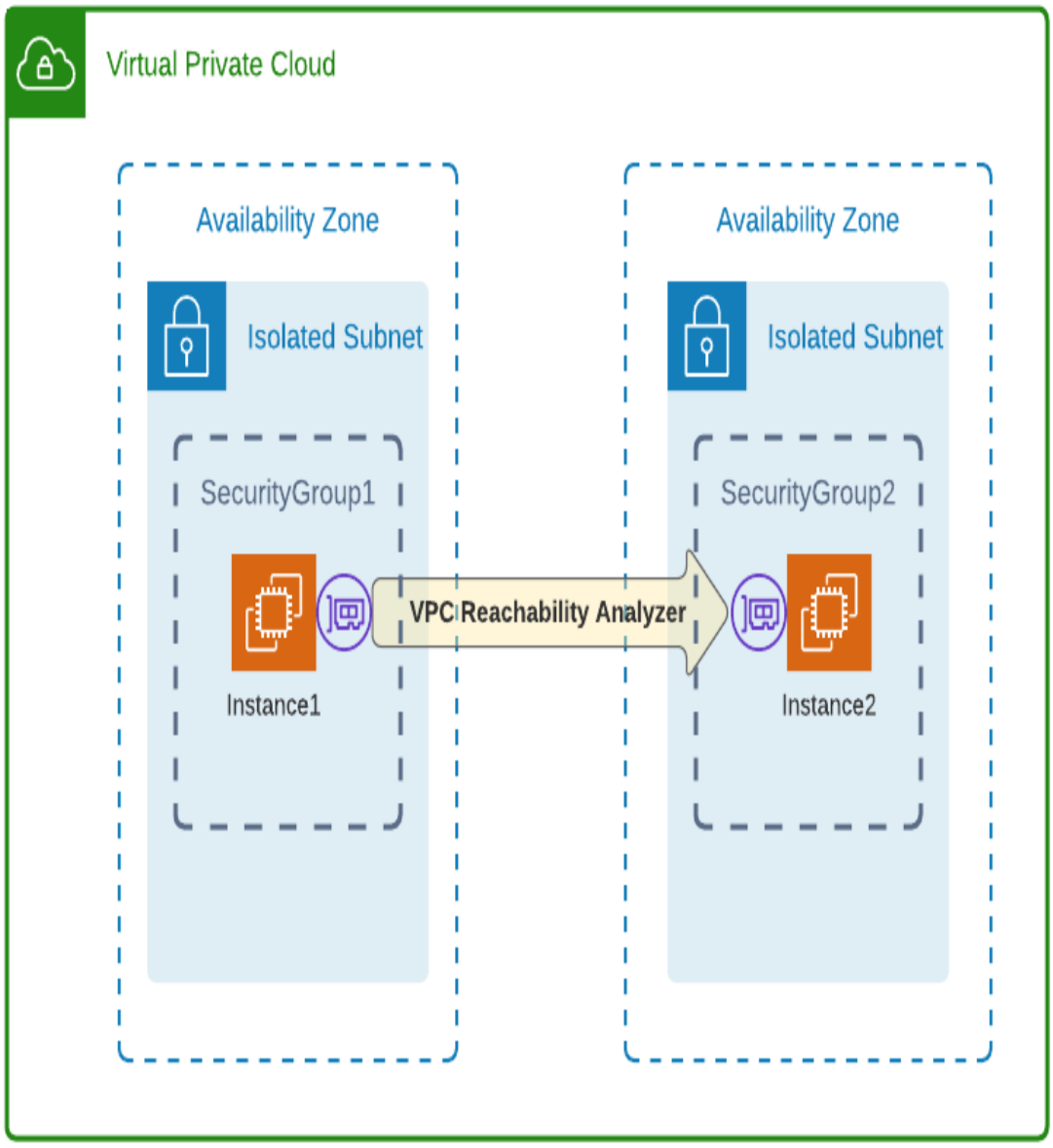


Figure 1-8. VPC Reachability Analyzer

You will create, analyze, and describe network insights using the VPC Reachability Analyzer. Based on the results, you will add a rule to the security group of Instance 2 which allows the SSH port (TCP port 22) from Instance 1's security group. Finally you will rerun the VPC Reachability Analyzer and view the updated results.

Prerequisites

- VPC with public subnets in 2 AZs and associated route tables
- Isolated subnets created in 2 AZs (we will turn these into the private subnets) and associated route tables
- Two EC2 instances deployed in the isolated subnets. You will need the ability to connect to these for testing.

Preparation

This recipe requires some "prep work" which deploys resources that you'll build the solution on. You will use the AWS CDK to deploy these resources

In the root of this Chapter's repo **cd** to the "206-VPC-Reachability-Analyzer/cdk-AWS-Cookbook-206/" directory and follow the subsequent steps:

```
cd 206-VPC-Reachability-Analyzer/cdk-AWS-Cookbook-206/
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt --no-dependencies
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

Steps

Create a Network Insights Path specifying both of the EC2 instances you deployed and TCP port 22:

```
INSIGHTS_PATH_ID=$(aws ec2 create-network-insights-path \
        --source $INSTANCE_ID_1 --destination-port 22 \
        --destination $INSTANCE_ID_2 --protocol tcp \
        --output text --query
NetworkInsightsPath.NetworkInsightsPathId)
```

Start the Network Insights Analysis between the two instances using the INSIGHTS_PATH_ID created in the previous step:

```
ANALYSIS_ID_1=$(aws ec2 start-network-insights-analysis \
        --network-insights-path-id $INSIGHTS_PATH_ID --output text
\
        --query NetworkInsightsAnalysis.NetworkInsightsAnalysisId)
```

Wait a few seconds until the analysis is done running and then view the results:

```
aws ec2 describe-network-insights-analyses \
        --network-insights-analysis-ids $ANALYSIS_ID_1
```

Output snippet (Note the “NetworkPathFound” and “ExplanationCode” fields)

```
{
    "NetworkInsightsAnalyses": [
        {
            "NetworkInsightsAnalysisId": "nia-0ae6e31d2fb4bd680",
            "NetworkInsightsAnalysisArn": "arn:aws:ec2:us-east-
1:111111111111:network-insights-analysis/nia-0ae6e31d2fb4bd680",
            "NetworkInsightsPathId": "nip-0080a415e43527d0f",
            "StartDate": "2020-12-22T02:12:36.836000+00:00",
            "Status": "succeeded",
            "NetworkPathFound": false,
            "Explanations": [
                {
                    "Direction": "ingress",
                    "ExplanationCode": "ENI_SG_RULES_MISMATCH",
                    "NetworkInterface": {
                        "Id": "eni-0676808b04add14c9",
                        "Arn": "arn:aws:ec2:us-east-
1:11111111111:network-interface/eni-0676808b04add14c9"
                    },
```

Update the security group attached to instance 2. Add a rule to allow access from instance 1’s security group to TCP port 22 (SSH)

```
aws ec2 authorize-security-group-ingress \
        --protocol tcp --port 22 \
        --source-group $INSTANCE_SG_ID_1 \
        --group-id $INSTANCE_SG_ID_2
```

Rerun the network insights analysis. Use the same INSIGHTS_PATH_ID as you did previously.

```
ANALYSIS_ID_2=$(aws ec2 start-network-insights-analysis \
        --network-insights-path-id $INSIGHTS_PATH_ID --output text
\
        --query NetworkInsightsAnalysis.NetworkInsightsAnalysisId)
```

Show the results of the new analysis:

```
aws ec2 describe-network-insights-analyses \
        --network-insights-analysis-ids $ANALYSIS_ID_2
```

Output snippet (Note the “NetworkPathFound” field)

```
{
        "NetworkInsightsAnalyses": [
        {
                "NetworkInsightsAnalysisId": "nia-
0f6c22b6429feb378",
                "NetworkInsightsAnalysisArn": "arn:aws:ec2:us-east-
1:111111111111:network-insights-analysis/nia-0f6c22b6429feb378",
                "NetworkInsightsPathId": "nip-09f7e16b46836b0c6",
                "StartDate": "2021-02-21T23:52:15.565000+00:00",
                "Status": "succeeded",
                "NetworkPathFound": true,
                "ForwardPathComponents": [
                {
                        "SequenceNumber": 1,
                        "Component": {
                        "Id": "i-0f945e41551cf0235",
```

Validation Steps

Create and populate some SSM parameters to store values so that you can retrieve them from your EC2 instance

```
aws ssm put-parameter \
    --name "Cookbook206Instance2Ip" \
    --type "String" \
    --value $(aws ec2 describe-instances --instance-ids
$INSTANCE_ID_2 --output text --query
Reservations[0].Instances[0].PrivateIpAddress)
```

Ensure your EC2 instance #1 has registered with SSM. Use this command to check the status. This command should return the instance ID

```
aws ssm describe-instance-information \
        --filters Key=ResourceType,Values=EC2Instance \
        --query "InstanceInformationList[].InstanceId" --output
text
```

Connect to your EC2 instance using SSM Session Manager

```
aws ssm start-session --target $INSTANCE_ID_1
```

Install the Ncat utility

```
sudo yum -y install nc
```

Set the region by grabbing the value from the instance's metadata

```
export AWS_DEFAULT_REGION=$(curl --silent
http://169.254.169.254/latest/dynamic/instance-identity/document \
| awk -F'"' ' /region/ {print $4}')
```

Retrieve the IP for Instance 2

```
INSTANCE_IP_2=$(aws ssm get-parameters \
    --names "Cookbook206Instance2Ip" \
        --query "Parameters[*].Value" --output text)
```

Test ssh connectivity to the other instance

```
nc -vz $INSTANCE_IP_2 22
```

Example Output:

```
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Connected to 10.10.0.48:22.
Ncat: 0 bytes sent, 0 bytes received in 0.01 seconds.
sh-4.2$
```

Exit the Session Manager Session

```
exit
```

Challenge

TODO

Clean Up

Delete the SSM Parameters that you created

```
aws ssm delete-parameter --name "Cookbook206Instance2Ip"
```

Delete the analyses:

```
aws ec2 delete-network-insights-analysis \
        --network-insights-analysis-id $ANALYSIS_ID_1
```

```
aws ec2 delete-network-insights-analysis \
        --network-insights-analysis-id $ANALYSIS_ID_2
```

Delete the path:

```
aws ec2 delete-network-insights-path \
        --network-insights-path-id $INSIGHTS_PATH_ID
```

To clean up the environment variables, run the helper.py script in this recipe's cdk- directory with the --unset flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Unset the environment variable that you created manually

```
unset INSIGHTS_PATH_ID
unset ANALYSIS_ID_1
unset ANALYSIS_ID_2
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && rm -r .venv/ && cd ../..
```

Discussion

You configured a network insights path which specified TCP port 22 between two EC2 instances. A network insights path is a definition of what connectivity you want to test. Next you performed a network insights analysis for the path and reviewed the results. Initially, there wasn't ssh connectivity between the instances because the security group on this destination (instance 2) did not allow access. After you updated the security group associated with instance 2 and reran and analysis, you were able to verify successful connectivity. Using the VPC Reachability Analyzer is an effective capability for network troubleshooting and validating configuration in a "serverless" manner; it does not require you to provision infrastructure to analyze, verify, and troubleshoot network connectivity.

> **NOTE**
>
> VPC reachability has broad support of sources and destinations for resources within your VPCs. For a complete list of supported sources and destinations, see this support document.

VPC Reachability Analyzer provides explanation codes that describe the result of a network path analysis. In this recipe, you observed the code ENI_SG_RULES_MISMATCH which indicates that the security groups are not allowing traffic between the source and destination. The next recipe contains more information on modifying security groups. The complete list of explanation codes are available in the Amazon VIrtual Private Cloud documentation.

1.7 Redirecting HTTP Traffic to HTTPS with an Application Load Balancer

Problem

You have a containerized application running in a private subnet. Users on the internet need to access this application. To help secure the application, you would like to redirect all requests from HTTP to HTTPS.

Solution

Create an Application Load Balancer. Next create Listeners for port 80 and 443, target groups for your containerized application, and Listener rules. Configure the Listener rules to send traffic to your target group for port 443. Finally, configure an action to redirect with a HTTP 301 response code to port 443 while preserving the URL in the request.

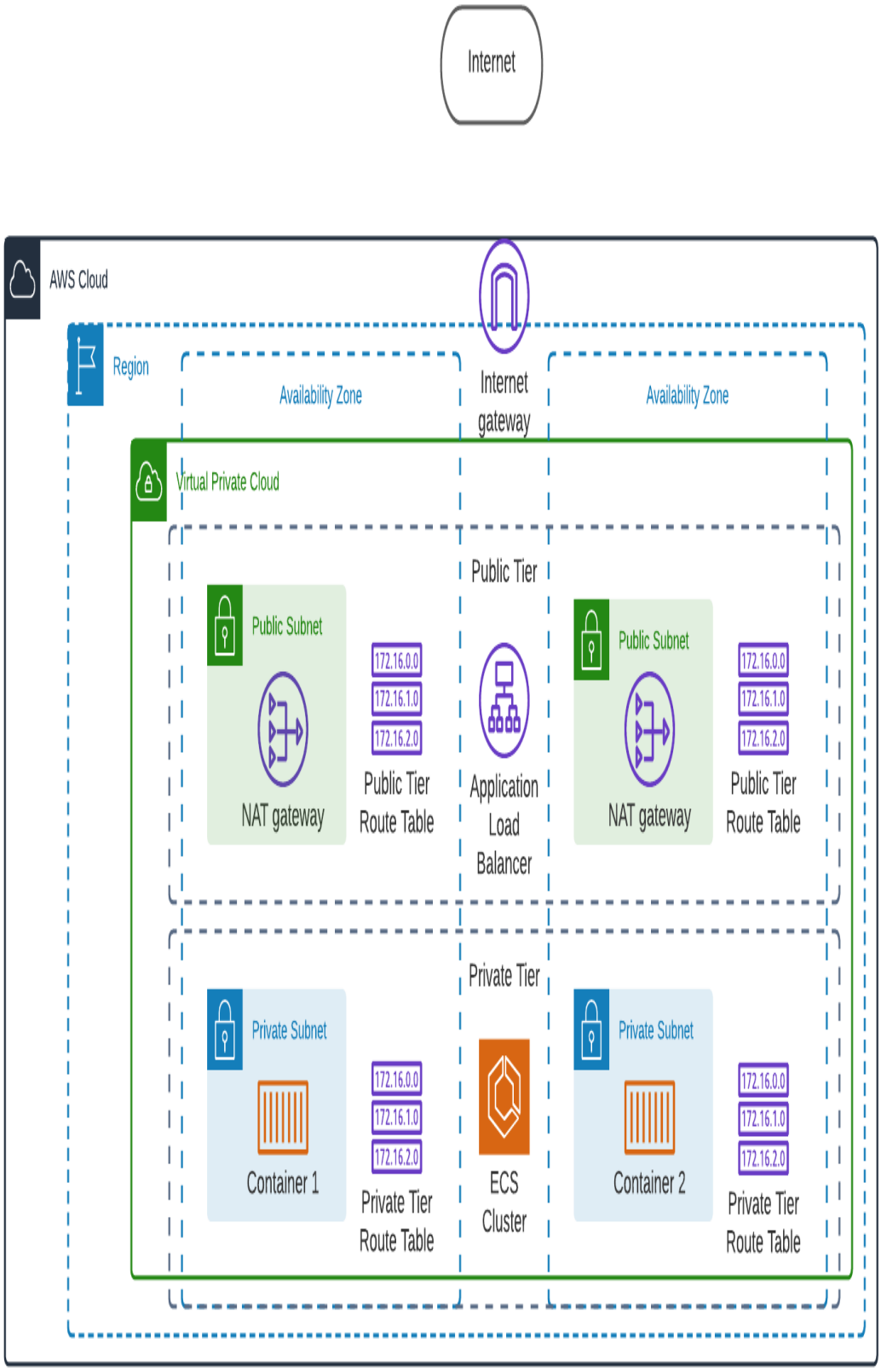

Internet
AWS Cloud
Region
Availability Zone
Internet gateway
Availability Zone
Virtual Private Cloud
Public Tier
Public Subnet
NAT gateway
172.16.0.0
172.16.1.0
172.16.2.0
Public Tier Route Table
Application Load Balancer
Public Subnet
NAT gateway
172.16.0.0
172.16.1.0
172.16.2.0
Public Tier Route Table
Private Tier
Private Subnet
Container 1
172.16.0.0
172.16.1.0
172.16.2.0
Private Tier Route Table
ECS Cluster
Private Subnet
Container 2
172.16.0.0
172.16.1.0
172.16.2.0
Private Tier Route Table

Figure 1-9. VPC with ALB serving internet traffic to containers in private subnets

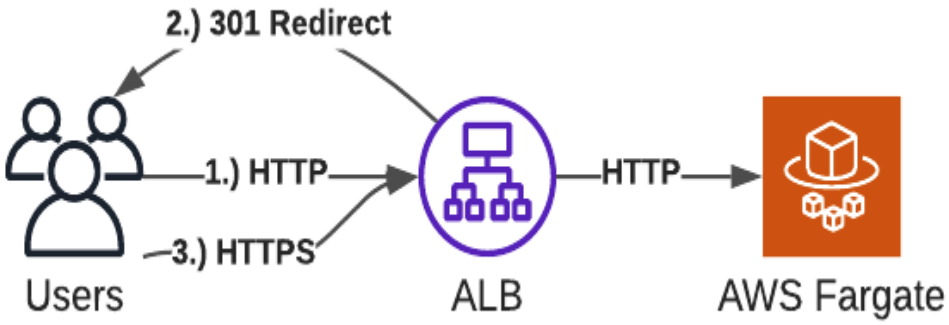


Figure 1-10. Redirecting HTTP to HTTPs with an ALB

Prerequisites

- VPC with public subnets in 2 AZs and associated route tables
- Private subnets created in 2 AZs and associated route tables
- An ECS Cluster and container definition exposing a web application on port 80
- A Fargate service which runs two tasks on the ECS cluster
- OpenSSL (you can install this using `brew install openssl` or `yum install openssl`)

Preparation

This recipe requires some "prep work" which deploys resources that you'll build the solution on. You will use the AWS CDK to deploy these resources

In the root of this Chapter's repo **cd** to the "207-Using-Load-Balancers-for-HTTPS-Redirection/cdk-AWS-Cookbook-207" directory and follow the subsequent steps:

```
cd 207-Using-Load-Balancers-for-HTTPS-Redirection/cdk-AWS-Cookbook-
207
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt --no-dependencies
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

Navigate up to the main directory for this recipe (out of the “cdk-AWS-Cookbook-207” directory)

```
cd ..
```

Modify the VPCPublicSubnets variable to work with the commands below

```
VPC_PUBLIC_SUBNETS=$(echo $VPC_PUBLIC_SUBNETS | tr -d ',"')
```

Steps

Create a new private key to be used for the certificate.

```
openssl genrsa 2048 > my-private-key.pem
```

Generate a self-signed certificate using OpenSSL CLI

```
openssl req -new -x509 -nodes -sha256 -days 365 \
        -key my-private-key.pem -outform PEM -out my-
certificate.pem
```

> **NOTE**
>
> You are using a self-signed certificate for this recipe, which will throw a warning when you access the Load Balancer DNS name in most browsers. You can generate a trusted certificate for your own DNS record by using AWS Certificate Manager (ACM): *https://docs.aws.amazon.com/acm/latest/userguide/acm-overview.html*

Upload the generated certificate into IAM:

```
CERT_ARN=$(aws iam upload-server-certificate \
        --server-certificate-name AWSCookbook207 \
        --certificate-body file://my-certificate.pem \
        --private-key file://my-private-key.pem \
        --query ServerCertificateMetadata.Arn --output text)
```

Create a security group to use with the ALB that you will create later

```
ALB_SG_ID=$(aws ec2 create-security-group --group-name
Cookbook207SG \
        --description "ALB Security Group" --vpc-id $VPC_ID \
        --output text --query GroupId)
```

Add rules to the Security group to allow HTTP and HTTPS traffic from the world

```
aws ec2 authorize-security-group-ingress \
        --protocol tcp --port 443 \
        --cidr '0.0.0.0/0' \
        --group-id $ALB_SG_ID
aws ec2 authorize-security-group-ingress \
        --protocol tcp --port 80 \
        --cidr '0.0.0.0/0' \
        --group-id $ALB_SG_ID
```

Authorize the EC2 Instance's security group to allow ingress traffic from the ALB

```
aws ec2 authorize-security-group-ingress \
        --protocol tcp --port 80 \
        --source-group $ALB_SG_ID \
        --group-id $APP_SG_ID
```

> **NOTE**
>
> Todo - talk about best practice to only allow traffic to target from the ALB SG

Create an ALB across the public subnets and assign it the previously created security group

```
LOAD_BALANCER_ARN=$(aws elbv2 create-load-balancer \
        --name aws-cookbook207-alb  \
        --subnets $VPC_PUBLIC_SUBNETS --security-groups $ALB_SG_ID
\
        --scheme internet-facing \
        --output text --query LoadBalancers[0].LoadBalancerArn)
```

Create target groups for the Load Balancer

```
TARGET_GROUP=$(aws elbv2 create-target-group \
        --name aws-cookbook207-tg --vpc-id $VPC_ID \
        --protocol HTTP --port 80 --target-type ip \
        --query "TargetGroups[0].TargetGroupArn" \
        --output text)
```

Register the EC2 Instance with the target group:

```
aws elbv2 register-targets --targets Id=$CONTAINER_IP_1 \
        --target-group-arn $TARGET_GROUP
```

Create an HTTPS listener on the ALB that uses the certificate you imported and forwards traffic to your target group

```
HTTPS_LISTENER_ARN=$(aws elbv2 create-listener \
        --load-balancer-arn $LOAD_BALANCER_ARN \
        --protocol HTTPS --port 443  \
        --certificates CertificateArn=$CERT_ARN \
        --default-actions Type=forward,TargetGroupArn=$TARGET_GROUP
\
        --output text --query Listeners[0].ListenerArn)
```

Add a rule for the Listener on port 443 to forward traffic to the target group that you created

```
aws elbv2 create-rule \
    --listener-arn $HTTPS_LISTENER_ARN \
    --priority 10 \
    --conditions        '{"Field":"path-
pattern","PathPatternConfig":{"Values":["/*"]}}' \
    --actions Type=forward,TargetGroupArn=$TARGET_GROUP
```

Create a redirect response for all HTTP traffic which sends a 301 response to the browser while preserving the full URL for the HTTPS redirect:

```
aws elbv2 create-listener --load-balancer-arn $LOAD_BALANCER_ARN \
        --protocol HTTP --port 80  \
        --default-actions "Type=redirect,RedirectConfig=
{Protocol=HTTPS,Port=443,Host='#{host}',Query='#{query}',Path='/#
{path}',StatusCode=HTTP_301}"
```

Verify the health of the targets:

```
aws elbv2 describe-target-health --target-group-arn $TARGET_GROUP \
        --query TargetHealthDescriptions[*].TargetHealth.State
```

Validation Steps

Get the URL of the Load balancer so that you can test it

```
LOAD_BALANCER_DNS=$(aws elbv2 describe-load-balancers \
        --names aws-cookbook207-alb \
        --output text --query LoadBalancers[0].DNSName)
```

Display the URL and test it in your browser. You should notice that you end up at a https URL. You will most likely receive a warning from your browser because of the self-signed cert.

```
echo $LOAD_BALANCER_DNS
```

Or Test from the command line

cURL the Load Balancer DNS over HTTP and observe the 301 code

```
curl -v http://$LOAD_BALANCER_DNS
```

cURL the Load Balancer DNS and specify to follow the redirect to HTTPS

```
curl -vkL http://$LOAD_BALANCER_DNS
```

Clean Up

Delete the ALB

```
aws elbv2 delete-load-balancer --load-balancer-arn
$LOAD_BALANCER_ARN
```

Revoke the security group authorization for the ECS security group

```
aws ec2 revoke-security-group-ingress \
        --protocol tcp --port 80 \
        --source-group $ALB_SG_ID \
        --group-id $APP_SG_ID
```

When the ALB is done deleting, delete the ALB security group

```
aws ec2 delete-security-group --group-id $ALB_SG_ID
```

Delete the target groups

```
aws elbv2 delete-target-group --target-group-arn $TARGET_GROUP
```

Delete the certificate

```
aws iam delete-server-certificate \
        --server-certificate-name AWSCookbook207
```

Go to the cdk-AWS-Cookbook-207 directory

```
cd cdk-AWS-Cookbook-207/
```

To clean up the environment variables, run the `helper.py` script in this recipe's `cdk-` directory with the `--unset` flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Unset your manually created environment variables

```
unset TARGET_GROUP
unset ALB_SG_ID
unset CERT_ARN
unset LOAD_BALANCER_ARN
unset HTTPS_LISTENER_ARN
unset CONTAINER_IP_1
unset CONTAINER_IP_2
unset TASK_ARN_1
unset TASK_ARN_2
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && rm -r .venv/ && cd ../..
```

Discussion

You created a self-signed SSL certificate and imported it into your AWS account. You also created a security group with authorizations to allow ingress traffic on TCP ports 80 and 443. Then, you created an Application Load Balancer (ALB) in your VPC public subnets and associated the security group with the ALB. Next you created and configured a Target Group, added a container-based web application running on ECS Fargate to the Target it created two listeners (one for port 80 HTTP and the other for port 443 HTTPS), and listener rules to send traffic to your desired locations. You added a 301 redirect rule for the port 80 listener. This allows the ALB to instruct clients to follow the redirect to port 443 so that users of your application will be automatically redirected to HTTPS. The redirect rule also preserves the URL path in the original request.

Application Load Balancers (ALBs) operate on Layer 7 of the OSI model. The ALB documentation lists the available target types of: EC2 instances, IP addresses, and Lambda functions. You can create internet-facing ALBs (when your VPC has an Internet Gateway attached) and internal ALBs for usage within your internal network only. The ALB provisions Elastic Network Interfaces (ENIs) which have IP addresses within your chosen subnets to communicate with your services. ALBs continuously run health checks for members of your associated target groups that allow the ALB to detect healthy components of your application to route traffic to. ALBs are also a great layer to add in front of your applications for increased security, since you can only allow the targets to be accessed by the load balancer, and not by clients directly.

AWS offers multiple types of load balancers for specific use cases. You should choose the load balancer that best fits your needs. For example, for high-performance Layer 4 load balancing with static IP address capability, you might consider Network Load Balancers, and for Virtual Network Appliances like virtual firewalls and security appliances, you might consider Gateway Load Balancers. For more information on the different types of load balancers available in AWS and a comparison between the offerings, see the following support document.

1.8 Simplifying Management of CIDRs in Security Groups with Prefix Lists

Problem

You have two applications hosted in public subnets. The applications are hosted on instances with specific access requirements for each application. During normal operation, these applications need to be accessed from virtual desktops in another region. However during testing, you need to reach them from your home PC.

Solution

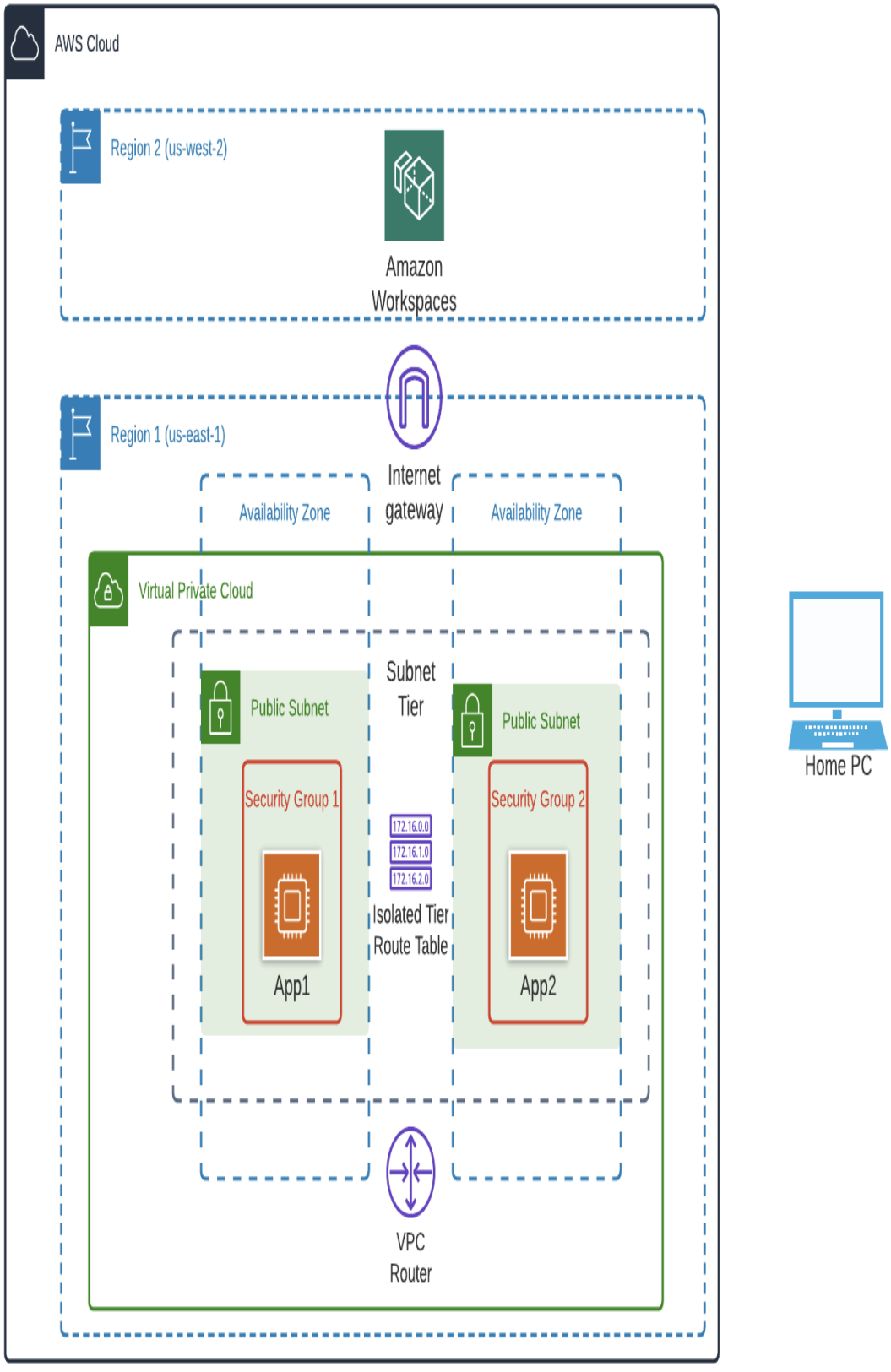

AWS Cloud
Region 2 (us-west-2)
Amazon
Workspaces
Region 1 (us-east-1)
Internet
gateway
Availability Zone
Availability Zone
Virtual Private Cloud
Subnet
Tier
Public Subnet
Public Subnet
Security Group 1
Security Group 2
172.16.0.0
172.16.1.0
172.16.2.0
Isolated Tier
Route Table
App1
App2
VPC
Router
Home PC

Figure 1-11. Two applications in public subnets protected by security groups

Using the AWS provided IP address ranges list, create a managed prefix list that contains a list of CIDR ranges for Workspaces Gateways in us-west-2 and associate it with each security group. Update the prefix list with your home IP for testing and then optionally remove it.

Prerequisites

- VPC with public subnets in 2 AZs and associated route tables
- Two EC2 instances in each public subnet running a web server on port 80
- Two security groups, one associated with each EC2 instance

Preparation

This recipe requires some "prep work" which deploys resources that you'll build the solution on. You will use the AWS CDK to deploy these resources

In the root of this Chapter's repo **cd** to the "208-Leveraging-Managed-Prefix-Lists/cdk-AWS-Cookbook-208" directory and follow the subsequent steps:

- cd 208-Leveraging-Managed-Prefix-Lists/cdk-AWS-Cookbook-208
- test -d .venv || python3 -m venv .venv
- source .venv/bin/activate
- pip install --upgrade pip setuptools wheel
- pip install -r requirements.txt --no-dependencies
- cdk deploy

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

Navigate up to the main directory for this recipe (out of the “cdk-AWS-Cookbook-208” directory)

```
cd ..
```

Download the AWS IP address ranges JSON file

(More info: https://docs.aws.amazon.com/general/latest/gr/aws-ip-ranges.html#aws-ip-egress-control)

```
curl -o ip-ranges.json https://ip-ranges.amazonaws.com/ip-ranges.json
```

Generate a list of the CIDR Ranges for Workspaces Gateways in us-west-2

(Note you will need to install the jq utility if your workstation doesn’t already have it - E.g: `brew install jq`)

```
jq -r '.prefixes[] | select(.region=="us-west-2") | select(.service=="WORKSPACES_GATEWAYS") | .ip_prefix' < ip-ranges.json
```

Steps

Use the IP ranges for Amazon CloudFront from `ip-ranges.json` to create a Managed Prefix list

```
PREFIX_LIST_ID=$(aws ec2 create-managed-prefix-list \
        --address-family IPv4 \
        --max-entries 15 \
        --prefix-list-name allowed-us-east-1-cidrs \
        --output text --query "PrefixList.PrefixListId" \
--entries
Cidr=44.234.54.0/23,Description=workspaces-us-west-2-cidr1
Cidr=54.244.46.0/23,Description=workspaces-us-west-2-cidr2)
```

> **NOTE**
>
> At this point your workstation should not be able to reach either of the instances. If you try a command below you will receive a “Connection timed out” error.
>
> ```
> curl -m 2 $INSTANCE_IP_1
> curl -m 2 $INSTANCE_IP_2
> ```

Get your workstation’s public IPv4 address

```
MY_IP_4=$(curl myip4.com | tr -d ' ')
```

Update your Managed Prefix List and add your workstation’s public IPv4 address

```
aws ec2 modify-managed-prefix-list \
        --prefix-list-id $PREFIX_LIST_ID \
        --current-version 1 \
        --add-entries Cidr=${MY_IP_4}/32,Description=my-
workstation-ip
```

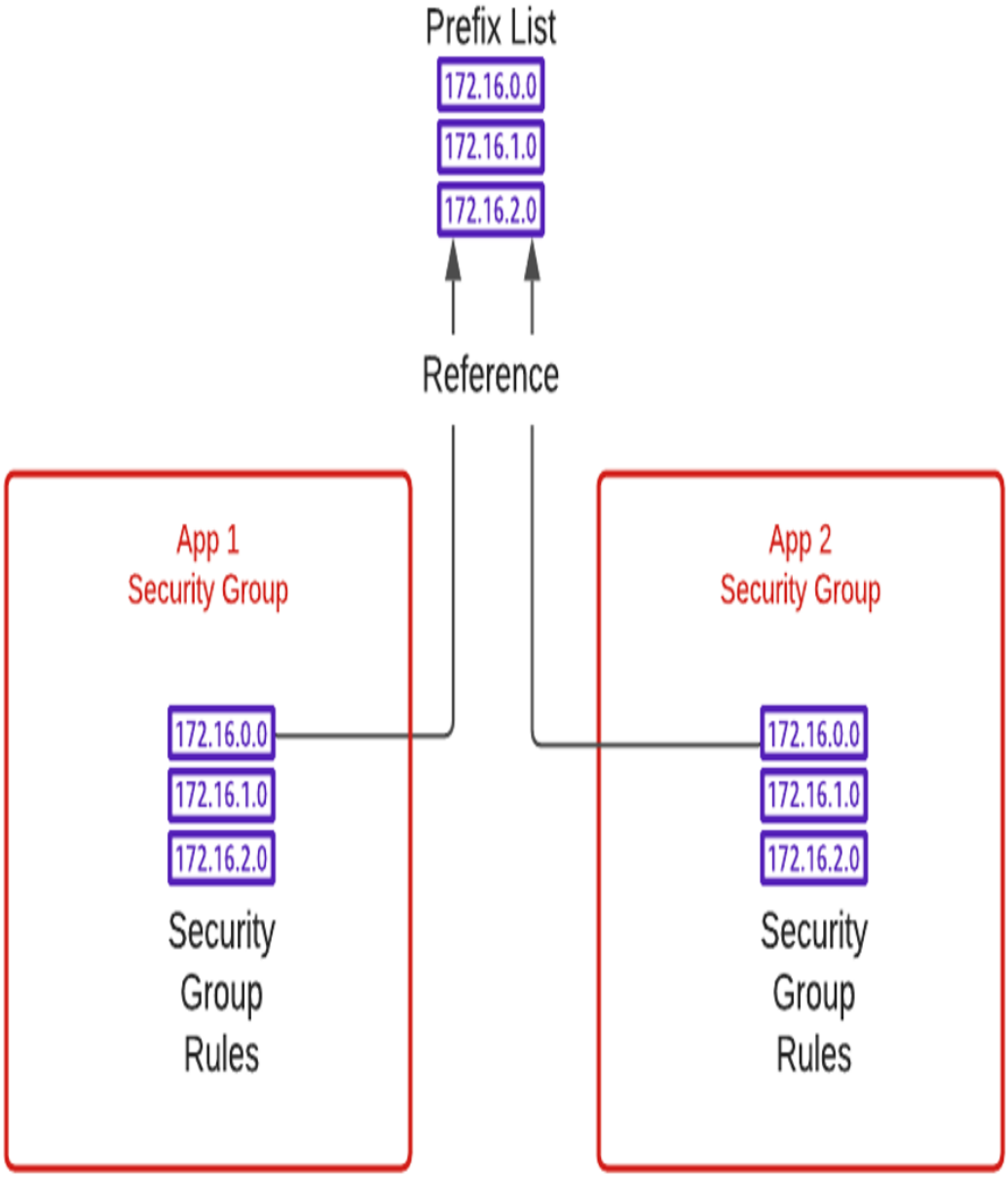


Figure 1-12. Security group rules referencing a Prefix list

> **NOTE**
>
> There is an AWS provided Managed Prefix list for S3 but we are going to combine these two ranges into one security group rule so that we don't hit the quota on the number of rules and / or routes.

Output

```
{
    "PrefixList": {
        "PrefixListId": "pl-013217b85144872d2",
        "AddressFamily": "IPv4",
        "State": "modify-in-progress",
        "PrefixListArn": "arn:aws:ec2:us-east-
1:111111111111:prefix-list/pl-013217b85144872d2",
        "PrefixListName": "allowed-us-east-1-cidrs",
        "MaxEntries": 10,
        "Version": 1,
        "OwnerId": "111111111111"
    }
}
```

For each Application's security group, add an inbound rule that allows TCP port 80 access from the prefix list.

```
aws ec2 authorize-security-group-ingress \
        --group-id $INSTANCE_SG_1 --ip-permissions \
IpProtocol=tcp,FromPort=80,ToPort=80,PrefixListIds="
[{Description=http-from-prefix-list,PrefixListId=$PREFIX_LIST_ID}]"
aws ec2 authorize-security-group-ingress \
        --group-id $INSTANCE_SG_2 --ip-permissions \
IpProtocol=tcp,FromPort=80,ToPort=80,PrefixListIds="
[{Description=http-from-prefix-list,PrefixListId=$PREFIX_LIST_ID}]"
```

> **NOTE**
>
> TIP Find out where your Managed List is used. This is command is helpful for auditing where prefix lists are used throughout your AWS environments
>
> ```
> aws ec2 get-managed-prefix-list-associations \
>     --prefix-list-id $PREFIX_LIST_ID
> ```

Validation Steps

Test Access to both instances from your workstation's PC

```
curl -m 2 $INSTANCE_IP_1
curl -m 2 $INSTANCE_IP_2
```

Challenge

Revert the active version of the Prefix list so that your workstation IP is removed and you can no longer access either application

Run this command with the prefix list version you would like to restore in --previous-version:

```
aws ec2 restore-managed-prefix-list-version \
        --prefix-list-id $PREFIX_LIST_ID \
        --previous-version 1 --current-version 2
```

Verify that you can no longer access the EC2 instances

```
curl -m 2 $INSTANCE_IP_1
curl -m 2 $INSTANCE_IP_2
```

Clean Up

Go to the cdk-AWS-Cookbook-208 directory

```
cd cdk-AWS-Cookbook-208/
```

To clean up the environment variables, run the `helper.py` script in this recipe's `cdk-` directory with the `--unset` flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Delete the Managed Prefix List

```
aws ec2 delete-managed-prefix-list \
    --prefix-list-id $PREFIX_LIST_ID
```

Unset your manually created environment variables

```
unset PREFIX_LIST_ID
unset MY_IP_4
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && rm -r .venv/ && cd ../..
```

Discussion

You created a managed prefix list with entries from a list of AWS service public CIDR ranges. Then, you associated the managed prefix list with two security group ingress rules for EC2 instances. The rules only allowed inbound communication with that prefix list for HTTP (TCP port 80).

As a result, CIDR ranges in the prefix list are only permitted to communicate inbound to port 80 with your EC2 instances. You then added your own workstation IP address to the prefix list and verified that the access was allowed by referencing the prefix list.

If you need to update the list of CIDR blocks allowing ingress communication to your instances, you can simply update the prefix list instead of the security group. This helps reduce the amount of maintenance overhead if you need to use this type of authorization across many security groups; you only need to update the prefix list in a single location rather than modify every security group authorization that requires this network security configuration. You can also use prefix lists for egress security group authorizations.

Prefix lists can be associated with route tables, and also useful for blackholing traffic (prohibit access to a specific list of IP addresses and CIDR blocks) and can also simplify your route table configuration. For example, you could maintain a prefix list of branch office CIDR ranges and use them to implement your routing and security group authorizations, simplifying your management for network flow and security configuration. An example of associating a prefix list with a route looks like:

```
aws ec2 create-route --route-table-id $Sub1RouteTableID \
--destination-prefix-list-id $PREFIX_LIST_ID \
--instance-id $INSTANCE_ID
```

Prefix lists also provide a powerful versioning mechanism, allowing you to roll back to previous known working states quickly. If, for example, you updated a prefix list and found that the change broke some existing functionality, you can roll back to a previous version of a prefix list to

restore previous functionality while you investigated the root cause of the error. If you decided to roll back to a previous version for some reason, first describe the prefix list to get the current version number:

```
aws ec2 describe-prefix-lists --prefix-list-ids $PREFIX_LIST_ID
```

1.9 Controlling Network Access to S3 from your VPC using VPC Endpoints

Problem

Your company's security team is worried about data exfiltration. Resources within your VPC should only be able to access a specific S3 bucket.. Also this S3 traffic should not traverse the internet and keep costs low.

Solution

You will create a Gateway VPC endpoint for S3, associate it with a route table, and customize it's policy document.

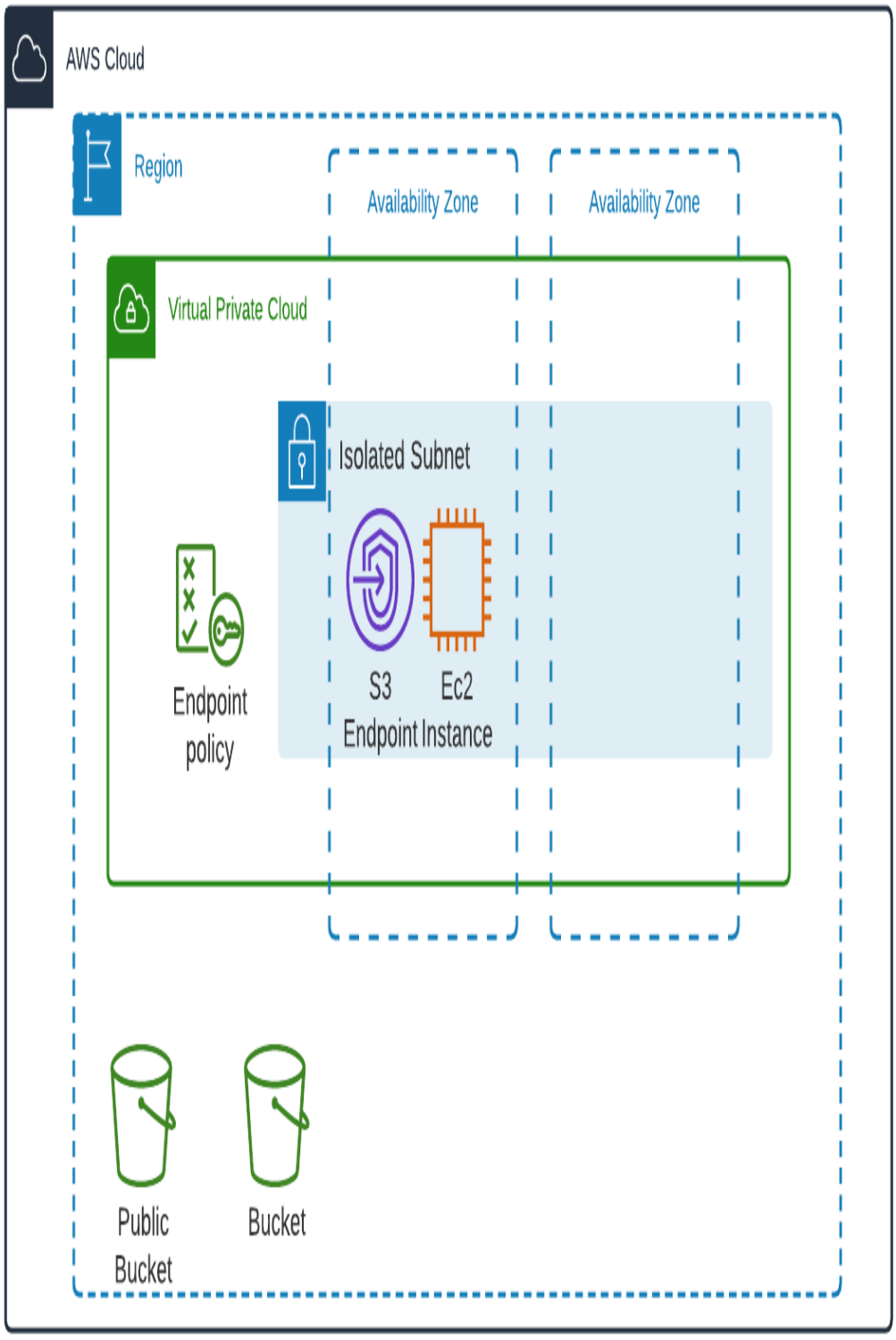

AWS Cloud
Region
Availability Zone
Availability Zone
Virtual Private Cloud
Isolated Subnet
Endpoint policy
S3 Endpoint
Ec2 Instance
Public Bucket
Bucket

Figure 1-13. Placeholder caption

Prerequisites

- VPC with isolated subnets in 2 AZs and associated route tables
- One EC2 instance in a public subnet that you can access for testing
- An existing S3 bucket that you want to limit access to

Preparation

This recipe requires some “prep work” which deploys resources that you’ll build the solution on. You will use the AWS CDK to deploy these resources

In the root of the Chapter 2 repo **cd** to the “209-Using-Gateway-VPC-Endpoints-with-S3/cdk-AWS-Cookbook-209” directory and follow the subsequent steps:

```
cd 209-Using-Gateway-VPC-Endpoints-with-S3/cdk-AWS-Cookbook-209/
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt --no-dependencies
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

Navigate up to the main directory for this recipe (out of the “cdk-AWS-Cookbook-209” directory)

```
cd ..
```

Steps

Create a gateway endpoint in your VPC and associate the endpoint with the isolated route tables

```
END_POINT_ID=$(aws ec2 create-vpc-endpoint \
        --vpc-id $VPC_ID \
        --service-name com.amazonaws.$AWS_REGION.s3 \
        --route-table-ids $RT_ID_1 $RT_ID_2 \
        --query VpcEndpoint.VpcEndpointId --output text)
```

Create a template endpoint policy file called `policy.json` with the following content (included in repo). This is used to limit access to only the S3 bucket that you created in the preparation steps.

```
{
  "Statement": [
    {
      "Sid": "RestrictToOneBucket",
      "Principal": "*",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::S3BucketName",
                   "arn:aws:s3:::S3BucketName/*"]
    }
  ]
}
```

Insert your S3_BUCKET_NAME in the policy-template.json file

```
sed -e "s/S3BucketName/${BUCKET_NAME}/g" \
        policy-template.json > policy.json
```

Modify the endpoint's policy document. Endpoint policies limit or restrict the resources which can be accessed through the VPC endpoint

```
aws ec2 modify-vpc-endpoint \
        --policy-document file://policy.json \
        --vpc-endpoint-id $END_POINT_ID
```

Validation Steps

Create and populate some SSM parameters to store values so that you can retrieve them from your EC2 instance

```
aws ssm put-parameter \
    --name "Cookbook209S3Bucket" \
    --type "String" \
    --value $BUCKET_NAME
```

Ensure your EC2 instance has registered with SSM. Use this command to check the status

```
aws ssm describe-instance-information \
        --filters Key=ResourceType,Values=EC2Instance \
        --query "InstanceInformationList[].InstanceId" --output
text
```

Connect to your EC2 instance using SSM Session Manager

```
aws ssm start-session --target $INSTANCE_ID
```

Set the region by grabbing the value from the instance's metadata

```
export AWS_DEFAULT_REGION=$(curl --silent
http://169.254.169.254/latest/dynamic/instance-identity/document \
| awk -F'"' ' /region/ {print $4}')
```

Retrieve the allowed S3 bucket name

```
BUCKET=$(aws ssm get-parameters \
    --names "Cookbook209S3Bucket" \
        --query "Parameters[*].Value" --output text)
```

Test access by trying to copy a file from the S3 bucket

```
aws s3 cp s3://${BUCKET}/test_file /home/ssm-user/
```

Output:

```
download: s3://cdk-aws-cookbook-209-awscookbookrecipe20979239201-
115xoj77fgxoh/test_file to ./test_file
```

> **NOTE**
>
> The command below is attempting to list a public S3 bucket. However because of the endpoint policy that we have configured, it is expected that this will fail.

Try to list the contents of a ***public*** S3 bucket associated with the OpenStreetMap Foundation Public Dataset Initiative

```
aws s3 ls s3://osm-pds/
```

Output:

```
An error occurred (AccessDenied) when calling the ListObjectsV2
operation: Access Denied
```

Exit the Session Manager Session

```
exit
```

Challenge

Modify the bucket policy for the S3 bucket, only allow access from the VPC endpoint that you created. For some tips on this, check out the S3 user guide.

Clean Up

Delete the SSM Parameter that you created

```
aws ssm delete-parameter --name "Cookbook209S3Bucket"
```

Delete the VPC Endpoint

```
aws ec2 delete-vpc-endpoints --vpc-endpoint-ids $END_POINT_ID
```

Go to the cdk-AWS-Cookbook-209 directory

```
cd cdk-AWS-Cookbook-209/
```

To clean up the environment variables, run the `helper.py` script in this recipe's `cdk-` directory with the `--unset` flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Unset your manually created environment variables

```
unset END_POINT_ID
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && rm -r .venv/ && cd ../..
```

Discussion

You created a Gateway VPC Endpoint and associated an Endpoint Policy which only permits access to a specific S3 bucket. This is a useful security implementation to restrict access to S3 buckets. This applies not only to S3

bucket owned by your account but for all S3 buckets globally on AWS. When you used the aws s3 cp command to interact with the bucket specified in your endpoint policy, you saw that the operation was successful. Conversely, when you tried to access a public S3 bucket in another AWS account, you saw the operation was unsuccessful.

> **NOTE**
>
> TIP Recently AWS announced support for S3 interface endpoints. However it is worth noting that while these are great for some use cases (e.g when you want to control traffic with security groups), they are not ideal for this problem because of the costs associated with interface end points.

Per the VPC user guide, Gateway VPC Endpoints are free and used within your VPC's route tables to keep traffic bound for AWS services within the AWS backbone network without traversing the network. This allows you to create VPCs that do not need Internet Gateways for applications which do not require them, but need access to other AWS services like S3 and DynamoDB. All traffic bound for these services will be directed by the route table to the VPC Endpoint rather than the public internet route, since the VPC Endpoint route table entry is more specific than the default 0.0.0.0/0 route.

S3 VPC Endpoint policies leverage JSON policy documents which can be as fine-grained as your needs require. You can use conditionals, source IP addresses, VPC Endpoint IDs, S3 bucket names, and more. For more information on the policy elements available, see this support document.

1.10 Enabling Transitive Cross-VPC Connections using Transit Gateway

Problem

Option1: You need to implement transitive routing across all of your VPCs and share internet egress from a shared services VPC to your other VPCs to

reduce the amount of NAT Gateways you have to deploy.

OR

Option2: You need all of your VPCs to be able to communicate with one and other and would like to minimize the amount of VPC peering connections that you would have to manage to accomplish this with VPC peering. You would also like to share NAT gateways with connected VPCs to reduce costs.

OR

Option3: From the docs “You need to connect your VPCs and on-premises networks through a central hub” although we don’t connect on-prem/VPN so maybe not

Solution

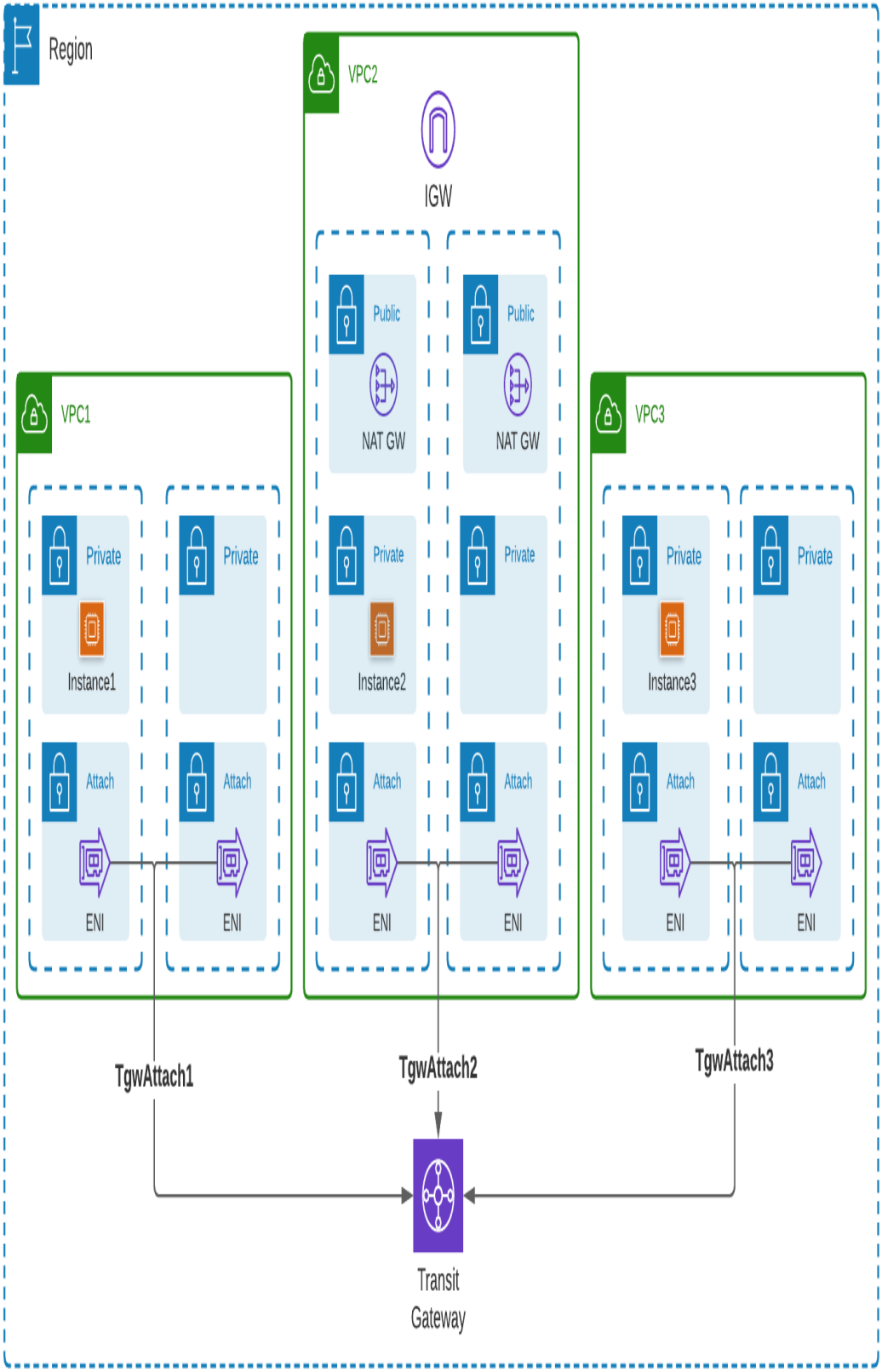

Region
VPC2
IGW
Public
NAT GW
Public
NAT GW
VPC1
VPC3
Private
Instance1
Private
Private
Instance2
Private
Private
Instance3
Private
Attach
ENI
Attach
ENI
Attach
ENI
Attach
ENI
Attach
ENI
Attach
ENI
TgwAttach1
TgwAttach2
TgwAttach3
Transit
Gateway

Figure 1-14. Transit Gateway with 3 VPCs

Deploy a Transit Gateway (TGW) and configure Transit Gateway VPC Attachments for all of your VPCs. Update your VPC route tables of each VPC to send all non-local traffic to the Transit Gateway and enable sharing of the NAT Gateway in your shared services VPC for all of your spoke VPCs.

> **WARNING**
>
> The default initial quota of VPCs per region per account is 5. This solution will deploy 3 VPCs. If you already have more than 2 VPCs, you can decide between 3 choices: Deploy to a different region, delete any existing VPCs that are no longer needed, or request a quota increase.

Prerequisites

- 3 VPCs in the same region with private and isolated subnet tiers
- Internet Gateway attached to a VPC (VPC #2 in our example)

  — NAT Gateway deployed in Public subnets

Preparation

This recipe requires some "prep work" which deploys resources that you'll build the solution on. You will use the AWS CDK to deploy these resources

In the root of this Chapter's repo **cd** to the "210-Using-a-Transit-Gateway/cdk-AWS-Cookbook-210" directory and follow the subsequent steps:

```
cd 210-Using-a-Transit-Gateway/cdk-AWS-Cookbook-210/
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt --no-dependencies
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

For this recipe, you will need to create some modified environment variables to use:

```
ATTACHMENT_SUBNETS_VPC_1=$(echo ${ATTACHMENT_SUBNETS_VPC_1} | tr -d
','"')
ATTACHMENT_SUBNETS_VPC_2=$(echo ${ATTACHMENT_SUBNETS_VPC_2} | tr -d
','"')
ATTACHMENT_SUBNETS_VPC_3=$(echo ${ATTACHMENT_SUBNETS_VPC_3} | tr -d
','"')
```

Steps

Create a Transit Gateway.

```
TGW_ID=$(aws ec2 create-transit-gateway \
        --description AWSCookbook210 \
        --
options=AmazonSideAsn=65010,AutoAcceptSharedAttachments=enable,Defa
ultRouteTableAssociation=enable,DefaultRouteTablePropagation=enable
,VpnEcmpSupport=enable,DnsSupport=enable \
        --output text --query TransitGateway.TransitGatewayId)
```

Wait until the Transit Gateway's state has reached "available". This may take several minutes

```
aws ec2 describe-transit-gateways \
        --transit-gateway-ids $TGW_ID \
        --output text --query TransitGateways[0].State
```

Create a Transit Gateway attachment for the VPC 1

```
TGW_ATTACH_1=$(aws ec2 create-transit-gateway-vpc-attachment \
        --transit-gateway-id $TGW_ID \
        --vpc-id $VPC_ID_1 \
        --subnet-ids $ATTACHMENT_SUBNETS_VPC_1 \
        --query
TransitGatewayVpcAttachment.TransitGatewayAttachmentId \
        --output text)
```

Create a Transit Gateway attachment for the VPC 2

```
TGW_ATTACH_2=$(aws ec2 create-transit-gateway-vpc-attachment \
        --transit-gateway-id $TGW_ID \
```

```
        --vpc-id $VPC_ID_2 \
        --subnet-ids $ATTACHMENT_SUBNETS_VPC_2 \
        --query
TransitGatewayVpcAttachment.TransitGatewayAttachmentId \
        --output text)
```

Create a Transit Gateway attachment for the VPC 3

```
TGW_ATTACH_3=$(aws ec2 create-transit-gateway-vpc-attachment \
        --transit-gateway-id $TGW_ID \
        --vpc-id $VPC_ID_3 \
        --subnet-ids $ATTACHMENT_SUBNETS_VPC_3 \
        --query
TransitGatewayVpcAttachment.TransitGatewayAttachmentId \
        --output text)
```

Add route for all private subnets in VPCs 1 and 3 to target theTGW for destinations of 0.0.0.0/0 This enables consolidated internet egress through the NAT Gateway in VPC2 and transitive routing to other VPCs.

```
aws ec2 create-route --route-table-id $VPC_1_RT_ID_1 \
        --destination-cidr-block 0.0.0.0/0 \
        --transit-gateway-id $TGW_ID
aws ec2 create-route --route-table-id $VPC_1_RT_ID_2 \
        --destination-cidr-block 0.0.0.0/0 \
        --transit-gateway-id $TGW_ID
aws ec2 create-route --route-table-id $VPC_3_RT_ID_1 \
        --destination-cidr-block 0.0.0.0/0 \
        --transit-gateway-id $TGW_ID
aws ec2 create-route --route-table-id $VPC_3_RT_ID_2 \
        --destination-cidr-block 0.0.0.0/0 \
        --transit-gateway-id $TGW_ID
```

Now add a route to your 10.10.0.0/24 supernet in the Private subnets VPC2, pointing its destination to the Transit Gateway. This is more specific than the 0.0.0.0/0 destination this is already present and therefore takes higher priority in routing decisions. This directs traffic bound for VPCs 1, 2 and 3 to the TGW:

```
aws ec2 create-route --route-table-id $VPC_2_RT_ID_1 \
        --destination-cidr-block 10.10.0.0/24 \
        --transit-gateway-id $TGW_ID
aws ec2 create-route --route-table-id $VPC_2_RT_ID_2 \
        --destination-cidr-block 10.10.0.0/24 \
        --transit-gateway-id $TGW_ID
```

Query for the Nat Gateways in use, we'll need these to add routes to them for internet traffic.

```
NAT_GW_ID_1=$(aws ec2 describe-nat-gateways \
        --filter "Name=subnet-id,Values=$VPC_2_PUBLIC_SUBNET_ID_1"
\
        --output text --query NatGateways[*].NatGatewayId)
NAT_GW_ID_2=$(aws ec2 describe-nat-gateways \
        --filter "Name=subnet-id,Values=$VPC_2_PUBLIC_SUBNET_ID_2"
\
        --output text --query NatGateways[*].NatGatewayId)
```

Add a route for the Attachment subnet in VPC2 to direct internet traffic to the NAT Gateway

```
aws ec2 create-route --route-table-id $VPC_2_ATTACH_RT_ID_1 \
        --destination-cidr-block 0.0.0.0/0 \
        --nat-gateway-id $NAT_GW_ID_1
aws ec2 create-route --route-table-id $VPC_2_ATTACH_RT_ID_2 \
        --destination-cidr-block 0.0.0.0/0 \
        --nat-gateway-id $NAT_GW_ID_2
```

Add a static route to the route tables associated with the public subnet in VPC2. The enables communication back to the TGW to enable sharing the NAT Gateway with all attached VPCs

```
aws ec2 create-route --route-table-id $VPC_2_PUBLIC_RT_ID_1 \
        --destination-cidr-block 10.10.0.0/24 \
        --transit-gateway-id $TGW_ID
aws ec2 create-route --route-table-id $VPC_2_PUBLIC_RT_ID_2 \
        --destination-cidr-block 10.10.0.0/24 \
        --transit-gateway-id $TGW_ID
```

Add a static route for the private subnets in VPC2 to allow communication back to the TGW attachments from VPC2 private subnets

```
aws ec2 create-route --route-table-id $VPC_2_RT_ID_1 \
        --destination-cidr-block 10.10.0.0/24 \
        --transit-gateway-id $TGW_ID

aws ec2 create-route --route-table-id $VPC_2_RT_ID_2 \
        --destination-cidr-block 10.10.0.0/24 \
        --transit-gateway-id $TGW_ID
```

Get the Transit route table ID

```
TRAN_GW_RT=$(aws ec2 describe-transit-gateways \
        --transit-gateway-ids $TGW_ID --output text \
```

```
        --query
TransitGateways[0].Options.AssociationDefaultRouteTableId)
```

Add a static route in the Transit Gateway route table for VPC 2 (with the NAT Gateways) to send all internet traffic over this path

```
aws ec2 create-transit-gateway-route \
  --destination-cidr-block 0.0.0.0/0 \
  --transit-gateway-route-table-id $TRAN_GW_RT \
  --transit-gateway-attachment-id $TGW_ATTACH_2
```

Validation Steps:

Ensure your EC2 instance #1 has registered with SSM. Use this command to check the status. This command should return the instance ID

```
aws ssm describe-instance-information \
        --filters Key=ResourceType,Values=EC2Instance \
        --query "InstanceInformationList[].InstanceId" --output
text
```

Connect to your EC2 instance using SSM Session Manager

```
aws ssm start-session --target $INSTANCE_ID_1
```

Test internet access

```
ping -c 4 aws.amazon.com
Output:
PING dr49lng3n1n2s.cloudfront.net (99.86.187.73) 56(84) bytes of
data.
64 bytes from server-99-86-187-73.iad79.r.cloudfront.net
(99.86.187.73): icmp_seq=1 ttl=238 time=3.44 ms
64 bytes from server-99-86-187-73.iad79.r.cloudfront.net
(99.86.187.73): icmp_seq=2 ttl=238 time=1.41 ms
64 bytes from server-99-86-187-73.iad79.r.cloudfront.net
(99.86.187.73): icmp_seq=3 ttl=238 time=1.43 ms
64 bytes from server-99-86-187-73.iad79.r.cloudfront.net
(99.86.187.73): icmp_seq=4 ttl=238 time=1.44 ms
--- dr49lng3n1n2s.cloudfront.net ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.411/1.934/3.449/0.875 ms
sh-4.2$
```

Exit the Session Manager Session

```
exit
```

Challenge 1

You can limit which VPCs can access the internet through the NAT Gateway in VPC2 by modifying the route tables. Try adding a more specific route of 10.10.0.0/24 instead of the 0.0.0.0/0 destination for VPC3 to see how you can customize the internet egress sharing.

Challenge 2

You may not want to allow VPC1 and VPC3 to be able to communicate with each other. Try adding a new Transit Gateway route table updating the attachments to accomplish this.

Challenge 3

In the solution you deployed 3 VPCs each of /26 subnet size within the 10.10.0.0/24 supernet. There is room for an additional /26 subnet. Try adding an additional VPC with a /26 CIDR with subnets, route tables and attach it to the Transit Gateway.

Clean Up

Delete the Transit Gateway attachments. These take a moment to delete.

```
aws ec2 delete-transit-gateway-vpc-attachment \
        --transit-gateway-attachment-id $TGW_ATTACH_1
aws ec2 delete-transit-gateway-vpc-attachment \
        --transit-gateway-attachment-id $TGW_ATTACH_2
aws ec2 delete-transit-gateway-vpc-attachment \
-transit-gateway-attachment-id $TGW_ATTACH_3
```

After the Transit Gateway attachments have been deleted, delete the Transit Gateway

```
aws ec2 delete-transit-gateway --transit-gateway-id $TGW_ID
```

To clean up the environment variables, run the `helper.py` script in this recipe's `cdk-` directory with the `--unset` flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Unset your manually created environment variables

```
unset TRAN_GW_RT
unset TGW_ID
unset ATTACHMENT_SUBNETS_VPC_1
unset ATTACHMENT_SUBNETS_VPC_2
unset ATTACHMENT_SUBNETS_VPC_3
unset NAT_GW_ID_1
unset NAT_GW_ID_2
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && rm -r .venv/ && cd ../..
```

Discussion

You created a Transit Gateway (TGW), Transit Gateway route tables, and Transit Gateway Attachments. Then you modified your VPC route tables by adding a route to the Transit. Transit Gateways allow you to quickly implement a multi-VPC "hub and spoke" network topology for your network in AWS. In the past, you may have had to use many peering connections to achieve similar results or use 3rd party software on instances in a "Transit VPC" architecture. Transit Gateway also supports cross-region peering of Transit Gateways and cross-account sharing via Resource Access Manager (RAM).

When you attached your VPCs to the Transit Gateway, you used subnets in each Availability Zone for resiliency. You also used dedicated "Attachment" subnets for the VPC attachments. You can attach any subnet within your VPC to the Transit Gateway, but using a dedicated subnet gives you flexibility to granularly define subnets you choose to route to the TGW (ie. if you attached the private subnet, it would always have a route to the TGW; this might not be intended based on your use case). In your case, you configured routes for your private subnets to send all traffic to the transit gateway which enabled sharing of the NAT gateway and Internet Gateway which results in cost savings over having to deploy multiple NAT gateways (one for each VPC you might have).

You can connect your on-premises network or any virtual network directly to a transit gateway, as it acts as a hub for all of your AWS network traffic. You

can connect IPsec VPNs, Direct Connect (DX), SD-WAN, and 3rd party network appliances to the Transit Gateway to extend your AWS network to non-AWS networks. This also allows you to consolidate VPN connections and/or Direct Connect (DX) connections by connecting one directly to the Transit Gateway to access all of your VPCs in a region. Border Gateway Protocol (BGP) is supported over these types of network extensions for dynamic route updates in both directions.

1.11 Peering Two VPCs Together for Inter-VPC Network Communication

Problem

You need to enable two instances in separate VPCs to communicate with each other in a simple and cost effective manner.

Solution

Request a peering connection between two VPCs, accept the peering connection, update the route tables for each VPC subnet, and finally test the connection from one instance to another.

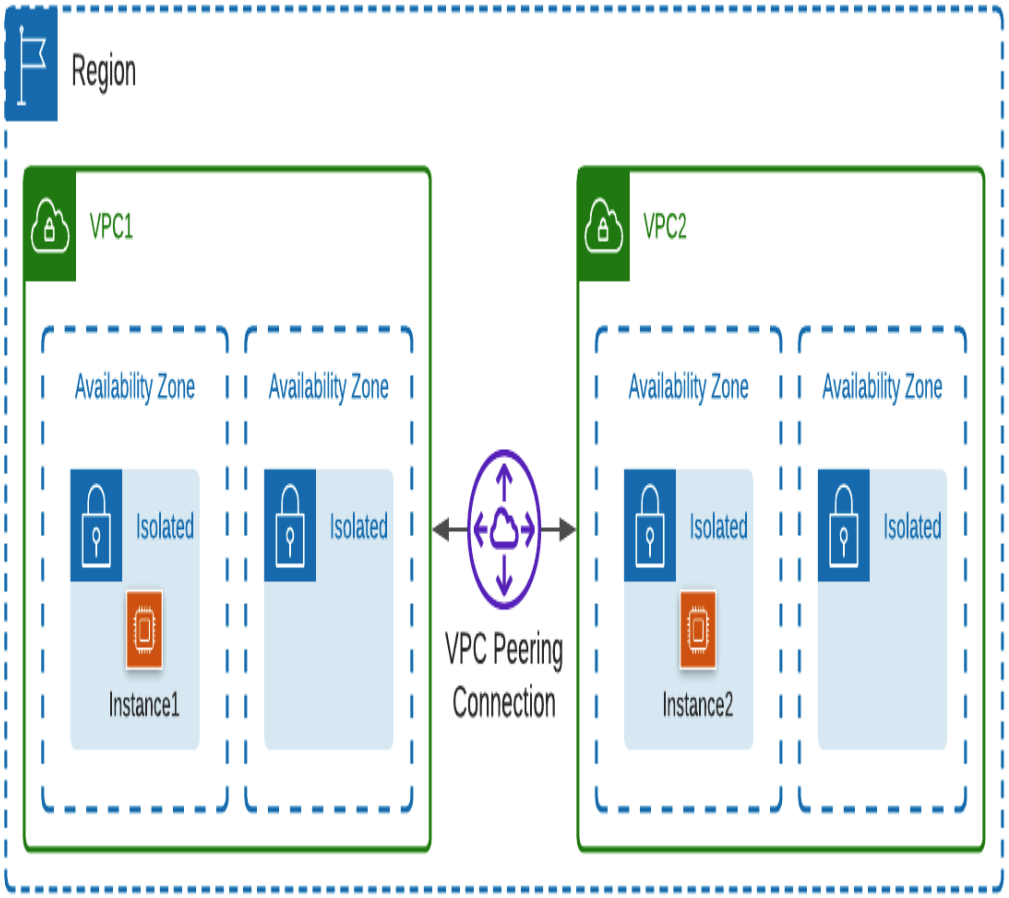


Figure 1-15. Communication between instance in peered VPCs

Prerequisites

- Two VPCs. Each with isolated subnets in 2 AZs and associated route tables
- In each VPC, one EC2 instance that you can assess for testing

Preparation

This recipe requires some “prep work” which deploys resources that you’ll build the solution on. You will use the AWS CDK to deploy these resources

In the root of this Chapter's repo **cd** to the "211-Peering-VPCs/cdk-AWS-Cookbook-211" directory and follow the subsequent steps:

```
cd 211-Peering-VPCs/cdk-AWS-Cookbook-211/
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt --no-dependencies
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

Steps

Create a VPC peering connection to connect VPC 1 to VPC 2

```
VPC_PEERING_CONNECTION_ID=$(aws ec2 create-vpc-peering-connection \
        --vpc-id $VPC_ID_1 --peer-vpc-id $VPC_ID_2 --output text \
        --query VpcPeeringConnection.VpcPeeringConnectionId)
```

Accept the peering connection

```
aws ec2 accept-vpc-peering-connection \
        --vpc-peering-connection-id $VPC_PEERING_CONNECTION_ID
```

> **NOTE**
>
> VPC peering connections can be established from one AWS account to a different AWS account. If you choose to peer VPCs across AWS accounts, you need to ensure you have the correct IAM configuration to create and accept the peering connection within each account.

In the route tables associated with each subnet, add a route to direct traffic destined for the peered VPC's CIDR range to the VPC_PEERING_CONNECTION_ID

```
aws ec2 create-route --route-table-id $VPC_SUBNET_RT_ID_1 \
        --destination-cidr-block $VPC_CIDR_2 \
```

```
        --vpc-peering-connection-id $VPC_PEERING_CONNECTION_ID
aws ec2 create-route --route-table-id $VPC_SUBNET_RT_ID_2 \
        --destination-cidr-block $VPC_CIDR_1 \
        --vpc-peering-connection-id $VPC_PEERING_CONNECTION_ID
```

Add an ingress rule to Instance2SG that allows ICMPv4 access on from Instance1SG:

```
aws ec2 authorize-security-group-ingress \
        --protocol icmp --port -1 \
        --source-group $INSTANCE_SG_1 \
        --group-id $INSTANCE_SG_2
```

Validation Steps

Get Instance 2's IP

```
aws ec2 describe-instances --instance-ids $INSTANCE_ID_2\
        --output text \
        --query Reservations[0].Instances[0].PrivateIpAddress
```

Ensure your EC2 instance #1 has registered with SSM. Use this command to check the status

```
aws ssm describe-instance-information \
        --filters Key=ResourceType,Values=EC2Instance \
        --query "InstanceInformationList[].InstanceId" --output
text
```

Connect to your EC2 instance using SSM Session Manager

```
aws ssm start-session --target $INSTANCE_ID_1
```

Ping Instance 2 from Instance 1

```
ping -c 4 <<INSTANCE_IP_2>>
Output:
PING 10.20.0.242 (10.20.0.242) 56(84) bytes of data.
64 bytes from 10.20.0.242: icmp_seq=1 ttl=255 time=0.232 ms
64 bytes from 10.20.0.242: icmp_seq=2 ttl=255 time=0.300 ms
64 bytes from 10.20.0.242: icmp_seq=3 ttl=255 time=0.186 ms
64 bytes from 10.20.0.242: icmp_seq=4 ttl=255 time=0.183 ms
--- 10.20.0.242 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3059ms
rtt min/avg/max/mdev = 0.183/0.225/0.300/0.048 ms
```

Exit the Session Manager Session

```
exit
```

NOTE

TIP You can search for a security group ID in the VPC console to show all security groups which reference others. You can also run the aws ec2 describe-security-group-references

CLI command to accomplish this. This is helpful in gaining insight into which security groups reference others. You can reference security groups in peered VPCs owned by other AWS accounts but not located in other regions.

Challenge 1

VPC peering connections can be established across AWS regions. Connect a VPC in another region to the VPC you deployed in the region used for the recipe.

Challenge 2

VPC peering connections can be established from one AWS account to a different AWS account. If you choose to peer VPCs across AWS accounts, you need to ensure you have the correct IAM configuration to create and accept the peering connection across AWS accounts.

Clean Up

Delete the security group rule

```
aws ec2 revoke-security-group-ingress \
        --protocol icmp --port -1 \
        --source-group $INSTANCE_SG_1 \
        --group-id $INSTANCE_SG_2
```

Delete the Peering connection

```
aws ec2 delete-vpc-peering-connection \
        --vpc-peering-connection-id $VPC_PEERING_CONNECTION_ID
```

To clean up the environment variables, run the `helper.py` script in this recipe's `cdk-` directory with the `--unset` flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Unset your manually created environment variables

```
unset VPC_PEERING_CONNECTION_ID
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && rm -r .venv/ && cd ../..
```

Discussion

You created a VPC peering connection between two VPCs by initiating the peering connection and accepting the peering connection. Then you updated the VPC route tables of each VPC to direct traffic destined for the other VPC through the peering connection. VPC peering connections are non-transitive. Each VPC needs to peer with every other VPC that they need to communicate with. This type of connection is ideal in a case where you might have a VPC hosting shared-services that other VPCs need to access, while not having the other VPCs communicate with each other.

In addition to the peering connections, you need to configure the route tables associated with the VPC subnets to send traffic destined for the peered VPC's CIDR to the peering connection (PCX). In other words, to enable VPC1 to be able to communicate with VPC2, the destination route must be present in VPC1 and the return route also must be present in VPC2.

If you were to add a 3rd VPC to this recipe, and you needed all VPCs to be able to communicate with each other, you would need to peer that 3rd VPC with the previous two and update all of the VPC route tables accordingly to allow for all of the VPCs to have communication with each other. As you continue to add more VPCs to a network architecture like this, you may notice that the amount of peering connections and route table updates required begin to increase exponentially. Because of this, Transit Gateway is a better choice for transitive VPC communication using Transit Gateway route tables.

You can use VPC peering cross-account if needed, and you can also reference security groups in peered VPCs in a similar way of referencing security groups within a single VPC. This allows you to use the same type of

strategy with how you manage security groups across your AWS environment when using VPC Peering.

> **WARNING**
>
> Connecting VPCs together requires non-overlapping CIDR ranges in order for routing to work normally. The VPC route tables must include a specific route directing traffic destined for the peered VPC to the peering connection.

1.12 Optimizing End User Load Time for S3 Static Web Content using CloudFront

Problem

You currently serve static web content in S3 and want to optimize your web assets for a global audience.

Prerequisites

- S3 bucket with static web content

Preparation

Create an index.html file

```
echo AWSCookbook > index.html
```

Generate a unique S3 bucket name to use for the CloudFront origin

```
BUCKET_NAME=awscookbook213-$(aws secretsmanager get-random-password
\
        --exclude-punctuation --exclude-uppercase \
        --password-length 6 --require-each-included-type \
        --output text \
        --query RandomPassword)
`
```

Create a Source S3 bucket

```
aws s3api create-bucket --bucket $BUCKET_NAME
```

Copy the previously created files to the bucket (fix this so you don't need public read)

```
aws s3 cp index.html s3://$BUCKET_NAME/
```

Solution

Create a CloudFront distribution and set the origin to your S3 bucket. Then configure an Origin Access Identity (OAI) to require the bucket dto be only accessible from CloudFront.

Steps

Create a CloudFront OAI to reference in a S3 bucket policy

```
OAI=$(aws cloudfront create-cloud-front-origin-access-identity \
        --cloud-front-origin-access-identity-config \
        CallerReference="awscookbook",Comment="AWSCookbook OAI" \
        --query CloudFrontOriginAccessIdentity.Id --output text)
```

Use the sed command to replace the values in the distribution-config-template.json with your CloudFront OAI and S3 bucket name:

```
sed -e "s/CLOUDFRONT_OAI/${OAI}/g" \
        -e "s|S3_BUCKET_NAME|${BUCKET_NAME}|g" \
        distribution-template.json > distribution.json
```

Create a CloudFront distribution which uses the distribution configuration json file that you just created

```
DISTRIBUTION_ID=$(aws cloudfront create-distribution \
        --distribution-config file://distribution.json \
        --query Distribution.Id --output text)
```

The distribution will take a few minutes to create, use this command to check the status. Wait until the Status reaches "Deployed"

```
aws cloudfront get-distribution --id $DISTRIBUTION_ID
```

Configure the S3 bucket policy to only allow requests from CloudFront using a bucket policy like this, we have provided a template in the repository you can use

```
{
    "Version": "2012-10-17",
```

```
    "Id": "PolicyForCloudFrontPrivateContent",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::cloudfront:user/CloudFront
Origin Access Identity CLOUDFRONT_OAI"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::S3_BUCKET_NAME/*"
        }
    ]
}
```

Use the sed command to replace the values in the bucket-policy-template.json with the CloudFront OAI and S3 bucket name:

```
sed -e "s/CLOUDFRONT_OAI/${OAI}/g" \
        -e "s|S3_BUCKET_NAME|${BUCKET_NAME}|g" \
        bucket-policy-template.json > bucket-policy.json
```

Apply the bucket policy to the S3 bucket with your static web content

```
aws s3api put-bucket-policy --bucket $BUCKET_NAME \
        --policy file://bucket-policy.json
```

Get the DOMAIN_NAME of the distribution that you created

```
DOMAIN_NAME=$(aws cloudfront get-distribution --id $DISTRIBUTION_ID
\
        --query Distribution.DomainName --output text)
```

Validation Steps

Try to access the S3 bucket directly using HTTPS to verify the bucket does not serve content directly

```
curl https://$BUCKET_NAME.s3.$AWS_REGION.amazonaws.com/index.html
```

```
Output:
212-Optimizing-S3-with-CloudFront:$ curl
https://$BUCKET_NAME.s3.$AWS_REGION.amazonaws.com/index.html
<?xml version="1.0" encoding="UTF-8"?>
<Error><Code>AccessDenied</Code><Message>Access Denied</Message>
<RequestId>0AKQD0EFJC9ZHPCC</RequestId>
<HostId>gfld4qKp9A93G8ee7VPBFrXBZV1HE3jiOb3bNB54fPEPTihit/OyFh7hF2N
u4+Muv6JEc0ebLL4=</HostId></Error>
212-Optimizing-S3-with-CloudFront:$
```

Use `curl` to observe that your index.html file is served from the private S3 bucket through CloudFront

```
curl $DOMAIN_NAME
Output:
212-Optimizing-S3-with-CloudFront:$ curl $DOMAIN_NAME
AWSCookbook
212-Optimizing-S3-with-CloudFront:$
```

Challenge

Configure a TTL on your S3 objects for 30 days so that your S3 bucket is not accessed directly more often than 30 days to retrieve specific objects. You can modify the distribution.json file and reconfigure the CloudFront distribution using the `aws cloudfront update-distribution` CLI command.

Clean Up

Disable the CloudFront distribution by logging into the console and clicking the Disable button for the distribution that you created (this process can take up to 15 minutes):

Delete the CloudFront distribution

```
aws cloudfront delete-distribution --id $DISTRIBUTION_ID --if-
match $(aws cloudfront get-distribution --id $DISTRIBUTION_ID --
query ETag --output text)
```

Delete the Origin Access Identity

```
aws cloudfront delete-cloud-front-origin-access-identity --id $OAI
--if-match $(aws cloudfront get-cloud-front-origin-access-identity
--id $OAI --query ETag --output text)
```

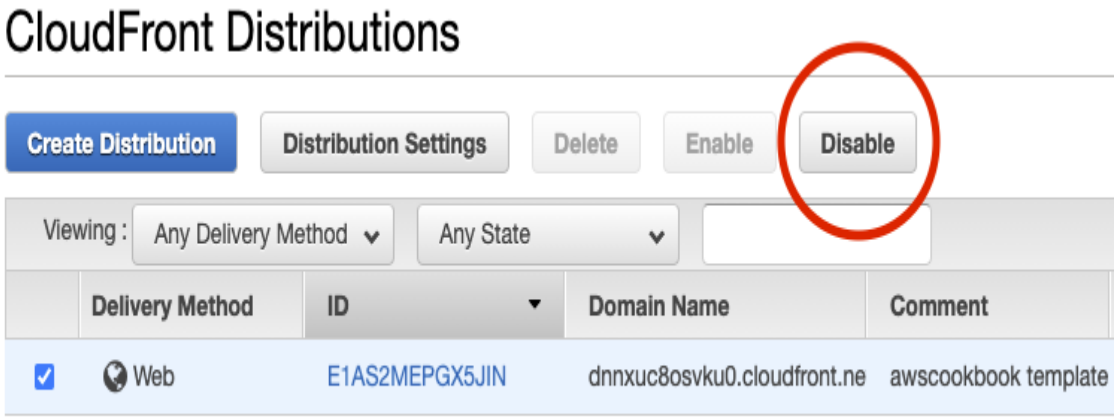
Clean up the bucket

```
aws s3 rm s3://$BUCKET_NAME/index.html
```

Delete the S3 bucket

```
aws s3api delete-bucket --bucket $BUCKET_NAME
```

Unset your manually created environment variables




```
unset DISTRIBUTION_ID
unset DOMAIN_NAME
unset OAI
```

Discussion

You created a CloudFront Origin Access Identity (OAI) and a CloudFront distribution using a S3 bucket as an origin. You configured the bucket to only

allow access from the CloudFront OAI. This configuration allows you to keep the S3 bucket private and only allows the CloudFront distribution to be able to access objects in the bucket. When securing web content, it is desirable to not open up your S3 bucket as public.

Amazon CloudFront is a Content Distribution Network (CDN) that peers with thousands of telecom carriers globally. CloudFront Edge locations are connected to the AWS Regions through the AWS network backbone. When your end users fetch content from your CloudFront distribution, they are optimally routed to a point of presence with the lowest latency.This allows you to centrally host web assets and avoid the storage costs associated with replicating data to multiple regions to be closer to the users. Using CloudFront also allows your content to leverage the AWS network backbone as much as possible rather than taking a path that relies on traversing the public internet to an AWS Region. You can also configure and tune time-to-live (TTL) caching for your content at edge locations; this reduces the frequency that CloudFront has to pull content from your origin when your users request it.

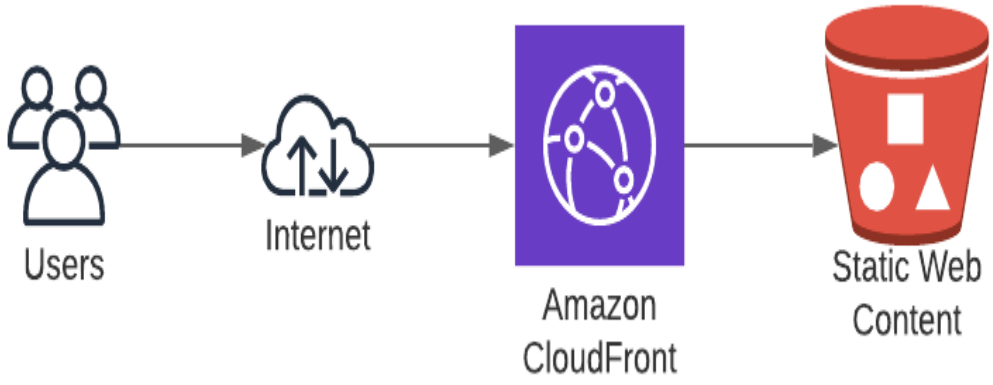


Figure 1-16. CloudFront and S3

You can associate your own custom domain name with CloudFront, force HTTPS, customize cache behavior, invoke Lambda functions (Lambda

@Edge), and more with CloudFront. When you use a CDN like CloudFront, you also benefit from S3 access cost savings for hosting content through CloudFront rather than S3 directly. S3 is billed for storage and per request. CloudFront reduces the number of requests to the bucket directly by caching content and serving directly from the CloudFront network.

Chapter 2. Databases

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 4th chapter of the final book. If you have feedback or content suggestions for the authors, please email *awscookbook@gmail.com*.

2.0 Introduction

Wikipedia defines a database as "an organized collection of data, generally stored and accessed electronically from a computer system"[1]. In 1960 Charles W. Bachman began designing the Integrated Data Store and released it four years later. The hierarchical model was nothing more than a representation of a tree-like structure with one field representing a parent and child. Today, databases are present in many different flavors and offer developers powerful options for storing and interacting with data. Some things that databases enable:

- Secure storage, retrieval, and analysis of medical records.
- Historical weather comparison
- Massively multiplayer online games
- Online shopping

You have a myriad of choices for using databases with AWS. Installing and running a database on EC2 provides you the largest choice of database engines and custom configurations, but brings about challenges like patching, backups, configuring high-availability, replication, performance tuning, etc. AWS offers managed database services which help address these challenges and cover a broad range of database types (Relational, Key-value/NoSQL, In-memory, Document, Wide column, Graph, Time series, Ledger.)[2] When

choosing a database type and data model for your application, you must define the requirements in terms of relationships, speed, volume, and access patterns.

The managed database services on AWS integrate with many services to provide you additional functionality from the security, operations, and development perspectives. In this chapter you will explore relational databases and their use cases with Amazon RDS (Relational Database Services), NoSQL usage with Amazon DynamoDB, and the ways to migrate, secure, and operate these databases types at scale. For example, you will learn how to integrate Secrets Manager with an RDS database to automatically rotate database user passwords. You will also learn how to leverage IAM authentication to reduce the application dependency on database passwords entirely, granting access to RDS through IAM permissions instead. You'll explore Auto Scaling with DynamoDB and learn about why this might be important from a cost and performance perspective.

Workstation Configuration

You will need a few things installed to be ready for the recipes in this chapter:

General Setup

Set and export your default region in your terminal

```
AWS_REGION=us-east-1
```

Validate AWS Command Line Interface (AWS CLI) setup and access

```
aws ec2 describe-instances
```

Set your AWS ACCOUNT ID by parsing output from the `aws sts get-caller-identity` operation.

```
AWS_ACCOUNT_ID=$(aws sts get-caller-identity \
--query Account --output text)
```

> **NOTE**
>
> The aws sts get-caller-identity operation "returns details about the IAM user or role whose credentials are used to call the operation." From: https://awscli.amazonaws.com/v2/documentation/api/latest/reference/sts/get-caller-identity.html

> **WARNING**
>
> During some of the steps of this chapter, you will create passwords using AWS SecretsManager and temporarily save them as environment variables to use in subsequent steps. Make sure that you unset the environment variables when you complete the steps of the recipe.

Checkout this Chapter's repo

```
git clone https://github.com/AWSCookbook/Databases
```

2.1 Creating an Aurora Serverless PostgreSQL Database

Problem

You need a database for infrequent, intermittent, and unpredictable usage.

Solution

Configure and create an Aurora Serverless Database Cluster with a strong password. Then, apply a customized scaling configuration and enable automatic pause after inactivity.

Preparation

This recipe requires some "prep work" which deploys resources that you'll build the solution on. You will use the AWS CDK to deploy these resources

In the root of this Chapter's repo **cd** to the "401-Creating-an-Aurora-Serverless-DB/cdk-AWS-Cookbook-401" folder and follow the subsequent steps:

```
cd 401-Creating-an-Aurora-Serverless-DB/cdk-AWS-Cookbook-401/
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt --no-dependencies
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

Steps

Use AWS Secrets Manager to generate a password which meet the requirements

```
MasterPassword=$(aws secretsmanager get-random-password \
--exclude-punctuation \
--password-length 41 --require-each-included-type \
--output text \
--query RandomPassword)
```

> **NOTE**
>
> We are excluding punctuation characters from the password that we are creating because PostgreSQL does not support them.

Create a RDS Subnet Group

```
aws rds create-db-subnet-group \
--db-subnet-group-name awscookbook401subnetgroup \
```

```
--db-subnet-group-description "AWSCookbook401 subnet group" \
--subnet-ids $Subnet1ID $Subnet2ID
```

Create a RDS Parameter Group

```
aws rds create-db-cluster-parameter-group \
--db-cluster-parameter-group-name awscookbook401paramgroup \
--db-parameter-group-family aurora-postgresql10 \
--description "AWSCookbook401 DB Cluster parameter group"
```

Create a VPC security group for the database

```
DBSecurityGroupId=$(aws ec2 create-security-group \
--group-name AWSCookbook401sg \
--description "Aurora Serverless Security Group" \
--vpc-id $VPCId --output text --query GroupId)
```

Create a database cluster

```
aws rds create-db-cluster \
--db-cluster-identifier awscookbook401dbcluster \
--engine aurora-postgresql \
--engine-mode serverless \
--engine-version 10.14 \
--db-cluster-parameter-group-name awscookbook401paramgroup \
--master-username master \
--master-user-password $MasterPassword \
--db-subnet-group-name awscookbook401subnetgroup \
--vpc-security-group-ids $DBSecurityGroupId
```

Wait for the Status to reaches “available”

```
aws rds describe-db-clusters \
--db-cluster-identifier awscookbook401dbcluster \
--output text --query DBClusters[0].Status
```

Modify the database to automatically with new autoscaling capacity targets and enable AutoPause after 5 minutes of inactivity

```
aws rds modify-db-cluster \
--db-cluster-identifier awscookbook401dbcluster --scaling-
configuration \
```

```
MinCapacity=8,MaxCapacity=16,SecondsUntilAutoPause=300,TimeoutAct
ion='ForceApplyCapacityChange',AutoPause=true
```

Wait at least 5 minutes, and observe that the database's capacity has scaled down to 0

```
aws rds describe-db-clusters \
--db-cluster-identifier awscookbook401dbcluster \
--output text --query DBClusters[0].Capacity
```

> **NOTE**
>
> The auto pause feature automatically sets the capacity of the cluster to 0 after inactivity. When your database activity resumes (e.g. with a query or connect), the capacity value is automatically set to your configured minimum scaling capacity value.

Grant the Instance's Security Group access to p

```
aws ec2 authorize-security-group-ingress \
--protocol tcp --port 5432 \
--source-group $InstanceSG \
--group-id $DBSecurityGroupId
```

Create and populate some ssm parameter to store values

```
aws ssm put-parameter \
--name "Cookbook201Endpoint" \
--type "String" \
--value $(aws rds describe-db-clusters --db-cluster-identifier
awscookbook401dbcluster --output text --query
DBClusters[0].Endpoint) aws ssm put-parameter \
--name "Cookbook401MasterPassword" \
--type "String" \
--value $MasterPassword
```

Ensure the provided Instance has registered with SSM. Use this command to check the status. This command should return the instance ID

```
aws ssm describe-instance-information \
--filters Key=ResourceType,Values=EC2Instance \
--query "InstanceInformationList[].InstanceId" --output text
```

Connect to the EC2 instance

```
aws ssm start-session --target $InstanceID
```

Install psql

```
sudo yum -y install postgresql
```

Set the region

```
export AWS_DEFAULT_REGION=us-east-1
```

Retrieve the Hostname Master Password

```
HostName=$(aws ssm get-parameters \
--names "Cookbook401Endpoint" \
  --query "Parameters[*].Value" --output text)

aws ssm get-parameters \
--names "Cookbook401MasterPassword" \
  --query "Parameters[*].Value" --output text
```

Connect to the Database - This may take a few seconds as the db capacity is scaling up. You'll need to copy and paste the password in - outputted above.

```
psql -h $HostName -U master -W -d postgres
```

Quit psql

```
\q
```

Log out of the EC2 instance

```
exit
```

Check the Capacity of the Cluster again

```
aws rds describe-db-clusters \
--db-cluster-identifier awscookbook401dbcluster \
--output text --query DBClusters[0].Capacity
```

Clean Up

Delete the SSM Parameters that you created

```
aws ssm delete-parameter --name "Cookbook401Endpoint"
aws ssm delete-parameter --name "Cookbook401MasterPassword"
```

Revoke the access from the instance to that database

```
aws ec2 revoke-security-group-ingress \
--protocol tcp --port 5432 \
--source-group $InstanceSG \
--group-id $DBSecurityGroupId
```

Delete the RDS database cluster

```
aws rds delete-db-cluster \
--db-cluster-identifier awscookbook401dbcluster \
--skip-final-snapshot
```

Wait for the Status to reach “deleted”

```
aws rds describe-db-clusters \
--db-cluster-identifier awscookbook401dbcluster \
--output text --query DBClusters[0].Status
```

When the cluster has finished deleting, delete the RDS Parameter Group

```
aws rds delete-db-cluster-parameter-group \
--db-cluster-parameter-group-name awscookbook401paramgroup
```

Delete the RDS Subnet Group

```
aws rds delete-db-subnet-group \
--db-subnet-group-name awscookbook401subnetgroup
```

Delete the security group for the database

```
aws ec2 delete-security-group \
--group-id $DBSecurityGroupId
```

To clean up the environment variables, run the `helper.py` script in this recipe's `cdk-` folder with the `--unset` flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Unset the environment variable that you created manually

```
unset MasterPassword
unset DBSecurityGroupId
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && cd ../..
```

Discussion

You created an Amazon Aurora Serverless cluster in your VPC by using the `aws rds create-db-cluster` command. If you inspect the command arguments, you will see that you use the `serverless` engine mode and did not provision any fixed capacity.

> **WARNING**
>
> Not all database engines and versions are available with the serverless engine. At the time of this writing, Aurora Serverless is currently available for Aurora with MySQL 5.6 compatibility and for Aurora with PostgreSQL 10.7+ compatibility[3]

The cluster will automatically scale capacity to meet the needs of your usage. Setting `MaxCapacity=16` limits the upper bound of your capacity to prevent runaway usage and unexpected costs. The cluster will set its Capacity to 0 when no connection or activity is detected. This is triggered when the `SecondsUntilAutoPause` value is reached. When you enable `AutoPause=true` for your cluster, you only pay for the underlying storage during idle times. The default (and minimum) "inactivity period" is 5 minutes. Connecting to a paused cluster will cause the capacity to scale up to `MinCapacity`. Aurora Serverless scaling is measured in capacity units (CUs) which correspond to compute and memory reserved for your cluster[4]. This capability is a good fit for many workloads and use cases from development, to batch-based workloads, and production workloads where traffic is unpredictable and costs associated with potential over-provisioning are a concern. By not needing to calculate baseline usage patterns, you can start developing quickly and the cluster will automatically respond to the demand that your application requires.

Aurora Serverless has many of the same features of the Amazon Aurora (provisioned) database service[5]. If you currently use a "provisioned" capacity type database on Amazon RDS and would like to start using Aurora Serverless, you can snapshot your current database and restore it from within the AWS Console or from the Command Line to perform a migration. If your current database is not running on RDS, you can use your database engine's dump and restore features or use the Amazon Database Migration Service (Amazon DMS).

Aurora Serverless further builds on the existing Aurora platform[6] which replicates your database's underlying storage 6 ways across 3 Availability Zones[7]. While this replication is a benefit for resiliency, you should still use automated backups for your database. Aurora Serverless has automated backups enabled by default, and the backup retention can be increased up to 35 days if needed.

> **NOTE**
>
> If your DB cluster has been for more than seven days, the DB cluster might be backed up with a snapshot. In this case, the DB cluster is restored when there is a request to connect to it.[8]

2.2 Using IAM Authentication with a RDS Database

Problem

You have a server which connects to a database with a password and you would like to use role based rotating credentials.

Solution

First you will enable IAM Authentication for your database. You will then configure the IAM permissions for the EC2 instance to use. Finally, create a new user on the database, retrieve the IAM authentication token, and verify connectivity.

Preparation

This recipe requires some "prep work" which deploys resources that you'll build the solution on. You will use the AWS CDK to deploy these resources

In the root of this Chapter's repo **cd** to the "402-Using-IAM-Authentication-with-RDS/cdk-AWS-Cookbook-402" folder and follow the subsequent steps:

```
cd 402-Using-IAM-Authentication-with-RDS/cdk-AWS-Cookbook-402/
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt --no-dependencies
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

For this recipe, we will need to create a modified environment variable from the output:

```
IsolatedSubs_list=$(echo ${IsolatedSubnets} | tr -d ' ' | tr -d
'"')
```

Navigate up to the main directory for this recipe (out of the “cdk-AWS-Cookbook-402” folder)

```
cd ..
```

Steps

Enable IAM DB authentication on the RDS DB instance

```
aws rds modify-db-instance \
--db-instance-identifier $RdsDatabaseId \
--enable-iam-database-authentication \
--apply-immediately
```

WARNING

At the time of this writing, IAM database authentication is available for the following database engines[9]:

- MySQL 8.0, minor version 8.0.16 or higher
- MySQL 5.7, minor version 5.7.16 or higher
- MySQL 5.6, minor version 5.6.34 or higher
- PostgreSQL 12, all minor versions
- PostgreSQL 11, all minor versions
- PostgreSQL 10, minor version 10.6 or higher
- PostgreSQL 9.6, minor version 9.6.11 or higher
- PostgreSQL 9.5, minor version 9.5.15 or higher

Retrieve the RDS Database Instance Resource ID and set it as an environment variable:

```
DBResourceId=$(aws rds describe-db-instances \
--query \
'DBInstances[?DBName==`AWSCookbookRecipe402`].DbiResourceId' \
--output text)
```

Create a file called policy.json with the following content (Provided in repo):

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-
db:AWS_REGION:AWS_ACCOUNT_ID:dbuser:DBResourceId/db_user"
         ]
      }
   ]
}
```

Replace the values in the template file using the sed command with environment variables you have set:

```
sed -e "s/AWS_ACCOUNT_ID/${AWS_ACCOUNT_ID}/g" \
-e "s|AWS_REGION|${AWS_REGION}|g" \
-e "s|DBResourceId|${DBResourceId}|g" \
policy-template.json > policy.json
```

Create an IAM Policy from using the file you just created

```
aws iam create-policy --policy-name AWSCookbook402EC2RDSPolicy \
--policy-document file://policy.json
```

Attach the IAM policy for AWSCookbook402LambdaRDSPolicy to the IAM role

```
aws iam attach-role-policy --role-name $EC2RoleName \
   --policy-arn
arn:aws:iam::$AWS_ACCOUNT_ID:policy/AWSCookbook402EC2RDSPolicy
```

Retrieve the RDS Admin Password from SecretsManager

```
RdsAdminPassword=$(aws secretsmanager get-secret-value --secret-
id $RdsSecretArn --query SecretString | jq -r | jq .password | tr
-d '"')
```

Set some SSM Parameters. This will make it easy to pull the values while testing on the provided EC2 instance (created in preparation steps)

```
aws ssm put-parameter \
--name "Cookbook402Endpoint" \
--type "String" \
--value $RdsEndpointaws ssm put-parameter \
--name "Cookbook402AdminPassword" \
--type "String" \
--value $RdsAdminPassword
```

Ensure the provided Instance has registered with SSM. Use this command to check the status. This command should return the instance ID

```
aws ssm describe-instance-information \
--filters Key=ResourceType,Values=EC2Instance \
--query "InstanceInformationList[].InstanceId" --output text
```

Connect to the EC2 instance

```
aws ssm start-session --target $InstanceID
```

Install mysql

```
sudo yum -y install mysql
```

Set the region

```
export AWS_DEFAULT_REGION=us-east-1
```

Retrieve the RDS hostname

```
hostname=$(aws ssm get-parameters \
--names "Cookbook402Endpoint" \
  --query "Parameters[*].Value" --output text)
```

Retrieve the Admin Password

```
password=$(aws ssm get-parameters \
--names "Cookbook402AdminPassword" \
  --query "Parameters[*].Value" --output text)
```

Connect to the Database

```
mysql -u admin -p$password -h $hostname
```

Create a new database user to associate with the IAM authentication:

```
CREATE USER db_user@'%' IDENTIFIED WITH AWSAuthenticationPlugin
as 'RDS';
```

```
GRANT SELECT ON *.* TO 'db_user'@'%';
```

Now, exit the mysql prompt

```
quit
```

Download the RDS Root CA file

```
cd /tmp
wget https://s3.amazonaws.com/rds-downloads/rds-ca-2019-root.pem
```

Generate the RDS auth token and save it as a variable

```
TOKEN="$(aws rds generate-db-auth-token --hostname $hostname --
port 3306 --username db_user)"
```

Connect to the database using the RDS auth token with the new db_user

```
mysql --host=$hostname --port=3306 --ssl-ca=/tmp/rds-ca-2019-
root.pem --user=db_user --password=$TOKEN
```

Run a SELECT query at the mysql prompt to verify that this user has the SELECT *.* grant that you applied

```
SELECT user FROM mysql.user;
```

Exit the mysql prompt

```
quit
```

Log out of the EC2 instance

```
exit
```

Clean Up

Delete the SSM parameters

```
aws ssm delete-parameter --name Cookbook402Endpoint
aws ssm delete-parameter --name Cookbook402AdminPassword
```

Detach the policy from the role

```
aws iam detach-role-policy --role-name $EC2RoleName \
--policy-arn
arn:aws:iam::$AWS_ACCOUNT_ID:policy/AWSCookbook402EC2RDSPolicy
```

Delete the IAM Policy

```
aws iam delete-policy --policy-arn \
arn:aws:iam::$AWS_ACCOUNT_ID:policy/AWSCookbook402EC2RDSPolicy
```

Go to the cdk-AWS-Cookbook-402 directory

```
cd cdk-AWS-Cookbook-402/
```

To clean up the environment variables, run the `helper.py` script in this recipe's `cdk-` folder with the `--unset` flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Unset the environment variable that you created manually

```
unset DBResourceId
unset RdsAdminPassword
unset IsolatedSubs_list
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && cd ../..
```

Discussion

You enabled IAM authentication on your RDS database by first modifying it's configuration. Then, you created an IAM Policy with the necessary permissions required for IAM authentication to be used with a new database user, attached it to your EC2's instance role, connected to the EC2 instance, authenticated to the database and ran two SQL statements which configured the AWS Authentication Plugin for a new database user. This database user was associated with the IAM policy attached to the role. Finally, you tested the connectivity from your EC2 instance to your database by using the token retrieved from IAM as the database user password.

Instead of a password in your mysql connection string, you retrieved a token associated with it's IAM role that lasts for 15 minutes[10]. If you install an application on this EC2 instance, the code can continuously refresh this token or you can also use a caching mechanism like redis to store and share the token with other components of your application. This is useful if you have a distributed architecture and/or horizontal scaling (multiple instances of your application running at once). There is no need to rotate passwords for your database user because the old token will be invalidated after 15 minutes.

You can create multiple database users associated with specific grants to allow your application to different levels of access to your database. The grants happen within the database, not within the IAM permissions. IAM only controls the `db-connect` action for the specific user, this only allows the authentication token to be retrieved. That username is mapped from IAM to the GRANT(s) by using the same username within the database as in the `policy.json` file below:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-
```

```
db:AWS_REGION::dbuser:DBResourceId/db_user"
            ]
        }
    ]
}
```

In this recipe, you also enabled encryption in transit by specifying the SSL certificate bundle that you downloaded to the EC2 instance in your database connection command. This encrypts the connection between your application and your database. This is a good security posture and is often required for many compliance standards. The connection string you used to connect with the IAM authentication token indicated an SSL certificate as one of the connection parameters. The Certificate Authority bundle is available to download and use within your application[11].

2.3 Leveraring RDS Proxy For Database Connections From Lambda

Problem

You have a serverless function that is accessing a database and you want to implement connection pooling.

Solution

Create an RDS Proxy, associate it with your RDS MySQL database, and configure your lambda to connect to the proxy instead of accessing the database directly.

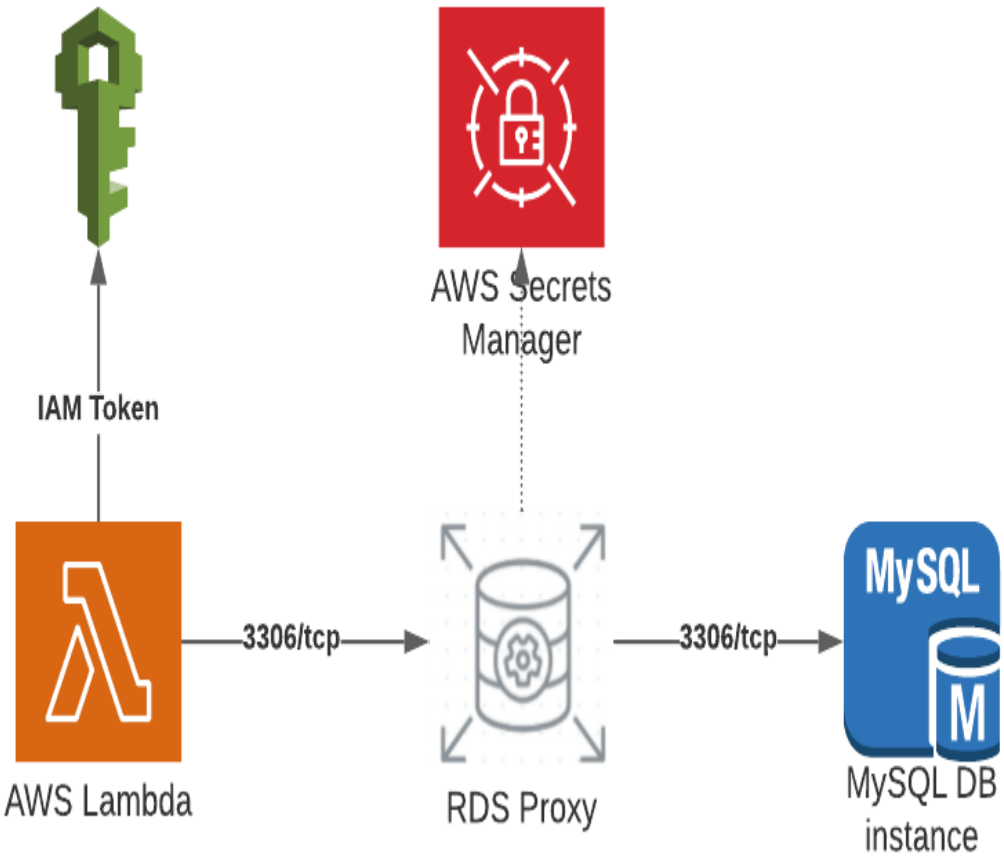


Figure 2-1. Lambda Connection Path to Database via RDS Proxy

Preparation

This recipe requires some "prep work" which deploys resources that you'll build the solution on. You will use the AWS CDK to deploy these resources

In the root of this Chapter's repo **cd** to the "403-Leveraring-RDS-Proxy-For-Db-Conns/cdk-AWS-Cookbook-403" folder and follow the subsequent steps:

```
cd 403-Leveraring-RDS-Proxy-For-Db-Conns/cdk-AWS-Cookbook-403/
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt --no-dependencies
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

Navigate up to the main directory for this recipe (out of the “cdk-AWS-Cookbook-403” folder)

```
cd ..
```

For this recipe, we will need to create a modified environment variable from the output:

```
IsolatedSubs_list=$(echo ${IsolatedSubnets} | tr -d ',' | tr -d
'"')
```

Steps

Create a file called assume-role-policy.json with the following content: (Provided in Repo)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rds.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Create an IAM Role for the RDS Proxy using the assume-role-policy.json

```
aws iam create-role --assume-role-policy-document \
file://assume-role-policy.json --role-name AWSCookbook403RDSProxy
```

Create a security group for the RDS Proxy

```
RDSProxySgId=$(aws ec2 create-security-group \
--group-name AWSCookbook403RDSProxySG \
--description "Lambda Security Group" --vpc-id $VPCId \
--output text --query GroupId)
```

Create the RDS Proxy

```
RDSProxyEndpointArn=$(aws rds create-db-proxy \
--db-proxy-name $DbName \
--engine-family MYSQL \
--auth '{
        "AuthScheme": "SECRETS",
        "SecretArn": "'"$RdsSecretArn"'",
        "IAMAuth": "REQUIRED"
      }' \
--role-arn
arn:aws:iam::$AWS_ACCOUNT_ID:role/AWSCookbook403RDSProxy \
--vpc-subnet-ids $IsolatedSubs_list \
--vpc-security-group-ids $RDSProxySgId \
--require-tls --output text \
--query DBProxy.DBProxyArn)
```

Wait for the RDS Proxy to become “available”

```
aws rds describe-db-proxies \
--db-proxy-name $DbName \
--query DBProxies[0].Status \
--output text
```

Retrieve the RDSProxyEndpoint and set it to an environment variable

```
RDSProxyEndpoint=$(aws rds describe-db-proxies \
--db-proxy-name $DbName \
--query DBProxies[0].Endpoint \
--output text)
```

Create a policy that allows the Lambda Function to generate IAM authentication tokens

Create a file called policy.json with the following content:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-
db:AWS_REGION:AWS_ACCOUNT_ID:dbuser:DBResourceId/admin"
         ]
      }
   ]
}
```

Separate out the ProxyID from the Endpoint ARN

```
RDSProxyID=$(echo $RDSProxyEndpointArn | awk -F: '{ print $7} ')
```

Replace the values in the template file using the sed command with environment variables you have set:

```
sed -e "s/AWS_ACCOUNT_ID/${AWS_ACCOUNT_ID}/g" \
-e "s|AWS_REGION|${AWS_REGION}|g" \
-e "s|RDSProxyID|${RDSProxyID}|g" \
policy-template.json > policy.json
```

Create an IAM Policy from using the file you just created

```
aws iam create-policy --policy-name AWSCookbook403RdsIamPolicy \
--policy-document file://policy.json
```

Attach the policy to the DBAppFunction Lambda Role to allow IAM Auth Token retrieval:

```
aws iam attach-role-policy --role-name $DbAppFunctionRoleName \
--policy-arn
arn:aws:iam::$AWS_ACCOUNT_ID:policy/AWSCookbook403RdsIamPolicy
```

Use this command to wait for the proxy to enter the “available” Status

```
aws rds describe-db-proxies --db-proxy-name $DbName \
        --query DBProxies[0].Status \
        --output text
```

Attach the SecretsManagerReadWrite policy to the RDS Proxy Role

```
aws iam attach-role-policy --role-name AWSCookbook403RDSProxy \
--policy-arn arn:aws:iam::aws:policy/SecretsManagerReadWrite
```

> **TIP**
>
> In a production scenario, you would want to scope this permission down to the minimal secret resources that your application needs to access, rather than grant `SecretsManagerReadWrite` which allows read/write for all secrets.

Add an ingress rule to the RDS Instance’s Security group that allows access on TCP port 3306 from the RDS Proxy Security Group:

```
aws ec2 authorize-security-group-ingress \
--protocol tcp --port 3306 \
--source-group $RDSProxySgId \
--group-id $RdsSecurityGroup
```

Register targets with the RDS Proxy

```
aws rds register-db-proxy-targets \
--db-proxy-name $DbName \
--db-instance-identifiers $RdsDatabaseId
```

Check that status with this command. Wait until the State reaches “AVAILABLE”

```
aws rds describe-db-proxy-targets \
--db-proxy-name awscookbookrecipe403 \
--query Targets[0].TargetHealth.State \
--output text
```

Monitor the state returned by the previous command before proceeding

```
watch -g !!
```

Add an ingress rule to the RDS Proxy Security group that allows access on TCP port 3306 from the Lambda App Function:

```
aws ec2 authorize-security-group-ingress \
--protocol tcp --port 3306 \
--source-group $DbAppFunctionSgId \
--group-id $RDSProxySgId
```

Modify the Lambda function to now use the RDS Proxy Endpoint as the DB_HOST, instead of connecting directly to the database

```
aws lambda update-function-configuration \
--function-name $DbAppFunctionName \
--environment Variables={DB_HOST=$RDSProxyEndpoint}
```

Run the Lambda Function with this command to validate that the function can connect to RDS using your RDS Proxy:

```
aws lambda invoke \
--function-name $DbAppFunctionName \
response.json && cat response.json
```

Invoke the function in the Lambda console multiple times to observe the database connections in CloudWatch Metrics

Clean Up

Delete the RDS DB Proxy

```
aws rds delete-db-proxy --db-proxy-name $DbName
```

The proxy will take some time to delete, monitor the deletion status with this command:

```
aws rds describe-db-proxies --db-proxy-name $DbName
```

The Elastic Network Interfaces for the RDS DB Proxy will remain, use this command to delete the associated network interfaces (answer ‘y’ to any that are found to delete):

```
aws ec2 describe-network-interfaces \
--filters Name=group-id,Values=$RDSProxySgId \
--query NetworkInterfaces[*].NetworkInterfaceId \
--output text | tr '\t' '\n' | xargs -p -I % \
aws ec2 delete-network-interface --network-interface-id %
```

Revoke security group authorization for RDS Proxy

```
aws ec2 revoke-security-group-ingress \
--protocol tcp --port 3306 \
--source-group $RDSProxySgId \
--group-id $RdsSecurityGroup
```

```
aws ec2 revoke-security-group-ingress \
--protocol tcp --port 3306 \
--source-group $DbAppFunctionSgId \
--group-id $RDSProxySgId
```

Delete the security group you created for RDS Proxy:

```
aws ec2 delete-security-group --group-id $RDSProxySgId
```

Detach the AWSCookbook403RdsIamPolicy policy from the Lambda role

```
aws iam detach-role-policy --role-name $DbAppFunctionRoleName \
--policy-arn
arn:aws:iam::$AWS_ACCOUNT_ID:policy/AWSCookbook403RdsIamPolicy
```

Delete the AWSCookbook403RdsIamPolicy policy

```
aws iam delete-policy --policy-arn
arn:aws:iam::$AWS_ACCOUNT_ID:policy/AWSCookbook403RdsIamPolicy
```

Detach the SecretsManager policy from the RDS Proxy role

```
aws iam detach-role-policy --role-name AWSCookbook403RDSProxy \
--policy-arn arn:aws:iam::aws:policy/SecretsManagerReadWrite
```

Delete the IAM Role for the proxy

```
aws iam delete-role --role-name AWSCookbook403RDSProxy
```

Go to the cdk-AWS-Cookbook-403 directory

```
cd cdk-AWS-Cookbook-403/
```

To clean up the environment variables, run the `helper.py` script in this recipe's `cdk-` folder with the `--unset` flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Unset the environment variable that you created manually

```
unset master_password
unset RDSProxySgId
unset RDSProxyEndpointArn
unset RDSProxyEndpoint
unset DBResourceId
unset RDSProxyID
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && cd ../..
```

Discussion

You connected a lambda function to your database using RDS Proxy. You first created the permissions and policies required, then created the RDS Proxy, associated a database as a target, modified and ran a Lambda function to connect using IAM Authentication to the proxy, and the proxy to your database.

Connection pooling is important to consider when you use Lambda with RDS as the database. SInce the function could be executed with a lot of concurrency and frequency depending on your application, the number of raw connections to your database can grow and impact performance on your database. By using RDS Proxy to manage the connections to the database, fewer connections are needed to the actual database. This setup increases performance and efficiency.

Without RDS Proxy, a Lambda function might establish a new connection to the database each time the function is invoked. This behavior depends on the execution environment, runtimes context (Python, NodeJS, Go, etc) and the way you instantiate connections to the database from the function code[12]. In cases with large amounts of function concurrency, this could result in large amounts of TCP connections to your database, reducing database performance and increasing latency. RDS Proxy helps manage the connections from Lambda by managing them as a “pool”, so that as concurrency increases, RDS Proxy only increases the actual connections to the database as-needed, offloading the TCP overhead to RDS proxy[13].

SSL encryption in transit is supported by RDS Proxy when you include the certificate bundle provided by AWS in your database connection string. RDS Proxy supports MySQL and PostgreSQL RDS databases. For a complete listing of all supported database engines and versions, see the following support document:
https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/rds-proxy.html

2.4 Encrypting the Storage of an Existing Amazon RDS for MySQL Database

Problem

You need to encrypt the storage for an existing database

Solution

Create a read-replica of your existing database, take a snapshot of the read-replica, copy the snapshot to an encrypted snapshot, and restore the encrypted snapshot to a new encrypted database.

Preparation

This recipe requires some “prep work” which deploys resources that you’ll build the solution on. You will use the AWS CDK to deploy these resources

In the root of this Chapter’s repo **cd** to the “404-Encrypt-Existing-RDS-MySQL-DB/cdk-AWS-Cookbook-404” folder and follow the subsequent steps:

```
cd 404-Encrypt-Existing-RDS-MySQL-DB/cdk-AWS-Cookbook-404/
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt --no-dependencies
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

Steps

Verify that the Storage is not encrypted

```
aws rds describe-db-instances \
--db-instance-identifier $RdsDatabaseId \
--query DBInstances[0].StorageEncrypted
```

Create a KMS key to use for your encrypted DB and store the key ID in an environment variable

```
key_id=$(aws kms create-key \
--tags TagKey=Name,TagValue=AWSCookbook404RDS \
--description "AWSCookbook RDS Key" \
--query KeyMetadata.KeyId \
--output text)
```

Create an Alias to reference your Key

```
aws kms create-alias \
--alias-name alias/awscookbook404 \
--target-key-id $key_id
```

Create a read-replica of your existing unencrypted database

```
aws rds create-db-instance-read-replica \
--db-instance-identifier awscookbook404db-rep \
--source-db-instance-identifier $RdsDatabaseId \
--max-allocated-storage 10
```

Wait for the “DBInstanceStatus” to become “available”

```
aws rds describe-db-instances \
        --db-instance-identifier awscookbook404db-rep \
        --output text --query DBInstances[0].DBInstanceStatus
```

Take an unencrypted snapshot of your read-replica

```
aws rds create-db-snapshot \
--db-instance-identifier awscookbook404db-rep \
--db-snapshot-identifier awscookbook404-snapshot
```

NOTE

In production environments, you should quiesce database write activity from your live application traffic before taking a snapshot. This ensures that you do not lose any data during the period of taking the snapshot and restoring the snapshot to a new encrypted database.

Wait for the “Status” of the snapshot to become available

```
aws rds describe-db-snapshots \
--db-snapshot-identifier awscookbook404-snapshot \
--output text --query DBSnapshots[0].Status
```

Copy the unencrypted snapshot to a new and encrypted snapshot by specifying your KMS key

```
aws rds copy-db-snapshot \
--copy-tags \
--source-db-snapshot-identifier awscookbook404-snapshot \
--target-db-snapshot-identifier awscookbook404-snapshot-enc \
--kms-key-id alias/awscookbook404
```

Wait for the “Status” of the encrypted snapshot to become available

```
aws rds describe-db-snapshots \
--db-snapshot-identifier awscookbook404-snapshot-enc \
--output text --query DBSnapshots[0].Status
```

Restore the encrypted snapshot to a new RDS instance

```
aws rds restore-db-instance-from-db-snapshot \
--db-subnet-group-name $RdsSubnetGroup \
--db-instance-identifier awscookbook404db-enc \
--db-snapshot-identifier awscookbook404-snapshot-enc
```

Wait for the “DBInstanceStatus” to become available

```
aws rds describe-db-instances \
--db-instance-identifier awscookbook404db-enc \
--output text --query DBInstances[0].DBInstanceStatus
```

Verify that the Storage is now encrypted

```
aws rds describe-db-instances \
--db-instance-identifier awscookbook404db-enc \
--query DBInstances[0].StorageEncrypted
```

Clean Up

Delete the read replica

```
aws rds delete-db-instance --skip-final-snapshot \
--delete-automated-backups \
--db-instance-identifier awscookbook404db-rep
```

Delete the encrypted RDS database you created

```
aws rds delete-db-instance --skip-final-snapshot \
--delete-automated-backups \
--db-instance-identifier awscookbook404db-enc
```

Delete the two snapshots

```
aws rds delete-db-snapshot \
--db-snapshot-identifier awscookbook404-snapshot

aws rds delete-db-snapshot \
--db-snapshot-identifier awscookbook404-snapshot-enc
```

Disable the KMS Key

```
aws kms disable-key --key-id $key_id
```

Scheduled the KMS Key for deletion

```
aws kms schedule-key-deletion \
--key-id $key_id \
--pending-window-in-days 7
```

Delete the Key Alias

```
aws kms delete-alias --alias-name alias/awscookbook404
```

To clean up the environment variables, run the `helper.py` script in this recipe's `cdk-` folder with the `--unset` flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Unset the environment variable that you created manually

```
unset key_id
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && cd ../..
```

Discussion

You migrated an existing database to a new database with storage encrypted at rest. First you created a read-replica. Then you took a snapshot of an unencrypted database's read-replica, encrypted the snapshot, and restored the encrypted snapshot to a new database. Specifying a KMS key with the `copy-snapshot` command encrypts the copied snapshot. Restoring an encrypted database to a new database results in an encrypted database[14].

By creating a read-replica to perform the snapshot from, you ensure that the snapshot process will not impact your database from a performance perspective. When you complete the steps, you need to reconfigure your application to point to a new database endpoint hostname. To perform this with minimal downtime, you can configure a Route53 DNS record as an alias to your DB endpoint (your application could be configured to use a DNS alias) and shift your application traffic over to the new encrypted database by updating the DNS record with the new database endpoint DNS.

Encryption at rest is a security approach left up to end users in the AWS shared responsibility model[15], and oftentimes it is required to achieve or maintain compliance with regulatory standards. The encrypted snapshot you took could automatically be copied to another region, as well as copied to S3 for archival/backup purposes. You left the original database in place during the steps of this recipe, it is safe to delete this database because the

unencrypted snapshot contains all of the data required to re-create your existing database from scratch.

2.5 Automating Password Rotation for RDS Databases

Problem

You would like to implement automatic password rotation for a database user.

Solution

Create a password and it in AWS Secrets Manager. Configure a rotation interval for the secret containing the password. Finally, create a Lambda function using AWS provided code, and configure the function to perform the password rotation.

Preparation

This recipe requires some "prep work" which deploys resources that you'll build the solution on. You will use the AWS CDK to deploy these resources

In the root of this Chapter's repo **cd** to the "405-Rotating-Database-Passwords-in-RDS/cdk-AWS-Cookbook-405" folder and follow the subsequent steps:

```
cd 405-Rotating-Database-Passwords-in-RDS/cdk-AWS-Cookbook-405/
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt --no-dependencies
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the

output to your terminal to export variables:

```
python helper.py
```

For this recipe, we will need to create a modified environment variable from the output:

```
IsolatedSubs_list=$(echo ${IsolatedSubnets} | tr -d ' ' | tr -d
'"')
```

Navigate up to the main directory for this recipe (out of the “cdk-AWS-Cookbook-405” folder)

```
cd ..
```

Steps

Use AWS Secrets Manager to generate a password which meet the RDS requirements

```
RdsAdminPassword=$(aws secretsmanager get-random-password \
--exclude-punctuation \
--password-length 41 --require-each-included-type \
--output text --query RandomPassword)
```

> **NOTE**
>
> The --exclude-punctuation flag is specified to limit the character set used to generate the password to ensure compatibility with the Database Engines supported by RDS. For more information, see the official documentation: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Limits.html

Change the Master Password for your RDS DB to the one you just created

```
aws rds modify-db-instance \
--db-instance-identifier $RdsDatabaseId \
```

```
--master-user-password $RdsAdminPassword \
--apply-immediately
```

Create a file with the following content called rdscreds-template.json (File included in repo)

```
{
    "username": "admin",
    "password": "PASSWORD",
    "engine": "mysql",
    "host": "HOST",
    "port": 3306,
    "dbname": "DBNAME",
    "dbInstanceIdentifier": "DBIDENTIFIER"
}
```

Use sed to modify the values in rdscreds-template.json to create rdscreds.json

```
sed -e "s/AWS_ACCOUNT_ID/${AWS_ACCOUNT_ID}/g" \
-e "s|PASSWORD|${RdsAdminPassword}|g" \
-e "s|HOST|${RdsEndpoint}|g" \
-e "s|DBNAME|${DbName}|g" \
-e "s|DBIDENTIFIER|${RdsDatabaseId}|g" \
rdscreds-template.json > rdscreds.json
```

Download code from the AWS Samples GitHub repository for the Rotation Lambda Function

```
wget https://raw.githubusercontent.com/aws-samples/aws-secrets-
manager-rotation-
lambdas/master/SecretsManagerRDSMySQLRotationSingleUser/lambda_fu
nction.py
```

NOTE

AWS provides information and templates for different DB rotation scenarios here:

https://docs.aws.amazon.com/secretsmanager/latest/userguide/reference_available-rotation-templates.html

Compress the file containing the code

```
zip lambda_function.zip lambda_function.py
```

Create a new Security Group for the Lambda Function to use:

```
LambdaSgId=$(aws ec2 create-security-group \
--group-name AWSCookbook405LambdaSG \
--description "Lambda Security Group" --vpc-id $VPCId \
--output text --query GroupId)
```

Add an ingress rule to the RDS Instances Security group that allows access on port 3306/tcp from the rotation Lambda's Security Group:

```
aws ec2 authorize-security-group-ingress \
--protocol tcp --port 3306 \
--source-group $EC2SecurityGroup \
--group-id $RdsSecurityGroup
```

Create a file named assume-role-policy.json with the following content (File included in repo):

```
{
        "Version": "2012-10-17",
        "Statement": [
        {
                "Effect": "Allow",
                "Principal": {
                "Service": "lambda.amazonaws.com"
                },
                "Action": "sts:AssumeRole"
        }
        ]
}
```

Create an IAM role using the statement in the provided assume-role-policy.json file using this command:

```
aws iam create-role --role-name AWSCookbook405Lambda \
--assume-role-policy-document file://assume-role-policy.json
```

Attach the IAM managed policy for AWSLambdaVPCAccess to the IAM role:

```
aws iam attach-role-policy --role-name AWSCookbook405Lambda \
--policy-arn arn:aws:iam::aws:policy/service-
role/AWSLambdaVPCAccessExecutionRole
```

Attach the IAM managed policy for SecretsManagerReadWrite to the IAM role:

```
aws iam attach-role-policy --role-name AWSCookbook405Lambda \
--policy-arn arn:aws:iam::aws:policy/SecretsManagerReadWrite
```

> **TIP**
>
> The IAM role that you associated with the Lambda function to rotate the password used the `SecretsManagerReadWrite` managed policy. In a production scenario, you would want to scope this down to limit which secrets the Lambda function can interact with.

Create a Lambda Function to perform the secret rotation using the code

```
LambdaRotateArn=$(aws lambda create-function \
--function-name AWSCookbook405Lambda \
--runtime python3.8 \
--package-type "Zip" \
--zip-file fileb://lambda_function.zip \
--handler lambda_function.lambda_handler --publish \
--environment Variables=
{SECRETS_MANAGER_ENDPOINT=https://secretsmanager.$AWS_REGION.amaz
onaws.com} \
--layers $PyMysqlLambdaLayerArn \
--role \
arn:aws:iam::$AWS_ACCOUNT_ID:role/AWSCookbook405Lambda \
--output text --query FunctionArn \
--vpc-config
SubnetIds=${IsolatedSubs_list},SecurityGroupIds=$LambdaSgId)
```

Use this command to determine when the Lambda Function has entered the “Active” State

```
aws lambda get-function --function-name $LambdaRotateArn \
--output text --query Configuration.State
```

Continuously monitor the state of the Lambda using watch and the last command you typed:

```
watch -g !! && say "Done"
```

Add a permission to the Lambda Function so that Secrets Manager can invoke it

```
aws lambda add-permission --function-name $LambdaRotateArn \
--action lambda:InvokeFunction --statement-id secretsmanager \
--principal secretsmanager.amazonaws.com
```

Set a unique suffix to use for the Secret Name to ensure you can re-use this pattern for additional user's automatic password rotations if desired

```
AWSCookbook405SecretName=AWSCookbook405Secret-$(aws
secretsmanager \
get-random-password \
--exclude-punctuation \
--password-length 6 --require-each-included-type \
--output text \
--query RandomPassword)
```

Create a Secret in Secrets Manager to store your Master Password

```
aws secretsmanager create-secret --name $AWSCookbook405SecretName
\
--description "My database secret created with the CLI" \
--secret-string file://rdscreds.json
```

Setup automatic rotation every 30 days and specify the Lambda to perform rotation for the secret you just created

```
aws secretsmanager rotate-secret \
--secret-id $AWSCookbook405SecretName \
--rotation-rules AutomaticallyAfterDays=30 \
--rotation-lambda-arn $LambdaRotateArn
```

> **NOTE**
>
> The `rotate-secret` command triggers an initial rotation of the password. You will trigger an extra rotation of the password in the next step to demonstrate how to perform rotations on-demand.

Perform another rotation of the secret, notice that the “VersionID” will be different than the last command indicating that the secret has been rotated

```
aws secretsmanager rotate-secret --secret-id
$AWSCookbook405SecretName
```

Validation Steps:

Retrieve the RDS Admin Password from SecretsManager

```
RdsAdminPassword=$(aws secretsmanager get-secret-value --secret-
id $AWSCookbook405SecretName --query SecretString | jq -r | jq
.password | tr -d '"')
```

Set some SSM Parameters. This will make it easy to pull the values while testing on the provided EC2 instance (created in preparation steps)

```
aws ssm put-parameter \
--name "Cookbook405Endpoint" \
--type "String" \
--value $RdsEndpoint --overwriteaws ssm put-parameter \
--name "Cookbook405AdminPassword" \
--type "String" \
--value $RdsAdminPassword --overwrite
```

Ensure the provided Instance has registered with SSM. Use this command to check the status. This command should return the instance ID

```
aws ssm describe-instance-information \
--filters Key=ResourceType,Values=EC2Instance \
--query "InstanceInformationList[].InstanceId" --output text
```

Connect to the provided EC2 instance

```
aws ssm start-session --target $InstanceID
```

Set and export your default region

```
export AWS_DEFAULT_REGION=us-east-1
```

Install the mysql client

```
sudo yum -y install mysql
```

Retrieve the rotated RDS Admin Password and RDS endpoint

```
hostname=$(aws ssm get-parameters \
--names "Cookbook405Endpoint" \
  --query "Parameters[*].Value" --output text)

password=$(aws ssm get-parameters \
--names "Cookbook405AdminPassword" \
  --query "Parameters[*].Value" --output text)
```

Connect to the Database to verify the latest rotated password is working

```
mysql -u admin -p$password -h $hostname
```

Run a SELECT statement on the mysql.user table to validate administrator permissions

```
SELECT user FROM mysql.user;
```

Exit from the mysql prompt

```
quit
```

Log out of the EC2 instance

```
exit
```

Clean Up

Delete the Secret in Secret Manager

```
aws secretsmanager delete-secret \
--secret-id $AWSCookbook405SecretName \
--recovery-window-in-days 7
```

Delete the SSM parameters

```
aws ssm delete-parameter --name Cookbook405Endpoint
aws ssm delete-parameter --name Cookbook405AdminPassword
```

Delete the Lambda Function

```
aws lambda delete-function --function-name AWSCookbook405Lambda
```

Detach the LambdaVPCAccessExecutionPolicy from the role

```
aws iam detach-role-policy --role-name AWSCookbook405Lambda \
--policy-arn arn:aws:iam::aws:policy/service-
role/AWSLambdaVPCAccessExecutionRole
```

Detach the SecretsManagerReadWrite policy from the role

```
aws iam detach-role-policy --role-name AWSCookbook405Lambda \
--policy-arn arn:aws:iam::aws:policy/SecretsManagerReadWrite
```

Delete the IAM Role

```
aws iam delete-role --role-name AWSCookbook405Lambda
```

Remove the ingress rule to the RDS Instance's Security group that allows access on port 3306/tcp from the Lambda's Security Group:

```
aws ec2 revoke-security-group-ingress \
--protocol tcp --port 3306 \
--source-group $LambdaSgId \
--group-id $RdsSecurityGroup
```

Delete the security group that you created for the Lambda Function:

```
aws ec2 delete-security-group --group-id $LambdaSgId
```

Go to the cdk-AWS-Cookbook-405 directory

```
cd cdk-AWS-Cookbook-405/
```

To clean up the environment variables, run the `helper.py` script in this recipe's `cdk-` folder with the `--unset` flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Unset the environment variable that you created manually

```
unset RdsAdminPassword
unset LambdaRotateArn
unset LambdaSgId
unset AWSCookbook405SecretName
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && cd ../..
```

Discussion

You configured automated password rotation for a user in your database. After creating the password, you used AWS Secrets Manager to store it, then configured the rotation interval. Finally you created and invoked a Lambda function to perform the rotation. In the validation steps, you used SSM Session Manager from the command line to remotely connect to the EC2 instance and verified that the new rotated password for the admin user was valid.

This method could be applied to rotate the passwords for non-admin database user accounts by following the same steps after you have created the user(s) in your database. You can configure your application to retrieve secrets from SecretsManager directly, or the Lambda function that you configured to update the SecretsManager values could also store the password in a secure location of your choosing. You would need to grant the Lambda additional permissions to interact with the secure location you choose and add some code to store the new value there.

The Lambda function that you deployed is Python-based and connects to a MySQL engine-compatible database. The Lambda runtime environment does not have this library included by default, so you specified a Lambda Layer within the `aws lambda create-function` command. This layer is required so that the PyMySQL library was available to the function in the Lambda runtime environment, and it was deployed for you as part of the Preparation Step when you ran `cdk deploy`. You can use layers to include packages and files required for your function to run[16].

2.6 Auto Scaling DynamoDB Table Provisioned Capacity

Problem

You have a database table with low provisioned throughput and you need additional capacity on the table for your application.

Solution

Configure Read and Write scaling by setting Scaling Target and a Scaling Policy for the Read and Write capacity of the DynamoDB table using AWS Application Auto Scaling.

Preparation

Create a DynamoDB table with fixed capacity of 1 Read Capacity Units and 1 Write Capacity Units

```
aws dynamodb create-table \
--table-name 'AWSCookbook406' \
--attribute-definitions 'AttributeName=UserID,AttributeType=S' \
--key-schema 'AttributeName=UserID,KeyType=HASH' \
--sse-specification 'Enabled=true,SSEType=KMS' \
--provisioned-throughput \
'ReadCapacityUnits=1,WriteCapacityUnits=1'
```

Put a few records in the table

```
aws ddb put AWSCookbook406 '[{UserID: value1}, {UserID: value2}]'
```

Steps

Navigate to this recipe’s folder in the chapter repository

```
cd 406-Auto-Scaling-DynamoDB
```

Register a ReadCapacityUnits Scaling Target for the DynamoDB table

```
aws application-autoscaling register-scalable-target \
--service-namespace dynamodb \
--resource-id "table/AWSCookbook406" \
--scalable-dimension "dynamodb:table:ReadCapacityUnits" \
--min-capacity 5 \
--max-capacity 10
```

Register a WriteCapacityUnits Scaling Target for the DynamoDB table

```
aws application-autoscaling register-scalable-target \
--service-namespace dynamodb \
--resource-id "table/AWSCookbook406" \
--scalable-dimension "dynamodb:table:WriteCapacityUnits" \
--min-capacity 5 \
--max-capacity 10
```

Create a scaling policy JSON file for read capacity scaling (read-scaling.json provided in repo)

```
{
    "PredefinedMetricSpecification": {
        "PredefinedMetricType": "DynamoDBReadCapacityUtilization"
    },
    "ScaleOutCooldown": 60,
    "ScaleInCooldown": 60,
    "TargetValue": 50.0
}
```

Create a scaling policy JSON file for write capacity scaling (write-scaling.json provided in repo)

```
{
    "PredefinedMetricSpecification": {
        "PredefinedMetricType":
"DynamoDBWriteCapacityUtilization"
    },
    "ScaleOutCooldown": 60,
    "ScaleInCooldown": 60,
    "TargetValue": 50.0
}
```

Apply the read scaling policy to the table using the read-scaling.json file

```
aws application-autoscaling put-scaling-policy \
--service-namespace dynamodb \
--resource-id "table/AWSCookbook406" \
--scalable-dimension "dynamodb:table:ReadCapacityUnits" \
--policy-name "AWSCookbookReadScaling" \
--policy-type "TargetTrackingScaling" \
--target-tracking-scaling-policy-configuration \
file://read-policy.json
```

Apply the write scaling policy to the table using the write-scaling.json file

```
aws application-autoscaling put-scaling-policy \
--service-namespace dynamodb \
--resource-id "table/AWSCookbook406" \
--scalable-dimension "dynamodb:table:WriteCapacityUnits" \
--policy-name "AWSCookbookWriteScaling" \
--policy-type "TargetTrackingScaling" \
--target-tracking-scaling-policy-configuration \
file://write-policy.json
```

You can observe the Auto Scaling configuration for your table in the DynamoDB console under the Additional settings tab

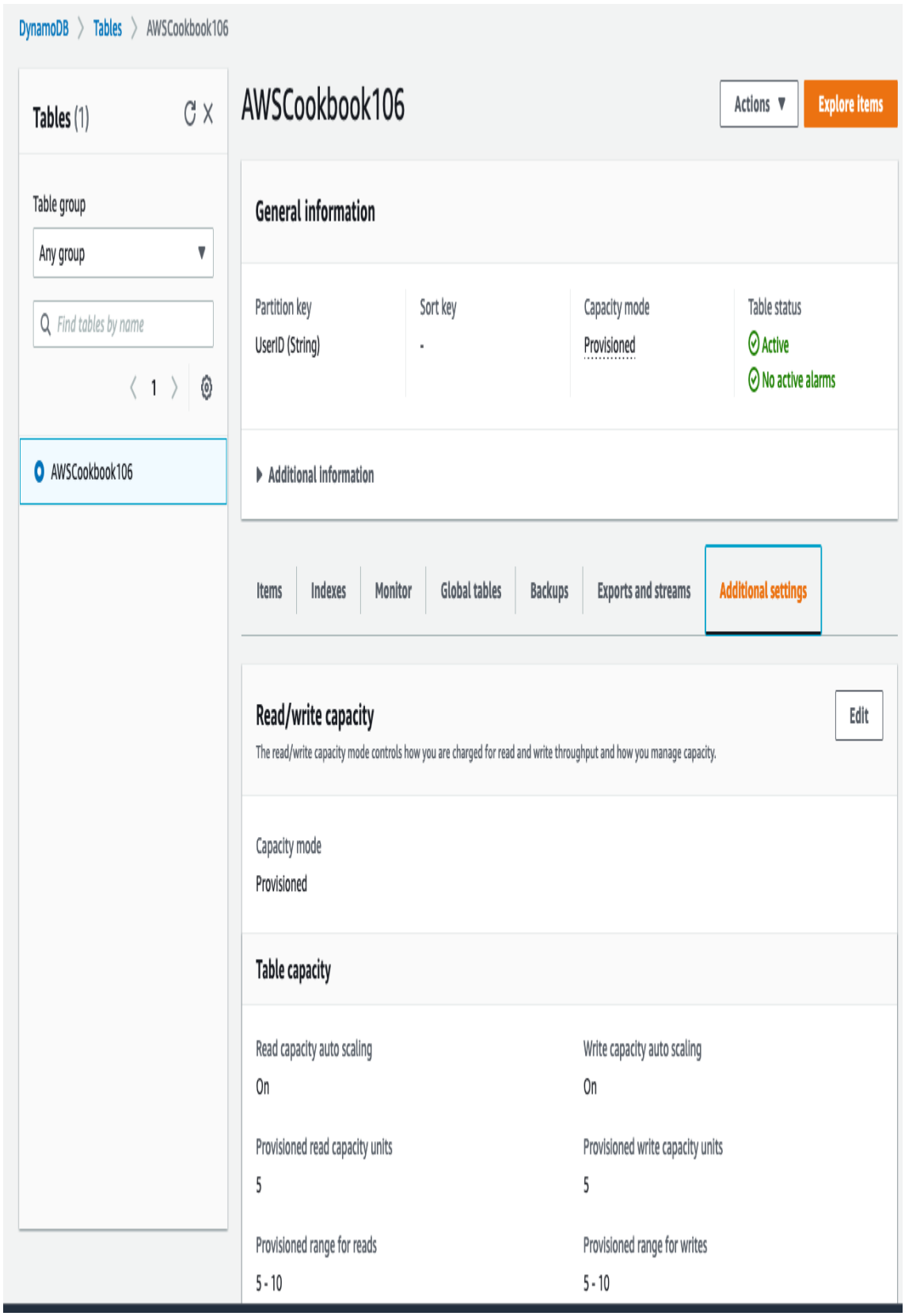

DynamoDB
Tables
AWSCookbook106
Tables (1)
Table group
Any group
Find tables by name
1
AWSCookbook106
AWSCookbook106
Actions
Explore items
General information
Partition key
UserID (String)
Sort key
-
Capacity mode
Provisioned
Table status
Active
No active alarms
Additional information
Items
Indexes
Monitor
Global tables
Backups
Exports and streams
Additional settings
Read/write capacity
The read/write capacity mode controls how you are charged for read and write throughput and how you manage capacity.
Edit
Capacity mode
Provisioned
Table capacity
Read capacity auto scaling
On
Write capacity auto scaling
On
Provisioned read capacity units
5
Provisioned write capacity units
5
Provisioned range for reads
5 - 10
Provisioned range for writes
5 - 10

Figure 2-2. DynamoDB Scaling Settings

Clean Up

Delete the DynamoDB table

```
aws dynamodb delete-table \
--table-name 'AWSCookbook406'
```

Discussion

You configured scaling targets for read and write capacity, defined a scaling policy, and applied the configuration to your DynamoDB table by using the `aws application-autoscaling put-scaling-policy` command. These steps enabled Auto Scaling for your DynamoDB table.

DynamoDB allows for two capacity modes: Provisioned, and On-Demand. With provisioned capacity mode, you specify the number of data reads and writes per second that you require for your application[17] and are charged for the capacity units that you specify. Conversely, with on-demand capacity mode, you pay per request for the data reads and writes your application performs on your tables. In general, using On-Demand mode can result in higher costs over provisioned mode for especially transactionally heavy applications.

You need to understand your application and usage patterns when selecting a provisioned capacity for your tables. If you set the capacity too low, you will experience slow database performance and your application could enter wait states. If you set the capacity too high, you are paying for unneeded capacity. Enabling Auto Scaling allows you to define minimum and maximum target values by setting a scaling target, while also allowing you to define when the Auto Scaling trigger should go into effect for scaling up, and when it should begin to scale your capacity down. This allows you to optimize for both cost and performance while taking advantage of the DynamoDB service. To see a list of the scalable targets that you configured for your table, you can use the following command:

```
aws application-autoscaling describe-scalable-targets \
--service-namespace dynamodb \
--resource-id "table/AWSCookbook406"
```

For more information on DynamoDB capacities and how they are measured, see this support document: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ProvisionedThroughput.html

2.7 Migrating Databases to Amazon RDS using Amazon DMS (Database Migration Service)

Problem

You need to move data from a source database to a target database.

Solution

Configure the VPC security groups & IAM permissions to allow DMS connectivity to the databases. Then, configure the DMS endpoints for the source and target databases. Next configure a DMS replication task. Finally start the replication task.

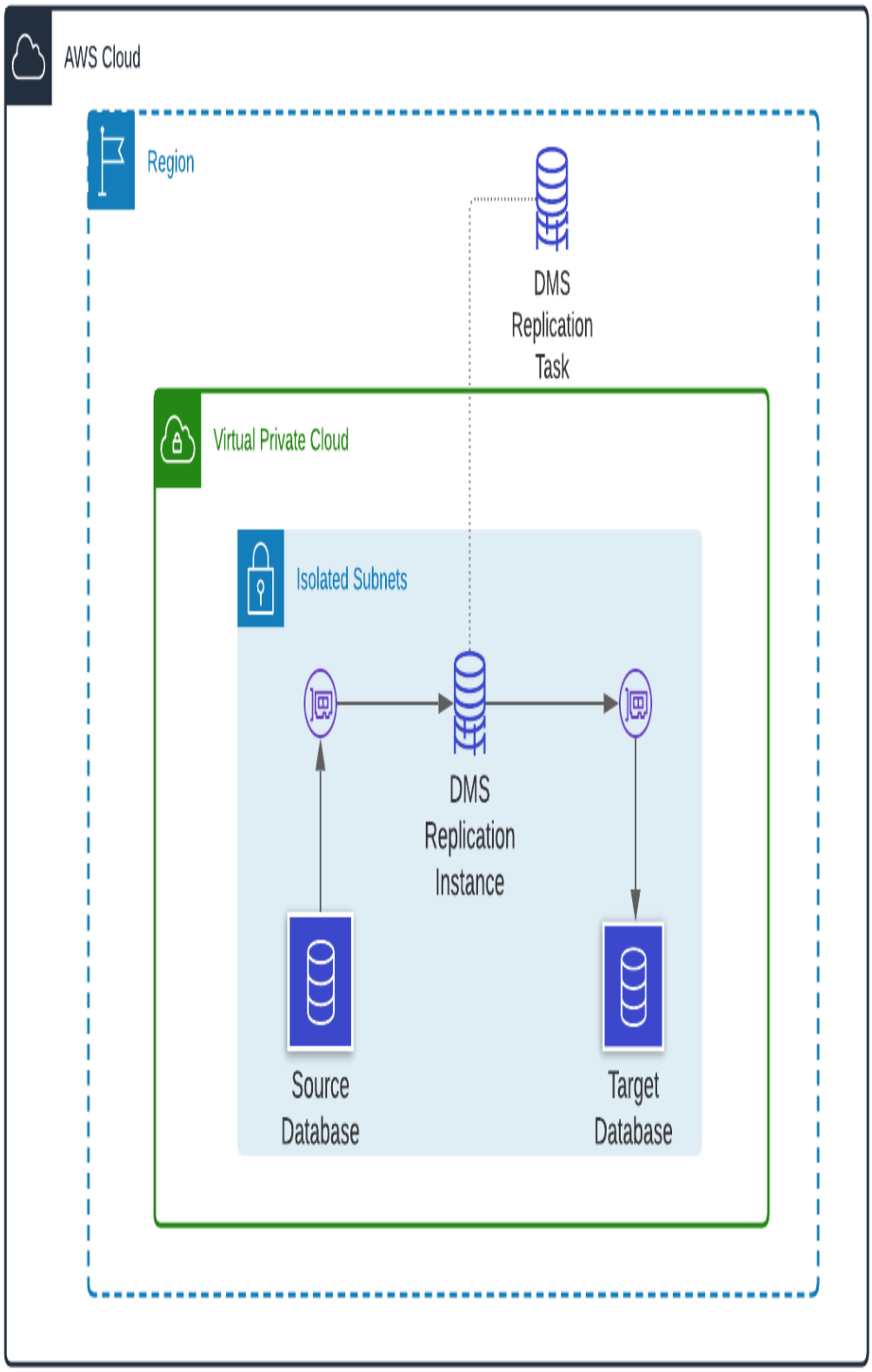

AWS Cloud
Region
DMS
Replication
Task
Virtual Private Cloud
Isolated Subnets
DMS
Replication
Instance
Source
Database
Target
Database

Figure 2-3. DMS Network Diagram

Preparation

This recipe requires some “prep work” which deploys resources that you’ll build the solution on. You will use the AWS CDK to deploy these resources

In the root of this Chapter’s repo **cd** to the “407-Migrating-Databases-to-Amazon-RDS/cdk-AWS-Cookbook-407” folder and follow the subsequent steps:

```
cd 407-Migrating-Databases-to-Amazon-RDS/cdk-AWS-Cookbook-407/
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt --no-dependencies
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

For this recipe, we will need to create a modified environment variable from the output:

```
IsolatedSubs_list=$(echo ${IsolatedSubnets} | tr -d ',' | tr -d
'"')
```

Navigate up to the main directory for this recipe (out of the “cdk-AWS-Cookbook-407” folder)

```
cd ..
```

Execute a lambda to seed the Database with some sample tables

```
aws lambda invoke \
--function-name $LambdaArn \
response.json
```

Steps

Create a security group for the replication instance

```
DMSSgId=$(aws ec2 create-security-group \
--group-name AWSCookbook407DMSSG \
--description "DMS Security Group" --vpc-id $VPCId \
--output text --query GroupId)
```

Grant the DMS Security Group access the Source and Target databases on port 3306

```
aws ec2 authorize-security-group-ingress \
--protocol tcp --port 3306 \
--source-group $DMSSgId \
--group-id $SourceRdsSecurityGroup
```

```
aws ec2 authorize-security-group-ingress \
--protocol tcp --port 3306 \
--source-group $DMSSgId \
--group-id $TargetRdsSecurityGroup
```

Create a Role for DMS using the assume-role-policy.json provided

```
aws iam create-role --role-name dms-vpc-role \
--assume-role-policy-document file://assume-role-policy.json
```

WARNING

The DMS service requires an IAM role with a specific name and a specific policy. The command you ran above satisfies this requirement. You may also already have this role in your account if you have used DMS previously. This command would result in an error if that is the case, and you can proceed with the next steps without concern.

Attach the managed DMS policy to the role

```
aws iam attach-role-policy --role-name dms-vpc-role --policy-arn
\
arn:aws:iam::aws:policy/service-role/AmazonDMSVPCManagementRole
```

Create a replication subnet group for the replication instance

```
RepSg=$(aws dms create-replication-subnet-group \
--replication-subnet-group-identifier awscookbook407 \
--replication-subnet-group-description "AWSCookbook407" \
--subnet-ids $IsolatedSubs_list \
--query ReplicationSubnetGroup.ReplicationSubnetGroupIdentifier \
--output text)
```

Create a replication instance and save the ARN in a variable

```
RepInstanceArn=$(aws dms create-replication-instance \
--replication-instance-identifier awscookbook407 \
--no-publicly-accessible \
--replication-instance-class dms.t2.medium \
--vpc-security-group-ids $DMSSgId \
--replication-subnet-group-identifier $RepSg \
--allocated-storage 8 \
--query ReplicationInstance.ReplicationInstanceArn \
--output text)
```

Wait until the ReplicationInstanceStatus reaches “available”, check the status use this command:

```
aws dms describe-replication-instances \
--filter=Name=replication-instance-id,Values=awscookbook407 \
--query ReplicationInstances[0].ReplicationInstanceStatus
```

Monitor the progress of the previous command in your terminal every 2 seconds

```
watch -g !!
```

Retrieve the source and target DB admin passwords from secretsmanager and save to environment variables

```
RdsSourcePassword=$(aws secretsmanager get-secret-value --secret-
id $RdsSourceSecretName --query SecretString | jq -r | jq
.password | tr -d '"')


RdsTargetPassword=$(aws secretsmanager get-secret-value --secret-
id $RdsTargetSecretName --query SecretString | jq -r | jq
.password | tr -d '"')
```

Create a source endpoint for DMS and save the ARN to a variable

```
SourceEndpointArn=$(aws dms create-endpoint \
--endpoint-identifier awscookbook407source \
--endpoint-type source --engine-name mysql \
--username admin --password $RdsSourcePassword \
--server-name $SourceRdsEndpoint --port 3306 \
--query Endpoint.EndpointArn --output text)
```

Create a target endpoint for DMS and save the ARN to a variable

```
TargetEndpointArn=$(aws dms create-endpoint \
--endpoint-identifier awscookbook407target \
--endpoint-type target --engine-name mysql \
--username admin --password $RdsTargetPassword \
--server-name $TargetRdsEndpoint --port 3306 \
--query Endpoint.EndpointArn --output text)
```

Create your replication task

```
ReplicationTaskArn=$(aws dms create-replication-task \
--replication-task-identifier awscookbook-task \
--source-endpoint-arn $SourceEndpointArn \
--target-endpoint-arn $TargetEndpointArn \
--replication-instance-arn $RepInstanceArn \
--migration-type full-load \
--table-mappings file://table-mapping-all.json \
--query ReplicationTask.ReplicationTaskArn --output text)
```

Wait for the status to reach “ready”. To check the status of the replication task, use the following

```
aws dms describe-replication-tasks \
--filters "Name=replication-task-arn,Values=$ReplicationTaskArn"
\
--query "ReplicationTasks[0].Status"
```

Monitor the progress of the previous command in your terminal every 2 seconds

```
watch -g !!
```

Start the replication task

```
aws dms start-replication-task \
--replication-task-arn $ReplicationTaskArn \
--start-replication-task-type start-replication
```

Monitor the progress of the replication task

```
aws dms describe-replication-tasks
```

Use the AWS console or the `aws dms describe-replication-tasks` operation to validate that your tables have been migrated

```
aws dms describe-replication-tasks \
--query ReplicationTasks[0].ReplicationTaskStats
```

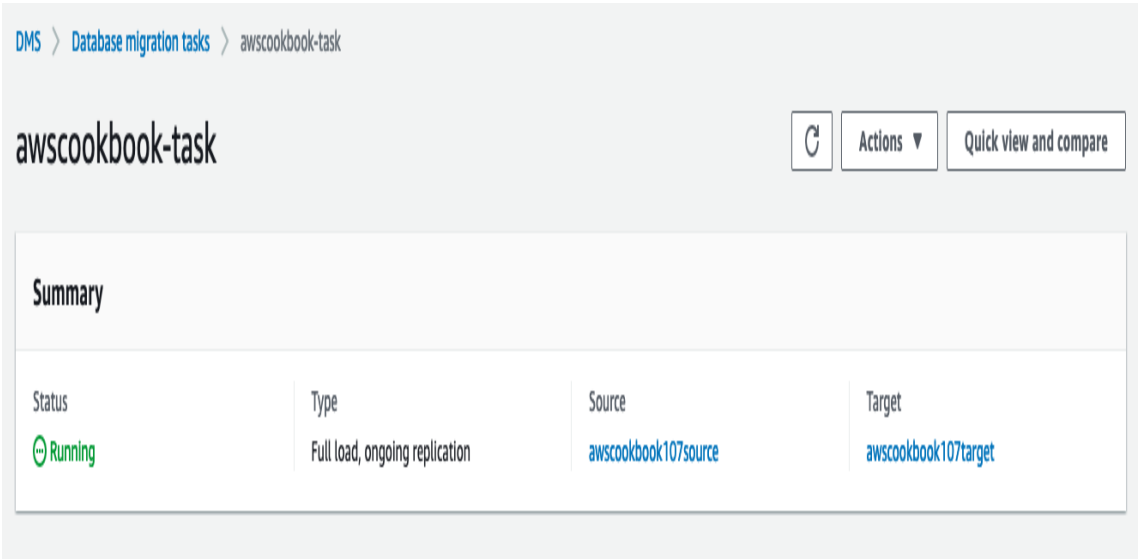


Figure 2-4. AWS Console DMS Task Overview

Clean Up

Delete the replication task

```
aws dms delete-replication-task \
--replication-task-arn $ReplicationTaskArn
```

After the replication task has finished deleting, delete the replication instance

```
aws dms delete-replication-instance \
--replication-instance-arn $RepInstanceArn
```

Detach the security group references from the RDS Security Groups

```
aws ec2 revoke-security-group-ingress \
--protocol tcp --port 3306 \
--source-group $DMSSgId \
--group-id $SourceRdsSecurityGroup

aws ec2 revoke-security-group-ingress \
--protocol tcp --port 3306 \
--source-group $DMSSgId \
--group-id $TargetRdsSecurityGroup
```

Detach the DMS policy from the role you created

```
aws iam detach-role-policy --role-name dms-vpc-role --policy-arn
\
arn:aws:iam::aws:policy/service-role/AmazonDMSVPCManagementRole
```

Delete the role you created for DMS

```
aws iam delete-role --role-name dms-vpc-role
```

Delete the Source and Target DMS endpoints

```
aws dms delete-endpoint --endpoint-arn $SourceEndpointArn
aws dms delete-endpoint --endpoint-arn $TargetEndpointArn
```

After the endpoints have been deleted, delete the DMS Security group you created

```
aws ec2 delete-security-group --group-id $DMSSgId
```

> **TIP**
>
> You may need to wait a few minutes for the network interfaces associated with the DMS endpoints to delete. If you would like to force the deletion, you can go to the EC2 console, select network interfaces, and delete the interface with the description "DMSNetworkInterface"

Delete the DMS subnet groups

```
aws dms delete-replication-subnet-group \
--replication-subnet-group-identifier awscookbook407
```

Go to the cdk-AWS-Cookbook-407 directory

```
cd cdk-AWS-Cookbook-407/
```

To clean up the environment variables, run the `helper.py` script in this recipe's `cdk-` folder with the `--unset` flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Unset the environment variable that you created manually

```
unset RepSg
unset RepInstanceArn
unset RdsSourcePassword
unset RdsTargetPassword
unset SourceEndpointArn
unset TargetEndpointArn
unset ReplicationTaskArn
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && cd ../..
```

Discussion

You completed a database migration using Amazon Database Migration Service (Amazon DMS). After configuring the security groups, subnets, and IAM role, you launched a DMS replication instance. This instance ran an engine for DMS which performs the migration from source to target follwing the specifications you passed in using the table-mapping.json file. The replication instance connects to the endpoints you configured for source and target and completes replication tasks that you initiate. You ran a one time `full-load` migration with this recipe. You could also run a `full-load-and-cdc` to continuously replicate changes on the source to the destination to minimize your application downtime when you cut over to the new database.

DMS comes with functionality to test source and destination endpoints from the replication instance. This is a handy feature to use when working with

DMS to validate that you have the configuration correct before you start to run replication tasks. Testing connectivity from the replication instance to both of the endpoints you configured can be done via the DMS console or via the command line with the following commands:

```
aws dms test-connection \
--replication-instance-arn $rep_instance_arn \
--endpoint-arn $source_endpoint_arn


aws dms test-connection \
--replication-instance-arn $rep_instance_arn \
--endpoint-arn $target_endpoint_arn
```

The `test-connection` takes a few moments to complete. You can check the status and the results of the test-connection operation by using this command:

```
aws dms describe-connections --filter \
"Name=endpoint-
arn,Values=$source_endpoint_arn,$target_endpoint_arn"
```

The DMS service supports a myriad of source and target databases. It can also transform data for you if your source and destination are different types of databases by using additional configuration in the table-mappings.json file. For example, the data type of a column in a Oracle Database may have a different format than the type in a PostgreSQL database. The AWS Schema Conversion Tool (SCT) can assist with identifying these necessary transforms, and also generate configuration files to use with DMS[18].

2.8 Enabling the Data API for a Web-Services to Aurora Serverless

Problem

You have a PostgreSQL database and you'd like to connect to it without having your application manage persistent database connections.

Solution

First, Enable the Data API for your database and configure the IAM permissions for your EC2 instance. Then, test from both the CLI and RDS console.

Preparation

This recipe requires some "prep work" which deploys resources that you'll build the solution on. You will use the AWS CDK to deploy these resources

In the root of this Chapter's repo **cd** to the "408-Working-with-Aurora-and-Data-APIs/cdk-AWS-Cookbook-408" folder and follow the subsequent steps:

```
cd 408-Working-with-Aurora-and-Data-APIs/cdk-AWS-Cookbook-408/
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt --no-dependencies
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

Navigate up to the main directory for this recipe (out of the "cdk-AWS-Cookbook-402" folder)

```
cd ..
```

Steps

Enable the Data API on your Aurora Serverless Cluster

```
aws rds modify-db-cluster \
--db-cluster-identifier $ClusterIdentifier \
--enable-http-endpoint \
--apply-immediately
```

Ensure that the HttpEndpointEnabled is set to true

```
aws rds describe-db-clusters \
--db-cluster-identifier $ClusterIdentifier \
--query DBClusters[0].HttpEndpointEnabled
```

Test a command from your CLI

```
aws rds-data execute-statement \
--secret-arn "$SecretArn" \
--resource-arn "$ClusterArn" \
--database "$DatabaseName" \
--sql "select * from pg_user" \
--output json
```

(Optional) You can also test access via the AWS Console using the Amazon RDS Query Editor. First run these two commands from your terminal so that you can copy and paste the values.

```
echo $SecretArn
echo $DatabaseName
```

Login to the AWS Console with Admin permissions and go to the RDS Console

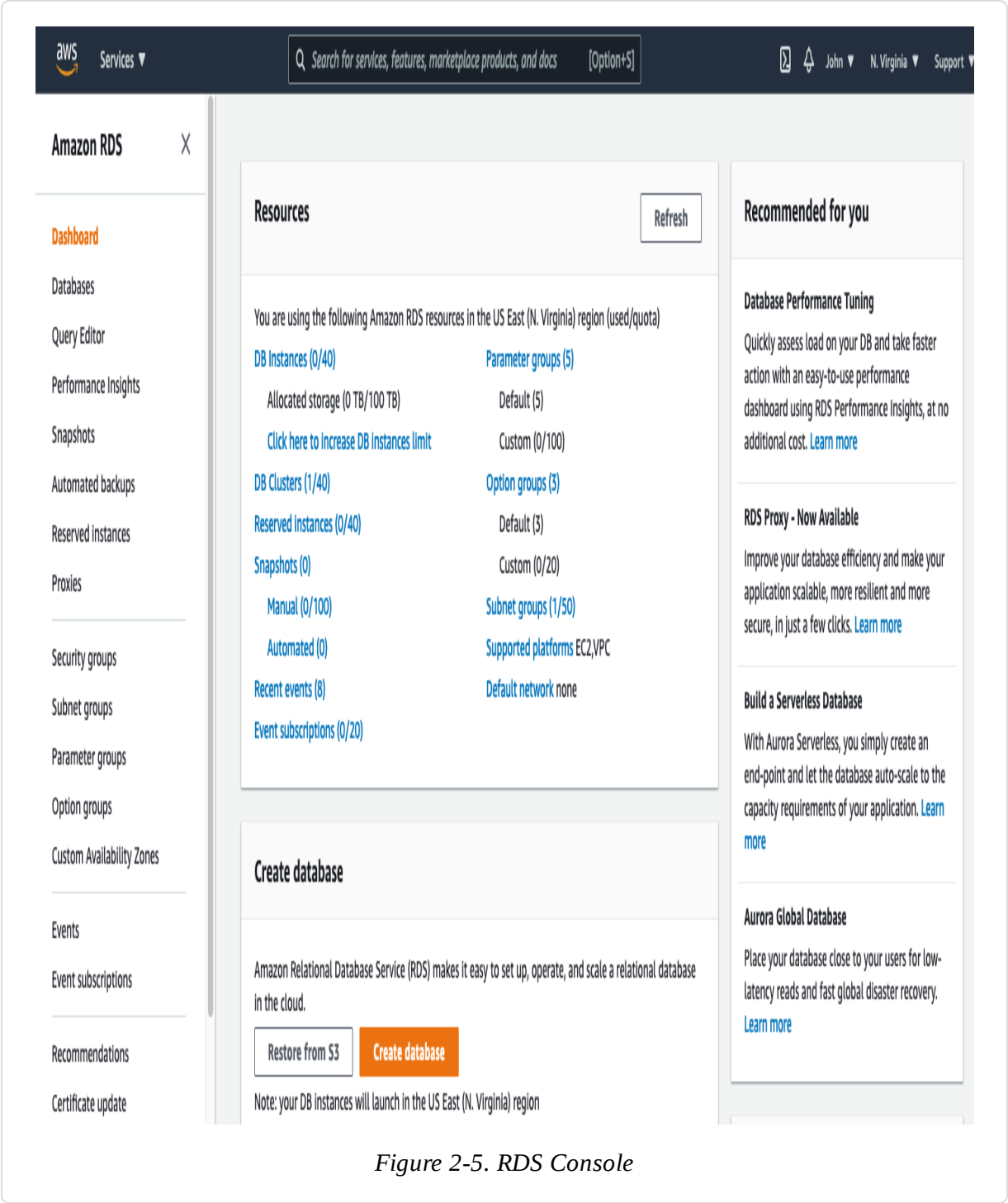


Figure 2-5. RDS Console

On the left hand sidebar menu, click on Query Editor

Fill out the values and select “Connect to database”

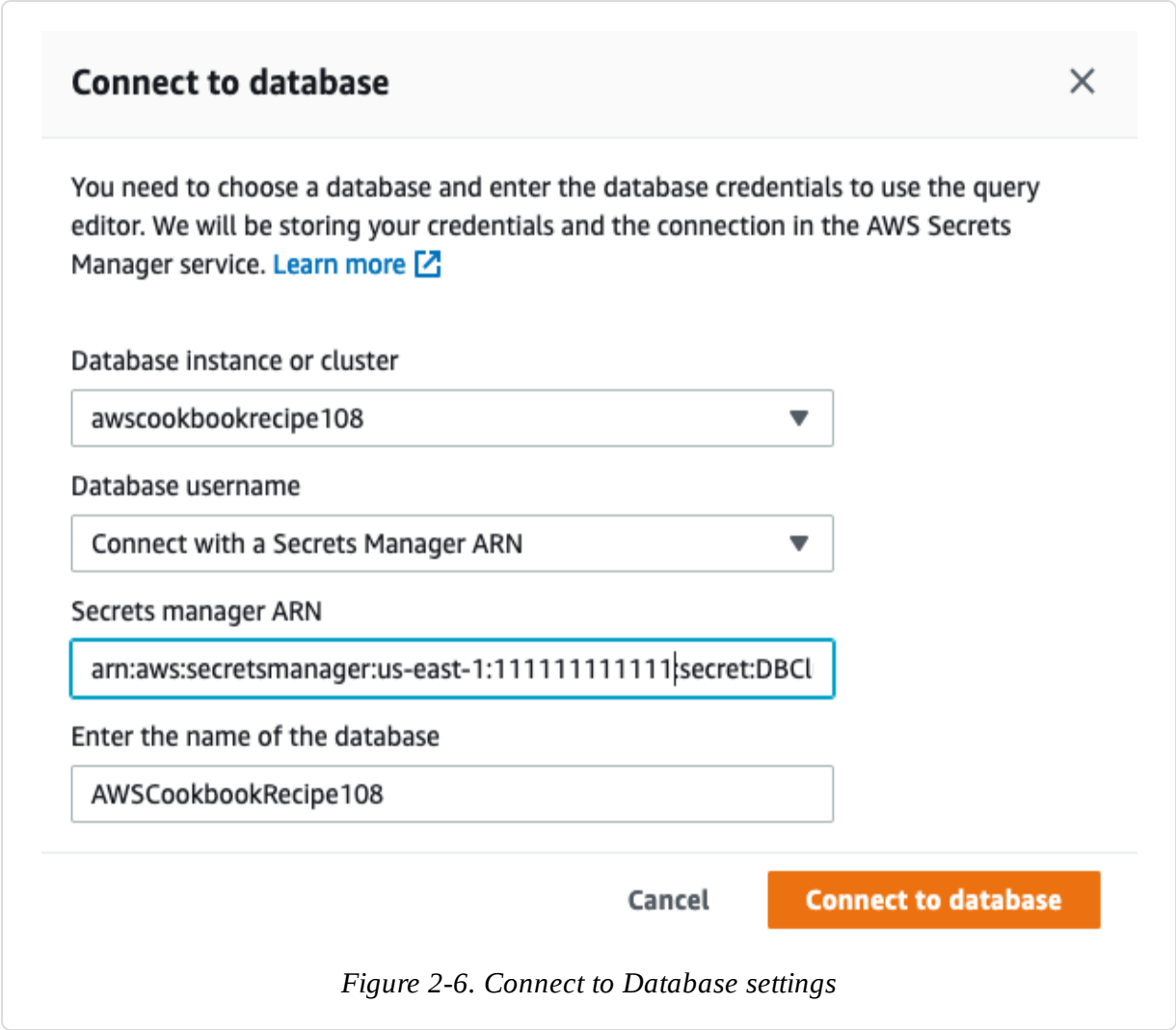


Figure 2-6. Connect to Database settings

Run the same query and view the results below the query editor

```
SELECT * from pg_user;
```

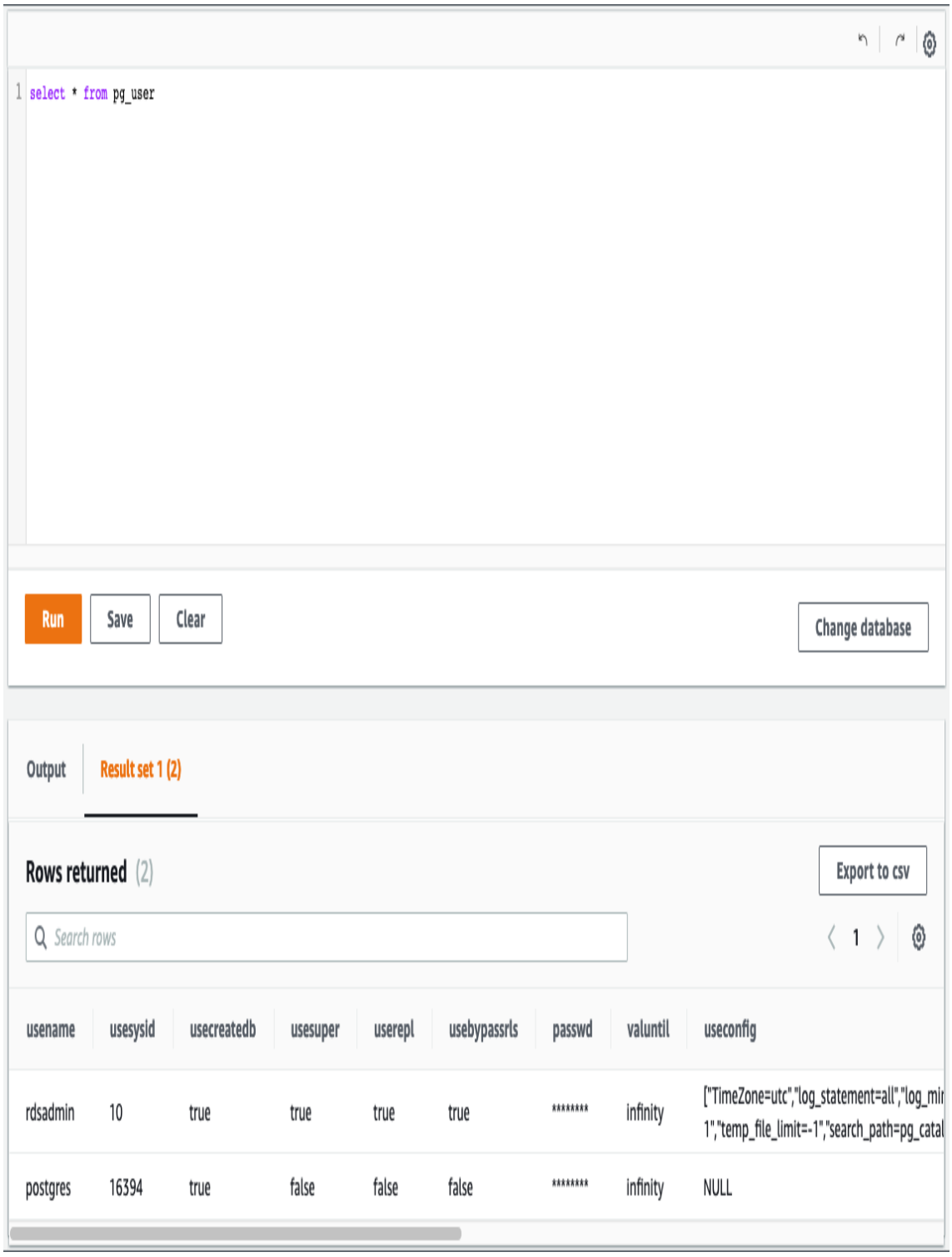


Figure 2-7. RDS Query Editor

Next you will configure your EC2 instance to use the data API with your database cluster.

To add permissions, create a file called policy-template.json with the following content (Provided in repo):

```
{
        "Version": "2012-10-17",
        "Statement": [
        {
                "Action": [
                "rds-data:BatchExecuteStatement",
                "rds-data:BeginTransaction",
                "rds-data:CommitTransaction",
                "rds-data:ExecuteStatement",
                "rds-data:RollbackTransaction"
                ],
                "Resource": "*",
                "Effect": "Allow"
        },
        {
                "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
                ],
                "Resource": "SecretArn",
                "Effect": "Allow"
        }
        ]
}
```

Replace the values in the template file using the sed command with environment variables you have set:

```
sed -e "s/SecretArn/${SecretArn}/g" \
policy-template.json > policy.json
```

Create an IAM Policy from using the file you just created

```
aws iam create-policy --policy-name AWSCookbook408RDSDataPolicy \
--policy-document file://policy.json
```

Attach the IAM policy for AWSCookbook408EC2RDSDataPolicy to your EC2 Instance’s IAM role

```
aws iam attach-role-policy --role-name $EC2RoleName \
--policy-arn
arn:aws:iam::$AWS_ACCOUNT_ID:policy/AWSCookbook408RDSDataPolicy
```

Set some SSM Parameters. This will make it easy to pull the values while testing on the provided EC2 instance (created in preparation steps)

```
aws ssm put-parameter \
--name "Cookbook408DatabaseName" \
--type "String" \
--value $DatabaseName aws ssm put-parameter \
--name "Cookbook408ClusterArn" \
--type "String" \
--value $ClusterArn


aws ssm put-parameter \
--name "Cookbook408SecretArn" \
--type "String" \
--value $SecretArn
```

Ensure the provided Instance has registered with SSM. Use this command to check the status. This command should return the instance ID

```
aws ssm describe-instance-information \
--filters Key=ResourceType,Values=EC2Instance \
--query "InstanceInformationList[].InstanceId" --output text
```

Connect to the EC2 instance

```
aws ssm start-session --target $InstanceID
```

Set the region

```
export AWS_DEFAULT_REGION=us-east-1
```

Retrieve the SSM Parameter values and set them to environment values

```
DatabaseName=$(aws ssm get-parameters \
--names "Cookbook408DatabaseName" \
  --query "Parameters[*].Value" --output text)SecretArn=$(aws ssm
```

```
get-parameters \
--names "Cookbook408SecretArn" \
  --query "Parameters[*].Value" --output text)ClusterArn=$(aws
ssm get-parameters \
--names "Cookbook408ClusterArn" \
  --query "Parameters[*].Value" --output text)
```

Run a query against that the Database

```
aws rds-data execute-statement \
--secret-arn "$SecretArn" \
--resource-arn "$ClusterArn" \
--database "$DatabaseName" \
--sql "select * from pg_user" \
--output json
```

Log out of the EC2 instance

```
exit
```

Clean Up

Delete the SSM parameters

```
aws ssm delete-parameter --name Cookbook408DatabaseName
aws ssm delete-parameter --name Cookbook408SecretArn
aws ssm delete-parameter --name Cookbook408ClusterArn
```

Detach the policy from the role

```
aws iam detach-role-policy --role-name $EC2RoleName \
--policy-arn
arn:aws:iam::$AWS_ACCOUNT_ID:policy/AWSCookbook408RDSDataPolicy
```

Delete the IAM Policy

```
aws iam delete-policy --policy-arn \
arn:aws:iam::$AWS_ACCOUNT_ID:policy/AWSCookbook408RDSDataPolicy
```

Go to the cdk-AWS-Cookbook-408 directory

```
cd cdk-AWS-Cookbook-408/
```

To clean up the environment variables, run the `helper.py` script in this recipe's `cdk-` folder with the `--unset` flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && cd ../..
```

Discussion

You enabled the Data API for your Aurora Serverless cluster. You then queried your database through a terminal and in the RDS console leveraging the Data API. The Data API exposes an HTTPS endpoint for usage with Aurora and uses IAM authentication to allow your application to execute SQL statements on your database over HTTPS instead of using classic TCP database connectivity[19].

> **TIP**
>
> All calls to the Data API are synchronous and the default timeout for a query is 45 seconds. If your queries take longer than 45 seconds, you can use the `continueAfterTimeout` parameter to facilitate long-running queries[20].

As is the case with other AWS service APIs which use IAM authentication, all activities performed with the Data API are captured in CloudTrail to ensure an audit trail is present, which can help satisfy your security and audit requirements[21]. You can control and delegate access to the Data API endpoint using IAM policies associated with roles for your application. For example, if you wanted to grant your application the ability to only read from your

database using the Data API, you could write a policy that omits the `rds-data:CommitTransaction` and `rds-data:RollbackTransaction` permissions.

The Query Editor within the RDS console provides a web-based means of access for executing SQL queries against your database. This is a convenient mechanism for developers and DBAs to quickly accomplish bespoke tasks. The same privileges that you assigned your EC2 instance in this recipe would need to be granted to your developer and DBA via IAM roles.

---

1 https://en.wikipedia.org/wiki/Database

2 https://aws.amazon.com/products/databases/

3 https://aws.amazon.com/rds/aurora/faqs/#Serverless

4 https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/aurora-serverless.modifying.html

5 https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/aurora-serverless.how-it-works.html

6 https://aws.amazon.com/rds/aurora/

7 https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/CHAP_AuroraOverview.html

8 https://awscli.amazonaws.com/v2/documentation/api/latest/reference/rds/modify-db-cluster.html

9 https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.IAMDBAuth.html#UsingWithRDS.IAMDBAuth.Availability

10 https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.IAMDBAuth.html

11 https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.SSL.html

12 https://docs.aws.amazon.com/lambda/latest/dg/runtimes-context.html

13 https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/rds-proxy.html

14 https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Overview.Encryption.html

15 https://aws.amazon.com/compliance/shared-responsibility-model/

16 https://docs.aws.amazon.com/lambda/latest/dg/configuration-layers.html

17 https://aws.amazon.com/dynamodb/pricing/provisioned/

18 https://aws.amazon.com/dms/schema-conversion-tool/

19 https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/data-api.html

20 https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/data-api.html#data-api.calling

21 https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/logging-using-cloudtrail-data-api.html

Chapter 3. Containers

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 6th chapter of the final book. If you have feedback or content suggestions for the authors, please email *awscookbook@gmail.com*.

3.0 Introduction

A container, put simply, packages application code, binaries, configuration files, and libraries together into a single executable package, called a container image. By packaging everything together in this way, you can develop, test, and run applications with control and consistency. Containers allow you to quickly start packaging up and testing things that you build locally, while ensuring that the exact same runtime environment is present regardless of where it is running. This generally reduces the time it takes to build something and offer it to a wide audience. Whether it's your personal blog, portfolio, or some cool new app you're building, making containers a part of your development workflow has many benefits.

Containers are wholly-"contained" environments that leverage the underlying compute and memory capabilities on the host where they are running (your laptop, a server in a closet, or the cloud). Multiple containers can be run on the same host at once without conflicts. You can also have multiple containers running with the intention of them communicating with one another. Think of a case where you have a front-end web application running as a container which accesses a container running a back-end for your website. This interoperability is especially important for what you will explore with containers on AWS. Running multiple containers at once and ensuring they are always available can present some challenges, which is

why you enlist the help of a container “orchestrator”. Popular orchestrators come in many flavors, but some of the common ones that you may have heard of are Kubernetes and Docker Swarm.

AWS has several choices for working with container-based workloads. You have options like Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS) as container orchestrators, and Amazon Elastic Cloud Compute (Amazon EC2) for deployments with custom requirements. Both of the AWS container orchestrator services mentioned (Amazon ECS and Amazon EKS) can run workloads on Amazon EC2 or on the fully-managed AWS Fargate compute engine. In other words, you can choose to control the underlying EC2 instance (or instances) responsible for running your containers on Amazon ECS and Amazon EKS, allowing some level of customization to your host, or, you can use Fargate, which is fully-managed by AWS so you don’t have to worry about instance management. AWS provides a comprehensive listing of all up to date container services here: https://aws.amazon.com/containers/

Some AWS services (AWS CodeDeploy, AWS CodePipeline and Amazon Elastic Container Registry) can help streamline the development lifecycle and provide automation to your workflow. These integrate well with Amazon ECS and Amazon EKS. Some examples of AWS services that provide Network capabilities are Amazon Virtual Private Cloud, Elastic Load Balancing, AWS Cloud Map, Amazon Route 53. Logging and monitoring concerns can be addressed by Amazon CloudWatch. Fine-grained security capabilities can be provided by AWS Identity and Access Management (IAM) and AWS Key Management System (KMS). By following the recipes in this chapter, you will see how these services combine to meet your needs.

Workstation Configuration

You will need a few things installed to be ready for the recipes in this chapter:

General Setup

Set and export your default region in your terminal

```
AWS_REGION=us-east-1
```

Validate AWS Command Line Interface (AWS CLI) setup and access

```
aws ec2 describe-instances
```

Set your AWS ACCOUNT ID by parsing output from the `aws sts get-caller-identity` operation.

```
AWS_ACCOUNT_ID=$(aws sts get-caller-identity \
--query Account --output text)
```

> **NOTE**
>
> The aws sts get-caller-identity operation "returns details about the IAM user or role whose credentials are used to call the operation." From: https://awscli.amazonaws.com/v2/documentation/api/latest/reference/sts/get-caller-identity.html

Checkout this Chapter's repo

```
git clone https://github.com/AWSCookbook/Containers
```

Docker Installation and Validation

Docker Desktop is recommended for Windows and Mac users, Docker Linux Engine is recommended for Linux users

In the following recipes, you'll use Docker to create a consistent working environment on your particular platform. Be sure to install the latest stable version of Docker for your OS.

MacOS

1. Follow instructions from Docker Desktop: https://docs.docker.com/docker-for-mac/install/

2. Run the Docker Desktop Application after installation

Windows

1. Follow instructions from Docker Desktop: https://docs.docker.com/docker-for-windows/install/
2. Run the Docker Desktop Application after installation

Linux

1. Follow instructions from Docker: https://docs.docker.com/engine/install/
2. Start the Docker Daemon on your distribution

CLI Docker Setup Validation

```
docker --version
```

Output:

```
Docker version 19.03.13, build 4484c46d9d
docker images
```

Output:

```
REPOSITORY      TAG           IMAGE ID            CREATED
SIZE
```

3.1 Building, Tagging, and Pushing a Container Image to Amazon ECR

Problem

You need a repository to store built and tagged container images.

Solution

First, you will create a repository in Amazon ECR. Next, you will create a Dockerfile and build a Docker image using it. Finally you will apply two tags to the container image and push them both to the newly created ECR repository.

Steps

Create a private repository in the AWS Management Console:

Log in to the console and search for “elastic container registry”

Figure 3-1. AWS Console

Click the “Create Repository” button. Give your repository a name, keep all defaults, scroll to the bottom, and click “Create Repository” again to finish

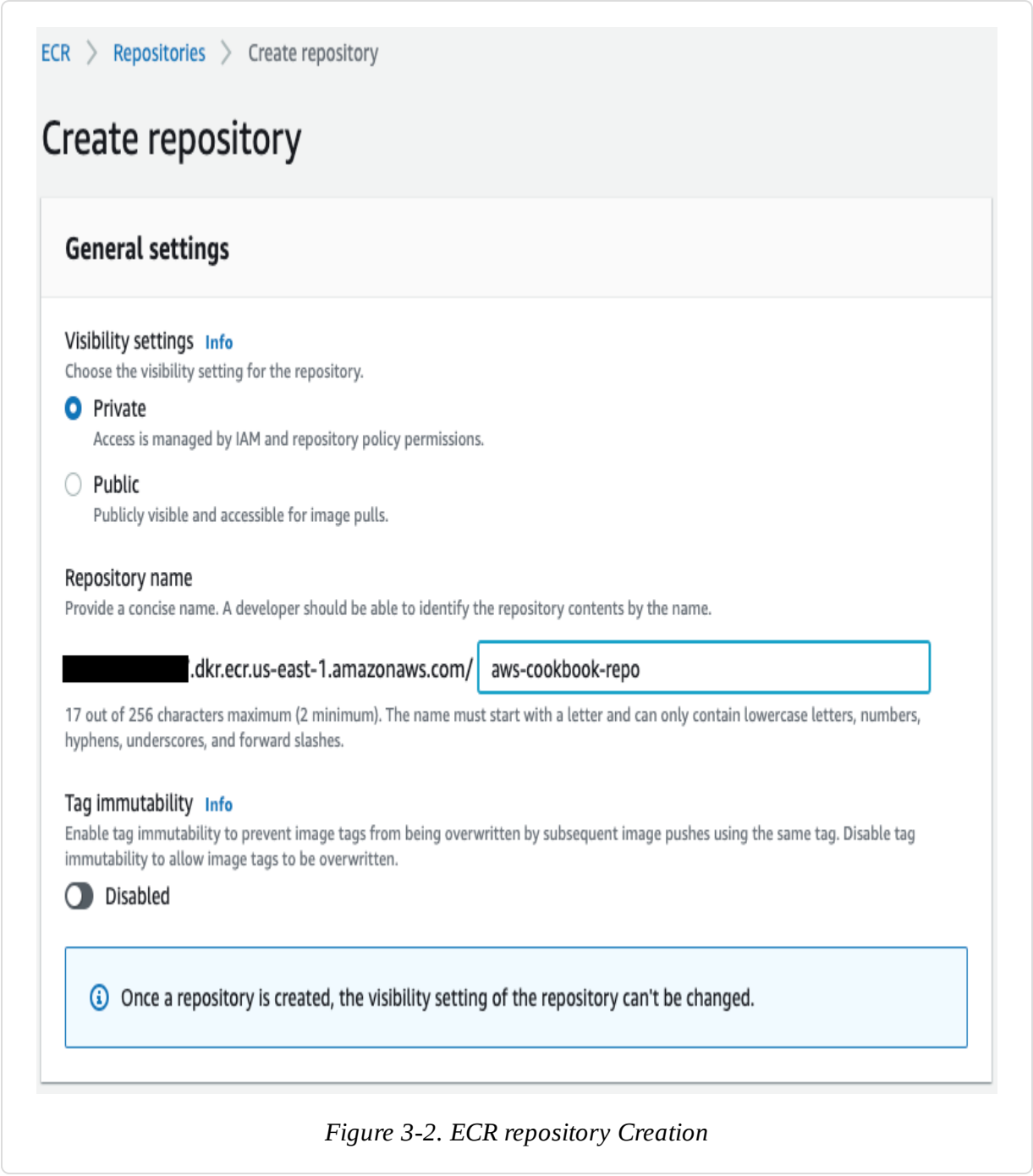


Figure 3-2. ECR repository Creation

You now have a repository created on Amazon ECR which you can use to push container images to!

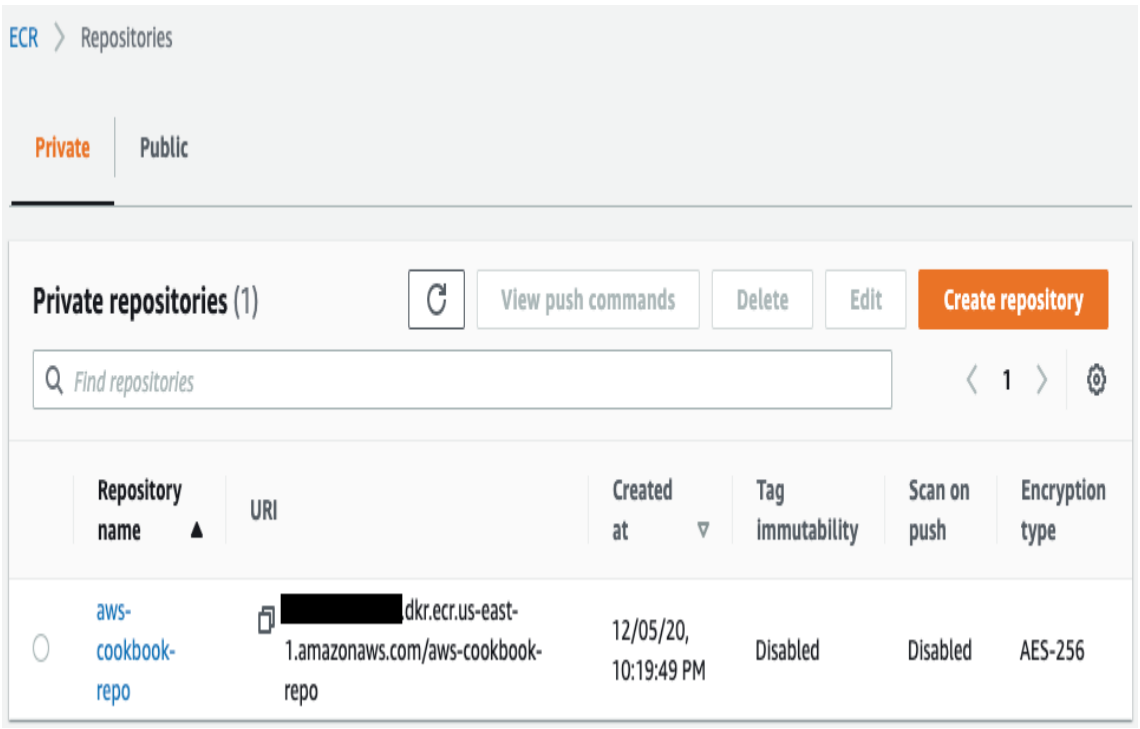


Figure 3-3. Screenshot of created ECR repository

As an alternate, you can also create an ECR repository from the command line:

```
aws ecr create-repository --repository-name aws-cookbook-repo
```

Whether you used the console or command line to create your ECR repository, use these commands to build, tag, and push a container image to the ECR repository:

Create a simple Dockerfile

```
echo FROM nginx:latest > Dockerfile
```

> **NOTE**
>
> This command creates a Dockerfile which contains a single line instructing the Docker Engine to use the nginx:latest image as the base image. Since you only use the base image with no other lines in the Dockerfile, the resulting image is identical to the nginx:latest image. You could include some HTML files within this image using the COPY and ADD Dockerfile directives.

Build and tag the image. This step may take a few moments as it downloads and combines the image layers.

```
docker build . -t \
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/aws-cookbook-
repo:latest
```

Add an additional tag.

```
docker tag \
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/aws-cookbook-
repo:latest \
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/aws-cookbook-
repo:1.0
```

Get docker login information:

```
aws ecr get-login-password | docker login --username AWS \
--password-stdin $AWS_ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com
```

Output:

```
Login Succeeded
```

> **TIP**
>
> Authentication is important to understand and get right with your repository. An authorization token needs to be provided each time an operation is executed against a private repository. Tokens last for twelve hours. An alternative mechanism that helps with frequent credential refreshes is the Amazon ECR Docker Credential Helper, available from the awslabs github repository.

Push each image tag to Amazon ECR:

```
docker push \
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/aws-cookbook-
repo:latest
```

```
docker push \
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/aws-cookbook-
repo:1.0
```

> **NOTE**
>
> You will see “Layer already exists” for the image layer uploads on the second push. This is because the image already exists in the ECR repository due to the first push, but this step is still required to add the additional tag.

Now you can view both of the tagged images in Amazon ECR from the console

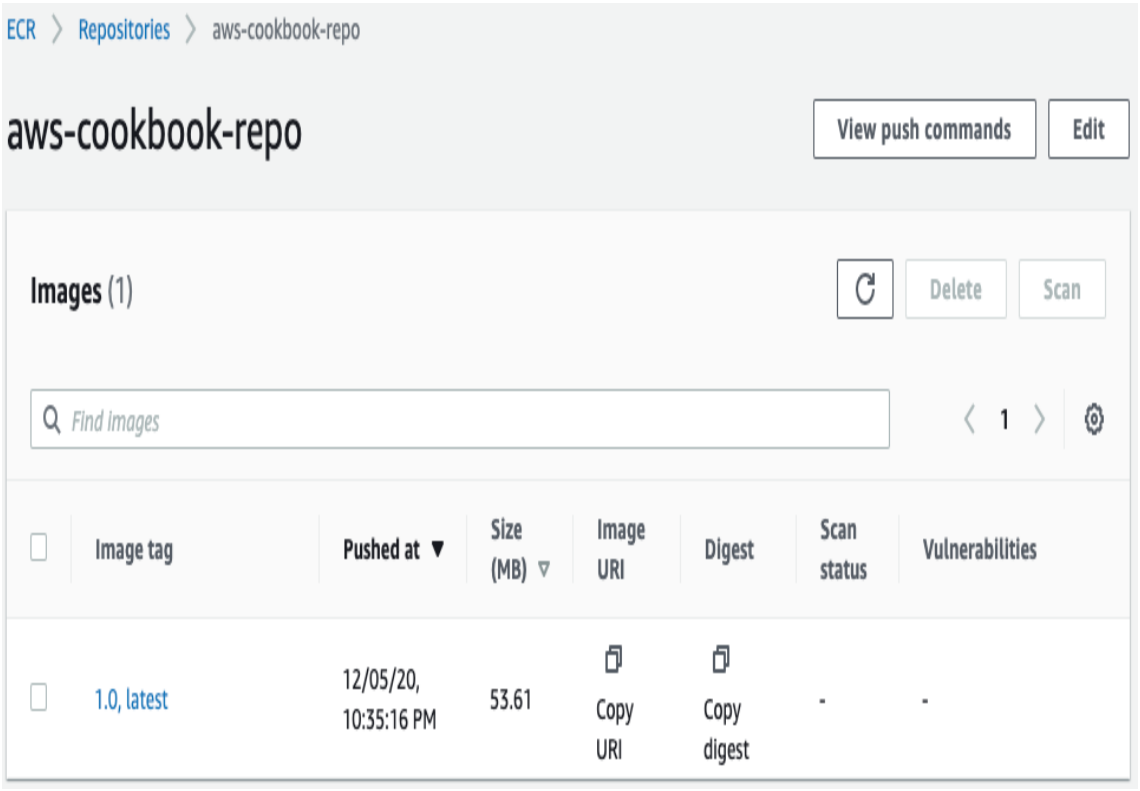


Figure 3-4. Screenshot of the image with two tags

Alternatively, you can use the AWS CLI to list the images

```
aws ecr list-images --repository-name aws-cookbook-repo
```

Output:

```
{
    "imageIds": [
        {
            "imageDigest":
"sha256:99d0a53e3718cef59443558607d1e100b325d6a2b678cd2a48b05e5e2
2ffeb49",
            "imageTag": "1.0"
        },
        {
            "imageDigest":
"sha256:99d0a53e3718cef59443558607d1e100b325d6a2b678cd2a48b05e5e2
2ffeb49",
```

```
            "imageTag": "latest"
        }
}
```

Clean Up

Remove the image from ECR

```
aws ecr batch-delete-image --repository-name aws-cookbook-repo \
--image-ids imageTag=latest aws ecr batch-delete-image --
repository-name aws-cookbook-repo \
--image-ids imageTag=1.0
```

Delete the image from your local machine

```
docker image rm \
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/aws-cookbook-
repo:1.0docker image rm \
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/aws-cookbook-
repo:latest
```

Delete the repository

```
>aws ecr delete-repository --repository-name aws-cookbook-repo
```

Discussion

You created a simple Dockerfile which defined a base image for your new container. Next you built an image from the Dockerfile. This pulled images layers down to your workstation and built a local image. Next, you created an ECR repository. One this was done, you then tagged the image twice with `:latest` and `:1.0`. The default behavior of the Docker CLI assumes Docker Hub as the repository destination. The whole universal resource identifier (URI) of the ECR repository is required with the tags so you can push to the ECR repository you created. You used a command line pipe to inject the ECR authorization token into the `docker login` command which gave your docker client access to the ECR repository. Finally you

pushed the image to the ECR repository, this uploaded all the image layers from your local machine to Amazon ECR.

Having a repository for your container images is an important foundational component of the application development process. You can grant access to other AWS accounts, IAM entities, and AWS services with permissions for Amazon ECR. Now that you know how to create an ECR repository, you will be able to store your container images and use them with AWS services.

> **NOTE**
>
> Amazon ECR supports classic Docker Image Manifest V2 Schema 2 and most recently OCI (Open Container Initiative) images. It can translate between these formats on pull. Legacy support is available for Manifest V2 Schema 1 and Amazon ECR can translate on the fly when interacting with legacy docker client versions. The experience should be seamless for most docker client versions in use today.

Container tagging allows you to version and keep track of your container images. You can apply multiple tags to an image. The Docker CLI pushes tagged images to the repository and the tags can be used with pulls. It is common in CI/CD to use the `:latest` tag for your builds, in addition to a version tag like `:1.0`. Since tags can be overwritten when you push, you can always use the `:latest` tag as part of your workflow to ensure that you will always be pushing a pointer to the latest built image for running your containers.

3.2 Scanning Images for Security Vulnerabilities on Push to Amazon ECR

Problem

You want to scan your container images for security vulnerabilities each time you push to a repository.

Solution

Enable automatic image scanning on a repository in Amazon ECR and observe the results.

Preparation

Create an ECR repository

```
aws ecr create-repository --repository-name aws-cookbook-repo
```

Rather than building a new container image from a Dockerfile (as you did in recipe 6.1), this time you are going to pull an old NGINX container image.

```
docker pull nginx:1.14.1
```

Steps

On the command line, apply the scanning configuration to the repository that you created:

```
aws ecr put-image-scanning-configuration \
--repository-name aws-cookbook-repo \
--image-scanning-configuration scanOnPush=true
```

Get docker login information:

```
aws ecr get-login-password | docker login --username AWS \
--password-stdin $AWS_ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com
```

Apply a tag to the image so that you can push it to the ECR repository:

```
docker tag nginx:1.14.1 \
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/aws-cookbook-
repo:old
```

Push the image:

```
docker push \
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/aws-cookbook-
repo:old
```

Shortly after the push is complete, you can examine the results of the security scan of the image in JSON format:

```
aws ecr describe-image-scan-findings \
--repository-name aws-cookbook-repo --image-id imageTag=old
```

Snippet of Output:

```
{
    "imageScanFindings": {
        "findings": [
            {
                "name": "CVE-2019-3462",
                "description": "Incorrect sanitation of the 302
redirect field in HTTP transport method of apt versions 1.4.8 and
earlier can lead to content injection by a MITM attacker,
potentially leading to remote code execution on the target
machine.",
                "uri": "https://security-
tracker.debian.org/tracker/CVE-2019-3462",
                "severity": "CRITICAL",
                "attributes": [
                    {
                        "key": "package_version",
                        "value": "1.4.8"
                    },
```

Clean Up

Delete the image from your local machine

```
docker image rm \
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/aws-cookbook-
repo:old
docker image rm nginx:1.14.1
```

Delete the image from ECR:

```
aws ecr batch-delete-image --repository-name aws-cookbook-repo \
--image-ids imageTag=old
```

Now delete the repository

```
aws ecr delete-repository --repository-name aws-cookbook-repo
```

> **TIP**
>
> Amazon ECR has a safety mechanism built-in which does not let you delete a repository containing images. If the repository is not empty and the delete-repository command is failing, you can bypass this check by adding `--force` to the `delete-repository` command.

Discussion

You created an ECR repository and enabled automatic scanning on push with the `put-image-scanning-configuration` command. You can enable this feature when you create a repository or anytime after. You then pulled an older version of an NGINX server image from the NGINX official Docker Hub repository, re-tagged it with your ECR repository, and pushed it. This triggered a vulnerability scan of the image. Lastly, you observed the security vulnerabilities associated with the older container image.

The Common Vulnerabilities and Exposures (CVEs) database from the open-source Clair project is used by Amazon ECR for vulnerability scanning[1]. You are provided a CVSS (Common Vulnerability Scoring System) score to indicate the severity of any detected vulnerabilities. This helps you detect and remediate vulnerabilities in your container image. You can configure alerts for newly discovered vulnerabilities in images using Amazon EventBridge and Amazon Simple Notification Service (Amazon SNS).

> **WARNING**
>
> The scanning feature does not continuously scan your images, so it is important to push your versions routinely (or trigger a manual scan).

You can retrieve the results of the last scan for an image at any time with the command used in the last step of this recipe. Furthermore, you can use these commands as part of an automated CI/CD process that may validate whether or not an image has a certain CVSS score before deploying.

3.3 Deploying a container using Amazon Lightsail

Problem

You need to quickly deploy a container and access it securely over the internet.

Solution

Deploy a plain NGINX container which listens on port 80 to Lightsail. Lightsail provides a way to quickly deploy applications to AWS.

Preparation

In addition to Docker Desktop and the AWS CLI (Version 2), you need to install the Lightsail Control plugin (lightsailctl) for the AWS CLI. It is a quick install supported on Windows, Mac, and Linux. You can follow the instructions for your platform here: https://lightsail.aws.amazon.com/ls/docs/en_us/articles/amazon-lightsail-install-software

> **NOTE**
>
> There are several power levels available for Lightsail, each of which is priced according to how much compute power your container needs. We selected nano in this example . A list of power levels and associated costs is available here: https://aws.amazon.com/lightsail/pricing/

Steps

Once you have `lightsailctl` installed, create a new container service and give it a name, power parameter, and scale parameter:

```
aws lightsail create-container-service \
--service-name awscookbook --power nano --scale 1
```

Output:

```
{
    "containerService": {
        "containerServiceName": "awscookbook",
        "arn": "arn:aws:lightsail:us-east-
1:111111111111:ContainerService/124633d7-b625-48b2-b066-
5826012904d5",
        "createdAt": "2020-11-15T10:10:55-05:00",
        "location": {
            "availabilityZone": "all",
            "regionName": "us-east-1"
        },
        "resourceType": "ContainerService",
        "tags": [],
        "power": "nano",
        "powerId": "nano-1",
        "state": "PENDING",
        "scale": 1,
        "isDisabled": false,
        "principalArn": "",
        "privateDomainName": "awscookbook.service.local",
        "url": "https://awscookbook.<<unique-id>>.us-east-
1.cs.amazonlightsail.com/"
    }
}
```

Pull a plain nginx container image to use which listens on port 80/tcp.

```
docker pull nginx
```

Use the following command to ensure that the state of your container service has entered the “READY” state. This may take a few minutes

```
aws lightsail get-container-services --service-name awscookbook
```

When the container service is ready, push the container image to Lightsail

```
aws lightsail push-container-image --service-name awscookbook \
--label awscookbook --image nginx
```

Output:

```
7b5417cae114: Pushed
Image "nginx" registered.
Refer to this image as ":awscookbook.awscookbook.1" in
deployments.
```

Now you will associate the image you pushed with the container service you created for deployment. Create a file with the following contents, and save it as *lightsail.json*:

```
{
  "serviceName": "awscookbook",
  "containers": {
     "awscookbook": {
        "image": ":awscookbook.awscookbook.1",
        "ports": {
           "80": "HTTP"
        }
     }
  },
  "publicEndpoint": {
     "containerName": "awscookbook",
```

```
      "containerPort": 80
    }
}
```

> **TIP**
>
> We have provided this file for you in this recipe's folder in the AWS Cookbook repo available at https://github.com/AWSCookbook/Containers

Create the deployment

```
aws lightsail create-container-service-deployment \
--service-name awscookbook --cli-input-json file://lightsail.json
```

View your container service again, and wait for the "ACTIVE" state. This may take a few minutes.

```
aws lightsail get-container-services --service-name awscookbook
```

Note the endpoint URL at the end of the output

Now, visit the endpoint URL in your browser, or use the `curl` on the command line:

E.g.: " url ": "https://awscookbook.un94eb3cd7hgk.us-east-1.cs.amazonlightsail.com/"

```
curl <<URL endpoint>>
```

Output:

```
...
<h1>Welcome to nginx!</h1>
...
```

Clean Up

Delete the local image from your workstation

```
docker image rm nginx
```

Delete the container service

```
aws lightsail delete-container-service --service-name awscookbook
```

Discussion

You configured a Lightsail Container Service and pushed a local container image to Lightsail. You used a JSON file with the required parameters for the deployment. This file references the tag of the container image that you pushed. After the deployment was completed, you validated the deployment in a browser or on the command line using a secure HTTPS connection.

Lightsail manages the TLS certificate, load balancer, compute, and storage. It can also manage MySQL and PostgreSQL databases as part of your deployment if your application requires it. Lightsail performs routine health checks on your application and will automatically replace a container you deploy that may have become unresponsive for some reason. Changing the power and scale parameters in the `lightsail create-container-service` command will allow you to create services for demanding workloads.

Using this recipe, you could deploy any common containerize application (e.g. Wordpress) and have it served on the internet in a short period of time. You could even point a custom domain alias at your Lightsail deployment for an SEO-friendly URL.

3.4 Deploying containers using AWS Copilot

Problem

You need to deploy a highly configurable Load Balanced Web Service quickly using best practices in a private network.

Solution

Starting with a Dockerfile, you can use AWS Copilot to quickly deploy an application using an architecture like this:

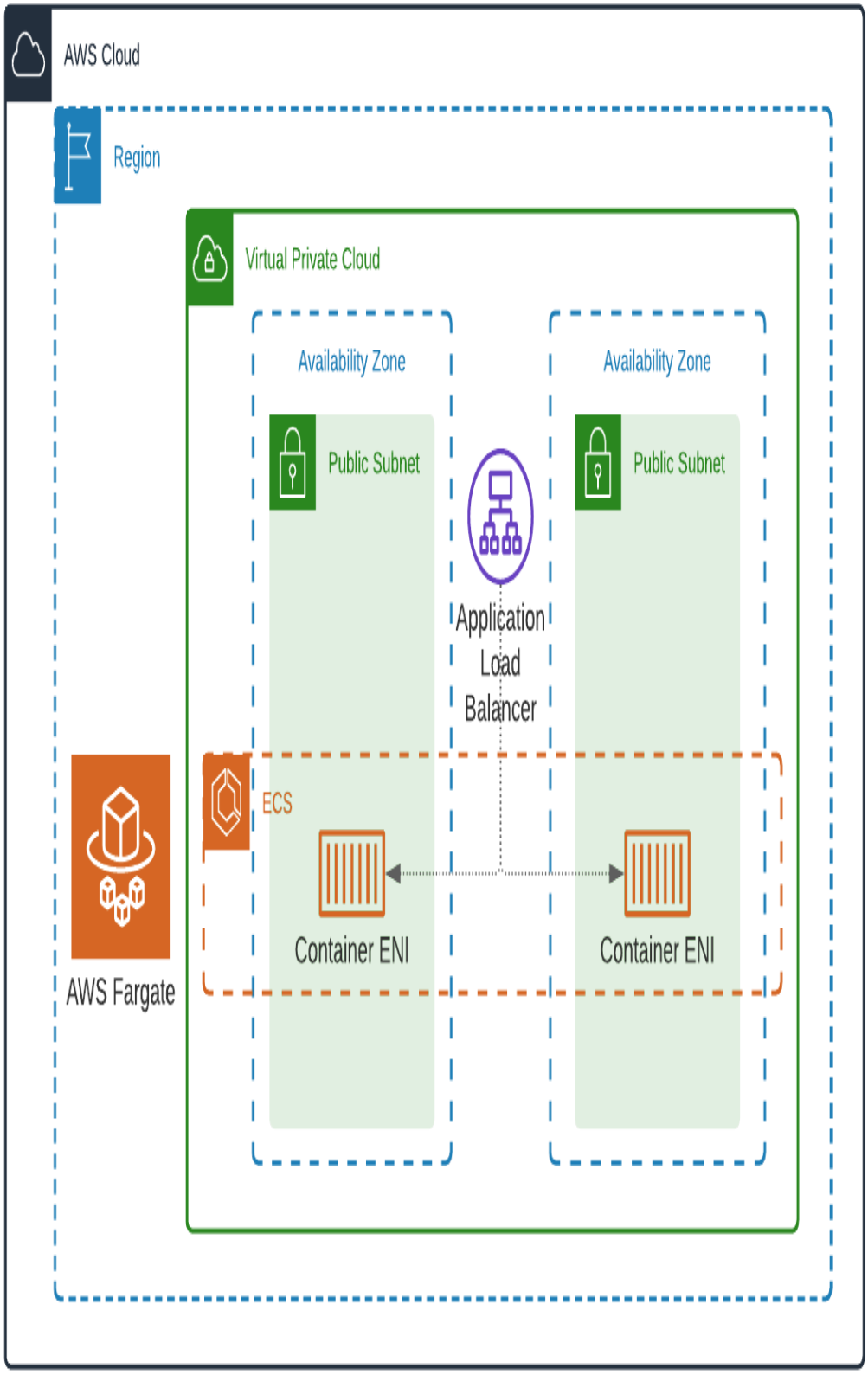

AWS Cloud
Region
Virtual Private Cloud
Availability Zone
Availability Zone
Public Subnet
Public Subnet
Application
Load
Balancer
ECS
Container ENI
Container ENI
AWS Fargate

Figure 3-5. AWS Copilot "Load Balanced Web Service" Infrastructure

Preparation

In addition to the workstation configuration steps in this chapter's introduction, you will also need to install the AWS Copilot CLI to complete this recipe.

> **NOTE**
>
> Refer to the AWS Copilot CLI installation instructions in the ECS Developer Guide for complete and up to date installation instructions.

To install the Copilot CLI using Homebrew, issue the following commands in your terminal:

```
brew install aws/tap/copilot-cli
```

Steps

Copilot requires an ECS service-linked role to allow Amazon ECS to perform actions on your behalf. This may already exist in your AWS account. To see if you have this role already, issue the following command:

```
aws iam list-roles --path-prefix /aws-service-
role/ecs.amazonaws.com/
```

(If the role is displayed, you can skip the following role creation step)

Create the ECS service-linked role if it does not exist:

```
aws iam create-service-linked-role --aws-service-name
ecs.amazonaws.com
```

> **NOTE**
>
> IAM Service linked roles allow AWS services to securely interact with other AWS services on your behalf. More Info: https://docs.aws.amazon.com/IAM/latest/UserGuide/using-service-linked-roles.html

`cd` to this recipe's directory in this Chapter's repository (https://github.com/AWSCookbook/Containers):

```
cd 604-Deploy-Container-With-Copilot-CLI
```

> **NOTE**
>
> You could provide your own Dockerfile and content for this recipe. If you choose to use your own container with this recipe, ensure that the container listens on port 80/tcp, or configure the alternate port with the `copilot init` command.

Now use AWS Copilot to deploy the sample NGINX Dockerfile to Amazon ECS:

```
copilot init --app web --name nginx --type 'Load Balanced Web
Service' \
--dockerfile './Dockerfile' --port 80 --deploy
```

> **NOTE**
>
> If you don't specify any arguments to the `copilot init` command, it will walk you through a menu of options for your deployment

The deployment will take a few moments. You can watch the progress of the deployment in your terminal.

After the deployment is complete, get information on the deployed service with this command:

```
copilot svc show
```

Clean Up

This command ensures that the deployed resources for theservice are removed, it will prompt for confirmation. The `app delete` command will take several minutes to complete.

```
copilot app delete
```

Discussion

You used AWS Copilot to deploy a “Load Balanced Web Service” in a new Amazon Virtual Private Cloud (Amazon VPC). You specified a name for the application (web), a Dockerfile, a type of service (*Load Balanced Web Service)* and a port (80). Once the infrastructure was deployed, Copilot built the container image, pushed it to an ECR repository, and deployed an Amazon ECS service. Finally you were presented with a URL to access the deployed application over the internet.

The `copilot init` command created a folder called “copilot” in your current working directory. You can view and customize the configuration using the manifest.yml that is associated with your application.

> **NOTE**
>
> The “test” environment is the default environment created. You can add additional environments to suit your needs and keep your environments isolated from each other by using the `copilot env init` command.

Copilot configures all of the required resources for hosting containers on Amazon ECS according to many best practices. Some examples are: deploying to multiple Availability Zones (AZs), using subnet tiers to segment traffic, using AWS KMS to encrypt, and more.

The AWS Copilot commands can also be embedded in your CI/CD pipeline to perform automated deployments. In fact, Copilot can orchestrate the creation and management of a CI/CD pipeline for you with the `copilot pipeline` command. For all of the current supported features and examples, visit the AWS Copilot Project Homepage.

3.5 Updating containers with blue/green deployments

Problem

You want to use a deployment strategy with your container based application so that you can update your application to the latest version without introducing downtime to customers while also being able to easily rollback if the deployment was not successful.

Solution

Use AWS CodeDeploy to orchestrate your application deployments to Amazon ECS with the Blue/Green strategy.

Preparation

In the root of this Chapter's repo **cd** to the "605-Updating-Containers-With-BlueGreen/cdk-AWS-Cookbook-605" folder

```
cd 605-Updating-Containers-With-BlueGreen/cdk-AWS-Cookbook-605/
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt --no-dependencies
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

Navigate up to the main directory for this recipe (out of the “cdk-AWS-Cookbook-605” folder)

```
cd ..
```

Steps

After the CDK deployment, visit the LoadBalancerDNS address that the CDK displayed in your browser, you will see the “Blue” application running:

```
E.g.: firefox http://fargateservicealb-925844155.us-east-
1.elb.amazonaws.com/
or
open http://$LoadBalancerDNS
```

Create an IAM role using the statement in the provided assume-role-policy.json file using this command:

```
aws iam create-role --role-name ecsCodeDeployRole \
--assume-role-policy-document file://assume-role-policy.json
```

Attach the IAM managed policy for CodeDeployRoleForECS to the IAM role:

```
aws iam attach-role-policy --role-name ecsCodeDeployRole \
--policy-arn arn:aws:iam::aws:policy/AWSCodeDeployRoleForECS
```

Create a new ALB target group to use as the “Green” target group with CodeDeploy:

```
aws elbv2 create-target-group --name "GreenTG" --port 80 \
--protocol HTTP --vpc-id $VPCId --target-type ip
```

Create the CodeDeploy Application:

```
aws deploy create-application --application-name awscookbook-605
\
--compute-platform ECS
```

CodeDeploy requires some configuration. We provide a template file (codedeploy-template.json) in this recipe’s folder of Chapter 6 repo.

Use the `sed` command to replace the values with the environment variables you exported with the helper.py script:

```
sed -e "s/AWS_ACCOUNT_ID/${AWS_ACCOUNT_ID}/g" \
-e "s|ProdListenerArn|${ProdListenerArn}|g" \
-e "s|TestListenerArn|${TestListenerArn}|g" \
codedeploy-template.json > codedeploy.json
```

> **TIP**
>
> `sed` (short for stream editor) is a great tool to use for text find and replace operations as well as other types of text manipulation in your terminal sessions and scripts. In this case, `sed` is used to replace values in a template file with values output from `cdk deploy` set as environment variables.

Now, create a deployment group:

```
aws deploy create-deployment-group --cli-input-json
file://codedeploy.json
```

The AppSpec-template.yaml contains information about the application you are going to update. The CDK pre-provisioned a task definition you can use.

Use the `sed` command to replace the value with the environment variable you exported with the helper.py script:

```
sed -e "s|FargateTaskGreenArn|${FargateTaskGreenArn}|g" \
appspec-template.yaml > appspec.yaml
```

Now copy the AppSpec file to S3 Bucket created by the CDK deployment so that CodeDeploy can use it to update the application:

```
aws s3 cp ./appspec.yaml s3://$S3BucketName
```

One final configuration file needs to be created, this contains the instructions about the deployment. Use `sed` to modify the S3 Bucket used in the deployment-template.json file.

```
sed -e "s|S3BucketName|${S3BucketName}|g" \
deployment-template.json > deployment.json
```

Now create a deployment with the deployment configuration:

```
aws deploy create-deployment --cli-input-json
file://deployment.json
```

To get the status of the deployment, observe the status in the AWS Console (Developer Tools --> CodeDeploy --> Deployment --> Click on the deployment ID) You should see CodeDeploy in progress with the deployment:

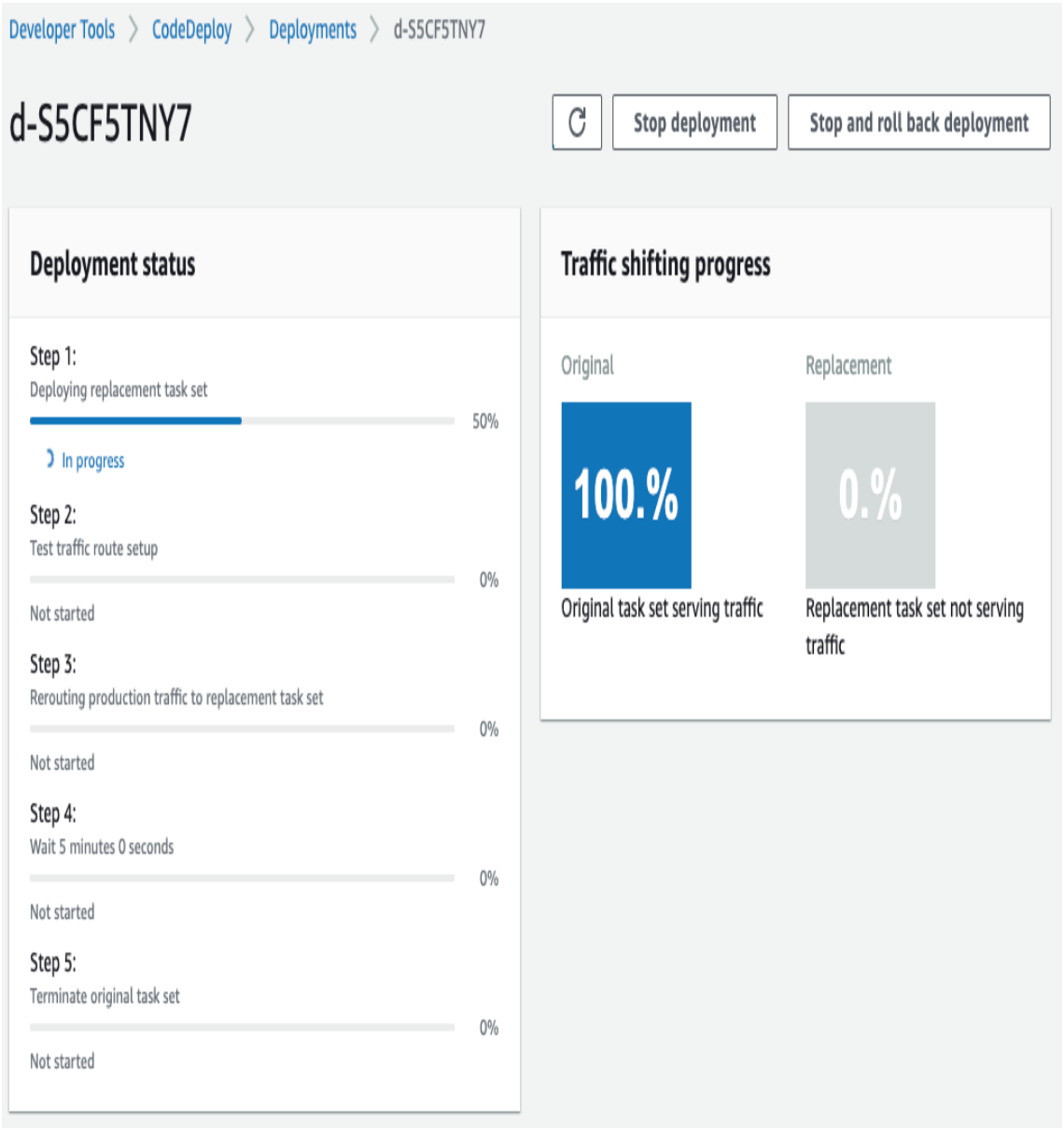


Figure 3-6. Initial Deployment status

Once the replacement task is serving 100% of the traffic, you can visit the same URL where you previously observed the Blue application running, replaced with the Green version of the application.

> **NOTE**
>
> You may need to refresh to see the updated Green application

Clean Up

Delete the CodeDeploy deployment group and application:

```
aws deploy delete-deployment-group \
--deployment-group-name awscookbook-605-dg \
--application-name awscookbook-605
```

```
aws deploy delete-application --application-name awscookbook-605
```

Detach the IAM policy from and delete the role used by CodeDeploy to update your application on Amazon ECS:

```
aws iam detach-role-policy --role-name ecsCodeDeployRole \
--policy-arn arn:aws:iam::aws:policy/AWSCodeDeployRoleForECS
```

```
aws iam delete-role --role-name ecsCodeDeployRole
```

Now remove the load balancer rules created by CodeDeploy during the deployment and the target group you created previously:

```
aws elbv2 delete-rule --rule-arn \
$(aws elbv2 describe-rules \
--listener-arn $ProdListenerArn \
--query 'Rules[?Priority==`"1"`].RuleArn' \
--output text)
```

```
aws elbv2 modify-listener --listener-arn $TestListenerArn \
--default-actions
Type=forward,TargetGroupArn=$DefaultTargetGroupArn
```

```
aws elbv2 delete-target-group --target-group-arn \
$(aws elbv2 describe-target-groups \
--names "GreenTG" \
--query 'TargetGroups[0].TargetGroupArn' \
--output text)
```

Delete the Blue and Green images

```
aws ecr batch-delete-image --repository-name aws-cdk/assets \
--image-ids imageTag=$(echo $BlueImage | cut -d : -f 2) \
imageTag=$(echo $GreenImage | cut -d : -f 2)
```

Go to the cdk-AWS-Cookbook-605 directory

```
cd cdk-AWS-Cookbook-605/
```

To clean up the environment variables, run the `helper.py` script in this recipe's `cdk-` folder with the `--unset` flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && cd ../..
```

Discussion

You created a CodeDeploy deployment group, associated it with an existing ECS service, and then used CodeDeploy to deploy a new version of your application with a Blue/Green strategy. CodeDeploy offers several deployment strategies (Canary, AllAtOnce, Blue/Green, etc) and you can also create your own custom deployment strategies. One reason to customize the strategy would be to define a longer wait period for the cutover window

or define other conditions to be met before traffic switchover occurs. In the default Blue/Green strategy, CodeDeploy keeps your previous version of the application running for 5 minutes while all traffic is routed to the new version. If you notice that the new version is not behaving properly, you can quickly route traffic back to the original version since it is still running in a separate AWS Application Load Balancer (ALB) Target Group.

CodeDeploy uses ALB Target Groups to manage which application is considered “production”. When you deployed the initial stack with the AWS CDK, the “V1-Blue” containers were registered with a target group associated with port 80 on the ALB. After you initiate the deployment of the new version, CodeDeploy starts a brand new version of the ECS service, associates it with the Green Target Group you created, and then gracefully shifts all traffic to the Green Target Group. The final result is the Green-V2 containers now being served on port 80 of the ALB. The previous target group is now ready to execute the next Blue/Green deployment.

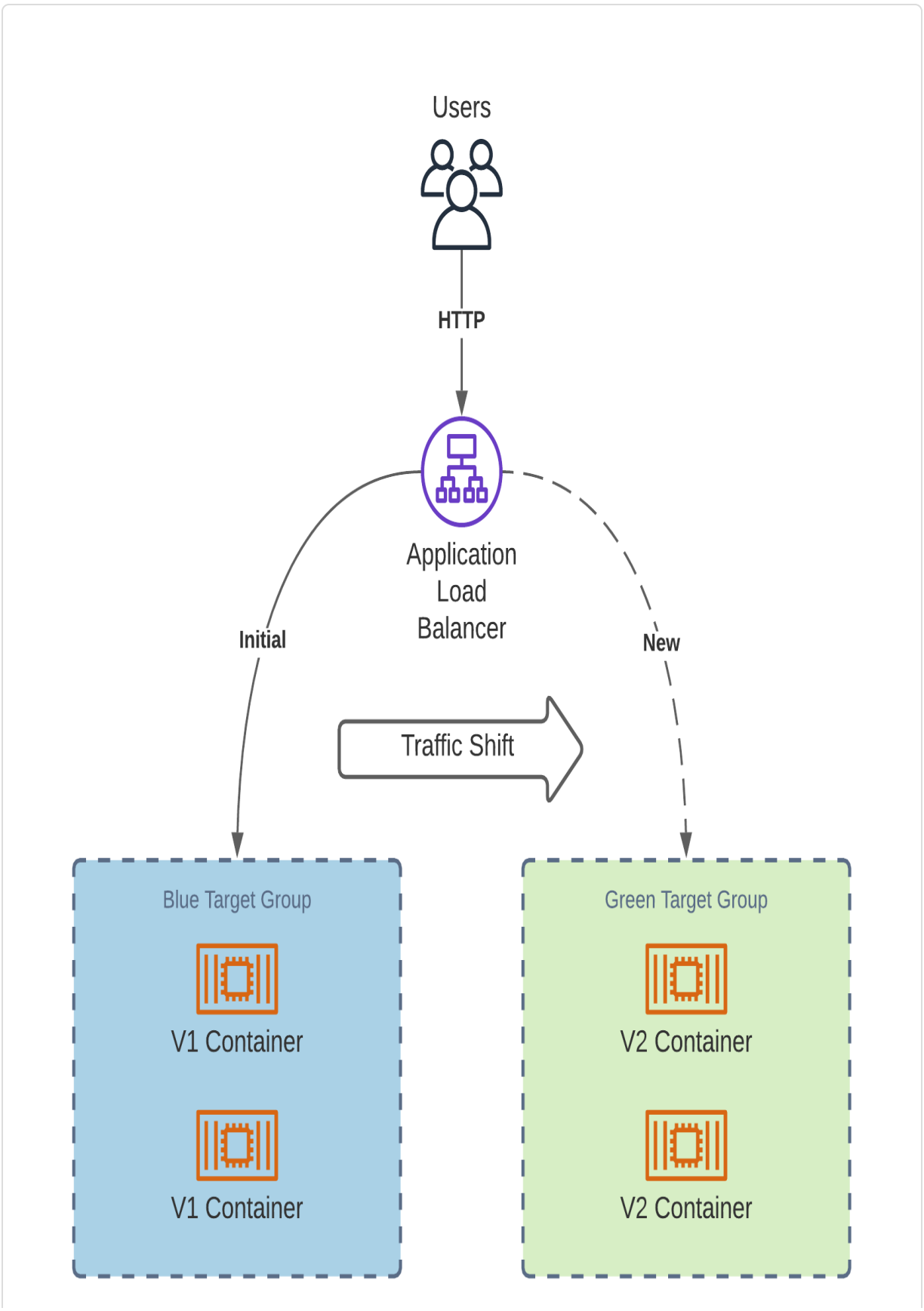

Users
HTTP
Application
Load
Balancer
Initial
New
Traffic Shift
Blue Target Group
V1 Container
V1 Container
Green Target Group
V2 Container
V2 Container

Figure 3-7. Blue/Green Target Group Association

This is a common pattern to utilize with CI/CD. Your previous version can quickly be reactivated with a seamless roll back. If no roll back is needed, the initial version (V1) is terminated and you can repeat the processes the next time you deploy putting V3 in the Blue Target Group, shifting traffic to it when you are ready. Using this strategy helps you minimize the customer impact of new application versions while allowing more frequent deployments.

> **TIP**
>
> Deployment Conditions allow you to define deployment success criteria. You can use a combination of a Custom Deployment Strategy and a Deployment Condition to build automation tests into your CodeDeploy process. This would allow you to ensure all of your tests run and pass before traffic is sent to your new deployment.

3.6 Auto Scaling container workloads on Amazon ECS

Problem

You need to deploy a containerized service which scales-out during times of heavy traffic to meet demand.

Solution

You will deploy CloudWatch Alarms and a scaling policy for an ECS service.

Preparation

This recipe requires some “prep work” which deploys resources that you’ll build the solution on. You will use the AWS CDK to deploy these resources

In the root of this Chapter's repo **cd** to the "606-Autoscaling-Container-Workloads/cdk-AWS-Cookbook-606" folder and follow the subsequent steps:

```
cd 606-Autoscaling-Container-Workloads/cdk-AWS-Cookbook-606/
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheelpip install -r
requirements.txt --no-dependencies
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

Navigate up to the main directory for this recipe (out of the "cdk-AWS-Cookbook-606" folder)

```
cd ..
```

Steps

Access the ECS service URL over the internet with the cURL command (or your web browser) to verify the successful deployment:

```
curl -v -m 3 $LoadBalancerDNS
```

Use verbose **(-v)** and 3 second timeout **(-m 3)** to ensure you see the entire connection and have a timeout set. Example command and output:

```
curl -v -m 10 http://AWSCookbook.us-east-
1.elb.amazonaws.com:8080/
```

```
*   Trying 1.2.3.4...
* TCP_NODELAY set
* Connected to AWSCookbook.us-east-1.elb.amazonaws.com (1.2.3.4)
port 8080
> GET / HTTP/1.1
> Host: AWSCookbook.us-east-1.elb.amazonaws.com:8080
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 200
< Content-Type: application/json
< Content-Length: 318
< Connection: keep-alive
<
{
  "URL":"http://awscookbookloadtestloadbalancer-36821611.us-east-
1.elb.amazonaws.com:8080/",
   "ContainerLocalAddress":"10.192.2.179:8080",
   "ProcessingTimeTotalMilliseconds":"0",
   "LoadBalancerPrivateIP":"10.192.2.241",
   "ContainerHostname":"ip-10-192-2-179.ec2.internal",
   "CurrentTime":"1605724705176"
}
Closing connection 0
```

> **TIP**
>
> Run this same curl command several times in a row, and you will notice the ContainerHostname and ContainerLocalAddress alternating between two addresses. This indicates that Amazon ECS is load balancing between the two containers you should expect to be running at all times as defined by the ECS service.

You will need to create a role for the Auto Scaling trigger to execute, this file is located in this solution's directory in the chapter repository:

```
aws iam create-role --role-name AWSCookbook606ECS \
--assume-role-policy-document file://task-execution-assume-
role.json
```

Attach the managed policy for Auto Scaling:

```
aws iam attach-role-policy --role-name AWSCookbook606ECS --
policy-arn arn:aws:iam::aws:policy/service-
```

```
role/AmazonEC2ContainerServiceAutoscaleRole
```

Register an Auto Scaling Target:

```
aws application-autoscaling register-scalable-target \
--service-namespace ecs \
--scalable-dimension ecs:service:DesiredCount \
--resource-id service/$ECSClusterName/AWSCookbook606 \
--min-capacity 2 \
--max-capacity 4
```

Set up an Auto Scaling policy for the Auto Scaling Target using the sample configuration file specifying a 50% average CPU target:

```
aws application-autoscaling put-scaling-policy --service-
namespace ecs \
--scalable-dimension ecs:service:DesiredCount \
--resource-id service/$ECSClusterName/AWSCookbook606 \
--policy-name cpu50-awscookbook-606 --policy-type
TargetTrackingScaling \
--target-tracking-scaling-policy-configuration file://scaling-
policy.json
```

Now, to trigger a process within the container which simulates high CPU load, run the same cURL command appending `cpu` to the end of the ServiceURL:

```
curl -v -m 3 $LoadBalancerDNS/cpu
```

This command will time out after 3 seconds, indicating that the container is running a CPU intensive process as a result of visiting that URL. Example command and output:

```
curl -v -m 10 http://AWSCookbookLoadtestLoadBalancer-36821611.us-
east-1.elb.amazonaws.com:8080/cpu
*   Trying 52.4.148.24...
* TCP_NODELAY set
```

```
* Connected to AWSCookbookLoadtestLoadBalancer-36821611.us-east-
1.elb.amazonaws.com (52.4.148.245) port 8080 (#0)
> GET /cpu HTTP/1.1
> Host: AWSCookbookLoadtestLoadBalancer-36821611.us-east-
1.elb.amazonaws.com:8080
> User-Agent: curl/7.64.1
> Accept: */*
>
* Operation timed out after 10002 milliseconds with 0 bytes
received
* Closing connection 0
curl: (28) Operation timed out after 10002 milliseconds with 0
bytes received
```

Wait approximately 5 minutes, then log into the AWS Console, locate Elastic Container Service, go to the Clusters page, select the cluster deployed and then select the ECS service. Verify that the Desired Count has increased to 4, the maximum scaling value that you configured. You can click the tasks tab to view 4 container tasks now running for your service.

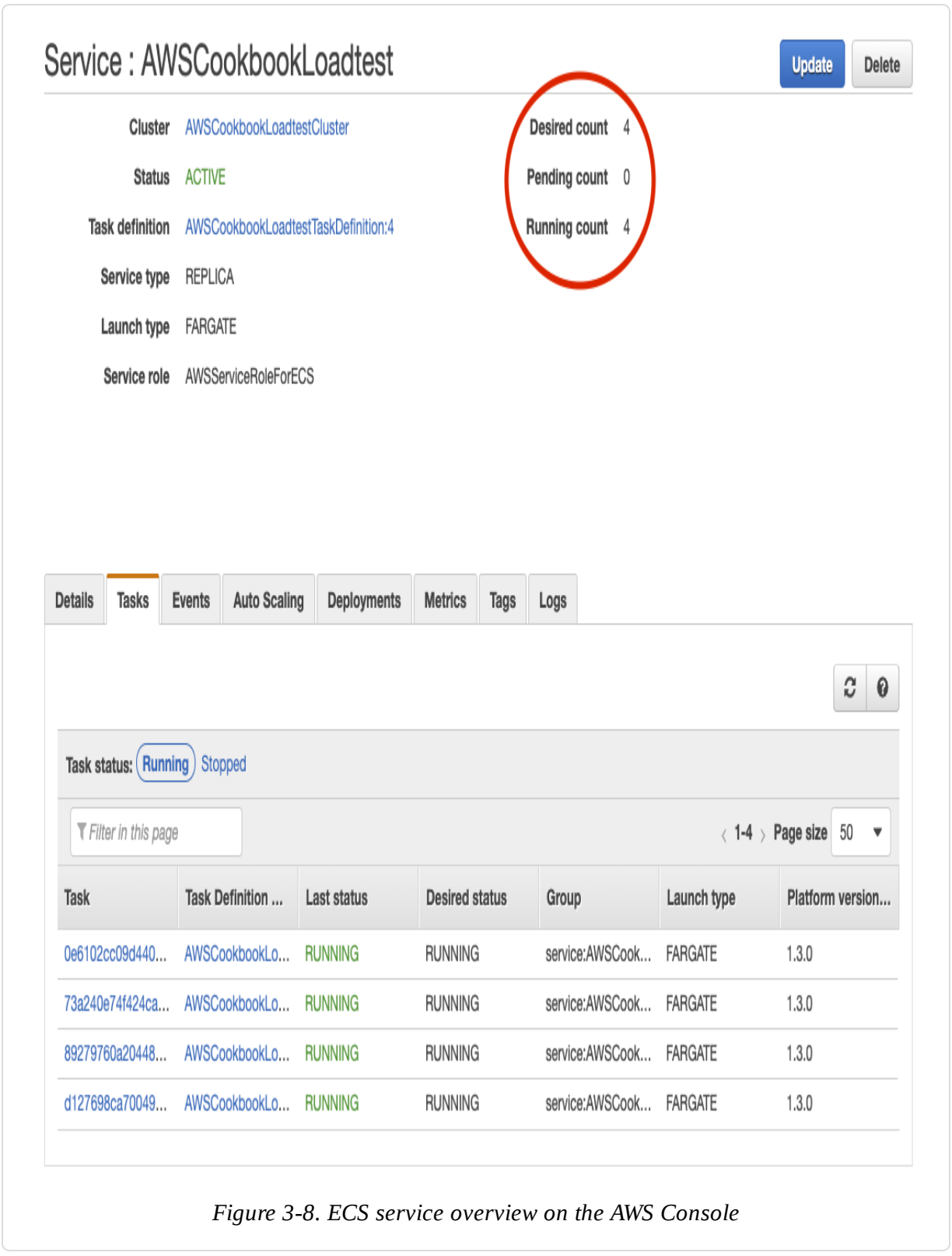


Figure 3-8. ECS service overview on the AWS Console

Click on the Metrics Tab to view the CPU Usage for the service. You set the scaling target at 50% to trigger the Autoscaling actions adding 2 additional containers to the service as a result of high CPU usage.

Figure 3-9. ECS service metrics on the AWS Console

Clean Up

Delete the container images

```
aws ecr batch-delete-image --repository-name aws-cdk/assets \
--image-ids imageTag=$(echo $ECRImage | cut -d : -f 2)
```

Go to the cdk-AWS-Cookbook-606 directory

```
cd cdk-AWS-Cookbook-606/
```

To clean up the environment variables, run the `helper.py` script in this recipe's `cdk-` folder with the `--unset` flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && cd ../..
```

Detach the managed Auto Scaling policy from the IAM role:

```
aws iam detach-role-policy --role-name AWSCookbook606ECS --
policy-arn \
arn:aws:iam::aws:policy/service-
role/AmazonEC2ContainerServiceAutoscaleRole
```

Delete the Auto Scaling IAM role:

```
aws iam delete-role --role-name AWSCookbook606ECS
```

Discussion

You deployed a CPU load simulation container to Amazon ECS using the AWS CDK, configured CloudWatch Alarms to monitor the CPU utilization metrics (within CloudWatch Metrics) of the running containers, set up an Auto Scaling trigger, and observed the behavior. You also created IAM roles which allowed CloudWatch to trigger Auto Scaling. Initially, there were two containers in the service. After you triggered the load simulation, the container count increased to 4 (which you specified as the maximum number for Auto Scaling of the service).

Auto Scaling is an important mechanism to implement to save costs associated with running your applications on AWS services. It allows your applications to provision their own resources as needed during times where load may increase and remove their own resources during times where the application may be idle. Note that in all cases where you have an AWS service doing something like this on your behalf, you have to specifically grant permission for services to execute these functions via IAM.

The underlying data that provides the metrics for such operations is contained in the CloudWatch Metrics service. There are many data points and metrics that you can use for configuring Auto Scaling, some of the most common ones are:

- Network I/O
- CPU Usage
- Memory Used
- Number of Transactions

In this recipe, you monitor the CPU Usage metric on the ECS service. You set the metric at 50% and trigger the CPU load with a cURL call to the HTTP endpoint of the ECS service. Scaling metrics are dependent upon the type of applications you are running and what technologies you use to build them. As a best-practice, you should observe your application metrics over a period of time to set a baseline before choosing metrics to implement Auto Scaling.

3.7 Launching a Fargate container task in response to an event

Problem

You need to launch a container task to process incoming files.

Solution

You will use Amazon EventBridge to trigger the launch of ECS container tasks on Fargate after a file is uploaded to S3.

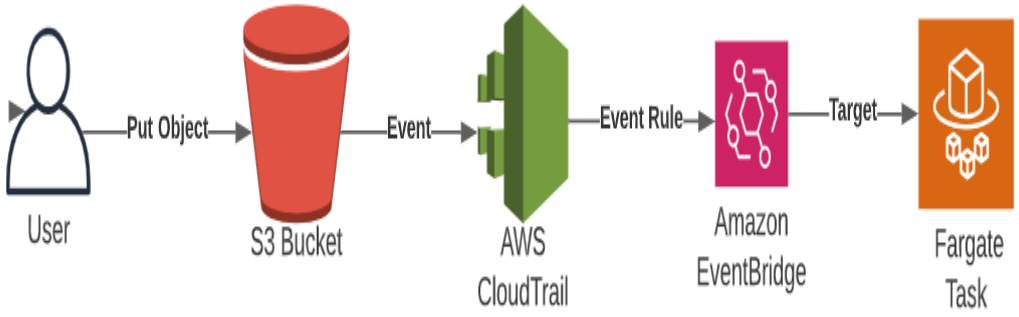


Figure 3-10. Flow of container EventBridge Pattern

Preparation

This recipe requires some "prep work" which deploys resources that you'll build the solution on. You will use the AWS CDK to deploy these resources

In the root of this Chapter's repo **cd** to the "607-Fargate-Task-With-Event/cdk-AWS-Cookbook-607" folder and follow the subsequent steps:

```
cd 607-Fargate-Task-With-Event/cdk-AWS-Cookbook-607/
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt --no-dependencies
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

Navigate up to the main directory for this recipe (out of the “cdk-AWS-Cookbook-607” folder)

```
cd ..
```

Steps

Configure CloudTrail to log events on the S3 bucket:

```
aws cloudtrail put-event-selectors --trail-name $CloudTrailArn  -
-event-selectors "[{ \"ReadWriteType\": \"WriteOnly\",
\"IncludeManagementEvents\":false, \"DataResources\": [{
\"Type\":  \"AWS::S3::Object\", \"Values\":
[\"arn:aws:s3:::$S3BucketName/input/\"] }],
\"ExcludeManagementEventSources\": [] }]"
```

Now create an assume-role policy JSON statement called policy1.json to use in the next step:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "events.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Create the role and specify the assume-role-policy.json file:

```
aws iam create-role --role-name AWSCookbook607RuleRole \
--assume-role-policy-document file://policy1.json
```

You will also need a policy document with the following content called policy2.json.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RunTask"
            ],
            "Resource": [
                "arn:aws:ecs:*:*:task-definition/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": "ecs-
tasks.amazonaws.com"
                }
            }
        }
    ]
}
```

Now attach the IAM policy json you just created to the IAM Role:

```
aws iam put-role-policy --role-name AWSCookbook607RuleRole \
--policy-name ECSRunTaskPermissionsForEvents \
--policy-document file://policy2.json
```

Create an EventBridge Rule which monitors the S3 bucket for file uploads:

```
aws events put-rule --name "AWSCookbookRule" --role-arn
"arn:aws:iam::$AWS_ACCOUNT_ID:role/AWSCookbook607RuleRole" --
event-pattern "{\"source\":[\"aws.s3\"],\"detail-type\":[\"AWS
API Call via CloudTrail\"],\"detail\":{\"eventSource\":
[\"s3.amazonaws.com\"],\"eventName\":
[\"CopyObject\",\"PutObject\",\"CompleteMultipartUpload\"],\"requ
estParameters\":{\"bucketName\":[\"$S3BucketName\"]}}}"
```

Modify the value in targets-template.json and create a targets.json for use:

```
sed -e "s|AWS_ACCOUNT_ID|${AWS_ACCOUNT_ID}|g" \
-e "s|AWS_REGION|${AWS_REGION}|g" \
-e "s|ECSClusterARN|${ECSClusterARN}|g" \
-e "s|TaskDefinitionARN|${TaskDefinitionARN}|g" \
-e "s|VPCPrivateSubnets|${VPCPrivateSubnets}|g" \
-e "s|VPCDefaultSecurityGroup|${VPCDefaultSecurityGroup}|g" \
targets-template.json > targets.json
```

Create a rule target which specifies the ECS cluster, ECS task definition, IAM Role, and networking parameters. This specifies what the rule will trigger, in this case launch a container on Fargate:

```
aws events put-targets --rule AWSCookbookRule --targets
file://targets.json
```

Output

```
{
    "FailedEntryCount": 0,
    "FailedEntries": []}
```

Check the S3 bucket to verify that its empty before we populate it:

```
aws s3 ls s3://$S3BucketName/
```

Copy the provided maze.jpg file to the S3 bucket. This will trigger the ECS task which launches a container with a Python library to process the file:

```
aws s3 cp maze.jpg s3://$S3BucketName/input/maze.jpg
```

This will trigger an ECS task to process the image file. Quickly, check the task with the `ecs list-tasks` command. The task will run for about 2-3 minutes.

```
aws ecs list-tasks --cluster $ECSClusterARN
```

Output:

```
{
    "taskArns": [        "arn:aws:ecs:us-east-
1:111111111111:task/cdk-aws-cookbook-607-
AWSCookbookEcsCluster46494E6E-
MX7kvtp1sYWZ/d86f16af55da56b5ca4874d6029"
    ]
}
```

After a few minutes, observe the output folder created in the S3 bucket:

```
aws s3 ls s3://$S3BucketName/output/
```

Download and view the output file:

```
aws s3 cp s3://$S3BucketName/output/output.jpg ./output.jpg
```

Open output.jpg with a file viewer of your choice to view file that was processed

Clean Up

Remove the EventBridge targets from the EventBridge rule:

```
aws events remove-targets --rule AWSCookbookRule --ids
AWSCookbookRuleID
```

Delete the EventBridge rule:

```
aws events delete-rule --name "AWSCookbookRule"
```

Detach the policies and delete the EventBridge Rule IAM role:

```
aws iam delete-role-policy --role-name AWSCookbook607RuleRole \
--policy-name ECSRunTaskPermissionsForEvents aws iam delete-role
--role-name AWSCookbook607RuleRole
```

Stop the Cloudtrail

```
aws cloudtrail stop-logging --name $CloudTrailArn
```

Go to the cdk-AWS-Cookbook-607 directory

```
cd cdk-AWS-Cookbook-607/
```

To clean up the environment variables, run the `helper.py` script in this recipe's `cdk-` folder with the `--unset` flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy
&&
deactivate && cd ../..
```

Discussion

You used a combination of ECS Fargate, S3, and EventBridge to create a serverless event-driven solution that processes files uploaded to S3 with ECS Fargate. You began by creating an IAM role for EventBridge to assume when launching ECS tasks. Next, you created an event selector to monitor an S3 Bucket for PutObject API requests. You then created a target for the event rule which specified the ECS task definition to run when the rule was run. When you ran the aws s3 cp command to copy the image file to the S3 bucket, this created a PutObject API call to be published to the default event bus,

which matched the rule you created. This ran a container on ECS which downloaded the file from S3, processed it (solved the maze) and uploaded the result back to S3.

Event-driven architecture is an important approach to application and process design in the cloud. This type of design allows for removing long-running application workloads in favor of serverless architectures which can be more resilient and easily scale to peaks of higher usage when needed. When there are no events to handle in your application, you generally do not pay much for compute resources (if at all) so potential cost savings is also a point to consider when choosing an application architecture.

> **NOTE**
>
> It is common to use Lambda functions with S3 for event-driven architectures, but for longer running data processing jobs and computational jobs like this one, Fargate is a better choice because the runtime is essentially infinite, while the maximum runtime for Lambda functions is limited.

Amazon ECS can run tasks and services. Services are made up of tasks, and generally, are long-running in that a service keeps a specific set of tasks running. Tasks can be short lived; a container may start, process some data, and then gracefully terminate after the task is complete. This is what you have achieved in this solution: a task was launched in response to an S3 event signalling a new object, the container read the object, processed the file, and shut down.

3.8 Capturing logs from containers running on Amazon ECS

Problem

You have an application running in a container and you want to inspect the application logs.

Solution

Send the logs from the container to Amazon Cloudwatch. By specifying the "awslogs" driver within an ECS task definition and providing an IAM role which allows the container to write to CloudWatch Logs, you are able to stream container logs to a location within Amazon CloudWatch.

Preparation

This recipe requires some "prep work" which deploys resources that you'll build the solution on. You will use the AWS CDK to deploy these resources

In the root of this Chapter's repo **cd** to the "608-Capturing-Logs-From-Containers-Running-On-ECS/cdk-AWS-Cookbook-608" folder and follow the subsequent steps:

```
cd 608-Capturing-Logs-From-Containers-Running-On-ECS/cdk-AWS-
Cookbook-608
test -d .venv || python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip setuptools wheelpip install -r
requirements.txt
cdk deploy
```

Wait for the `cdk deploy` command to complete.

We created a helper.py script to let you easily create and export environment variables to make subsequent commands easier. Run the script, and copy the output to your terminal to export variables:

```
python helper.py
```

Navigate up to the main directory for this recipe (out of the "cdk-AWS-Cookbook-608" folder)

```
cd ..
```

Steps

This solution, like the others using Amazon ECS, requires an ECS service-linked role to allow ECS to perform actions on your behalf. This may already exist in your AWS account. To see if you have this role already, issue the following command:

```
aws iam list-roles --path-prefix /aws-service-
role/ecs.amazonaws.com/
```

If the role is displayed, you can skip the creation step.

Create the ECS service-linked role if it does not exist (it is OK if the command fails indicating that the role already exists in your account):

```
aws iam create-service-linked-role --aws-service-name
ecs.amazonaws.com
```

Create a file called *task-execution-assume-role.json* with the following content. The file is provided in the root of this recipe's folder in the AWS Cookbook repo.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Create an IAM role using the statement in the file above.

```
aws iam create-role --role-name AWSCookbook608ECS \
```

```
--assume-role-policy-document file://task-execution-assume-
role.json
```

Attach the AWS managed IAM policy for ECS task execution to the IAM role that you just created:

```
aws iam attach-role-policy --role-name AWSCookbook608ECS --
policy-arn arn:aws:iam::aws:policy/service-
role/AmazonECSTaskExecutionRolePolicy
```

Create a Log Group in CloudWatch:

```
aws logs create-log-group --log-group-name AWSCookbook608ECS
```

Create a file called *taskdef.json* with the following content - FYI the file is provided in this recipe's folder in the AWS Cookbook repo.

```
{
    "networkMode": "awsvpc",
    "containerDefinitions": [
        {
            "portMappings": [
                {
                    "hostPort": 80,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "entryPoint": [
                "sh",
                "-c"
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "AWSCookbook608ECS",
                    "awslogs-region": "us-east-1",
                    "awslogs-stream-prefix": "LogStream"
                }
            },
            "name": "awscookbook608",
```

```
            "image": "httpd:2.4",
            "command": [
                "/bin/sh -c \"echo 'Hello AWS Cookbook Reader,
this container is running on ECS!'  >
/usr/local/apache2/htdocs/index.html && httpd-foreground\""
            ]
        }
    ],
    "family": "awscookbook608",
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "cpu": "256",
    "memory": "512"
 }
```

Now that you have an IAM role and an ECS task definition config, you need to create the ECS task using the config and associate the IAM role.

```
aws ecs register-task-definition --execution-role-arn \
"arn:aws:iam::$AWS_ACCOUNT_ID:role/AWSCookbook608ECS" \
--cli-input-json file://taskdef.json
```

Run the ECS task on the ECS cluster that you created earlier in this recipe with the AWS CDK:

```
aws ecs run-task --cluster $ECSClusterName \
--launch-type FARGATE --network-configuration
"awsvpcConfiguration={subnets=[$VPCPublicSubnets],securityGroups=
[$VPCDefaultSecurityGroup],assignPublicIp=ENABLED}" --task-
definition awscookbook608
```

Check the status of the task to make sure the task is running. First, find the Task's Amazon Resource Name (ARN):

```
aws ecs list-tasks --cluster $ECSClusterName
```

Output:

```
{
    "taskArns": [        "arn:aws:ecs:us-east-
1:1234567890:task/cdk-aws-cookbook-608-
AWSCookbookEcsCluster46494E6E-
MX7kvtp1sYWZ/d86f16af55da56b5ca4874d6029"
    ]
}
```

Then use the task ARN to check for the “RUNNING” state with the describe-tasks command output:

```
aws ecs describe-tasks --cluster $ECSClusterName --tasks
<<TaskARN>>
```

After the task has reached the “RUNNING” state (approximately 15 seconds), use the following commands to view logs.

```
aws logs describe-log-streams --log-group-name AWSCookbook608ECS
```

Output:

```
{
    "logStreams": [
        {
            "logStreamName":
"LogStream/webserver/97635dab942e48d1bab11dbe88c8e5c3",
            "creationTime": 1605584764184,
            "firstEventTimestamp": 1605584765067,
            "lastEventTimestamp": 1605584765067,
            "lastIngestionTime": 1605584894363,
            "uploadSequenceToken":
"49612420096740389364147985468451499506623702081936625922",
            "arn": "arn:aws:logs:us-east-1:123456789012:log-
group:AWSCookbook608ECS:log-
stream:LogStream/webserver/97635dab942e48d1bab11dbe88c8e5c3",
            "storedBytes": 0
        }
    ]
}
```

Note the logStreamName from the output and then run the get-log-events command

```
aws logs get-log-events --log-group-name AWSCookbook608ECS \
--log-stream-name <<logStreamName>>
```

Example Output:

```
{
    "events": [
        {
            "timestamp": 1605590555566,
            "message": "[Tue Nov 17 05:22:35.566054 2020]
[mpm_event:notice] [pid 7:tid 140297116308608] AH00489:
Apache/2.4.46 (Unix) configured -- resuming normal operations",
            "ingestionTime": 1605590559713
        },
        {
            "timestamp": 1605590555566,
            "message": "[Tue Nov 17 05:22:35.566213 2020]
[core:notice] [pid 7:tid 140297116308608] AH00094: Command line:
'httpd -D FOREGROUND'",
            "ingestionTime": 1605590559713
        }
    ],
    "nextForwardToken":
"f/35805865872844590178623550035180924397960264595350487 05",
    "nextBackwardToken":
"b/35805865872844590178623550035180924397960264595350487 04"
}
```

Clean Up

Stop the ECS task:

```
aws ecs stop-task --cluster $ECSClusterName --task <<TaskARN>>
```

Delete the IAM Policy Attachment and Role:

```
aws iam detach-role-policy --role-name AWSCookbook608ECS --
policy-arn \
```

```
arn:aws:iam::aws:policy/service-
role/AmazonECSTaskExecutionRolePolicy aws iam delete-role --role-
name AWSCookbook608ECS
```

Delete Log Group:

```
aws logs delete-log-group --log-group-name AWSCookbook608ECS
```

Deregister the Task Definition

```
aws ecs deregister-task-definition --task-definition
awscookbook608:1
```

Go to the cdk-AWS-Cookbook-608 directory

```
cd cdk-AWS-Cookbook-608/
```

To clean up the environment variables, run the `helper.py` script in this recipe's `cdk-` folder with the `--unset` flag, and copy the output to your terminal to export variables:

```
python helper.py --unset
```

Use the AWS CDK to destroy the resources, deactivate your Python virtual environment, and go to the root of the chapter:

```
cdk destroy && deactivate && cd ../..
```

Discussion

In this recipe you created a CloudWatch Logs group, registered a task definition for a simple web server with logging parameters defined, ran the task on Amazon ECS, and observed the web server output in CloudWatch Logs. You made use of the awslogs driver and an IAM role which allows the

running task to write to a CloudWatch Log Group. This is a common pattern when working with containers on AWS as you most likely need log output for troubleshooting and debugging your application. This configuration is handled by tools like Copilot since it is a common pattern, but when working with Amazon ECS directly like defining and running a task, the configuration is critical for developers to know about.

> **TIP**
>
> Containers send the PID 1 process stdout and stderr output - meaning the first process in the container is the only process logging to these streams. This is what is captured by the awslogs driver on Amazon ECS and many popular container engines. Be sure that your application that you would like to see logs from is running with PID 1.

In order for most AWS services to communicate with each other, you must assign a role to them which allows the required level of permissions for the communication. This holds true when configuring logging to CloudWatch from a container ECS task, the container must have a role associated with it which allows the CloudWatchLogs operations via the awslogs logConfiguration driver:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        }
    ]
}
```

CloudWatch Logs allow for a central logging solution for many AWS services. When running multiple containers, it is important to be able to quickly locate logs for debugging purposes.

---

[1] https://docs.aws.amazon.com/AmazonECR/latest/userguide/image-scanning.html

Appendix A. Fast Fixes

> **A NOTE FOR EARLY RELEASE READERS**
>
> With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.
>
> This will be the Appendix of the final book. Note that this early release content will be updated after re:Invent 2020. If you have feedback or content suggestions for the authors, please email *awscookbook@gmail.com*.

These useful 1-2 liners will help you use the AWS Cookbook.

Set your AWS ACCOUNT ID to a bash variable

```
export AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account
--output text)
```

Set your default region

```
export AWS_DEFAULT_REGION=us-east-1
```

Get the mostly recently created CloudWatch Log Group Name

```
aws logs describe-log-groups --output=yaml --query
'reverse(sort_by(logGroups,&creationTime))[:1].{Name:logGroupName}'
```

Tail the logs for the CloudWatch Group

```
aws logs tail <<LOGGROUPNAME>> --follow --since 10s
```

Delete all instances for your current working region (H/T: Curtis Rissi)

```
aws ec2 terminate-instances --instance-ids $(aws ec2 describe-
instances --filters  "Name=instance-state-
name,Values=pending,running,stopped,stopping" --query
"Reservations[].Instances[].[InstanceId]" --output text | tr '\n' '
')
```

Determine the user making cli calls

```
aws sts get-caller-identity
```

Generate Yaml input for your CLI command and use it

```
aws ec2 create-vpc --generate-cli-skeleton yaml-input > input.yaml
#Edit input.yaml - at a minimum modify CidrBlock, DryRun,
ResourceType, and Tags
aws ec2 create-vpc --cli-input-yaml file://input.yaml
```

List the AWS Regions Name and Endpoints in a table format

```
aws ec2 describe-regions --output table
```

Find Interface Vpc Endpoints for the region you are currently using

```
aws ec2 describe-vpc-endpoint-services | jq '.ServiceNames'
```

Simple put into a DynamoDB Table

```
aws ddb put table_name '[{key1: value1}, {key2: value2}]'
```

About the Authors

John Culkin has been a lifelong student of Technology. He acquired a BA from the University of Notre Dame and a MBA from the University of Scranton, both focusing on Management Information Systems. Embracing the cloud revolution has been a focus of his career. During his time as a Principal Cloud Architect Lead at Cloudreach, he led the delivery of cloud solutions aligned to business needs. During these migration and optimization engagements across many industries, he coached developers on how and what to engineer. Currently a Solutions Architect at AWS, he now focuses on creating business transformative solutions that utilize cloud services.

Mike Zazon's career and experience includes roles of software engineer, software architect, operations, physical IT infrastructure architecture in data centres and most recently cloud infrastructure architecture. Currently a Cloud Architect at AWS, he focuses on helping enterprise customers modernize their businesses using AWS. Mike enjoys working on all levels of the stack from foundational infrastructure to application refactoring.