STACS SOLUTIONS ENGINEERING

Continuous Delivery with AWS CodePipeline and ECS Fargate

PREFACE

The Hashstacs Solutions Engineering team has been building modern applications on the cloud and we faced many challenges ranging from minimizing application downtime to application scalability.

As we continue to deliver new product features and bug fixes, we have fine-tuned our process on the AWS cloud to take advantage of managed services to reduce dedicated manpower for devops and to enable rapid deployment from code to product.

In this article we would like to share our experience and know-how setting up a fully automated Continuous Delivery workflow on the AWS cloud.

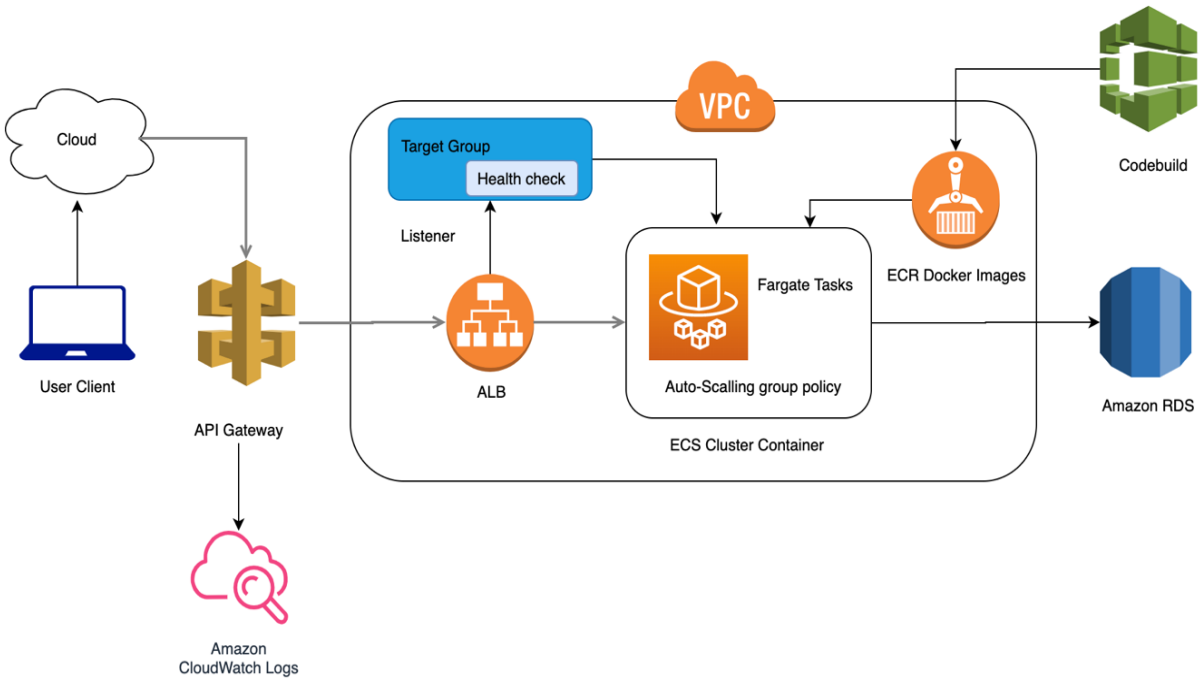
CONTENTS

1 High Level Architecture Design

The following is a high level solution design encompassing the many services used for our Continuous Delivery workflow.

We will split up the design into 2 separate portions:

1) Deployment of the source code as containers
2) Setup of the containers with API Gateway for secured access by the end user client



1.1 Deployment from Source code to Containers

As AWS Elastic Container Service (ECS) is a fast, highly scalable managed service that orchestrates and provisions Docker containers, we have chosen to use Fargate on ECS for our final deployment of applications.

This first portion of our deployment setup primarily resides in AWS Code Pipeline which covers the full range of activities from source code to containerized applications on ECS Fargate.

As our version control system is managed on Github, for the application deployment source code is first pushed to GitHub and pulled by AWS CodePipeline automatically via web hooks.

After that, CodeBuild will be triggered to build and deploy the package into the AWS Elastic Container Repository (ECR) as docker images where we track all versions of our containerized application.

Finally, CodePipeline will pull the latest image from ECR and deploy on ECS Fargate.

Using ECS Fargate, a fully managed service, we can organise the state of containers which are independently monitored, scaled up and down automatically, making sure the minimum set of tasks are running based on demand which is an incredibly cost-efficient solution that scales both up and down depending on real time demand.

1.2 Setting up ECS Fargate with API Gateway

The second portion of our deployment setup is to connect our containerized applications to API Gateway where our application APIs can be fully managed and secured by API Gateway.

1 of the key factors for using API Gateway is the usage of AWS Cognito for authentication and authorization of end users for each API that we deploy in ECS Fargate.

2 Setting Up Services

In this section, we cover some of the key steps required to implement the Continuous Delivery pipeline mentioned above.

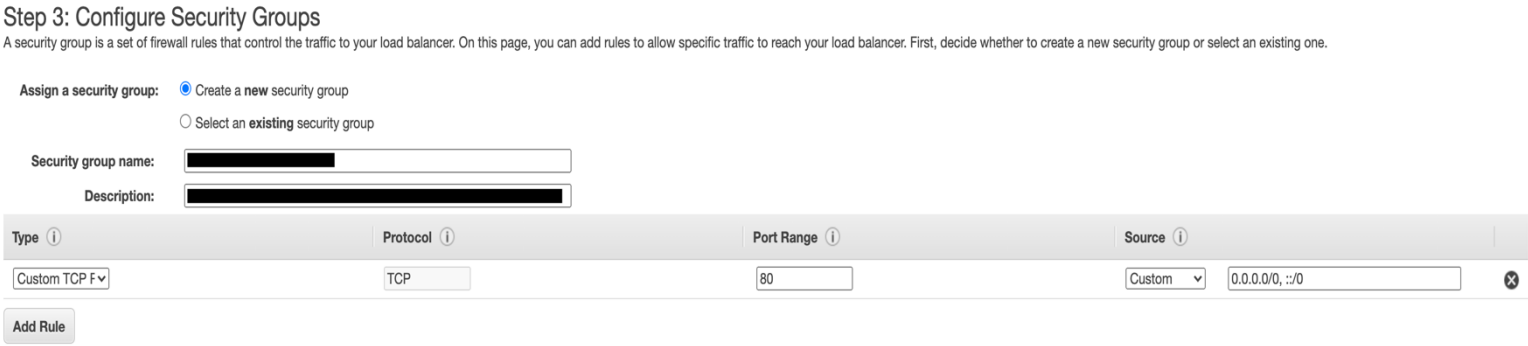
Aside from the AWS Code Pipeline setup that we have previously written in a separate article, the remaining setup on ECS Fargate and API Gateway will need to be setup in the following order:

1. Setting up the Load Balancer
2. Setting up the ECS Cluster and Target Definitions for the container
3. Integration of the Load Balancer with API Gateway

2.1 Load Balancer

Load Balancer is the single entry from cloud/user which distribute all the incoming traffic across multiple targets that are defined in the target group. You can create and define the security group and target group while creating ALB.

Security group works quite similarly to a firewall, controlling all the incoming and outgoing traffic which can define in inbound and outbound rules. The following diagram shown is simply an example and is NOT a secured way to enforce incoming traffic.



The target group tells the ALB where to route the traffic to and from while monitoring the health check response from ECS.

Step 4: Configure Routing

Your load balancer routes requests to the targets in this target group using the protocol ar

Target group

Target group	New target group
Name	[redacted]
Target type	(•) Instance ( ) IP ( ) Lambda function
Protocol	HTTP
Port	80

Health checks

Protocol	HTTP
Path	/health

EC2 > Target groups > [redacted]

[redacted]

arn:aws:elasticloadbalancing:[redacted]

Basic configuration

Target type	Protocol : Port	VPC
ip	HTTP : 80	[redacted]

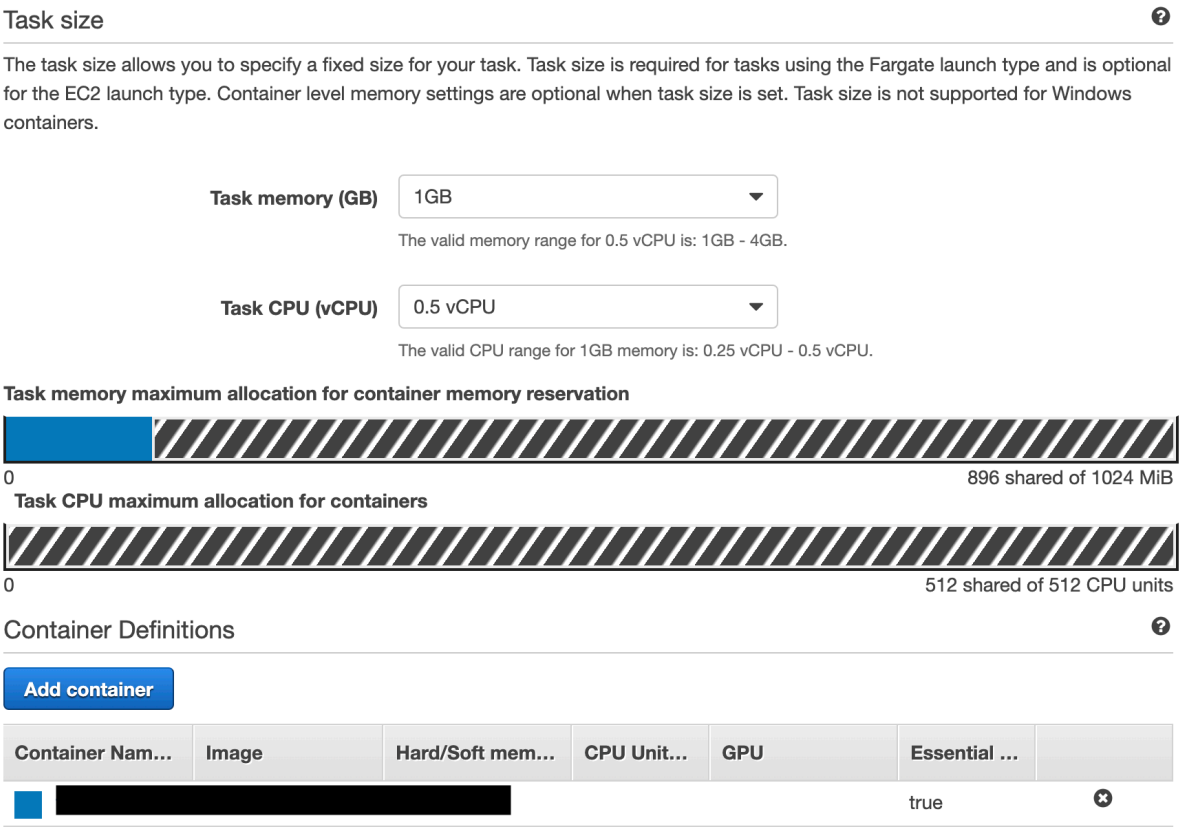
Group details | Targets | Monitoring | Tags

Health check settings

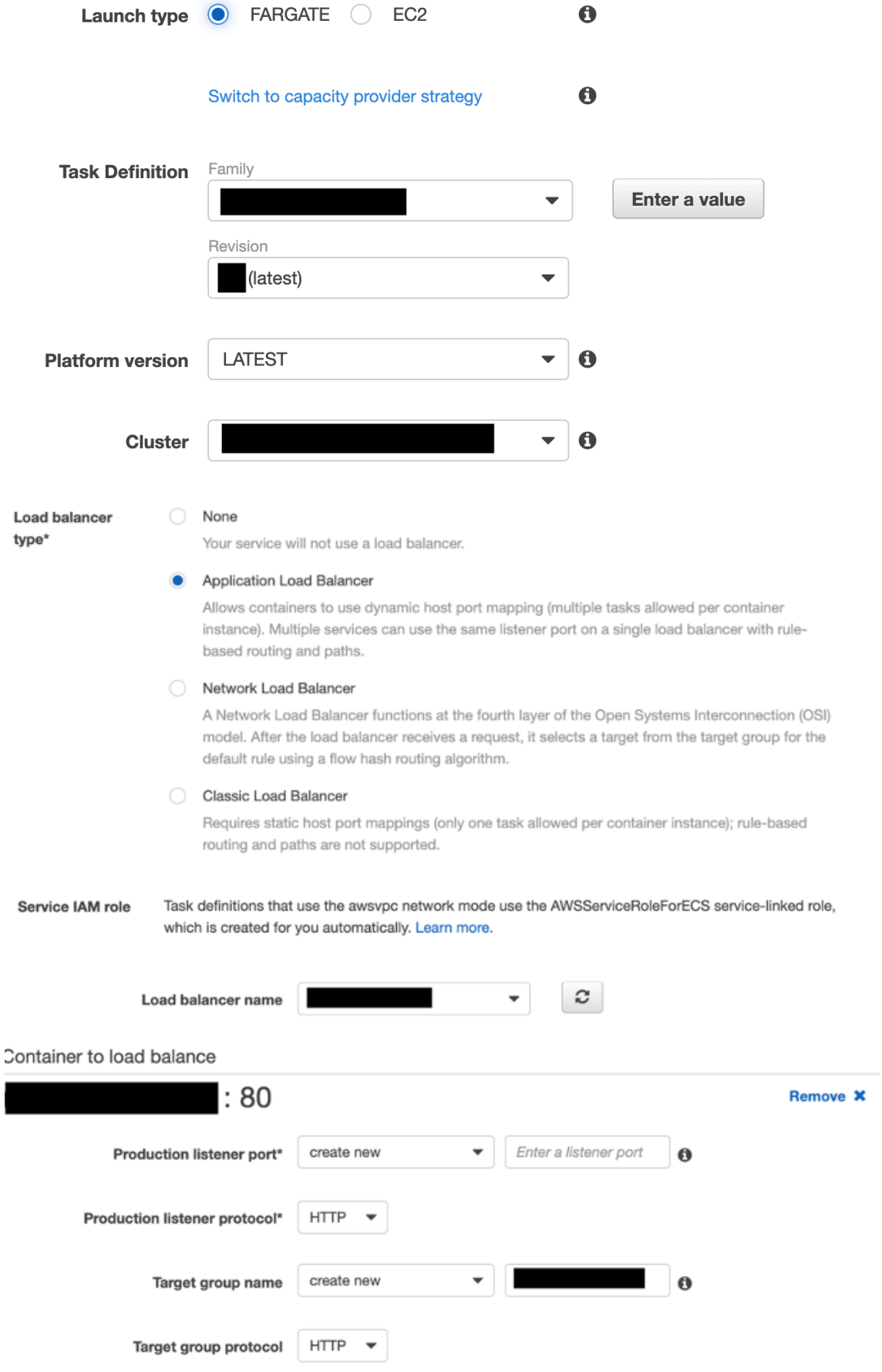
Protocol HTTP	Unhealthy threshold 2
Path [redacted]	Timeout 5
Port traffic-port	Interval 30
Healthy threshold 5	Success codes 200

2.2 ECS

Task Definitions are an important part of ECS, it describes how a docker container should launch (we choose Fargate as a launch type), and contains settings on defining how to launch the docker containers, memory and CPU usages. For the container definitions, we are simply using the latest docker image from ECR.



After creating Task Definitions, we create ECS cluster, is a group of services or tasks. Service is the long running task of the Task Definitions, where you can add the ALB and can adjust the number of running tasks depending on the requirement. At this point, the Code Pipeline Deploy stage can be updated with the Service and Cluster information. Task (Fargate Task) is the instance/running container defines in the task definition. This can be one or multiple tasks depending on the traffic and auto scaling policy defined in the service.

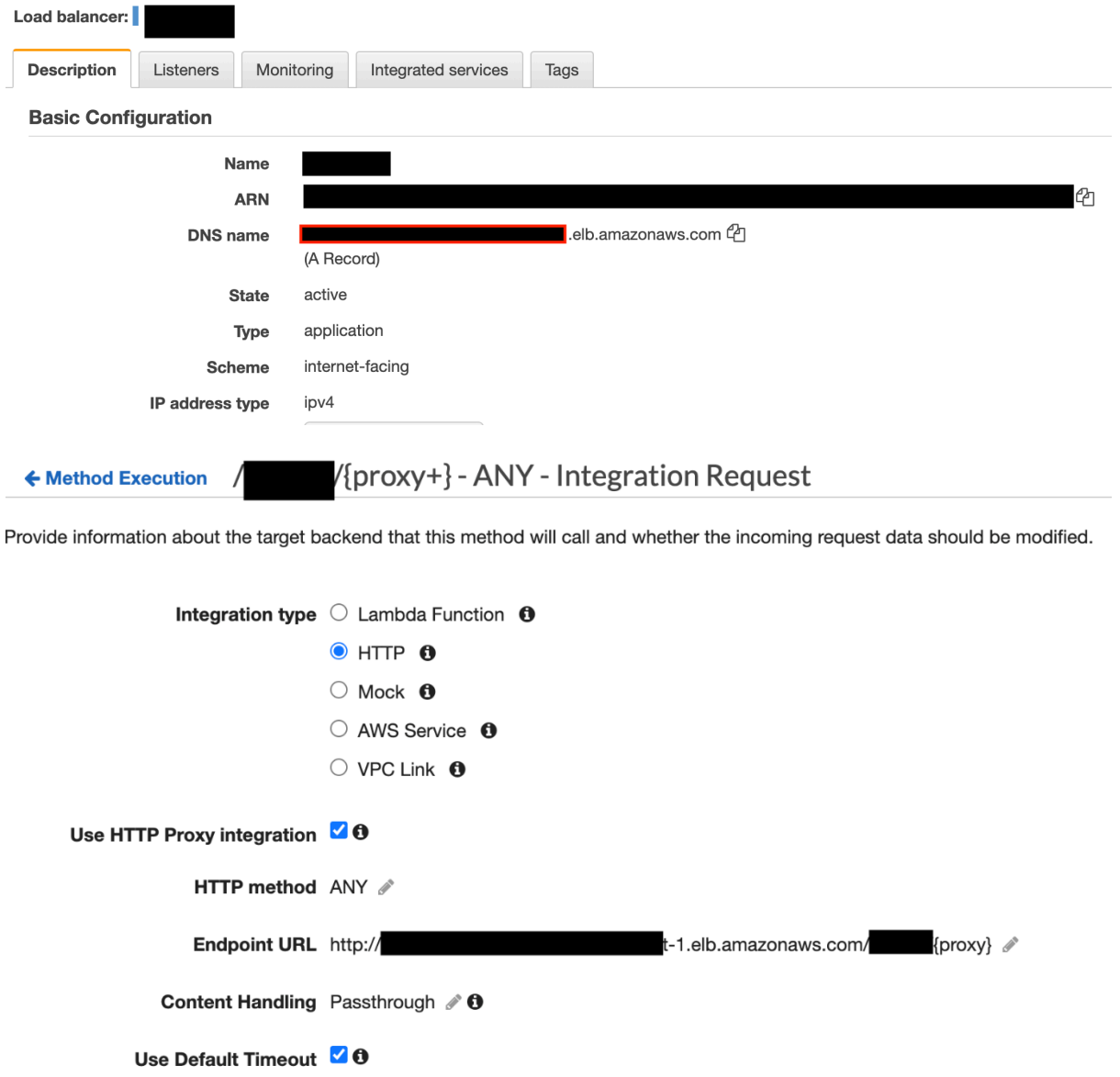

Launch type
FARGATE
EC2
Switch to capacity provider strategy
Task Definition
Family
[redacted]
Enter a value
Revision
[redacted] (latest)
Platform version
LATEST
Cluster
[redacted]
Load balancer type*
None
Your service will not use a load balancer.
Application Load Balancer
Allows containers to use dynamic host port mapping (multiple tasks allowed per container instance). Multiple services can use the same listener port on a single load balancer with rule-based routing and paths.
Network Load Balancer
A Network Load Balancer functions at the fourth layer of the Open Systems Interconnection (OSI) model. After the load balancer receives a request, it selects a target from the target group for the default rule using a flow hash routing algorithm.
Classic Load Balancer
Requires static host port mappings (only one task allowed per container instance); rule-based routing and paths are not supported.
Service IAM role
Task definitions that use the awsvpc network mode use the AWSServiceRoleForECS service-linked role, which is created for you automatically. Learn more.
Load balancer name
[redacted]
Container to load balance
[redacted] : 80
Remove
Production listener port*
create new
Enter a listener port
Production listener protocol*
HTTP
Target group name
create new
[redacted]
Target group protocol
HTTP


Minimum number of tasks
[redacted]
Automatic task scaling policies you set cannot reduce the number of tasks below this number.
Desired number of tasks
[redacted]
Maximum number of tasks
[redacted]
Automatic task scaling policies you set cannot increase the number of tasks above this number.
IAM role for Service Auto Scaling
[redacted]
Automatic task scaling policies
Scaling policy type
Target tracking
Step scaling
Policy name*
[redacted]
ECS service metric*
ECSServiceAverageCPUUtilization
Configure an ALB for the service in order to enable target tracking on ALB metrics
Target value*
[redacted]
Scale-out cooldown period
300
seconds between scaling actions
Scale-in cooldown period
300
seconds between scaling actions
Disable scale-in

2.3 API Gateway

After finished deploying the ECS and ALB, setup the load balancer DNS name in API gateway. The endpoint URL listed in the Integration Request of the API Gateway should be the DNS name of the load balancer.

Remember to deploy the API in API Gateway once ready to see the effects.



3 Conclusion

The current setup has allowed us to maintain full uptime of our applications without requiring much manual interventions since ECS Fargate ensures that the desired number of containers are deployed. Deploying upgrades to our containers also does not require downtime since ECS only decommissions the older version once the new version is up and running.

However, we do note that there are improvements to be made. The current architecture is not fully automated yet as we need to create services manually for each new application and service deployed. For this, we are looking into the use of template (CloudFormation) or configuration (Terraform) to manage and implement infrastructure as code which will be easy to maintain and also reusable.

We have barely scratched the surface of the ECS service and are looking forward to explore more features that can enable us to deliver better features to serve our clients, such as Blue green deployment to enable A/B testing of new features.