O'REILLY®

AWS Cookbook

Building Practical Solutions with AWS

Early Release

RAW & UNEDITED

John Culkin, Mike Zazon
& James Ferguson

AWS Cookbook

Building Practical Solutions with AWS

John Culkin, Mike Zazon, and James Ferguson

AWS Cookbook

by John Culkin, Mike Zazon, and James Ferguson Copyright © 2021 Culkins Coffee Shop LLC, Mike Zazon, and JumpBox Central LLC. 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

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

978-1-492-09260-5

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. 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*.

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 leverаring 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. 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 4th chapter 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*.

1.0 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). Many 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 your default region in your terminal

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

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

```
aws ec2 describe-instances
```

Set your AWS ACCOUNT ID

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

Checkout this Chapter’s repo

```
git clone https://github.com/AWSCookbook/Chapter4
```

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
```

1.1 Building, Tagging and, Pushing a Container Image to Amazon ECR

Problem

You need a container repository to store and container images that you build and tag.

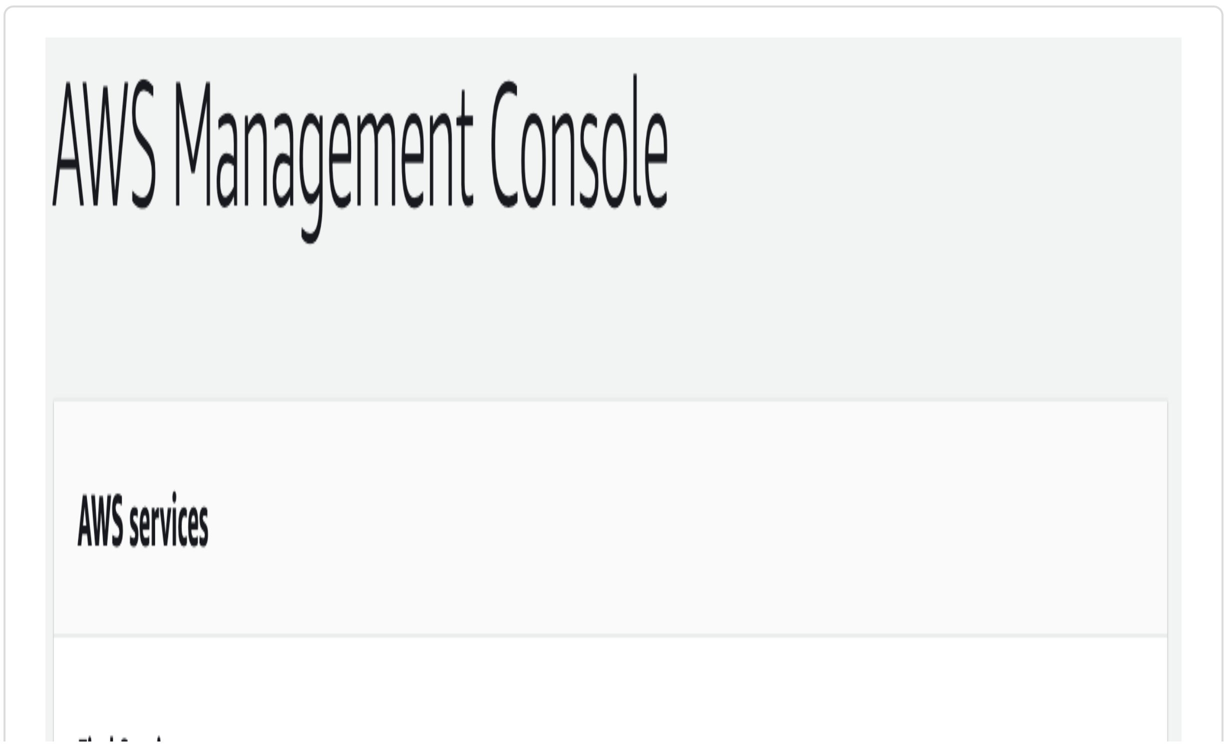
Solution

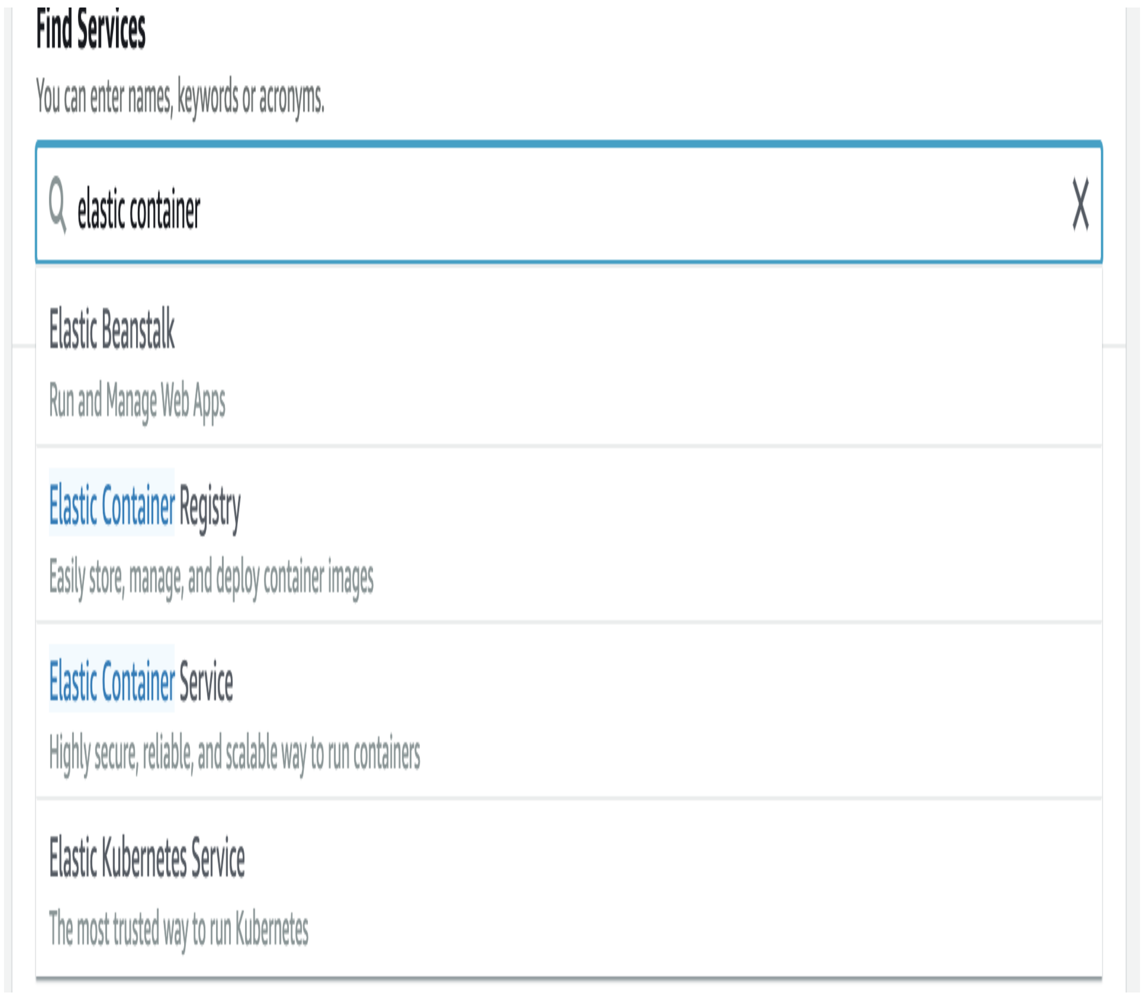
First, you will create a repository in Amazon ECR. Next, you will create a Dockerfile and build a Docker image using it. Finally you will apply two tags to the container image and push them both to the newly created ECR repository.

Steps

Create a private repository in the AWS Management Console:

Log in to the console and search for “elastic container registry”


AWS Management Console
AWS services



Find Services

You can enter names, keywords or acronyms.

elastic container

Elastic Beanstalk

Run and Manage Web Apps

Elastic Container Registry

Easily store, manage, and deploy container images

Elastic Container Service

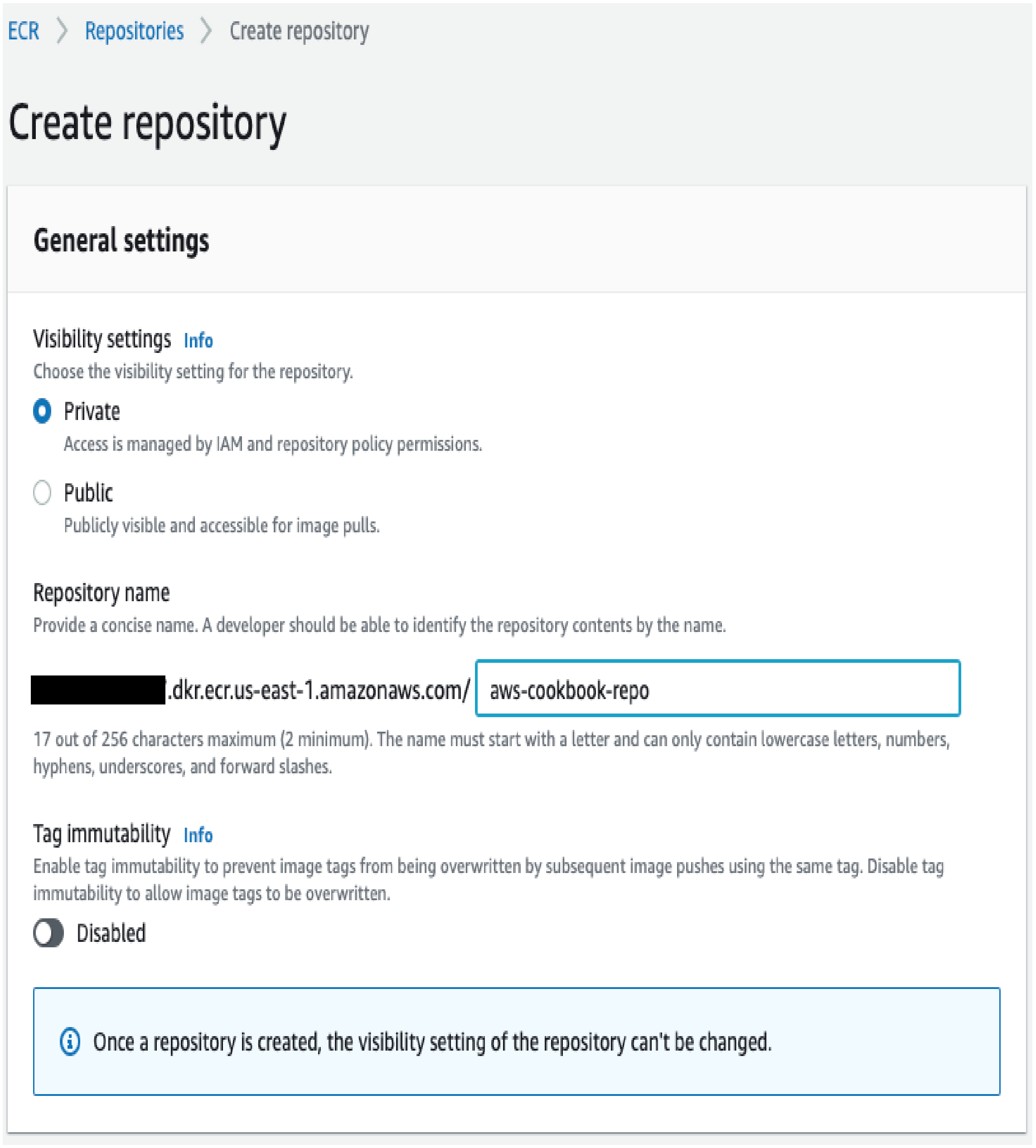
Highly secure, reliable, and scalable way to run containers

Elastic Kubernetes Service

The most trusted way to run Kubernetes

Figure 1-1. 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



Create repository

General settings

Visibility settings Info

Choose the visibility setting for the repository.

[✓ Private] Access is managed by IAM and repository policy permissions.

Public Publicly visible and accessible for image pulls.

Repository name

Provide a concise name. A developer should be able to identify the repository contents by the name.

[redacted].dkr.ecr.us-east-1.amazonaws.com/ aws-cookbook-repo

17 out of 256 characters maximum (2 minimum). The name must start with a letter and can only contain lowercase letters, numbers, hyphens, underscores, and forward slashes.

Tag immutability Info

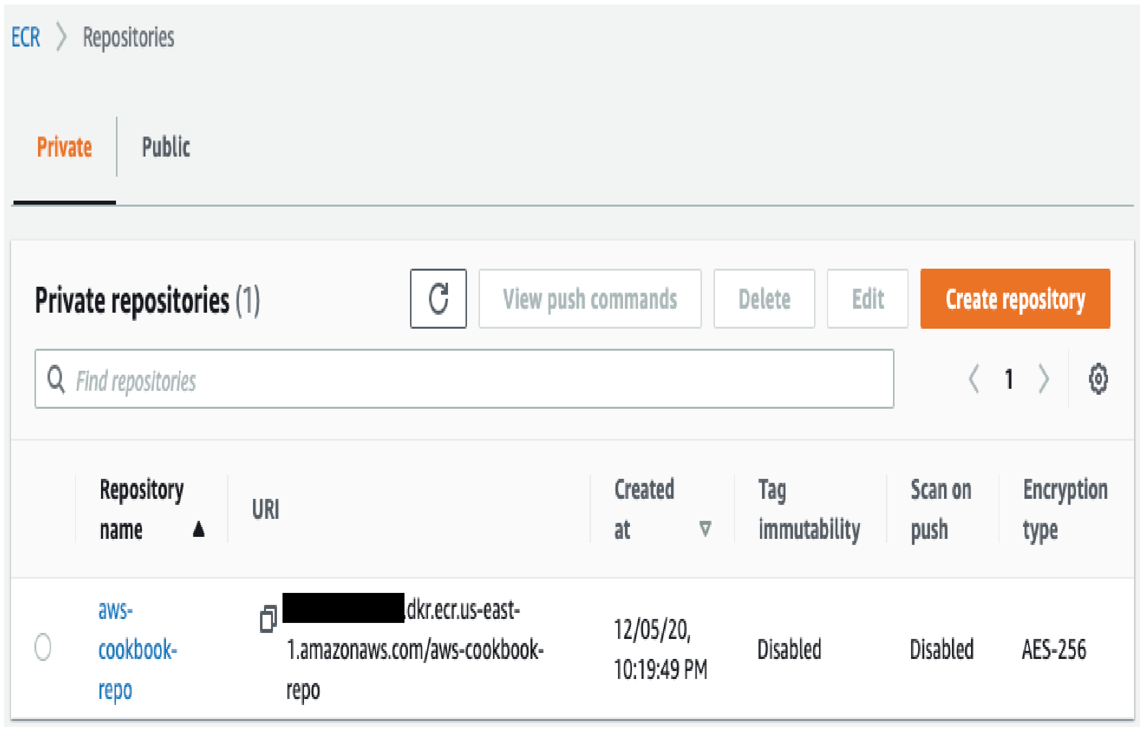
Enable tag immutability to prevent image tags from being overwritten by subsequent image pushes using the same tag. Disable tag immutability to allow image tags to be overwritten.

Disabled

Once a repository is created, the visibility setting of the repository can't be changed.

Figure 1-2. ECR repository Creation

You now have a repository created on Amazon ECR which you can use to push container images to!



Repository name	URI	Created at	Tag immutability	Scan on push	Encryption type
aws-cookbook-repo	[redacted].dkr.ecr.us-east-1.amazonaws.com/aws-cookbook-repo	12/05/20, 10:19:49 PM	Disabled	Disabled	AES-256

Figure 1-3. 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

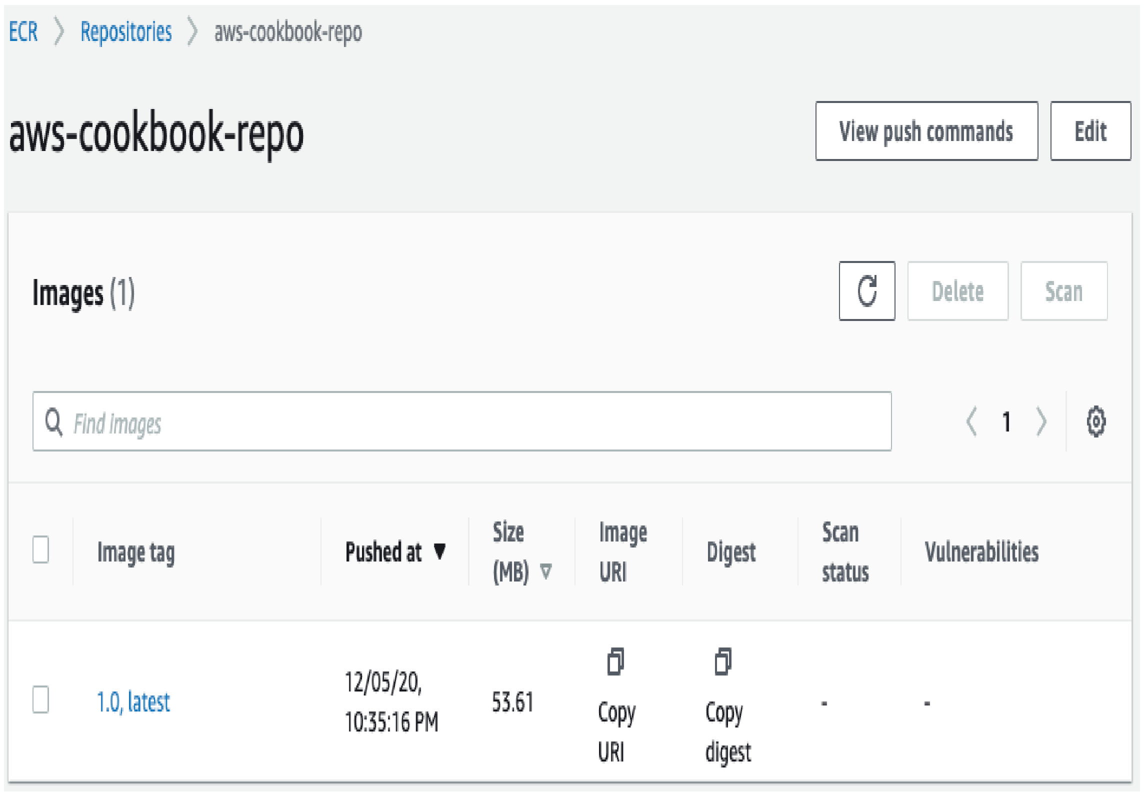


	Image tag	Pushed at ▼	Size (MB)	Image URI	Digest	Scan status	Vulnerabilities
☐	1.0, latest	12/05/20, 10:35:16 PM	53.61	Copy URI	Copy digest	-	-

Figure 1-4. 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":
"sha256:99d0a53e3718cef59443558607d1e100b325d6a2b678cd2a48b05e5e22f
feb49",
            "imageTag": "1.0"
        },
        {
            "imageDigest":
"sha256:99d0a53e3718cef59443558607d1e100b325d6a2b678cd2a48b05e5e22f
feb49",
            "imageTag": "latest"
        }
}
```

Clean Up

To clean up, first remove the image and then delete the empty repository.

```
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
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 whole universal resource identifier (URI) of the ECR repository is required with the tags so you can push to the 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.

1.2 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.

Steps

Create an ECR repository

```
aws ecr create-repository --repository-name aws-cookbook-repo
```

Now, 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
```

Rather than building a new container image from a Dockerfile (as you did in recipe 4.1), this time you are going to pull an old NGINX container image.

```
docker pull nginx:1.14.1
```

Apply a tag to this 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
```

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
```

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 in JSON format of the security scan on the image:

```
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

When you are finished viewing the vulnerabilities, 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
```

NOTE

[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.

1.3 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 will host the container and provide you with an HTTPS URL for your container.

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
  }
}
```

NOTE

[TIP]

We have provided this file for you in this recipe’s folder in the AWS Cookbook repo available at https://github.com/AWSCookbook/Chapter4

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 container image

```
aws lightsail delete-container-image \
--service-name awscookbook --image :awscookbook.awscookbook.1
```

Delete the container service..

```
aws lightsail delete-container-service --service-name awscookbook
```

Discussion

Lightsail provides a way to quickly deploy applications to AWS. 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 easily copy some html content into your container 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.

1.4 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:

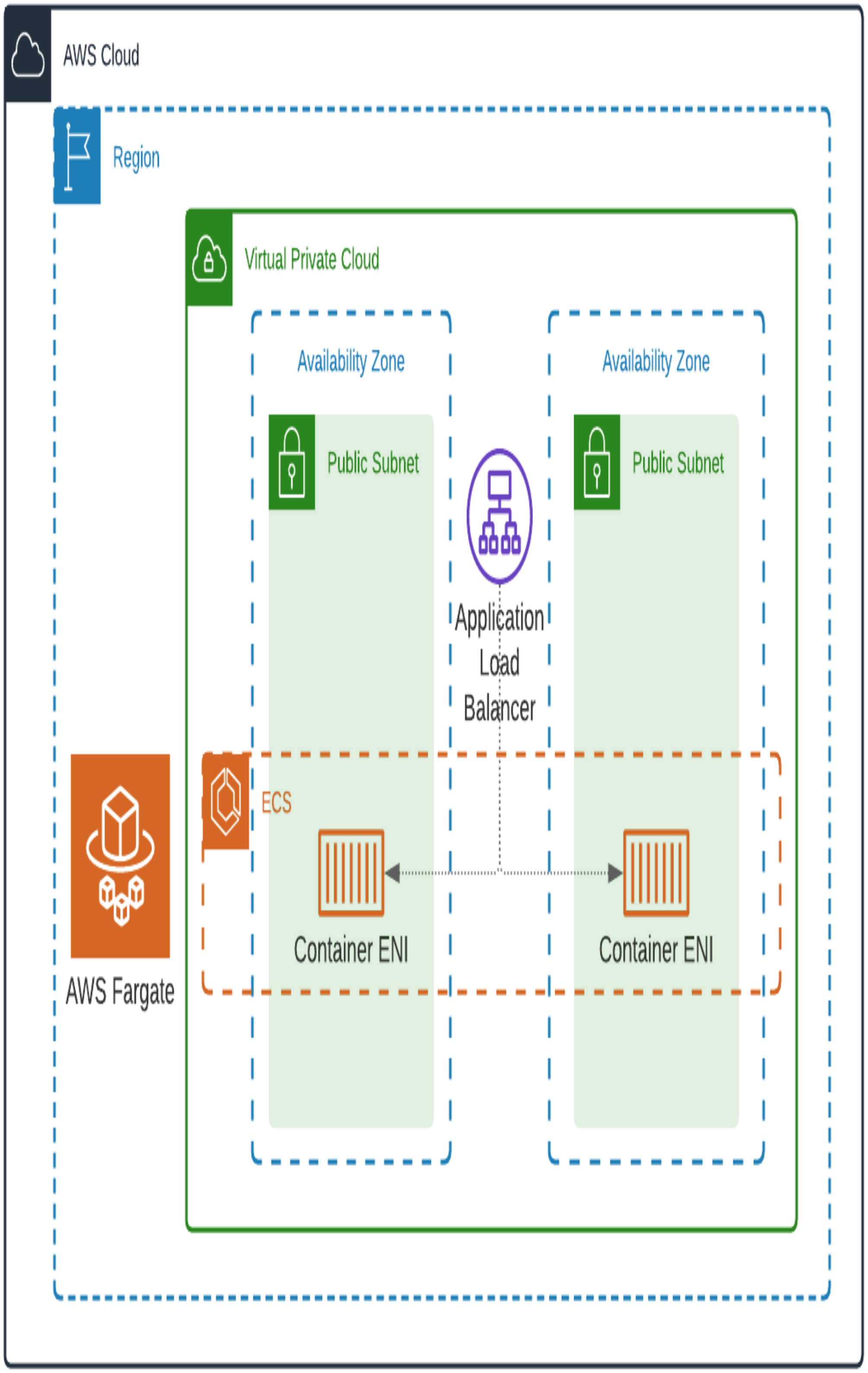

AWS Cloud
Region
Virtual Private Cloud
Availability Zone
Availability Zone
Public Subnet
Public Subnet
Application Load Balancer
ECS
Container ENI
Container ENI
AWS Fargate

Figure 1-5. 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/Chapter4):

```
cd 404-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 hte service 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.

1.5 Updating containers with blue/green deployments

Problem

You want to use a Blue/Green strategy for deployments of your application.

Solution

Use AWS CodeDeploy to orchestrate deployments to Amazon ECS with the Blue/Green strategy.

Preparation

In the root of the AWS Cookbook Chapter 4 repo `cd` to the “cdk-AWS-Cookbook-405” folder and follow the subsequent steps:

```
cd 405-Updating-Containers-With-BlueGreen/cdk-AWS-Cookbook-405/
```

Create a python virtual environment:

```
python3 -m venv .env
```

Activate the newly created python virtual environment:

```
source .env/bin/activate
```

Install the required python modules:

```
python -m pip install -r requirements.txt
```

If this is the first time you are using the AWS CDK, you'll need to bootstrap with the region you are working on with the AWS CDK Toolkit:

```
cdk bootstrap aws://$AWS_ACCOUNT_ID/$AWS_REGION
```

Deploy the resources (Hit "y" when prompted with "Do you wish to deploy these changes")

```
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

After the CDK deployment, visit the LoadBalancerDNS address in your browser, you will see the "Blue" application running:

```
E.g.: firefox http://fargateservicealb-925844155.us-east-
1.elb.amazonaws.com/
```

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

```
cd ..
```

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-405 \
--compute-platform ECS
```

CodeDeploy requires some configuration. We provide a template file (codedeploy-template.json) in this recipe’s folder of Chapter 4 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.

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
```

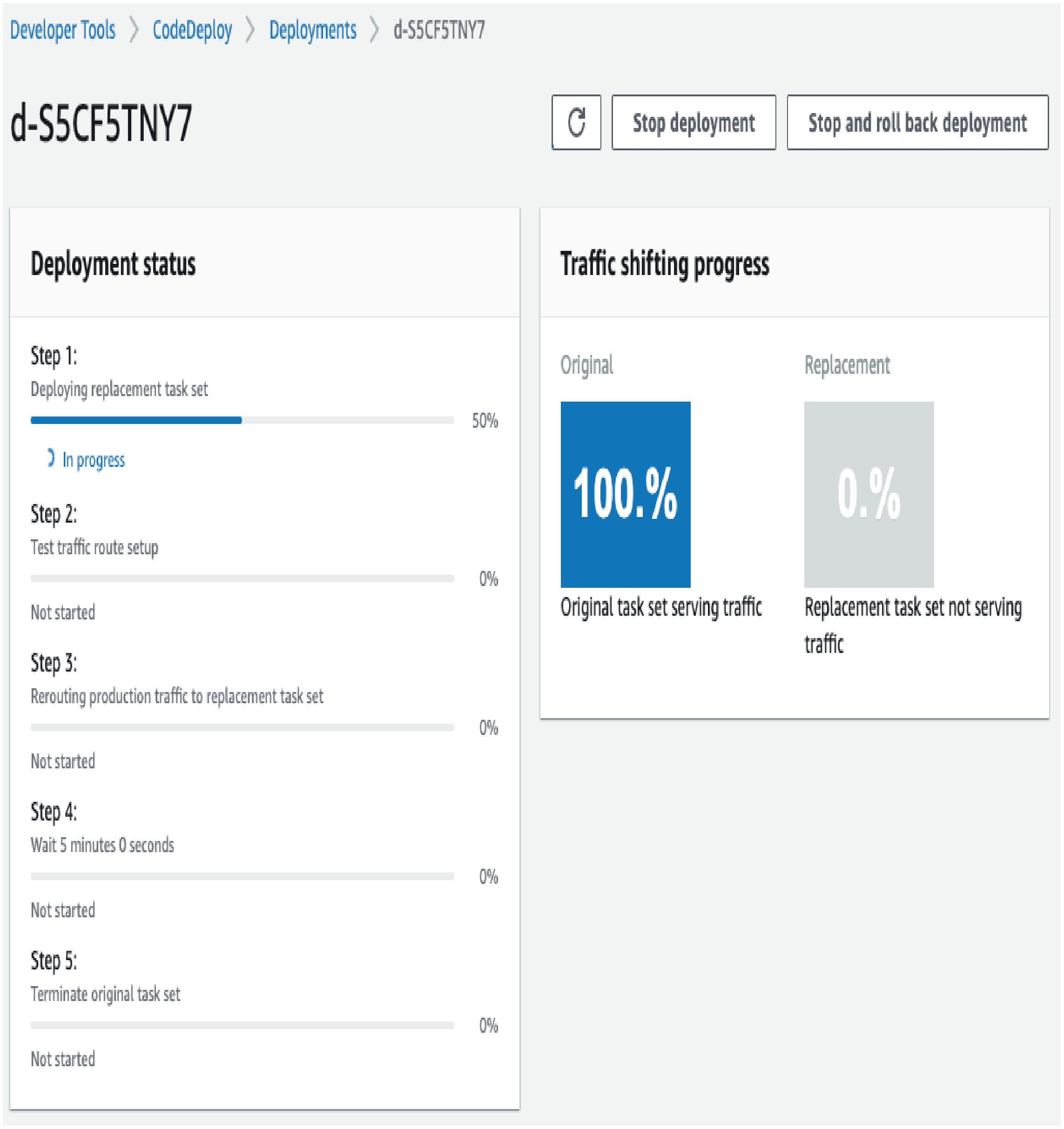
One final configuration file needs to be created, this contains the instructions about the deployment. Use `sed` to modify the S3 Bucket used in the deployment-template.json file.

```
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:


Developer Tools > CodeDeploy > Deployments > d-S5CF5TNY7

d-S5CF5TNY7

Deployment status

Step 1:
Deploying replacement task set
50%
In progress

Step 2:
Test traffic route setup
0%
Not started

Step 3:
Rerouting production traffic to replacement task set
0%
Not started

Step 4:
Wait 5 minutes 0 seconds
0%
Not started

Step 5:
Terminate original task set
0%
Not started

Traffic shifting progress

Original
100.%
Original task set serving traffic

Replacement
0.%
Replacement task set not serving traffic

Figure 1-6. Initial Deployment status

Once the replacement task is serving 100% of the traffic, you can visit the same URL you previously observed the Blue application running, replaced with the Green version of the application.

Clean Up

Delete the CodeDeploy deployment group and application:

```
aws deploy delete-deployment-group \ --deployment-group-name
awscookbook-405-dg \ --application-name awscookbook-405
aws deploy delete-application --application-name awscookbook-405
```

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)
```

Remove the S3 contents of the S3 Bucket to allow AWS CDK to remove it

```
aws s3 rm s3://$S3BucketName --recursive
```

Finally, use the AWS CDK to destroy the remaining resources:

```
cd cdk-AWS-Cookbook-405/
cdk destroy (Confirm with “y” when prompted with “Are you sure you
want to delete”)
```

Deactivate your python virtual environment:

```
deactivate
```

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 courier 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 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.

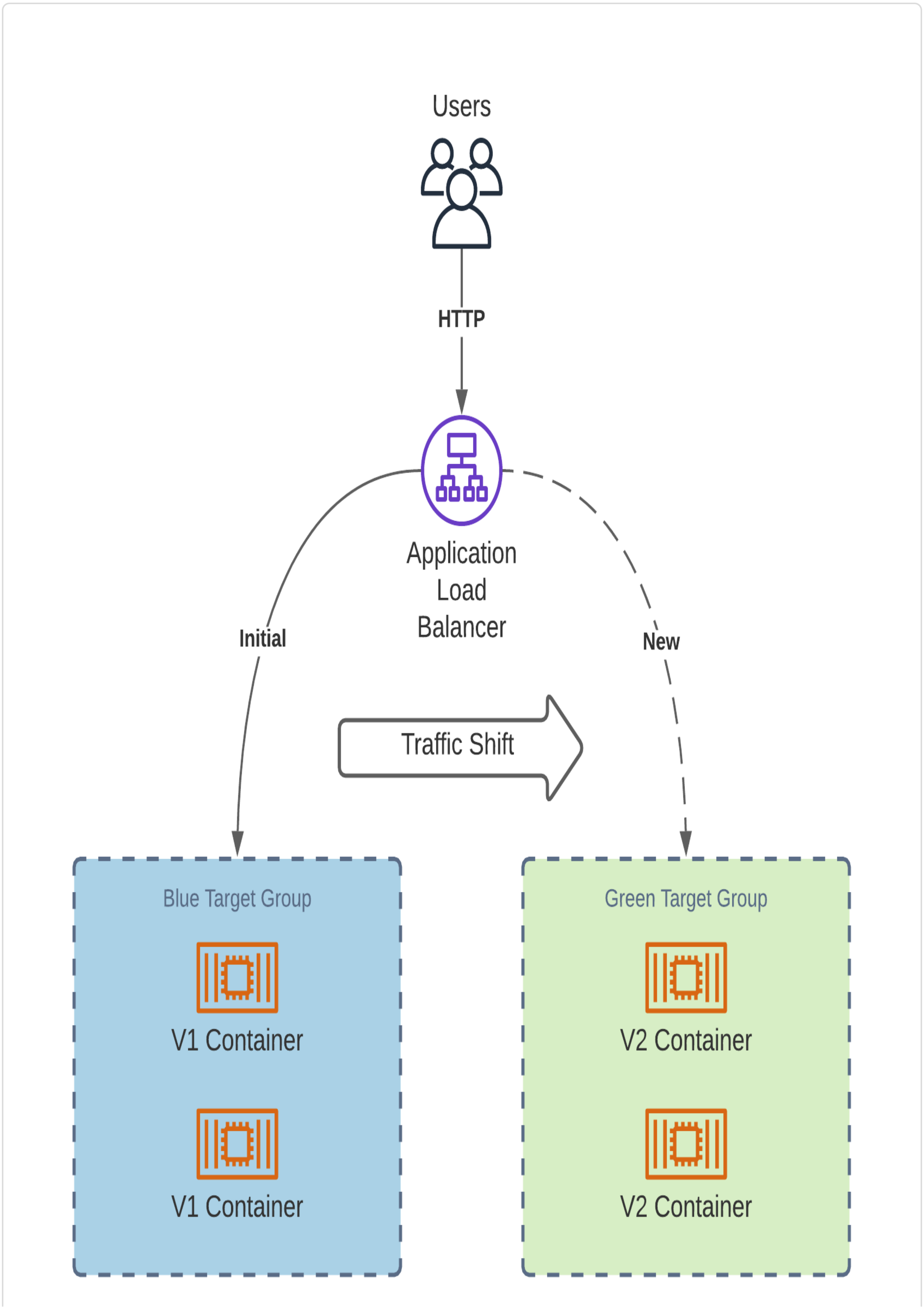

Users
HTTP
Application Load Balancer
Initial
New
Traffic Shift
Blue Target Group
V1 Container
V1 Container
Green Target Group
V2 Container
V2 Container

Figure 1-7. 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.

1.6 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 the Chapter 4 repo cd to the “406-Autoscaling-Container-Workloads/cdk-AWS-Cookbook-406” folder and follow the subsequent steps:

```
cd 406-Autoscaling-Container-Workloads/cdk-AWS-Cookbook-406/
python3 -m venv .env
source .env/bin/activate
python -m pip 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
```

Steps

Navigate up to the main directory for this recipe (out of the “cdk-AWS-Cookbook-406” folder)

```
cd ..
```

Access the ECS service URL over the internet with the cURL command (or your web browser) to verify the successful deployment:

```
curl -v -m 10 $LoadBalancerDNS
```

Use verbose (`-v`) and 10 second timeout (`-m 10`) 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 AWSCookbook406ECS \
--assume-role-policy-document file://task-execution-assume-
role.json
```

Attach the managed policy for Auto Scaling:

```
aws iam attach-role-policy --role-name AWSCookbook406ECS --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/loadtest \
    --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/loadtest \
    --policy-name cpu50-awscookbook-406 --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:

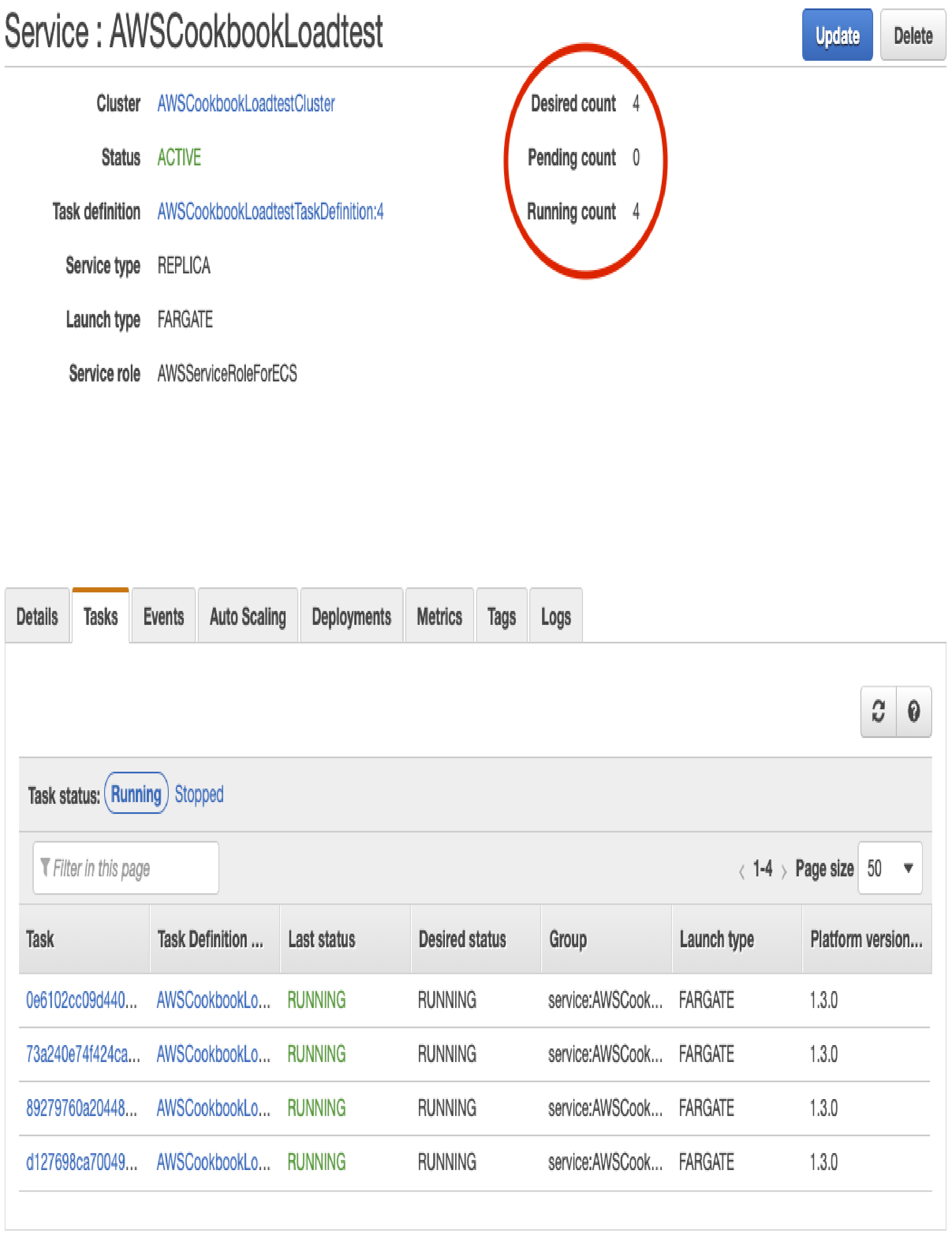
```
curl -v -m 10 $LoadBalancerDNS/cpu
```

This command will likely time out after 10 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
```

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 is now 4, the maximum scaling value that you configured. You can click the tasks tab to view 4 container tasks now running for your service.


Service : AWSCookbookLoadtest

Update Delete

Cluster	AWSCookbookLoadtestCluster
Status	ACTIVE
Task definition	AWSCookbookLoadtestTaskDefinition:4
Service type	REPLICA
Launch type	FARGATE
Service role	AWSServiceRoleForECS

Desired count	4
Pending count	0
Running count	4

Details Tasks Events Auto Scaling Deployments Metrics Tags Logs

Task status: Running Stopped

Filter in this page

1-4 Page size 50

Task	Task Definition ...	Last status	Desired status	Group	Launch type	Platform version...
0e6102cc09d440...	AWSCookbookLo...	RUNNING	RUNNING	service:AWSCook...	FARGATE	1.3.0
73a240e74f424ca...	AWSCookbookLo...	RUNNING	RUNNING	service:AWSCook...	FARGATE	1.3.0
89279760a20448...	AWSCookbookLo...	RUNNING	RUNNING	service:AWSCook...	FARGATE	1.3.0
d127698ca70049...	AWSCookbookLo...	RUNNING	RUNNING	service:AWSCook...	FARGATE	1.3.0

Figure 1-8. 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.

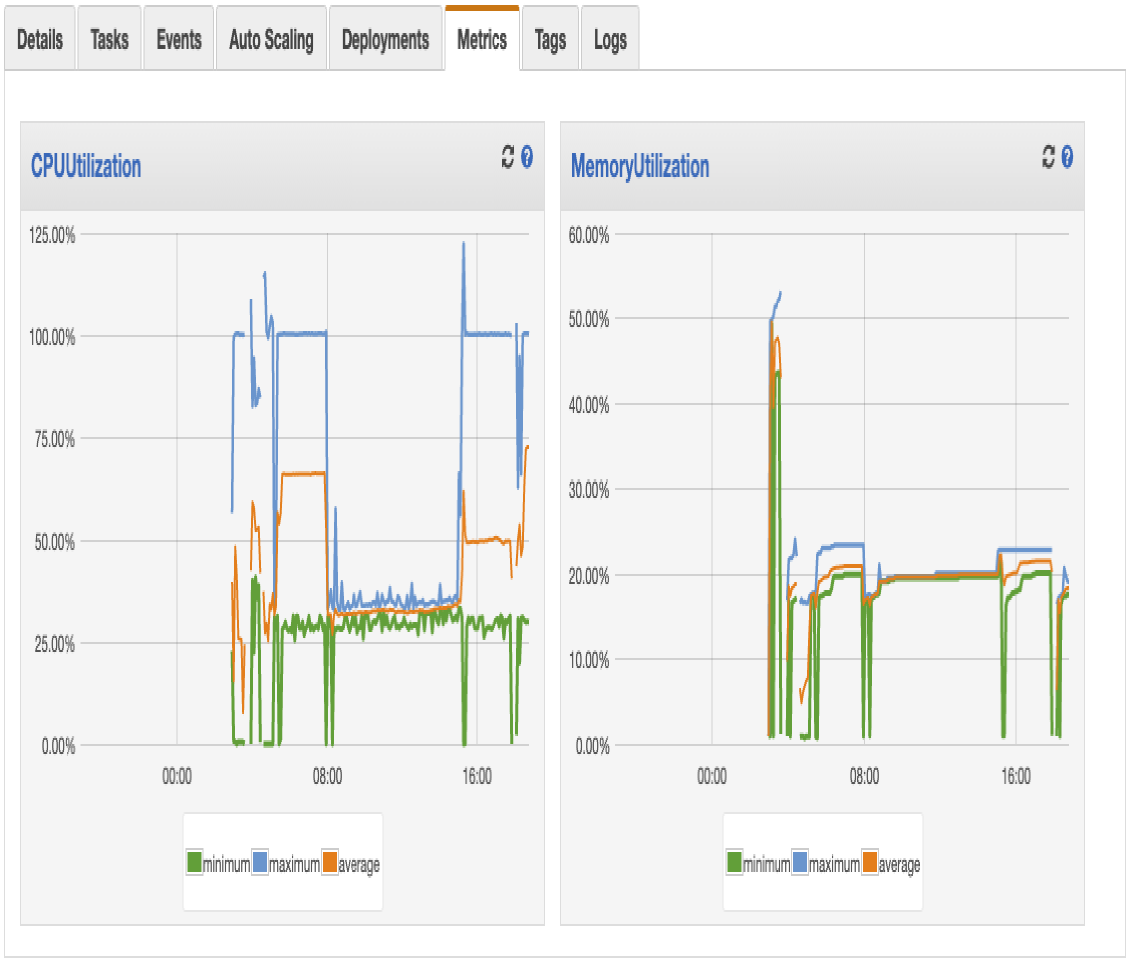


Figure 1-9. ECS service metrics on the AWS Console

Clean Up

Detach the managed Auto Scaling policy from the IAM role:

```
aws iam detach-role-policy --role-name AWSCookbook406ECS --policy-
arn \
arn:aws:iam::aws:policy/service-
role/AmazonEC2ContainerServiceAutoscaleRole
```

Delete the Auto Scaling IAM role:

```
aws iam delete-role --role-name AWSCookbook406ECS
```

Use the AWS CDK to destroy the resources:

```
cd cdk-AWS-Cookbook-406/
cdk destroy (Confirm with “y” when prompted with “Are you sure you
want to delete”)
```

Deactivate your python virtual environment:

```
deactivate
```

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
- Numnb

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. Network I/O and Memory metrics are also useful in cases; 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.

1.7 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.

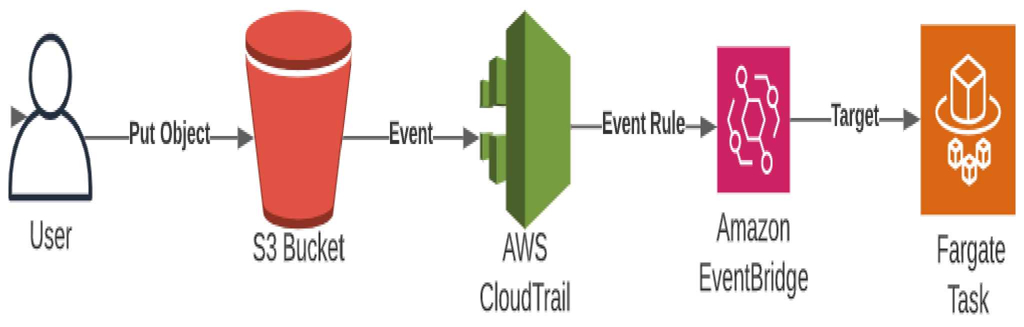


Figure 1-10. Flow of container EventBridge Pattern

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 the Chapter 4 repo cd to the “407-Fargate-Task-With-Event/cdk-AWS-Cookbook-407” folder and follow the subsequent steps:

```
cd 407-Fargate-Task-With-Event/cdk-AWS-Cookbook-407/
python3 -m venv .env
source .env/bin/activate
python -m pip 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
```

Steps

Navigate up to the main directory for this recipe (out of the “cdk-AWS-Cookbook-407” folder)

```
cd ..
```

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 AWSCookbook407RuleRole \ --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 AWSCookbook407RuleRole \
    --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::$AWS_ACCOUNT_ID:role/AWSCookbook407RuleRole" --event-
pattern "{\"source\":[\"aws.s3\"],\"detail-type\":[\"AWS API Call
via CloudTrail\"],\"detail\":{\"eventSource\":
[\"s3.amazonaws.com\"],\"eventName\":
[\"CopyObject\",\"PutObject\",\"CompleteMultipartUpload\"],\"reques
tParameters\":{\"bucketName\":[\"$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": []
}
```

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-1:1234567890:task/cdk-aws-cookbook-
407-AWSCookbookEcsCluster46494E6E-
MX7kvtp1sYWZ/d86f16af55da56b5ca4874d6029"
    ]
}
```

After a few minutes, observe the output.jpg file created in the S3 bucket:

```
aws s3 ls s3://$S3BucketName/
```

Download and view the output:

```
aws s3 cp s3://$S3BucketName/output.jpg .
```

Open output.jpg with a file viewer of your choice to view file that the

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 AWSCookbook407RuleRole \
--policy-name ECSRunTaskPermissionsForEvents
aws iam delete-role --role-name AWSCookbook407RuleRole
```

Remove the S3 contents of the S3 Bucket to allow AWS CDK to remove it

```
aws s3 rm s3://$S3BucketName --recursive
```

Finally, use the AWS CDK to destroy the remaining resources:

```
cd cdk-AWS-Cookbook-407/
cdk destroy (Confirm with “y” when prompted with “Are you sure you
want to delete”)
```

Deactivate your python virtual environment:

```
deactivate
```

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 alive. 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.

1.8 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 the Chapter 4 repo cd to the “408-Capturing-Logs-From-Containers-Running-On-ECS/cdk-AWS-Cookbook-408” folder and follow the subsequent steps:

```
cd 408-Capturing-Logs-From-Containers-Running-On-ECS/cdk-AWS-
Cookbook-408
python3 -m venv .env
source .env/bin/activate
python -m pip 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
```

Steps

Navigate up to the main directory for this recipe (out of the “cdk-AWS-Cookbook-408” folder)

```
cd ..
```

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 AWSCookbook408ECS \
--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 AWSCookbook408ECS --policy-
arn arn:aws:iam::aws:policy/service-
role/AmazonECSTaskExecutionRolePolicy
```

Create a Log Group in CloudWatch:

```
aws logs create-log-group --log-group-name AWSCookbook408ECS
```

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": "AWSCookbook408ECS",
                    "awslogs-region": "us-east-1",
                    "awslogs-stream-prefix": "LogStream"
                }
            },
            "name": "awscookbook408",
            "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": "awscookbook408",
    "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/AWSCookbook408ECS" \
--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 awscookbook408:1
```

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-
408-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 AWSCookbook408ECS{
    "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:AWSCookbook408ECS: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 AWSCookbook408ECS \
--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/35805865872844590178623550035180924397996026459535048705",
    "nextBackwardToken":
"b/35805865872844590178623550035180924397996026459535048704"
}
```

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 AWSCookbook408ECS --policy-
arn \
arn:aws:iam::aws:policy/service-
role/AmazonECSTaskExecutionRolePolicy
aws iam delete-role --role-name AWSCookbook408ECS
```

Delete Log Group:

```
aws logs delete-log-group --log-group-name AWSCookbook408ECS
```

Use the AWS CDK to destroy the remaining resources:

```
cd cdk-AWS-Cookbook-408/
cdk destroy (Confirm with “y” when prompted with “Are you sure you
want to delete”)
```

Deactivate your python virtual environment:

```
deactivate
```

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

Chapter 2. AWS Organizations

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 13th chapter 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*.

2.0 Introduction

AWS Accounts in the early days were singular (fig 13.1.1) management for all of your workloads. Since those days AWS has launched Organizations, organizational units and multi-account strategies that include:

- Multi-account governance
- Networking and Service Control Policies
- Consolidated Billing
- Organizational Policies for Backups and Tagging

AWS provides a pre-defined organization structure (fig 13.1.2) for you to create multi-account structure and organizational units. This structure provides a well-defined *landing zone* to centrally govern, secure and scale your usage of AWS. The following recipes provide easy to use structure, and defined usage patterns.

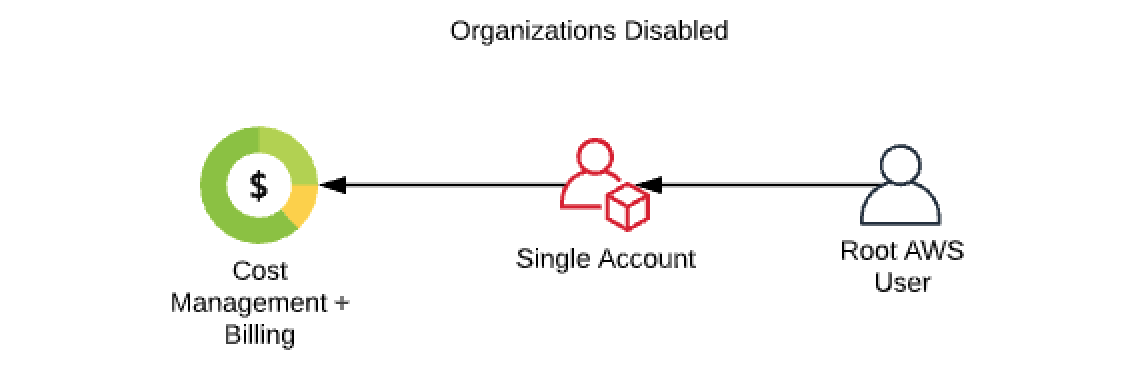


Figure 2-1. A single account setup without organizations. All workloads, stacks and dev to prod live in the same account.

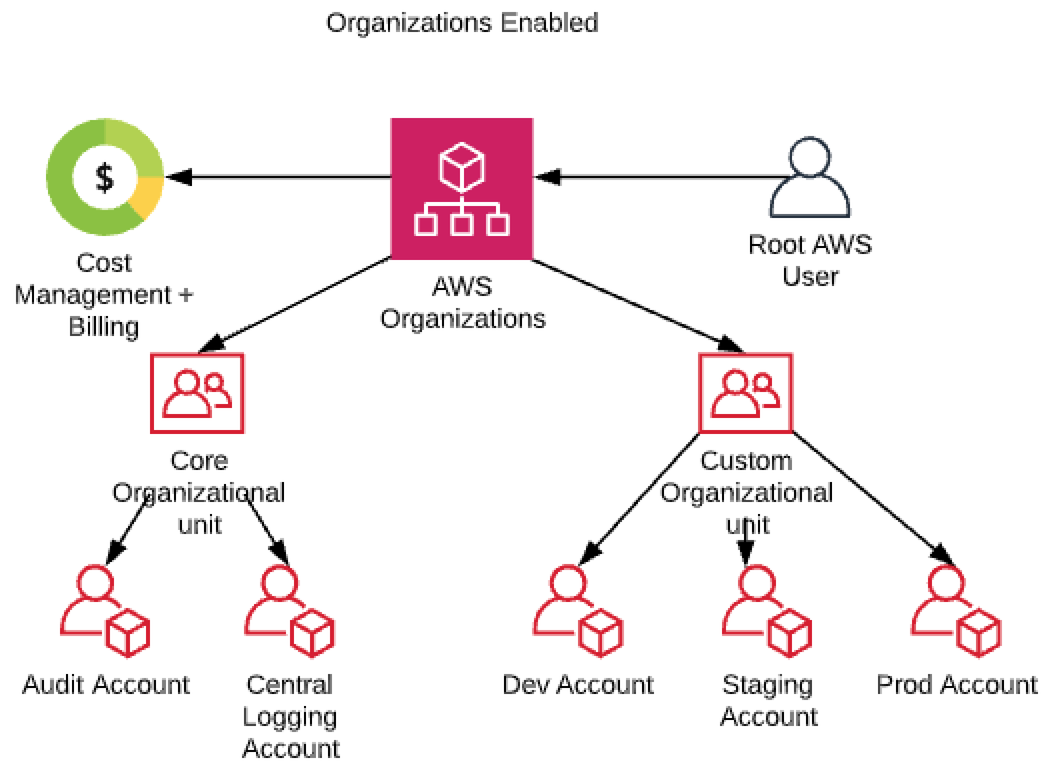


Figure 2-2. An AWS accounts setup using Organizations, broken out by Organizational units, stacks and workflows.

2.1 Setting up an Amazon Web Services Account

Problem

How do I set up an AWS Account?

Solution

You will set up a single payer account with the following steps.

1. Open the Amazon Web Services home page. https://aws.amazon.com/
2. Choose “Create an AWS Account”
3. Enter your account information, and then choose Continue.
4. Choose Personal or Professional. Though both accounts provide you the same services, professional should be chosen if using a company signup.
5. Enter your company or personal information.
6. Read and accept the AWS Customer Agreement.
7. Choose Create Account and Continue.

Once the account has been created, you are now the owner of an AWS management Account. This management account is what we will use for the remaining sections in this chapter.

Discussion

AWS defines the management account as “the AWS account you use to create your organization; it is also the primary account you receive when signing up for the first time with AWS. From the management account, you can create other accounts in your organization, invite and manage invitations for other accounts to join your organization, and remove accounts from your organization. You can also attach policies to entities such as administrative roots, organizational units (OUs), or accounts within your organization. The management account has the role of a payer account and is responsible for paying all charges accrued by the accounts in its organization. You cannot change which account in your organization is the management account.” It’s important to note that a management account and root account are the same. We have provided an example of what this structure looks like in the diagrams above. If you have set up a free AWS account and logged in, this is your management account, and the email you signed up with is the root user for the management account. If you have not set up an account, you can do so by following these simple steps.

You can create as many AWS accounts as you like; however, they all require unique email addresses. In addition, once an account is created, it can take up to 90 days for it to be disabled and removed. Please ensure you only set up accounts when needed. Generally a single account with no organizations attached is a valid way for a single developer to test and do POC requirements on their own time. If looking to do this as part of a larger group, the next section discusses this in more detail.

2.2 Organizing multiple accounts for enterprise deployments

Problem

Your company needs to segment accounts for PII (define these), PCI, and/or development resources as well as security, audit and monitoring.

Solution

Let’s create an Organizational structure following the Well Architected pattern as below.

management Payer Account (Payer Account) - MFA Enabled

- Core OU (Used for Audit, Central Logging)

  — Audit Account (SecOps, Compliance Validations)

  — Logging Account (Central Logging of all accounts)

- Custom OU

  — Production Resources Account

  — Staging Resources Account

  — Development Resources Account

Step 1: Log into your AWS Console (console.aws.amazon.com)

Step 2: Click the upper right Account Name menu and choose “My organization”

Step 3: Click on “Create Organization”

Step 4: Choose the Create Organization from the Modal window

Step 5: AWS organizations will send an email to validate your root email address of your management payer account. Once complete, continue with following steps.

Step 6: Click the “Organize Accounts” tab and create the following OU’s

- 6.1 Core OU
- 6.2 Custom OU

Step 7: Click on “Accounts” tab to create accounts to place under these OU’s (When adding an account, ensure you use a unique and accessible email. You cannot use the same email from any other account. If you do not have access to the email you input, you will not be able to access the new accounts without contacting support to assist). Recommendations are accountname+dept@domain.com

- 7.1 Add Account “Central Logging”
- 7.2 Add Account “Audit”
- 7.3 Add Account “Development”
- 7.4 Add Account “Staging”
- 7.5 Add Account “Production”

Now that you have created the primary accounts for our “Core” and “Custom” OU, we can now move them into their respective OUs. Inside the Organizations Dashboard, click on “Organize Accounts”, check the two accounts “Central logging” and “Audit”, then click the “move link” and place them into the Core OU. Repeat these steps for the Custom OU by moving “Development”, “Staging” and “Production” under the Custom OU. You may notice there are many options available for the OU and Accounts we don’t cover here, such as “Service Control Policies”, “Tag Policies” and more. We will cover these more in the following sections.

Step 8: Now that we have set up our accounts and our OUs, we can proceed to the following sections on logging, security and billing.

Discussion

AWS Organizations is a defined structure (Fig 13.1.2) that provides a hierarchy approach to Organization Units (OU), Accounts and Billing. You can nest multiple accounts under a single OUand up to 5 OUs nested under another. No account can be part of more than one OU at any time. In addition to the segmentation advantages for teams and the software development lifecycle, this also provides a great way to secure specific compliance requirements in their own accounts, scope of responsibility and billing resources. AWS has also launched a service called AWS Control Tower, that provides an automated setup much like the one below. However, to ensure you fully understand AWS Organizations, we recommend following this manual approach for now and utilizing the AWS Control Tower service from our Landing Zone section.

The Core OU that contains both the Audit and Logging accounts, is intended to provide a centralized location for the AWS Services to centralize AWS Cloudwatch logs, VPC Flow logs and AWS Cloudtrail logs. The logging account is where you can then set up your third party tools to stream logs from all accounts, or use the AWS pre-built solutions for automation and remediation as well as analytics via AWS Quicksight and Amazon Athena. The Audit account is where logs will be sent for a segmented and secured storage of all logs to help your security team audit and remediate accounts untouched by other teams.

When creating OUs and Accounts, many factors play into why you would choose one approach over another. A good rule of thumb is to ensure that you base your naming and decisions around access and blast radius. You may have two different accounts with resources that need to interact with each other, but may not be part of the same account due to security reasons. Place these under an OU that allows you to define how they can communicate and place all others in another OU or seperate account. Some segmentation ideas are mentioned here for review and ideation:

Segmentation Type	**Accounts**
Compliance	PII, FedRamp, PCI, 3rd Party Access for Audit, Mergers and Acquisitions, shared accounts between organizations.

Segmentation Type	Accounts
SDLC	Dev, Staging, Prod
Departmental	B2B, Consumer, Marketing

TIP

AWS has some additional info in the following White Paper: https://d0.awsstatic.com/aws-answers/AWS_Multi_Account_Security_Strategy.pdf

2.3 Service Control Policies

Problem

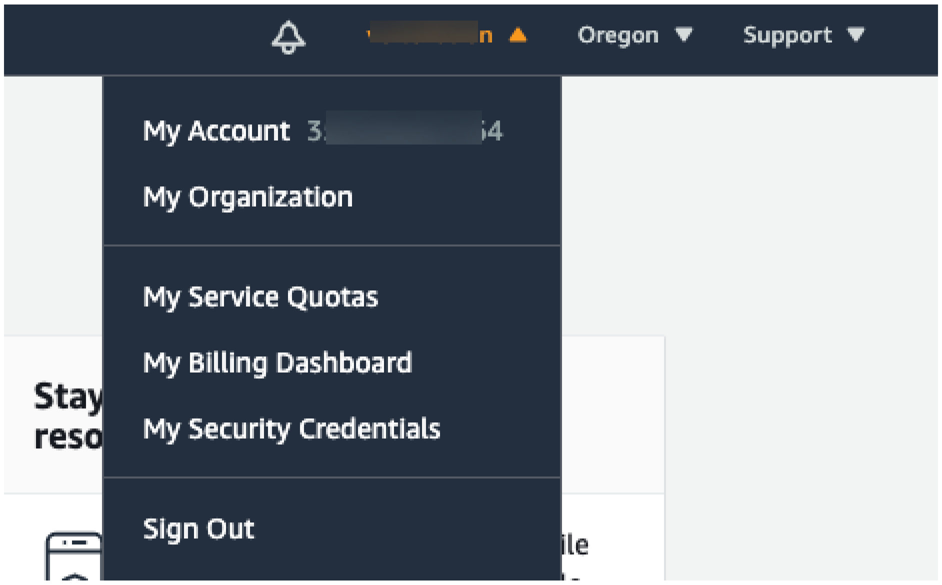
You need to block all internet access to any VPC inside a specific account(s). In addition, your boss has requested that an IAM principle cannot make certain changes globally.

Solution

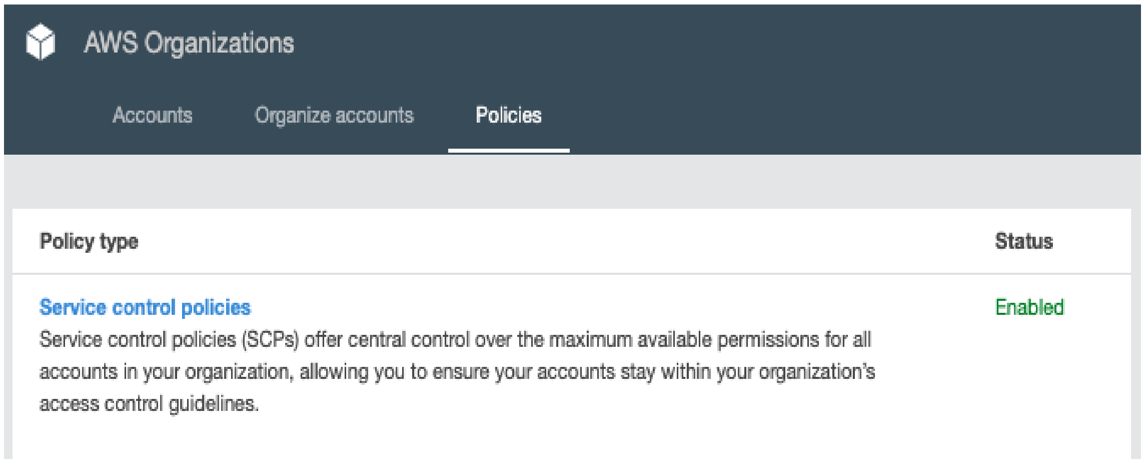
The following walks you through setting up a Service Control Policy and enabling that policy globally.

Step 1: Open up your AWS Console: console.aws.amazon.com

Step 2: Navigate to “My Organization” under your account name in the upper right.



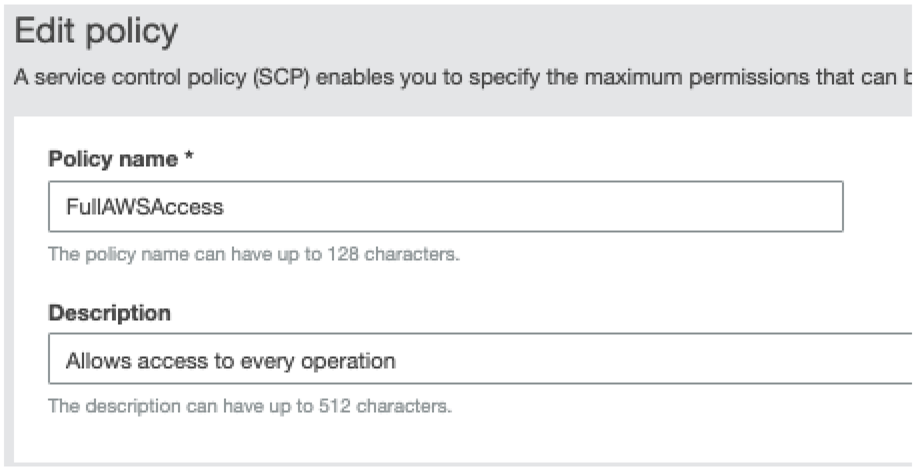
Step 3: Click on “Policies”, you should see a list of available policy services, click on “Service Control Policies” and enable the service.



Policy type	Status
Service control policies Service control policies (SCPs) offer central control over the maximum available permissions for all accounts in your organization, allowing you to ensure your accounts stay within your organization's access control guidelines.	Enabled

Step 4: Once enabled, click on the text “Service control policies” then the“Create Policy” button

Step 5: The create Console screen will show, this screen has the Policy Name, Description, and then a GUI for creating the policy. AWS has included an auto insert from all services listed on the left side menu, to make it easy to find what you are looking for. Fill in a name and description for this policy.



Edit policy

A service control policy (SCP) enables you to specify the maximum permissions that can b

Policy name *

FullAWSAccess

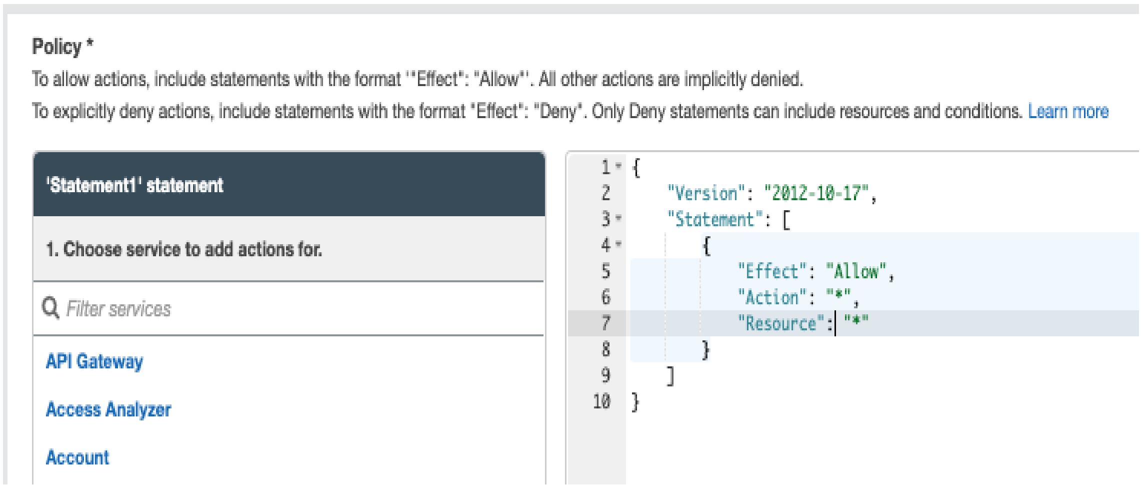
The policy name can have up to 128 characters.

Description

Allows access to every operation

The description can have up to 512 characters.

Step 6: Click inside the policy text section on the right, you should see a list of services on the left.



Policy *

To allow actions, include statements with the format '"Effect": "Allow"'. All other actions are implicitly denied.

To explicitly deny actions, include statements with the format "Effect": "Deny". Only Deny statements can include resources and conditions. Learn more

'Statement1' statement

1. Choose service to add actions for.

Filter services

API Gateway

Access Analyzer

Account

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

Click on or find EC2, then choose “AttachInternetGateway'', you will notice it inserts this method into the document under the “Deny” action statement which it defaults to at loadtime.

```
Example:
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Statement",
"Effect": "Deny",
      "Action": [
        "ec2:AttachInternetGateway”
      ],
      "Resource": "*"
    }
  ]
}
```

Step 7: Now follow these same steps until your document looks like the below by extracting the service name “Servicename:Action” and adding each action for each service. Do not hit save yet.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Statement",
"Effect": "Deny",
      "Action": [
        "ec2:AttachInternetGateway",
        "ec2:CreateInternetGateway",
        "ec2:CreateEgressOnlyInternetGateway",
        "ec2:CreateVpcPeeringConnection",
        "ec2:AcceptVpcPeeringConnection",
        "globalaccelerator:Create*",
        "globalaccelerator:Update*"
      ],
      "Resource": "*"
    }
  ]
}
```

Step 8: Success! You have now created your first SCP document policy. However, as we know, your boss asked you to also block all IAM principles from making changes. So, click on the “add Statement” link at the bottom of the current policy view. You will notice your policy is updated now with the first statement and has inserted a new statement section inside the policy for you. It should now look like the following:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Statement",
"Effect": "Deny",
      "Action": [
        "ec2:AttachInternetGateway",
        "ec2:CreateInternetGateway",
        "ec2:CreateEgressOnlyInternetGateway",
        "ec2:CreateVpcPeeringConnection",
        "ec2:AcceptVpcPeeringConnection",
        "globalaccelerator:Create*",
        "globalaccelerator:Update*"
      ],
      "Resource": "*"
    }
  ]
},
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Statement2",
"Effect": "Deny",
      "Action": [],
      "Resource": []
    }
  ]
}
```

Step 9: As in the preceding example, walk through each deny statement and add this to your policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Statement2",
      "Effect": "Deny",
      "Action": [
        "iam:AttachRolePolicy",
        "iam:DeleteRole",
        "iam:DeleteRolePermissionsBoundary",
        "iam:DeleteRolePolicy",
        "iam:DetachRolePolicy",
        "iam:PutRolePermissionsBoundary",
        "iam:PutRolePolicy",
        "iam:UpdateAssumeRolePolicy",
        "iam:UpdateRole",
        "iam:UpdateRoleDescription"
      ],
      "Resource": [
        "arn:aws:iam::*:role/role-that-users-can-never-change"
      ],
      "Condition": {
      "StringNotLike": {
"aws:PrincipalARN":"arn:aws:iam::*:role/role-needs-ability-to-make-changes"
        }
      }
    }
  ]
}
```

Step 10: Hold on, you say, what is the Resource ARN and Condition added at the end? SCPs have a fully documented syntax set as described here: https://docs.aws.amazon.com/organizations/latest/userguide/orgs_reference_scp-syntax.html#scp-elements-table

Since in the second statement we are adding a specific resource vs. all resources, we need to set the “ARN” of that resource. An ARN is an Amazon Resource Name, that identifies a resource in your account. In this case, we are specifying an IAM role as the resource we are denying. This role per the IAM chapter in this book could be the Power User role you set up for administering your AWS accounts, or could be specific to a custom role you created for your administrators.

Step 11: We are going to set a Condition clause that blocks this Deny action from being applied to myself or a role such as SecOps or the Root user from making these changes.

Step 12: Once completed, click the “Create Policy” and the policy has now been created. However we still need to apply the policy to the accounts and or Organizational Units.

Step 13: Navigate to the “Organize Accounts” section under “My Organization”. Choose the “Custom” OU by checking the box. On the right hand pane, you will see “Service Control Policies”, Click on this then choose the new policy you just created and click the “attach” link next to it. Your SCP has now been enabled across all accounts underneath the “Custom” OU.

Keep in mind, you can attach an SCP to a single account as well in addition to an OU. However since you are limited to 5 SCPs per account or OU, you can, as per the example earlier, embed multiple SCPs inside a single SCP with the use of multiple statement blocks.

Discussion

In this section we discussed what a Service Control Policy is, and what it can do. However we did not discuss strategies for this. How would you manage these? Who in your organization makes the approval decisions on these and executes them? How can they be used to successfully manage concerns around compliance and security in your organization? The following documentation from AWS provides some good baselines. However, as with all things, come up with a strategy your organization agrees on and is clearly stated and understood by all teams.

https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_policies.html

Service Control Policies (SCPs) are ways to control the broadest available permissions for all accounts in your organization. They ensure global control guidelines or guardrails across accounts. Remember that an SCP affects every user and role and even the root user in every OU and account that it's attached to. (Fig 13.3.1)

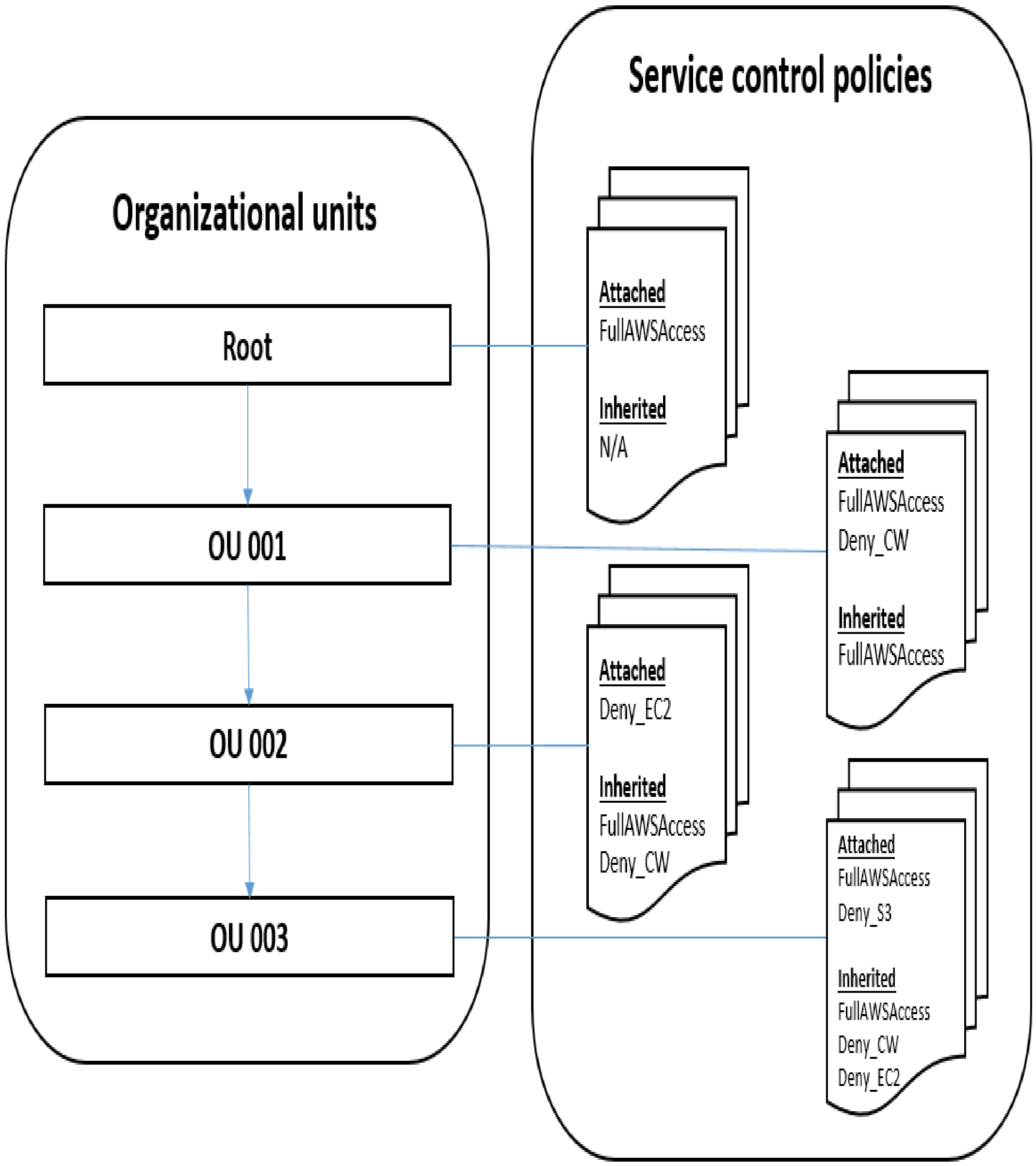
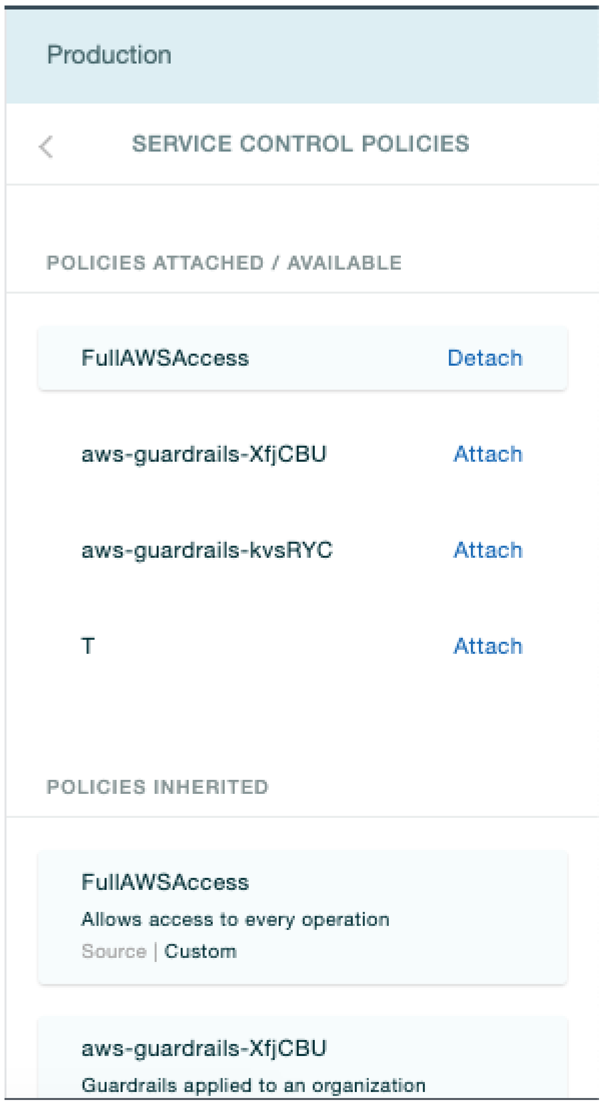


Figure 2-3. Caption to come

An SCP is inherited by any account and OU it’s attached to from its parent. So as an example, if you apply an SCP to the Management account, it applies to all accounts in your Organization. However if you apply to an account inside an OU, or directly to an OU, it only applies to that single account or all accounts inside that OU. You can see what SCP is attached or inherited by an account in the accounts section of Organizations, clicking on an account and viewing the Service Control Policies in the right pane (Fig 13.3.2).



Production

SERVICE CONTROL POLICIES

POLICIES ATTACHED / AVAILABLE

FullAWSAccess	Detach
aws-guardrails-XfjCBU	Attach
aws-guardrails-kvsRYC	Attach
T	Attach

POLICIES INHERITED

FullAWSAccess
Allows access to every operation
Source | Custom

aws-guardrails-XfjCBU
Guardrails applied to an organization

Figure 2-4. Caption to come

The table below (Table 13.3.1) gives the characteristics for this type of organizational policy.

Table 2-1. Caption to come

Policy type	**Affects management account**	**Maximum number you can attach to a root, OU, or account**	**Maximum size**	**Supports viewing effective policy for OU or account**
SCP	No	5*	5120 bytes	No

TIP

Although only 5 policies are allowed, you can embed multiple policies inside a single policy document up to the available size.

2.4 Tagging Policies and Resources

Problem

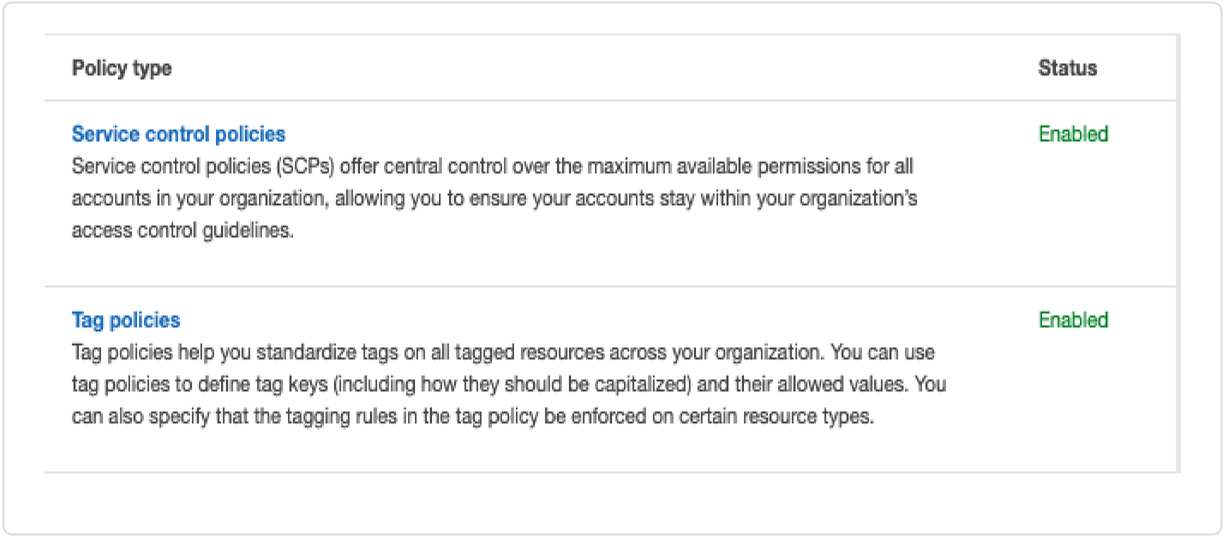
You want to ensure your organization tags all resources on creation. In addition, you want to ensure they only use defined values you have set for reporting purposes and enforcement of costs.

Solution

Adding a Tagging policy to your organization.

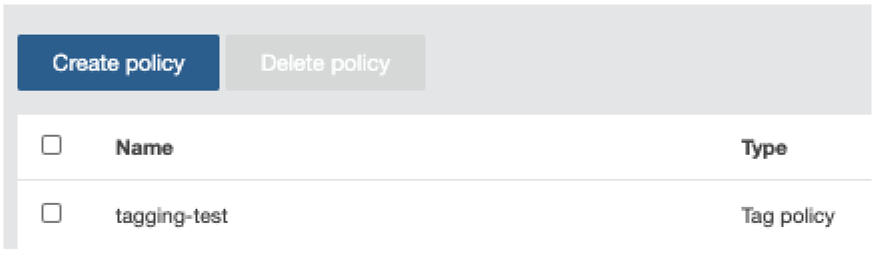
Step 1: Navigate to console.aws.amazon.com/organizations

Step 2: Click on “Policies” tab and then click on “Tagging Policies”



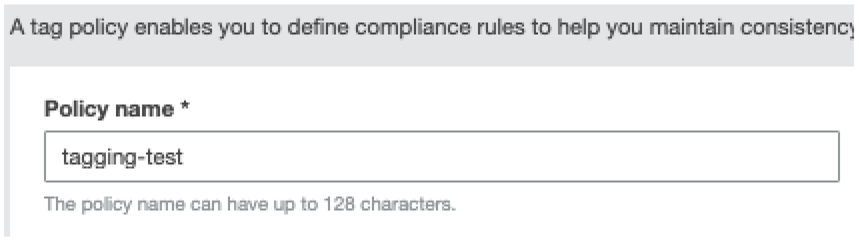
Policy type	Status
Service control policies Service control policies (SCPs) offer central control over the maximum available permissions for all accounts in your organization, allowing you to ensure your accounts stay within your organization’s access control guidelines.	Enabled
Tag policies Tag policies help you standardize tags on all tagged resources across your organization. You can use tag policies to define tag keys (including how they should be capitalized) and their allowed values. You can also specify that the tagging rules in the tag policy be enforced on certain resource types.	Enabled

Step 3: Click on “Create Policy”



	Name	Type
☐	tagging-test	Tag policy

Step 4: Fill in the name, in this example we will call it “tagging-test”. Fill in a description (optional), scroll down and you will see two sections, one says “Add Tag”, this section is not to be confused with the actual tags for resources, but the tags for the policy document itself. As you scroll down, you will see another section with a “visual editor” and “json” tab. This section of the document is where you define the tags key-value pairs, as well as the optional capitalization and resource enforcement.



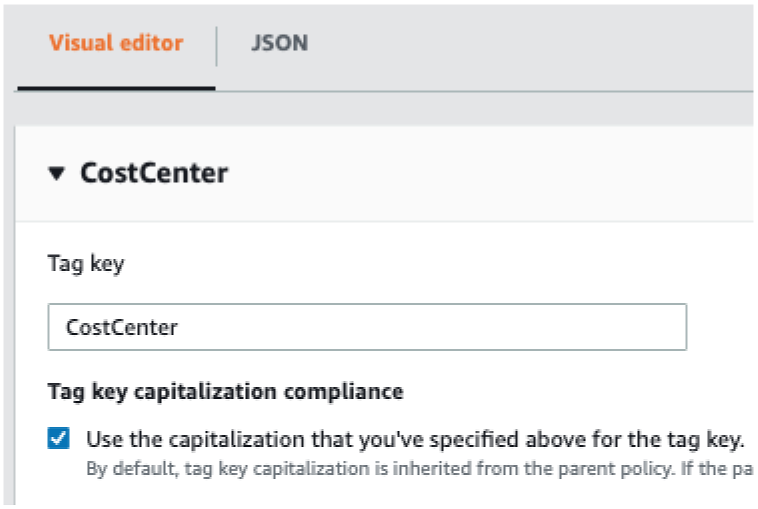
A tag policy enables you to define compliance rules to help you maintain consistenc

Policy name *

tagging-test

The policy name can have up to 128 characters.

Step 5: In the tag key, type in “CostCenter”, directly under that input, check the box for “Tag Key Capitalization enforcement”. This ensures that the tag key will use the same case as you desire.



▼ CostCenter

Tag key

CostCenter

Tag key capitalization compliance

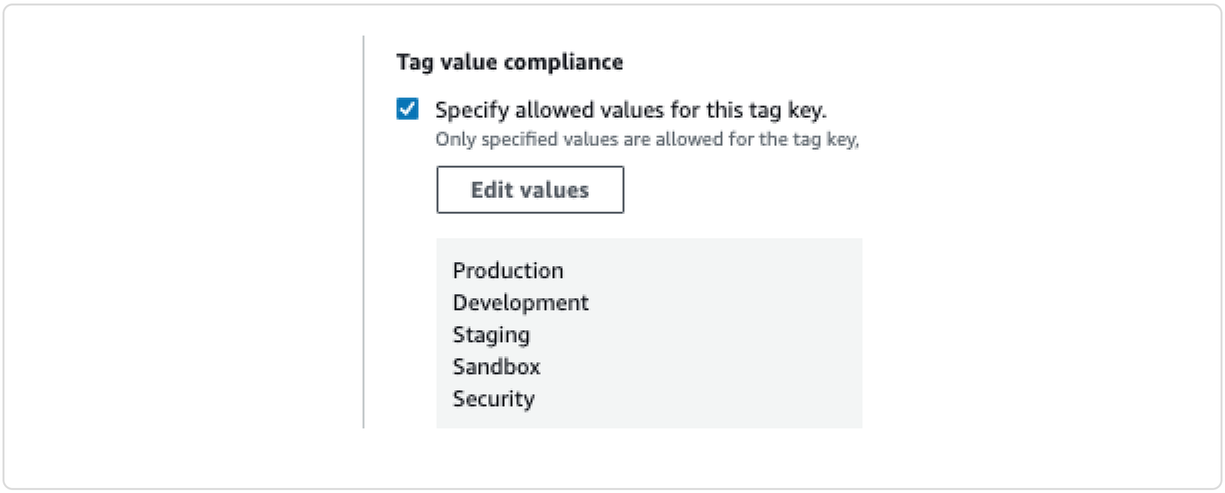
☑ Use the capitalization that you've specified above for the tag key.

By default, tag key capitalization is inherited from the parent policy. If the pa

Step 6: Under "Tag Value Compliance", check the box for "Specify allowed values for this tag key"

Step 7: We will now input values we expect to see and need to report on for our "CostCenter", click on "Edit Values", you will get a Modal window, input the following values, as you add a value, click the "Add another value" button until complete. The values to input are:

- Production
- Staging
- Development
- Sandbox
- Security



Tag value compliance

☑ Specify allowed values for this tag key.
Only specified values are allowed for the tag key,

Production
Development
Staging
Sandbox
Security

Once complete, click the “save changes”. The json syntax for this policy is here, and can be added by pasting into the “json” tab next to the “visual editor”

```
{
    "tags": {
        "CostCenter": {
            "tag_key": {
                "@@assign": "CostCenter"
            },
            "tag_value": {
                "@@assign": [
                    "Production",
                    "Development",
                    "Staging",
                    "Sandbox",
                    "Security"
                ]
            }
        }
    }
}
```

Step 8: The last option is “prevent non-compliant operations for this tag”, this option enforces the tags presence for the resources chosen. If not checked, or not chosen, any of the tags you have specified and not used, will show as non-compliant resources, but will still allow the resources to be created. This option can be seen as a proactive action, where when not chosen, this policy is simply seen as non-compliant, or reactive action.

Step 9: Choose “Create Policy”. This will then take you back to the tagging policy table.

Step 10: Click on the policy “tagging-test”, this is the name we had you call this policy. When clicked, you should see a list display to the right with “Accounts”, “Organizational Units”, and “Roots”. For this example, we will choose “Roots”.

Step 11: Click the “attach” action next to root. This in effect attaches this policy to the root (top) of your Organization. This ensures this policy will impact all accounts and OUs that roll up to your management account.

Step 12: To see the compliant and non-compliant resources for this policy, click on the upper left tab “accounts”, choose your root account (a star should show next to it), then click the far right list that pops up for “tag policies”. You should see two options at this point, “view effective tag policy” and “view compliance status”.

Step 13: Click “view compliance status”, and this should launch a new window where you can view all resources that are compliant and non compliant under your organization and all accounts.

Discussion

A tag is a custom attribute label that you add to an AWS resource to make it easier to identify, organize, and search for resources. Each tag has two parts:

- A tag key (for example, CostCenter, Environment, or Project). Tag keys are case sensitive.

- A tag value (for example, 111122223333 or Production). You can set the value of a tag to an empty string, but you can't set the value of a tag to null. Omitting the tag value is the same as using an empty string. Like tag keys, tag values are case sensitive.
- Tag Limitations:

  — Each tag key must be unique

  — Resources currently have a maximum of 50 user created tags

  — The tag value must be a minimum of 0 and a maximum of 256 Unicode characters in UTF-8.

  — The tag key must be a minimum of 1 and a maximum of 128 Unicode characters in UTF-8.

With tagging policies, you can stack the policies at an organization level for required organization wide reporting, then attach specific policies to accounts and organizational units that require different types of reporting and enforcement. As an example, if you wanted to see the utilization of resources based on a new schema, query or application codebase, you may want to add key-value pairs for version, schema etc… This would then allow you to cross report this with the Cost and Utilization report under billing (Chapter 6) and be able to see if the new code is more efficient or costly. You can use these approaches for many other use cases. Additional advanced methods can be seen here: https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_policies_example-tag-policies.html#tag-policy-syntax-reference

2.5 Backup Policies

Problem

You have been asked to create backups on Amazon DynamoDB once per day in all production accounts, and in developer accounts, once per week.

Solution

You will create backups for production and development accounts using Organization backup policies.

Step 1: Login to your root account, navigate to organizations and choose “Policies”, click on “backup policies” and choose enable.

Step 2: Click on “Create Policy”

Step 3: You will be met with the following sections, fill out for now the following

- Policy Name: Production
- Description: Production Backup Policy
- Backup Details

  — Backup plan name: production

  — Backup plan regions: us-west-2, us-east-1
- AWS Backup Rule

  — Rule name: daily_backup

  — Backup window: leave on defaults for now

  — Lifecycle: leave both transition to cold storage and expire to never

  — Backup vault: production

  — Copy to regions: leave blank for now, can be used to copy for disaster recovery.

  — Click “Add backup rule”

- Assign Resources:
  — Resource assignment name: production
  — IAM Role: default
  — Resource Tage Key: BackupPlan
  — Tag Values: Production
- Advanced backup settings: leave untouched for now
- Click on “Create policy”

Step 4: You will now be shown a “Targets” action. This allows us to apply the policy to the account and or OU of our choosing. Choose “Attach”.

Step 5: Click on an OU or individual account. If you choose an OU, the accounts under that OU get that policy applied. If you choose a specific account, only that account has the policy applied. You can stack targets or only choose one. For this example, choose the “Production” account under “Custom” OU from our “Organizing Multiple accounts” recipe earlier.

Step 6: Create the same type of policy for development with changes per section for development and ensure you attach the new policy to the “development” account under the “Custom” OU as its target.

Discussion

AWS Organizations provides a backup policy management tool in two locations. AWS Backup and AWS Organizations. These services partnered together to ensure customers could provide compliance and RPO targets at scale across multiple accounts and targets. Inside AWS Organizations you can create and manage your backup policies by OU and accounts. In addition, you can see these inside the AWS Backup service when configured and enabled.

AWS Backup policies allow for management of backups across multiple accounts and OU’s based on specific tag and value key stores. Utilizing this approach allows you to enable and enforce backups across your organization in conjunction with your tagging policy approach from 13.4 recipe. In addition, you can monitor and manage these directly inside the AWS Organizations or the AWS Backup service console. AWS Backup can be used for a variety of services, and every individual service itself has the ability to create backups as well if the service has enabled it. Organization-wide backups should be used because it simplifies auditing and compliance, as well as helps control costs and keep infra management away from individual teams

2.6 Leaving an Organization as a Member Account

Problem

As a member account administrator, you need to remove an account from an organization.

Solution

Removing an account from your organization that you no longer need can take two a couple paths. The first is removing it from your organization, thereby making the account being removed responsible for its own charges and billing. The second is deleting an account, which removes it from the organization and deletes all resources associated with that account.

We are going to walk through removing an account first, then discuss deleting an account.

Pre-requisites

- For each account that you want to remove from an organization and make a standalone account, you must choose a support plan, provide and verify required contact information, and provide payment methods. AWS uses the payment method to charge for any billable (not AWS Free Tier) AWS activity that occurs once the account isn't attached to an organization.
- The principals in the account are no longer affected by any policies that applied in the organization.
- Any services associated with the current organization may become inactive such as AWS SSO and the users associated with it.
- If this is for the current management account, this cannot leave the organization without first deleting the organization.

Required Minimum Permissions (if leaving as the leaving account admin). Note the Management account for the organization that currently administers the account can also remove and account.

If removing an account, these minimum permissions are not required:

- `organizations:DescribeOrganization` (console only).
- `organizations:LeaveOrganization` – Note that the organization administrator can apply a policy to your account that removes this permission, preventing you from removing your account from the organization.
- If you sign in as an IAM account user and the account is missing payment information, the IAM user must have the permissions `aws-portal:ModifyBilling` and `aws-portal:ModifyPaymentMethods`. Also, the member

account must have IAM user access to billing enabled. If this isn't already enabled, see Activating Access to the Billing and Cost Management Console in the *AWS Billing and Cost Management User Guide*.

Step 1: Sign in to the AWS Organizations console at https://console.aws.amazon.com/organizations/. You must be signed in as an IAM user, assume an IAM role, or sign in as the root user (not recommended) in the member account that's leaving the organization.

By default, you don't have access to the root user password in a member account that was created using AWS Organizations. If required, recover the root user password by walking through the steps at Accessing a member account as the root user.

Step 2: On the Organizations Overview page, choose "Leave Organization"

Step 3: Perform one of the following steps:

- If your account has all the required information to operate as a standalone account (see prerequisites above), a confirmation dialog box appears. Confirm your choice to remove the account. You are redirected to the Getting Started page of the AWS Organizations console. Here you can choose to create a new Organization for this standalone account, or accept invites to join other Organizations if any.
- If your account doesn't have all the required information, perform the following steps:

  — A dialog box appears to explain that you must complete some additional steps. Click the link.

  — Complete all the sign-up steps that are presented. They might include the following:

    - Provide contact information
    - Provide a valid payment method
    - Verify the phone number
    - Select a support plan option

— When you see the dialog box stating that the sign-up process is complete, choose Leave organization.

— A confirmation dialog box appears. Confirm your choice to remove the account. You are redirected to the Getting Started page of the AWS Organizations console. Here you can choose to create a new Organization for this standalone account, or accept invites to join other Organizations if any.

Step 4: Remove the IAM roles that grant access to your account from the Organization.

Discussion

By following this process you can move an account you administer from one organization to another or simply create its own standalone management account. Once you have done that, you can use this account to create a new organization and create member accounts under it, or join it to another Organization via invites from the other organization. Keep in mind that all history such as billing, governance policies such as guard rails and SCP’s that are assigned across an OU and account in organizations will be removed. Ensure you get a copy of this information if you intend to use it moving forward, including any policies that were custom for the organization you just left.

2.7 Removing accounts from your organization

Problem

As a Management Account Administrator, you need to remove a member account from your organization.

Solution

You will remove a member account from your organization with the following steps.

Pre-requisites

- For each account that you want to remove from an organization and make a standalone account, you must choose a support plan, provide and verify required contact information, and provide payment methods. AWS uses the payment method to charge for any billable (not AWS Free Tier) AWS activity that occurs once the account isn't attached to an organization.
- The principals in the account are no longer affected by any policies that applied in the organization.
- Any services associated with the current organization may become inactive such as AWS SSO and the users associated with it.
- If this is for the current management account, this cannot leave the organization without first deleting the organization.

Step 1: Sign in to the AWS Organizations console at https://console.aws.amazon.com/organizations/. You must sign in as an IAM user, assume an IAM role, or sign in as the root user (not recommended) in the organization's management account.

Step 2: On the Accounts tab, select the check box next to the member account that you want to remove from your organization. You can choose more than one.

Step 3: Choose Remove account.

Step 4: In the Remove account dialog box, choose Remove. A dialog box displays the success or failure status for each account.

Step 5: If AWS Organizations fails to remove one or more of the accounts, it’s typically because you have not provided all the required information for the account to operate as a standalone account. Perform the following steps:

1. Sign in to one of the failed accounts.
2. We recommend that you sign in to the member account by choosing Copy link, and then pasting it into the address bar of a new incognito browser window. If you don’t use an incognito window, you’re signed out of the management account and won’t be able to navigate back to this dialog box.
3. The browser takes you directly to the sign-up process to complete any steps that are missing for this account. Complete all the steps presented. They might include the following:
    - Provide contact information
    - Provide a valid payment method
    - Verify the phone number
    - Select a support plan option
4. After you complete the last sign-up step, AWS automatically redirects your browser to the AWS Organizations console for the member account. Choose Leave organization, and then confirm your choice in the confirmation dialog box. You are redirected to the

Getting Started page of the AWS Organizations console, where you can view any pending invitations for your account to join other organizations.

Discussion

By following this process you can move an account you administer from one organization to another or simply create its own standalone management account. Once you have done that, you can use this account to create a new organization and create member accounts under it, or join it to another Organization via invites from the other organization. Keep in mind that all history such as billing, governance policies such as guard rails and SCP’s that are assigned across an OU and account in organizations will be removed. Ensure you get a copy of this information if you intend to use it moving forward, including any policies that were custom for the organization you just left.

2.8 Deleting an account

Problem

You want to delete an account and remove all information, resources and data associated with it.

Solution

You will delete an account with the following steps.

Prerequisites

Before closing or deleting an account ensure you:

- Backup any applications and data you want to retain and move them to another account or offline to your local machine/network.

- AWS cannot recover or restore any data or resources once deleted.

Step 1: Sign in as the root user of the account that you want to close using the email and password associated with that account. You cannot sign in as an IAM user for this action.

Step 2: Open the Billing and Cost Management console at https://console.aws.amazon.com/billing/home#/.

Step 3: On the navigation bar in the upper-right corner, choose your account name (or alias) and then choose “My Account”.

Step 4: On the Account Settings page, scroll to the end of the page to the Close Account section. Read and ensure that you understand the text next to the check box.

Step 5: Select the check box to confirm your understanding of the terms and then choose “Close Account”.

Step 6: In the confirmation box, choose “Close Account”.

Discussion

After you close an AWS account, you can no longer use it to access AWS services or resources. You will be given 90 days after you close your account (the “Post-Closure Period”), to log in and view past bills and access AWS Support. You can contact AWS Support within the Post-Closure Period to reopen the account. For more information, see How do I reopen my closed AWS account? in the Knowledge Center.

Keep in mind that if this account was used to share resources, data or applications to other accounts, those will become unusable the moment you confirm this account’s closure. Ensure you either move those resources, data or applications to accounts you will remain open, or ensure they are good to be deleted and removed.

Appendix A. Fast Fixes

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)
```

S et 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'
```

S imple 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.

James Ferguson has been involved in technology for the past 25 years.James has seen the transformation from on-prem data centers to the Cloud and SOA based systems personally. His roles have included Developer, Principal Architect, C-Level and Engineering team leads for some of the worlds largest companies. He now is a technology leader and Sr. SA with AWS, helping teams transform their business into nimble and productive services.