Managing Deployed Containerized Web Application on AWS Using EKS on AWS Fargate

by

Bashair Abdullah M Algarni

This capstone project submitted in partial fulfillment of the requirements for the
degree of Master of Science in Networking and System Administration

Rochester Institute of Technology

B. Thomas Golisano College of Computing and Information Sciences

Department of Information Sciences and Technologies

2021

Table of Contents

INTRODUCTION AND BACKGROUND INFORMATION 6

OVERVIEW OF CONTAINERS 6
OVERVIEW OF AWS EKS 7
OVERVIEW AWS EKS ON AWS FARGATE 8

MOTIVATIONS AND GOALS 9

PROBLEM 9
PROJECT GOAL 9
SIGNIFICANCE AND POTENTIAL BENEFITS 9

PRIOR WORK 10

METHODOLOGY 13

PHASE 1: INSTALLATION AND CONFIGURATION 13
PHASE2: IMPLANTATION AND TESTING 27

CHALLENGES 35

LIMITATIONS 35

CONCLUSION 35

FUTURE WORK 36

REFERENCES 37

Table of Figures

Figure 1: Virtual machines vs. Containers [3] 7
Figure 2: Using AWS EKS to Deploy Applications [6] 8
Figure 3: Project Architecture 13
Figure 4: AWS Account 14
Figure 5: AWS cloud9 EC2 instance information 14
Figure 6: IAM role 14
Figure 7: Check eksctl version 15
Figure 8: Check AWS CLI version 15
Figure 9: Check Kubectl version 15
Figure 10: Check docker status 15
Figure 11: AWS ECR 16
Figure 12: Authenticate to the registry 16
Figure 13 : Build first docker image with tag latest 16
Figure 14 : First Image pushed to the ECR 16
Figure 15 : Build second docker image with tag latest2 17
Figure 16: Second image pushed to ECR 17
Figure 17 : Two images information from console 17
Figure 18: Two images information from terminal 17
Figure 19: Createcluster yaml file 18
Figure 20 : EKS cluster with Fargate profile creating using eksctl command 18
Figure 21: Cloud formation stacks 19
Figure 22: Cluster resources created automatically by CloudFormation 19
Figure 23: Configure access to the EKS cluster 20
Figure 24 : IAM OIDC provider 20
Figure 25 : IAM policy 20
Figure 26 : IAM service account 21
Figure 27: Install the TargetGroupBinding CRDs 21
Figure 28: Checking CRDs 21
Figure 29: Add the EKS repository to Helm and Install the AWS Load Balancer controller using iamserviceaccount 21
Figure 30: Check the pods by using kubectl get command 21
Figure 31: Check the deployment completed 21
Figure 32: Deployment part from the deployment yaml file 22
Figure 33: Service part from depolymnt.yaml file 25
Figure 34: Service information by run get svc 25
Figure 35: Installed Metrics Server 25
Figure 36: HPA part from deployment.yaml file 26
Figure 37: Ingress yaml file 26
Figure 38: Deployment implemented 1 27
Figure 39: Check the web application Hello Tech World V1 28
Figure 40: Deployment implemented 2 28
Figure 41: Check the web application Hello Tech World V2 28
Figure 42: Rolling update preformed 29
Figure 43: Rollout status 29
Figure 44: Check pod status 29
Figure 45: Check the web application Hello Tech World V1 30
Figure 46: pod information 30
Figure 47: Testing code 31
Figure 48: Rolling update preformed 31
Figure 49: Testing code result 31
Figure 50: Check pod status 32
Figure 51: Check the web application Hello Tech World V2 after rolling update 32
Figure 52: Generating load 33

Figure 53: Describe HPA..33
Figure 53: Auto scale out With CPU 25%..33
Figure 54: 8 pods running..34
Figure 55: Auto scale in with CPU 25%...34
Figure 56: Another try with CPU target 50 % 34
Figure 57: 4 pods running..34

Abstract: One of the most critical concepts in the information technology world is cloud computing, and many organizations and companies have moved toward using it recently. These organizations need to perform updates on scalable containerized applications without affecting their availability and providing the best experience to their customers. Amazon Web Services (AWS) is one of the popular cloud computing platforms, and it offers many on-demand services to individuals, companies, and organizations. Another concept involved in this project is containerized application, and it has many benefits including, consistency, cost-effective and scalability. This project aims to demonstrate through the use of Amazon Elastic Kubernetes Service (Amazon EKS) on AWS Fargate how organizations can build, deploy and manage containerized web applications in a manner that is uncomplicated, reliable, and scalable. The Rolling update deployment strategy was performed, and Kubernetes Horizontal Pod Autoscaler (HAP) was used to automatically scale the number of pods based on CPU utilization in this project. The outcome of this project is to accomplish zero-downtime when performing a rolling update deployment.

INTRODUCTION AND BACKGROUND INFORMATION

One of the most critical concepts in IT is cloud computing. The success of cloud computing relies on on-demand services, so the users are just paying for what services they need. Measured service, broad network access, rapid elasticity, resource pooling, and on-demand self-service are the characteristics that the National Institute of Standards and Technology (NIST) used to categorize if the service is cloud service [1]. There are three models of cloud services including Infrastructure as a service (IaaS), Platform as a service (PaaS), and Software as a service (SaaS), and these service models lie on top of Private Cloud, Public Cloud, and Hybrid Cloud, which are deployment models of the cloud. Users in the Infrastructure as a service (IaaS) are provisioned with networking, storage, and compute to control and manage different operating systems and applications. Users are provided with the operating system, runtime, and middleware in the Platform as a service (PaaS) model to manage data and applications. Developers used this model widely because a customized software can be created without worry about the fundamental system, while Software as a service (SaaS) responsible for delivering applications directly to the users through a third-party provider who managed these applications [1].

For this project, Amazon web services (AWS), a public cloud, will be used because of its comprehensive services such as AWS EKS and AWS Faregate. Containerization technology like Kubernetes and Docker will be used to accomplish this project.

Overview of Containers:

An executable unit of software in which application code is packaged in popular ways along with its libraries and dependencies is the definition of the Containers. These containers can be run anywhere locally or on the cloud. Unlike virtual machines, these Containers are fast, small, and portable, and with them, there is no need to have a guest operating system in each instance.

However, containers leverage the resources and features of the host operating system. One of the biggest issues’ developers face is creating software applications that work predictably on various computers. These software applications will need to run in different environments, including production, testing, development, and staging. Container technology solves these issues where each container contains an entire runtime environment, including the application, libraries, dependencies, and configuration files [2].

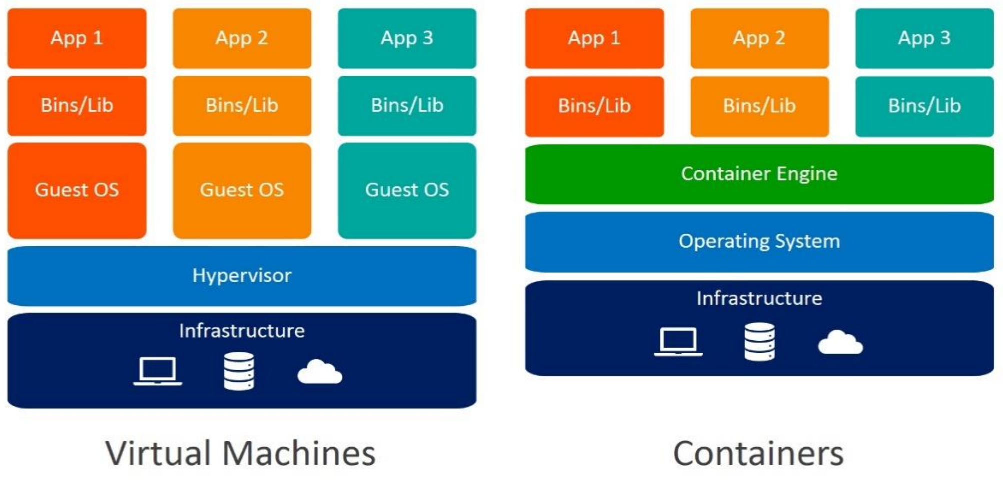


Figure 1: Virtual machines vs. Containers [3]

Overview of AWS EKS:

“Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications” [4]. Amazon Elastic Kubernetes Service (AWS EKS) is a managed service that allows users to run Kubernetes clusters on the AWS cloud. Users who use AWS EKS can manage, and provision meaning patching, updating, and securing the worker nodes where the pods run. The control panel, which is the master node, is managed, which means install, operate, and maintain by the cloud. Other services will be used on AWS besides EKS, including integration with other AWS services, such as AWS IAM for authentication, AWS ECR for container images, and AWS ALB for load distributions. Access to these services is one of the advantages of using EKS AWS [4].

Overview AWS EKS on AWS Fargate:

“AWS Fargate is a serverless compute engine for containers that work with Amazon Elastic Kubernetes Service (EKS)” [5]. In order to understand the concept of AWS Fargate, it is required to know the AWS Elastic Container Service (AWS ECS), an in-house container management solution. AWS ECS allows users to run Docker containers in clusters. These clusters are managed by AWS automatically. Like AWS EKS, this service is integrated with other AWS services such as container register AWS ECR, AWS IAM and AWS ALB. The AWS Fargate launch type is one of the AWS ECS launch types. This type allows users to use Fargate tasks to run containers in a serverless way, so users do not need to provision the instance that runs the containers. If the users want to run the entire Kubernetes cluster under the serverless compute model, they run AWS EKS on AWS Fargate. This approach allows users to run cluster master node in a managed way and use AWS Fargate profile to manage the infrastructure where pods are running. As a result, the infrastructure is completely managed when users run the Kubernetes cluster on AWS Fargate [6].

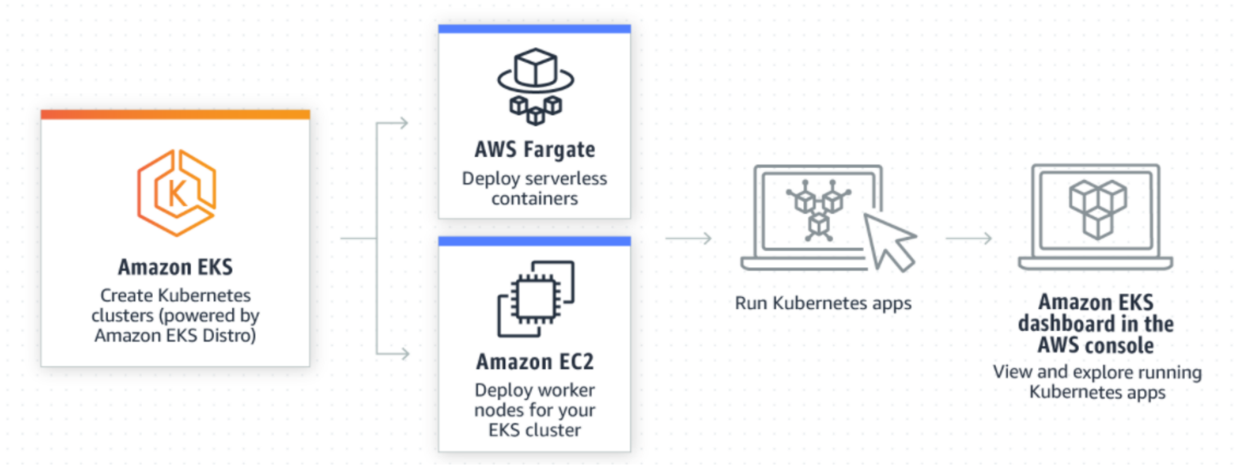

Amazon EKS
Create Kubernetes clusters (powered by Amazon EKS Distro)
AWS Fargate
Deploy serverless containers
Amazon EC2
Deploy worker nodes for your EKS cluster
Run Kubernetes apps
Amazon EKS dashboard in the AWS console
View and explore running Kubernetes apps

Figure 2: Using AWS EKS to Deploy Applications [6]

MOTIVATIONS AND GOALS

Problem

Organizations need to perform updates on scalable containerized applications without affecting their availability.

Project Goal

To demonstrate through the use of AWS EKS on AWS Fargate how organizations can build, deploy and manage containerized applications in a manner that is uncomplicated, reliable, and scalable.

Significance and Potential Benefits

There are many benefits of using AWS EKS service, including improving availability and observability, scaling and efficiently provisioning user resources, and also users can have a more secure Kubernetes environment. Across various AWS availability zones, the Kubernetes control plane is run by EKS. EKS offers on-demand upgrades and patching with zero downtime, which considers one of the important advantages for the users. In addition to that, EKS can automatically discover and replace unhealthy nodes. Users can identify and resolve problems faster because of the observability of user Kubernetes clusters provided by the EKS console. Moreover, there is no need for provisioning compute capacity separately to scale Kubernetes applications when users use EKS to manage node groups. In addition to this, users have the choice to automatically provision on-demand serverless compute for their applications by using AWS Fargate. Cost-saving for more efficiency is considered another advantage of using EKS on Fargate. Besides that, the security part is one of EKS's features because the latest security patches are automatically applied to the cluster's control plane by EKS, and AWS always works to ensure that each EKS cluster is secure. This proposed project can be used by many organizations which seek to develop their applications and provide a great experience for their users. It can be useful when there is a need to update the application without effecting its availability for users. Organizations can also use this proposed architecture if there is a need to have more than one version of the application, because of the increase demand, by scaling it out.

PRIOR WORK

Based on the paper "Deploying A Dockerized Application with Kubernetes on Google Cloud Platform", the use of containers has many benefits, including better scalability and lower overhead. The authors show in this paper that in recent years the applications that rely on microservices are becoming more popular and that because of the architecture of the applications. In addition to this, the author mentions that companies are using microservice-based architecture to develop their applications. The complexity of these applications is increased. The companies need to have many developers to manage them, which made it very challenging to implement and deploy newer application versions with various functions. There are many advantages of the microservices architecture mention in this paper because "In a microservice-based architecture, the application is structured and divided into several modules called microservices. Each microservice runs as an independent process and communicates with other microservices through synchronous protocols"[7]. Because microservices work as an independent process, they can be deployed and improved on their own. The authors believe in this paper that it is efficient to run the microservice-based application in containers. Docker container was discussed in this paper as one of the most used technologies that allow package, distribute, and run applications. Also, when the application gets large, that means there will be more microservices, and that leads to an increase in the number of containers. This number of containers needs more managing and configurations. Kubernetes is one of the open-source systems that help in this, and it also helps simplify the application deployments and allows quick increases in the number of the new versions. This paper shows that it is possible to run containers on any platform that supports Kubernetes [7].

"A Performance Evaluation of Containers running on Managed Kubernetes Services" paper shows that despite the benefits from running containerized workloads on the physical server directly, deploying containers on VMs has many advantages, mainly in the cloud because the infrastructure that is based on the cloud rely on virtualization technologies. This makes the cloud environment a proper environment to scale nodes, such as nodes on a Kubernetes cluster, efficiently. The authors in this paper agree that cloud service providers offer elasticity for the users by allowing them to use the computing resources as they need and pay for just these services. One of these services is Kubernetes services. Kubernetes on the cloud simplify deployment, management, and provision of worker nodes. Also, users do not need to maintain the complex clusters and their infrastructure by using this service. The performance of Docker containers running on Kubernetes-based cloud services is discussed in this paper. The Amazon Elastic Container Service for Kubernetes (EKS) and other Kubernetes services on other cloud platforms like Google and Microsoft are considered in this paper and how the Kubernetes is more mature and robust than other similar services such as Docker Swarm [8].

According to "Techniques to Secure Data on Cloud: Docker Swarm or Kubernetes?" paper, container technology, is being used to secure the data that upload on the cloud. A way of achieving that is discussed in this paper where a provision of a process that encrypts and decrypts the user data by launching a container for every user can help balance the load on many servers. The authors suggested improving this way by using Kubernetes. There are many reasons to choose Kubernetes over other technologies. It is a powerful tool because it can handle containers and, at the same time, provide enormous scalability and automation, which proves its efficiency. For monitoring and logging, Kubernetes contains an in-built library and process. Also, when Kubernetes is used, users do not need to use third-party applications to have these features. The authors agree that it is widely deployed, which means the node will be replaced so fast if it is failed. Kubernetes is considered a great choice to offer more efficiency and accuracy than other technologies and differs from other technologies in its outstanding performance [9].

One of the technologies that support microservices architecture smoothly and continuously is Kubernetes. Based on "Automation of Microservices Application Deployment Made Easy by Rundeck and Kubernetes", " Kubernetes can be easily integrated into the DevOps pipeline"[9]. The idea of Kubernetes focuses on self-healing, which means when the application goes down, this application is discovered immediately, and a new version of it is created. This guarantees the high availability of the application, which means it is up and running, and it is also easy to update it without bringing it down. With Kubernetes, "New pods with the new image are created while still redirecting the traffic to the previous version images. Once the new pods are up, slowly the traffic is redirected to the new version pods, and the old pods are killed at the end."[9]. Another feature of using Kubernetes is quickly scaling the number of pods, which means, in other words, replicas of the application. This paper concludes that using Kubernetes considers the most appropriate way that eases the deployment of microservices with features that meet companies' requirements like scalability and high availability for the application [10].

METHODOLOGY

In this project, many processes were followed to build an integrated environment to manage a deployed containerize application on AWS using EKS Fargate. The methodology of this project focused on rolling updates and scaling a deployed containerized web application on AWS EKS and Fargate. AWS services used in this project are EKS, Fargate, ALB, VPC, HPA, ECR, IAM, Cloud 9.

Phase 1: Installation and Configuration

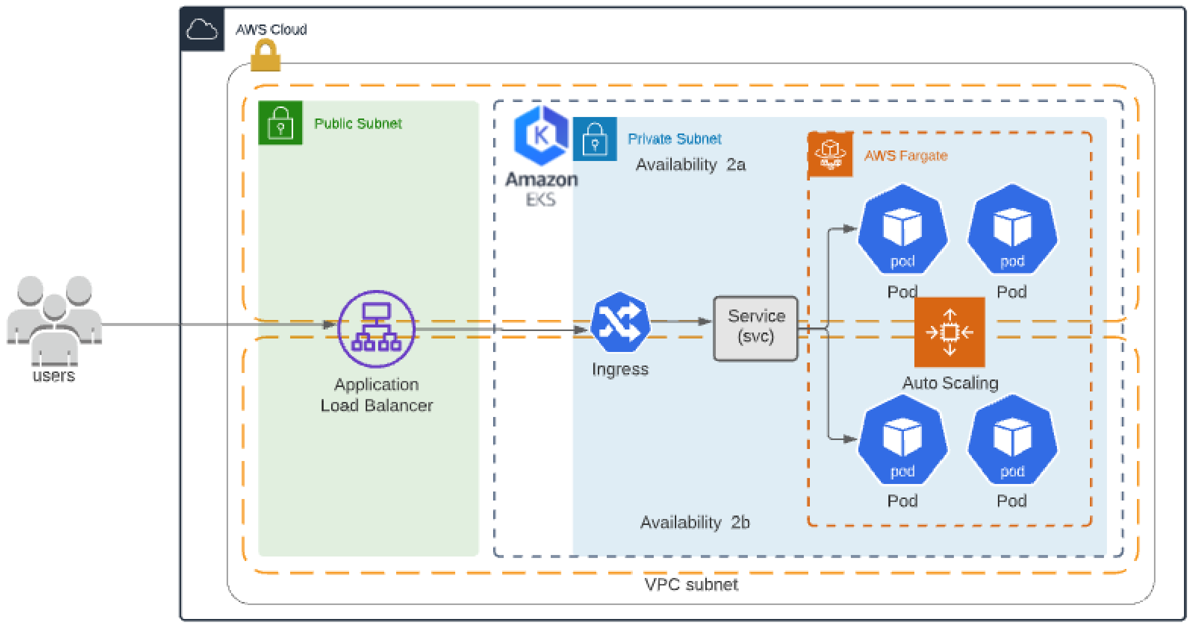
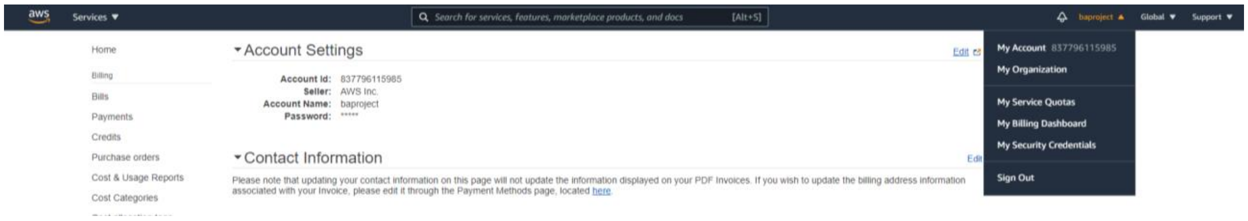


Figure 3: Project Architecture

Step1: Created an account on Amazon Web Services (AWS)

Because AWS has many features, and one of these features is pay-as-you-go, there is a need to add billing details when creating the account. This feature helps users and enterprises only pay for the services that they use.



Account Settings

Account Id:	837796115985
Seller:	AWS Inc.
Account Name:	baproject
Password:	*****

My Account 837796115985

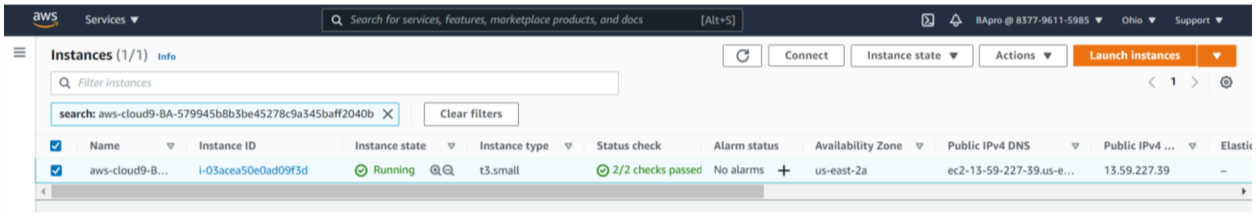
Contact Information

Please note that updating your contact information on this page will not update the information displayed on your PDF Invoices. If you wish to update the billing address information associated with your Invoice, please edit it through the Payment Methods page, located here.

Figure 4: AWS Account

Step 2: Setup Cloud 9 Environment

A workspace was created using AWS Cloud 9 service and named it as BApro. For the environment setting, creating a new EC2 instance was chosen and its type t3.small was chosen because it is recommended by AWS for small-sized web projects. Amazon Linux was chosen as the platform for this environment.



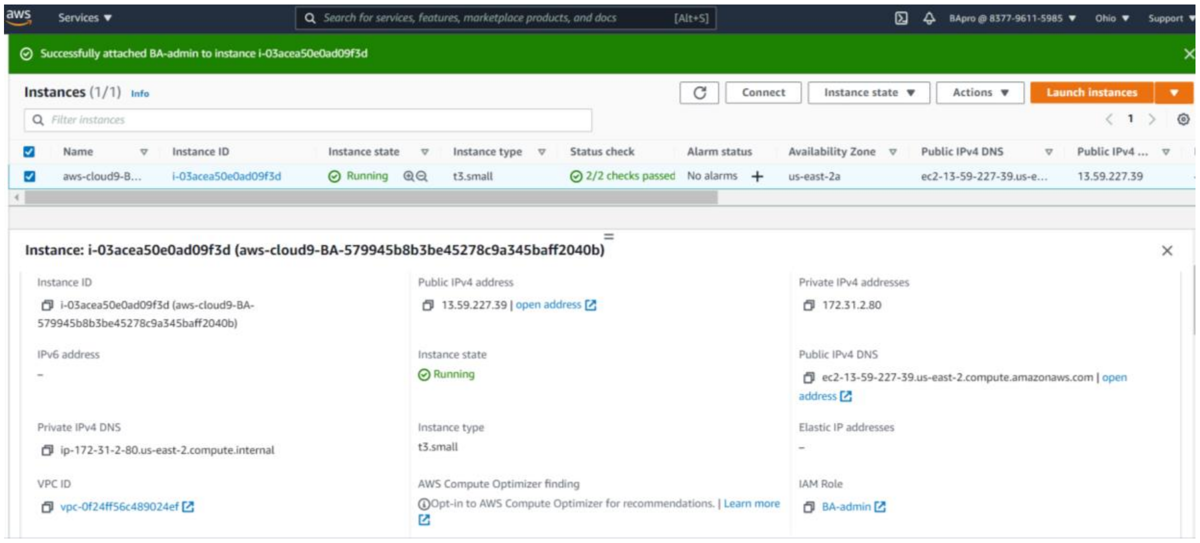
Instances (1/1)

search: aws-cloud9-BA-579945b8b3be45278c9a345baff2040b

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...
aws-cloud9-B...	i-03acea50e0ad09f3d	Running	t3.small	2/2 checks passed	No alarms	us-east-2a	ec2-13-59-227-39.us-e...	13.59.227.39

Figure 5: AWS cloud9 EC2 instance information

IAM role created for this workspace and name it as BA-admin and it attached to it.



Successfully attached BA-admin to instance i-03acea50e0ad09f3d

Instances (1/1)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...
aws-cloud9-B...	i-03acea50e0ad09f3d	Running	t3.small	2/2 checks passed	No alarms	us-east-2a	ec2-13-59-227-39.us-e...	13.59.227.39

Instance: i-03acea50e0ad09f3d (aws-cloud9-BA-579945b8b3be45278c9a345baff2040b)

Instance ID
i-03acea50e0ad09f3d (aws-cloud9-BA-579945b8b3be45278c9a345baff2040b)

Public IPv4 address
13.59.227.39 | open address

Private IPv4 addresses
172.31.2.80

IPv6 address
–

Instance state
Running

Public IPv4 DNS
ec2-13-59-227-39.us-east-2.compute.amazonaws.com | open address

Private IPv4 DNS
ip-172-31-2-80.us-east-2.compute.internal

Instance type
t3.small

Elastic IP addresses
–

VPC ID
vpc-0f24ff56c489024ef

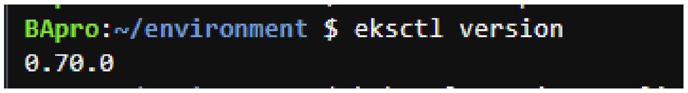
AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations. | Learn more

IAM Role
BA-admin

Figure 6: IAM role

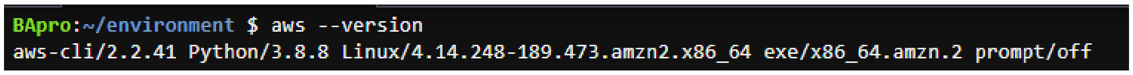
Step 3: Installed required tools

AWS CLI, Docker, kubectl, eksctl have been installed and used in this project.



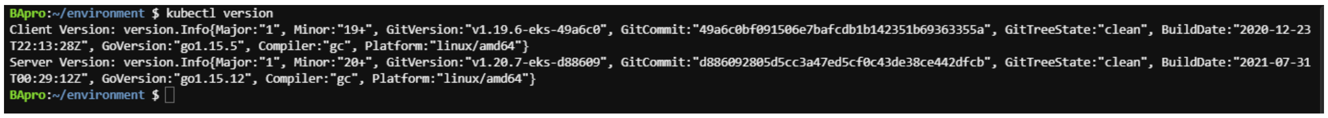
```
BApro:~/environment $ eksctl version
0.70.0
```

Figure 7: Check eksctl version



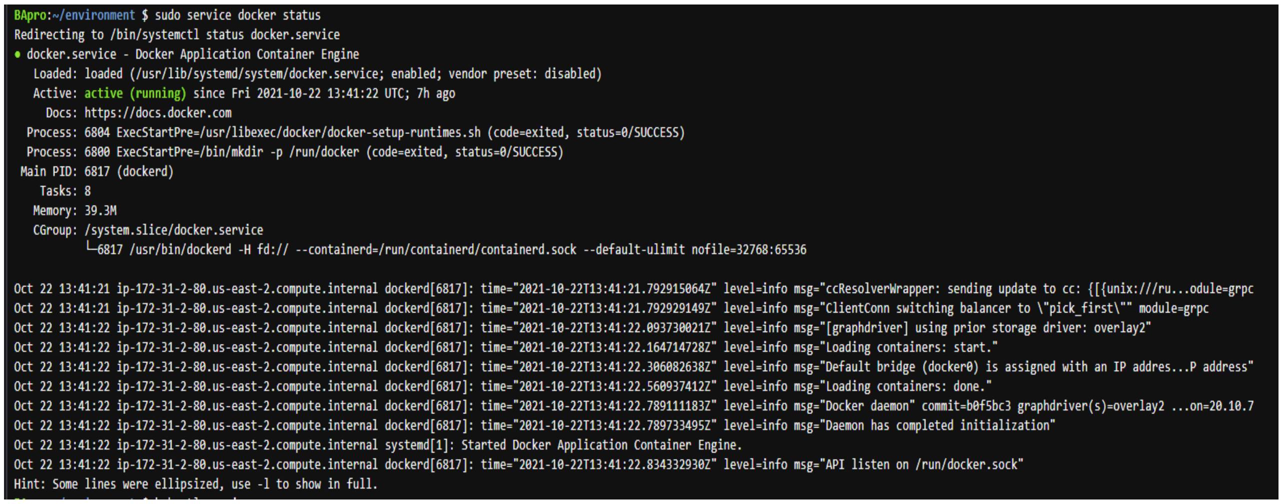
```
BApro:~/environment $ aws --version
aws-cli/2.2.41 Python/3.8.8 Linux/4.14.248-189.473.amzn2.x86_64 exe/x86_64.amzn.2 prompt/off
```

Figure 8: Check AWS CLI version



```
BApro:~/environment $ kubectl version
Client Version: version.Info{Major:"1", Minor:"19+", GitVersion:"v1.19.6-eks-49a6c0", GitCommit:"49a6c0bf091506e7bafcdb1b142351b69363355a", GitTreeState:"clean", BuildDate:"2020-12-23
T22:13:28Z", GoVersion:"go1.15.5", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"20+", GitVersion:"v1.20.7-eks-d88609", GitCommit:"d886092805d5cc3a47ed5cf0c43de38ce442dfcb", GitTreeState:"clean", BuildDate:"2021-07-31
T00:29:12Z", GoVersion:"go1.15.12", Compiler:"gc", Platform:"linux/amd64"}
BApro:~/environment $
```

Figure 9: Check Kubectl version

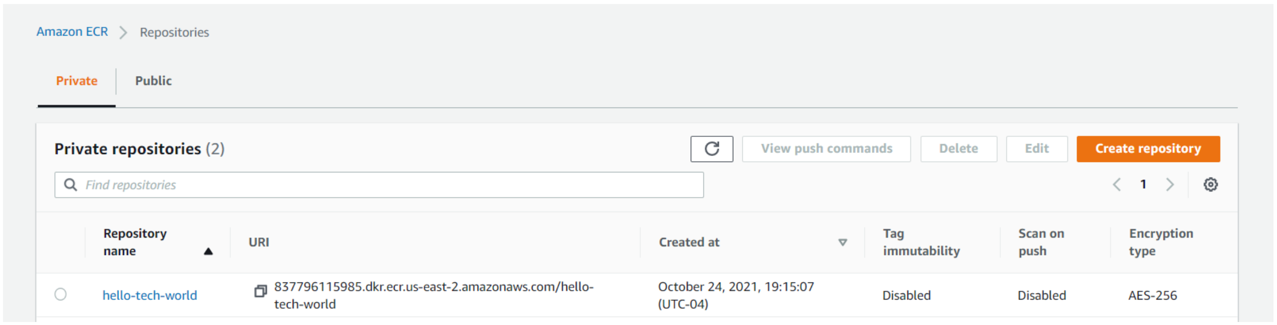


```
BApro:~/environment $ sudo service docker status
Redirecting to /bin/systemctl status docker.service
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since Fri 2021-10-22 13:41:22 UTC; 7h ago
     Docs: https://docs.docker.com
  Process: 6804 ExecStartPre=/usr/libexec/docker/docker-setup-runtimes.sh (code=exited, status=0/SUCCESS)
  Process: 6800 ExecStartPre=/bin/mkdir -p /run/docker (code=exited, status=0/SUCCESS)
 Main PID: 6817 (dockerd)
    Tasks: 8
   Memory: 39.3M
   CGroup: /system.slice/docker.service
           └─6817 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-ulimit nofile=32768:65536

Oct 22 13:41:21 ip-172-31-2-80.us-east-2.compute.internal dockerd[6817]: time="2021-10-22T13:41:21.792915064Z" level=info msg="ccResolverWrapper: sending update to cc: {[{unix:///ru...odule=grpc
Oct 22 13:41:21 ip-172-31-2-80.us-east-2.compute.internal dockerd[6817]: time="2021-10-22T13:41:21.792929149Z" level=info msg="ClientConn switching balancer to \"pick_first\"" module=grpc
Oct 22 13:41:22 ip-172-31-2-80.us-east-2.compute.internal dockerd[6817]: time="2021-10-22T13:41:22.093730021Z" level=info msg="[graphdriver] using prior storage driver: overlay2"
Oct 22 13:41:22 ip-172-31-2-80.us-east-2.compute.internal dockerd[6817]: time="2021-10-22T13:41:22.164714728Z" level=info msg="Loading containers: start."
Oct 22 13:41:22 ip-172-31-2-80.us-east-2.compute.internal dockerd[6817]: time="2021-10-22T13:41:22.306082638Z" level=info msg="Default bridge (docker0) is assigned with an IP addres...P address"
Oct 22 13:41:22 ip-172-31-2-80.us-east-2.compute.internal dockerd[6817]: time="2021-10-22T13:41:22.560937412Z" level=info msg="Loading containers: done."
Oct 22 13:41:22 ip-172-31-2-80.us-east-2.compute.internal dockerd[6817]: time="2021-10-22T13:41:22.789111183Z" level=info msg="Docker daemon" commit=b0f5bc3 graphdriver(s)=overlay2 ...on=20.10.7
Oct 22 13:41:22 ip-172-31-2-80.us-east-2.compute.internal dockerd[6817]: time="2021-10-22T13:41:22.789733495Z" level=info msg="Daemon has completed initialization"
Oct 22 13:41:22 ip-172-31-2-80.us-east-2.compute.internal systemd[1]: Started Docker Application Container Engine.
Oct 22 13:41:22 ip-172-31-2-80.us-east-2.compute.internal dockerd[6817]: time="2021-10-22T13:41:22.834332930Z" level=info msg="API listen on /run/docker.sock"
Hint: Some lines were ellipsized, use -l to show in full.
```

Figure 10: Check docker status

Step 4: Created AWS ECR repository to push docker images

Amazon Elastic Container Registry (ECR) created with name hello-tech-world. A simple web application have used [11] and edited it by added text Hello Tech World V1 and V2. Used Docker to build the two images and then push them to the ECR.

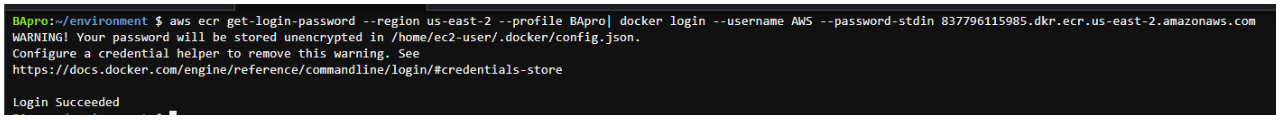


Private repositories (2)

Repository name	URI	Created at	Tag immutability	Scan on push	Encryption type
hello-tech-world	837796115985.dkr.ecr.us-east-2.amazonaws.com/hello-tech-world	October 24, 2021, 19:15:07 (UTC-04)	Disabled	Disabled	AES-256

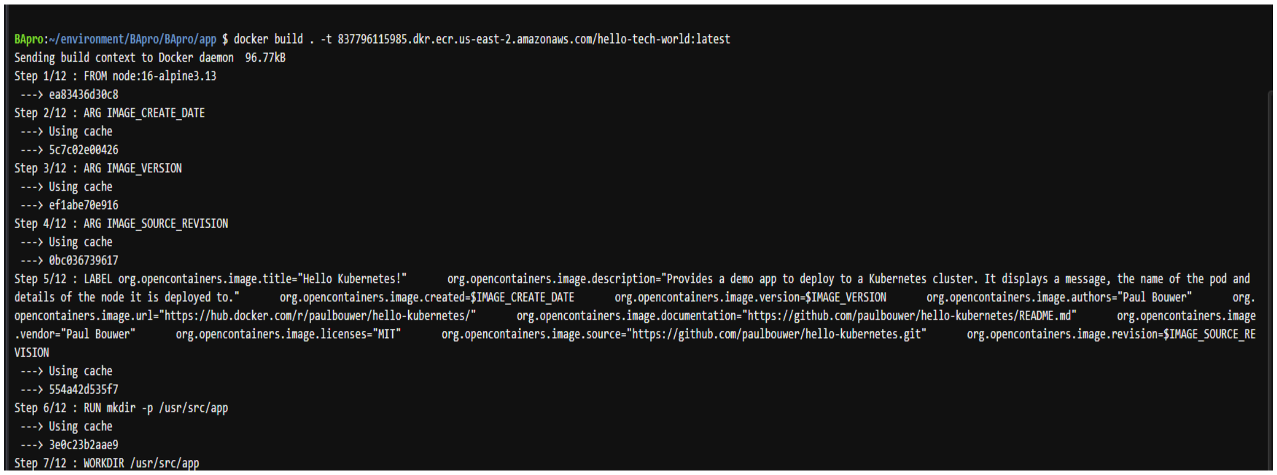
Figure 11: AWS ECR

It is important and required to authenticate the Docker CLI to the registry so docker command can push and pull images. [12]



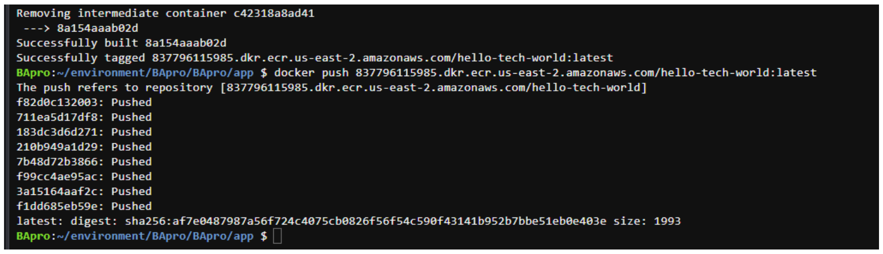
```
BApro:~/environment $ aws ecr get-login-password --region us-east-2 --profile BApro| docker login --username AWS --password-stdin 837796115985.dkr.ecr.us-east-2.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

Figure 12: Authenticate to the registry



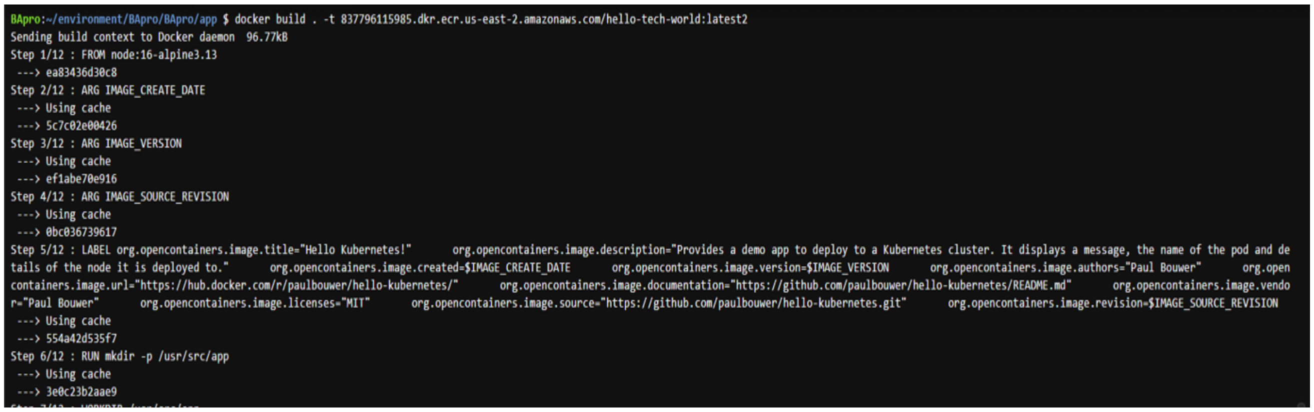
```
BApro:~/environment/BApro/BApro/app $ docker build . -t 837796115985.dkr.ecr.us-east-2.amazonaws.com/hello-tech-world:latest
Sending build context to Docker daemon  96.77kB
Step 1/12 : FROM node:16-alpine3.13
 ---> ea83436d30c8
Step 2/12 : ARG IMAGE_CREATE_DATE
 ---> Using cache
 ---> 5c7c02e00426
Step 3/12 : ARG IMAGE_VERSION
 ---> Using cache
 ---> ef1abe70e916
Step 4/12 : ARG IMAGE_SOURCE_REVISION
 ---> Using cache
 ---> 0bc036739617
Step 5/12 : LABEL org.opencontainers.image.title="Hello Kubernetes!"       org.opencontainers.image.description="Provides a demo app to deploy to a Kubernetes cluster. It displays a message, the name of the pod and
details of the node it is deployed to."       org.opencontainers.image.created=$IMAGE_CREATE_DATE       org.opencontainers.image.version=$IMAGE_VERSION       org.opencontainers.image.authors="Paul Bouwer"       org.
opencontainers.image.url="https://hub.docker.com/r/paulbouwer/hello-kubernetes/"       org.opencontainers.image.documentation="https://github.com/paulbouwer/hello-kubernetes/README.md"       org.opencontainers.image
.vendor="Paul Bouwer"       org.opencontainers.image.licenses="MIT"       org.opencontainers.image.source="https://github.com/paulbouwer/hello-kubernetes.git"       org.opencontainers.image.revision=$IMAGE_SOURCE_RE
VISION
 ---> Using cache
 ---> 554a42d535f7
Step 6/12 : RUN mkdir -p /usr/src/app
 ---> Using cache
 ---> 3e0c23b2aae9
Step 7/12 : WORKDIR /usr/src/app
```

Figure 13 : Build first docker image with tag latest



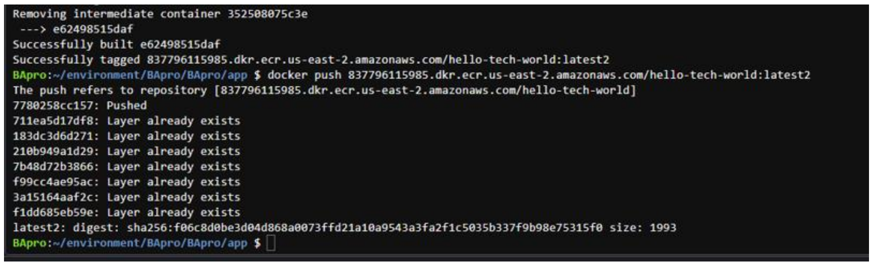
```
Removing intermediate container c42318a8ad41
 ---> 8a154aaab02d
Successfully built 8a154aaab02d
Successfully tagged 837796115985.dkr.ecr.us-east-2.amazonaws.com/hello-tech-world:latest
BApro:~/environment/BApro/BApro/app $ docker push 837796115985.dkr.ecr.us-east-2.amazonaws.com/hello-tech-world:latest
The push refers to repository [837796115985.dkr.ecr.us-east-2.amazonaws.com/hello-tech-world]
f82d0c132003: Pushed
711ea5d17df8: Pushed
183dc3d6d271: Pushed
210b949a1d29: Pushed
7b48d72b3866: Pushed
f99cc4ae95ac: Pushed
3a15164aaf2c: Pushed
f1dd685eb59e: Pushed
latest: digest: sha256:af7e0487987a56f724c4075cb0826f56f54c590f43141b952b7bbe51eb0e403e size: 1993
BApro:~/environment/BApro/BApro/app $
```

Figure 14 : First Image pushed to the ECR



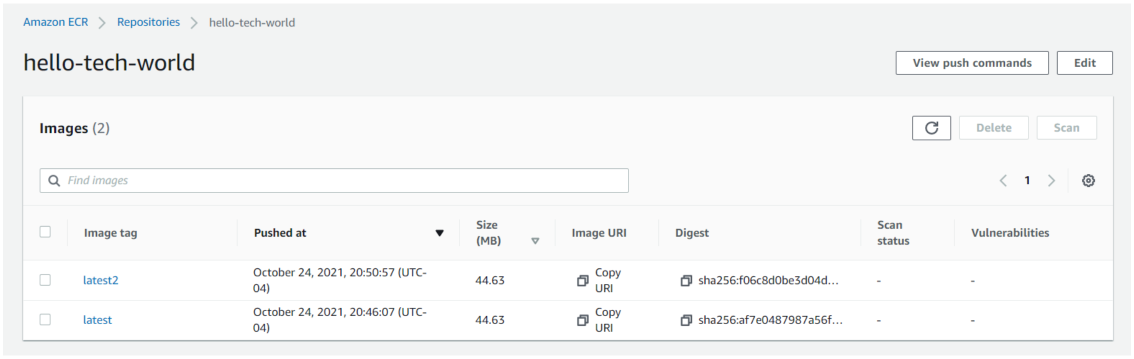
```
BApro:~/environment/BApro/BApro/app $ docker build . -t 837796115985.dkr.ecr.us-east-2.amazonaws.com/hello-tech-world:latest2
Sending build context to Docker daemon  96.77kB
Step 1/12 : FROM node:16-alpine3.13
 ---> ea83436d30c8
Step 2/12 : ARG IMAGE_CREATE_DATE
 ---> Using cache
 ---> 5c7c02e00426
Step 3/12 : ARG IMAGE_VERSION
 ---> Using cache
 ---> ef1abe70e916
Step 4/12 : ARG IMAGE_SOURCE_REVISION
 ---> Using cache
 ---> 0bc036739617
Step 5/12 : LABEL org.opencontainers.image.title="Hello Kubernetes!"       org.opencontainers.image.description="Provides a demo app to deploy to a Kubernetes cluster. It displays a message, the name of the pod and de
tails of the node it is deployed to."       org.opencontainers.image.created=$IMAGE_CREATE_DATE       org.opencontainers.image.version=$IMAGE_VERSION       org.opencontainers.image.authors="Paul Bouwer"       org.open
containers.image.url="https://hub.docker.com/r/paulbouwer/hello-kubernetes/"       org.opencontainers.image.documentation="https://github.com/paulbouwer/hello-kubernetes/README.md"       org.opencontainers.image.vendo
r="Paul Bouwer"       org.opencontainers.image.licenses="MIT"       org.opencontainers.image.source="https://github.com/paulbouwer/hello-kubernetes.git"       org.opencontainers.image.revision=$IMAGE_SOURCE_REVISION
 ---> Using cache
 ---> 554a42d535f7
Step 6/12 : RUN mkdir -p /usr/src/app
 ---> Using cache
 ---> 3e0c23b2aae9
```

Figure 15 : Build second docker image with tag latest2



```
Removing intermediate container 352508075c3e
 ---> e62498515daf
Successfully built e62498515daf
Successfully tagged 837796115985.dkr.ecr.us-east-2.amazonaws.com/hello-tech-world:latest2
BApro:~/environment/BApro/BApro/app $ docker push 837796115985.dkr.ecr.us-east-2.amazonaws.com/hello-tech-world:latest2
The push refers to repository [837796115985.dkr.ecr.us-east-2.amazonaws.com/hello-tech-world]
7780258cc157: Pushed
711ea5d17df8: Layer already exists
183dc3d6d271: Layer already exists
210b949a1d29: Layer already exists
7b48d72b3866: Layer already exists
f99cc4ae95ac: Layer already exists
3a15164aaf2c: Layer already exists
f1dd685eb59e: Layer already exists
latest2: digest: sha256:f06c8d0be3d04d868a0073ffd21a10a9543a3fa2f1c5035b337f9b98e75315f0 size: 1993
BApro:~/environment/BApro/BApro/app $
```

Figure 16: Second image pushed to ECR

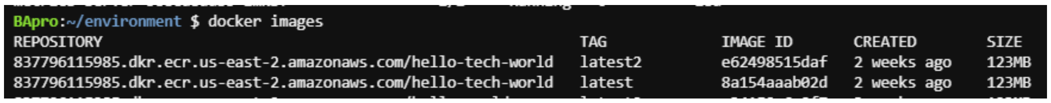


hello-tech-world

Images (2)

	Image tag	Pushed at	Size (MB)	Image URI	Digest	Scan status	Vulnerabilities
	latest2	October 24, 2021, 20:50:57 (UTC-04)	44.63	Copy URI	sha256:f06c8d0be3d04d...	-	-
	latest	October 24, 2021, 20:46:07 (UTC-04)	44.63	Copy URI	sha256:af7e0487987a56f...	-	-

Figure 17 : Two images information from console



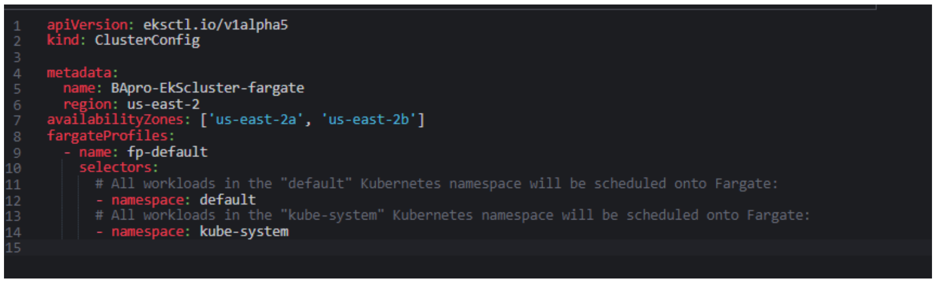
```
BApro:~/environment $ docker images
REPOSITORY                                                TAG       IMAGE ID       CREATED       SIZE
837796115985.dkr.ecr.us-east-2.amazonaws.com/hello-tech-world   latest2   e62498515daf   2 weeks ago   123MB
837796115985.dkr.ecr.us-east-2.amazonaws.com/hello-tech-world   latest    8a154aaab02d   2 weeks ago   123MB
```

Figure 18: Two images information from terminal

Step 5: Created EKS Cluster with Fargate profile

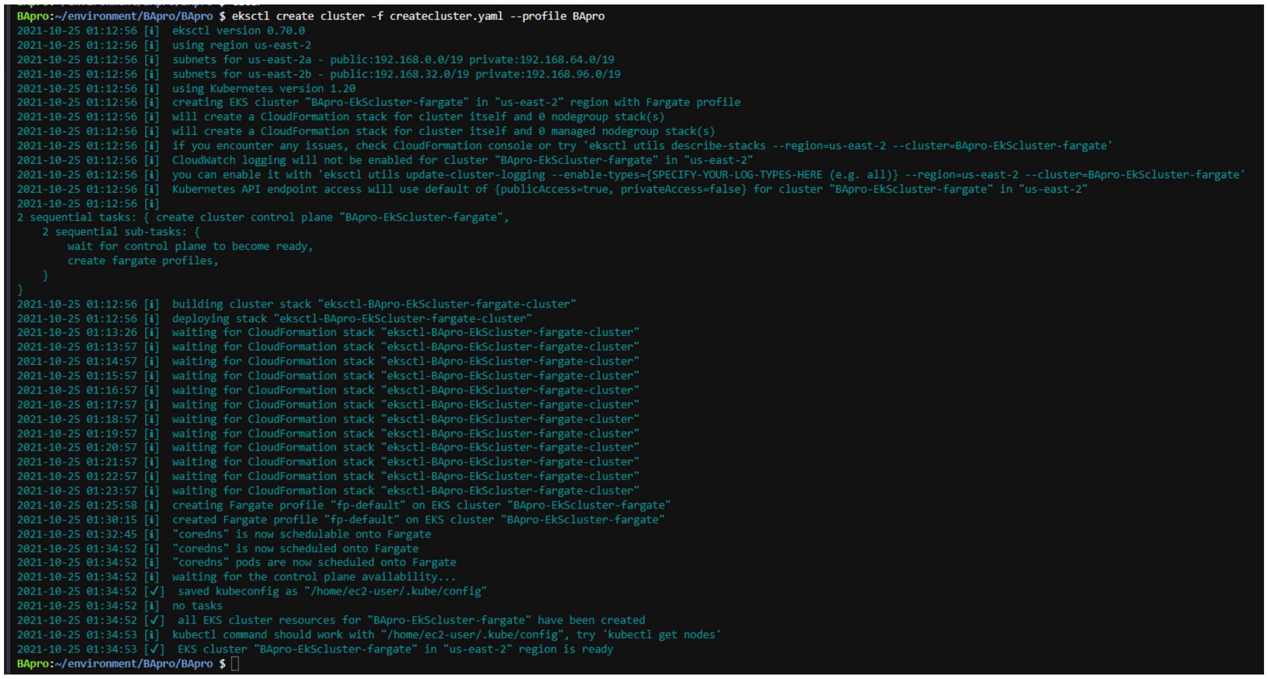
A YAML file written to configured EKS Cluster with Fargate profile and used eksctl command to create EKS cluster. Eksctl tool took care of creating the cloud formation stacks and all the resources.

The idea of having a Fargate profile is allowing the user to know pods that are running on Fargate. There are up to five selectors in each profile, and each selector should have a namespace and optional labels. Pods that match selectors in the namespace and labels are scheduled on Fargate. Using the namespace to deploy application workloads provided the user more abilities to manage the interaction between the pods that deployed on to EKS [13].



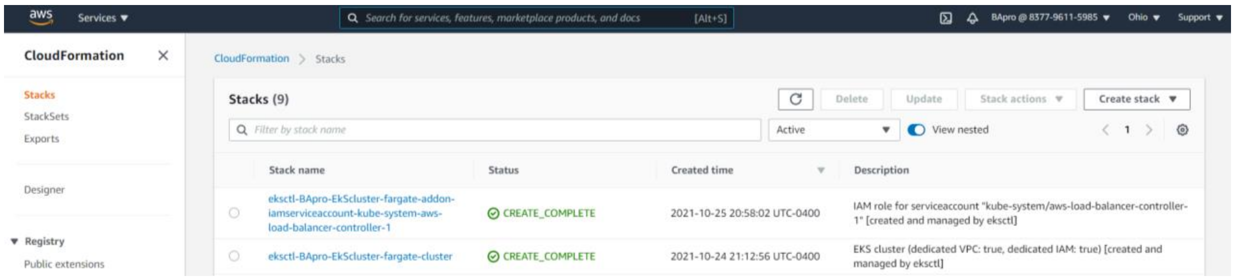
```yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: BApro-EkScluster-fargate
  region: us-east-2
availabilityZones: ['us-east-2a', 'us-east-2b']
fargateProfiles:
  - name: fp-default
    selectors:
      # All workloads in the "default" Kubernetes namespace will be scheduled onto Fargate:
      - namespace: default
      # All workloads in the "kube-system" Kubernetes namespace will be scheduled onto Fargate:
      - namespace: kube-system

```

Figure 19: Createcluster yaml file



```
BApro:~/environment/BApro/BApro $ eksctl create cluster -f createcluster.yaml --profile BApro
2021-10-25 01:12:56 [i]  eksctl version 0.70.0
2021-10-25 01:12:56 [i]  using region us-east-2
2021-10-25 01:12:56 [i]  subnets for us-east-2a - public:192.168.0.0/19 private:192.168.64.0/19
2021-10-25 01:12:56 [i]  subnets for us-east-2b - public:192.168.32.0/19 private:192.168.96.0/19
2021-10-25 01:12:56 [i]  using Kubernetes version 1.20
2021-10-25 01:12:56 [i]  creating EKS cluster "BApro-EkScluster-fargate" in "us-east-2" region with Fargate profile
2021-10-25 01:12:56 [i]  will create a CloudFormation stack for cluster itself and 0 nodegroup stack(s)
2021-10-25 01:12:56 [i]  will create a CloudFormation stack for cluster itself and 0 managed nodegroup stack(s)
2021-10-25 01:12:56 [i]  if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=us-east-2 --cluster=BApro-EkScluster-fargate'
2021-10-25 01:12:56 [i]  CloudWatch logging will not be enabled for cluster "BApro-EkScluster-fargate" in "us-east-2"
2021-10-25 01:12:56 [i]  you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=us-east-2 --cluster=BApro-EkScluster-fargate'
2021-10-25 01:12:56 [i]  Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "BApro-EkScluster-fargate" in "us-east-2"
2021-10-25 01:12:56 [i]
2 sequential tasks: { create cluster control plane "BApro-EkScluster-fargate",
    2 sequential sub-tasks: {
        wait for control plane to become ready,
        create fargate profiles,
    }
}
2021-10-25 01:12:56 [i]  building cluster stack "eksctl-BApro-EkScluster-fargate-cluster"
2021-10-25 01:12:56 [i]  deploying stack "eksctl-BApro-EkScluster-fargate-cluster"
2021-10-25 01:13:26 [i]  waiting for CloudFormation stack "eksctl-BApro-EkScluster-fargate-cluster"
2021-10-25 01:13:57 [i]  waiting for CloudFormation stack "eksctl-BApro-EkScluster-fargate-cluster"
2021-10-25 01:14:57 [i]  waiting for CloudFormation stack "eksctl-BApro-EkScluster-fargate-cluster"
2021-10-25 01:15:57 [i]  waiting for CloudFormation stack "eksctl-BApro-EkScluster-fargate-cluster"
2021-10-25 01:16:57 [i]  waiting for CloudFormation stack "eksctl-BApro-EkScluster-fargate-cluster"
2021-10-25 01:17:57 [i]  waiting for CloudFormation stack "eksctl-BApro-EkScluster-fargate-cluster"
2021-10-25 01:18:57 [i]  waiting for CloudFormation stack "eksctl-BApro-EkScluster-fargate-cluster"
2021-10-25 01:19:57 [i]  waiting for CloudFormation stack "eksctl-BApro-EkScluster-fargate-cluster"
2021-10-25 01:20:57 [i]  waiting for CloudFormation stack "eksctl-BApro-EkScluster-fargate-cluster"
2021-10-25 01:21:57 [i]  waiting for CloudFormation stack "eksctl-BApro-EkScluster-fargate-cluster"
2021-10-25 01:22:57 [i]  waiting for CloudFormation stack "eksctl-BApro-EkScluster-fargate-cluster"
2021-10-25 01:23:57 [i]  waiting for CloudFormation stack "eksctl-BApro-EkScluster-fargate-cluster"
2021-10-25 01:25:58 [i]  creating Fargate profile "fp-default" on EKS cluster "BApro-EkScluster-fargate"
2021-10-25 01:30:15 [i]  created Fargate profile "fp-default" on EKS cluster "BApro-EkScluster-fargate"
2021-10-25 01:32:45 [i]  "coredns" is now schedulable onto Fargate
2021-10-25 01:34:52 [i]  "coredns" is now scheduled onto Fargate
2021-10-25 01:34:52 [i]  "coredns" pods are now scheduled onto Fargate
2021-10-25 01:34:52 [i]  waiting for the control plane availability...
2021-10-25 01:34:52 [✓]  saved kubeconfig as "/home/ec2-user/.kube/config"
2021-10-25 01:34:52 [i]  no tasks
2021-10-25 01:34:52 [✓]  all EKS cluster resources for "BApro-EkScluster-fargate" have been created
2021-10-25 01:34:53 [i]  kubectl command should work with "/home/ec2-user/.kube/config", try 'kubectl get nodes'
2021-10-25 01:34:53 [✓]  EKS cluster "BApro-EkScluster-fargate" in "us-east-2" region is ready
BApro:~/environment/BApro/BApro $
```

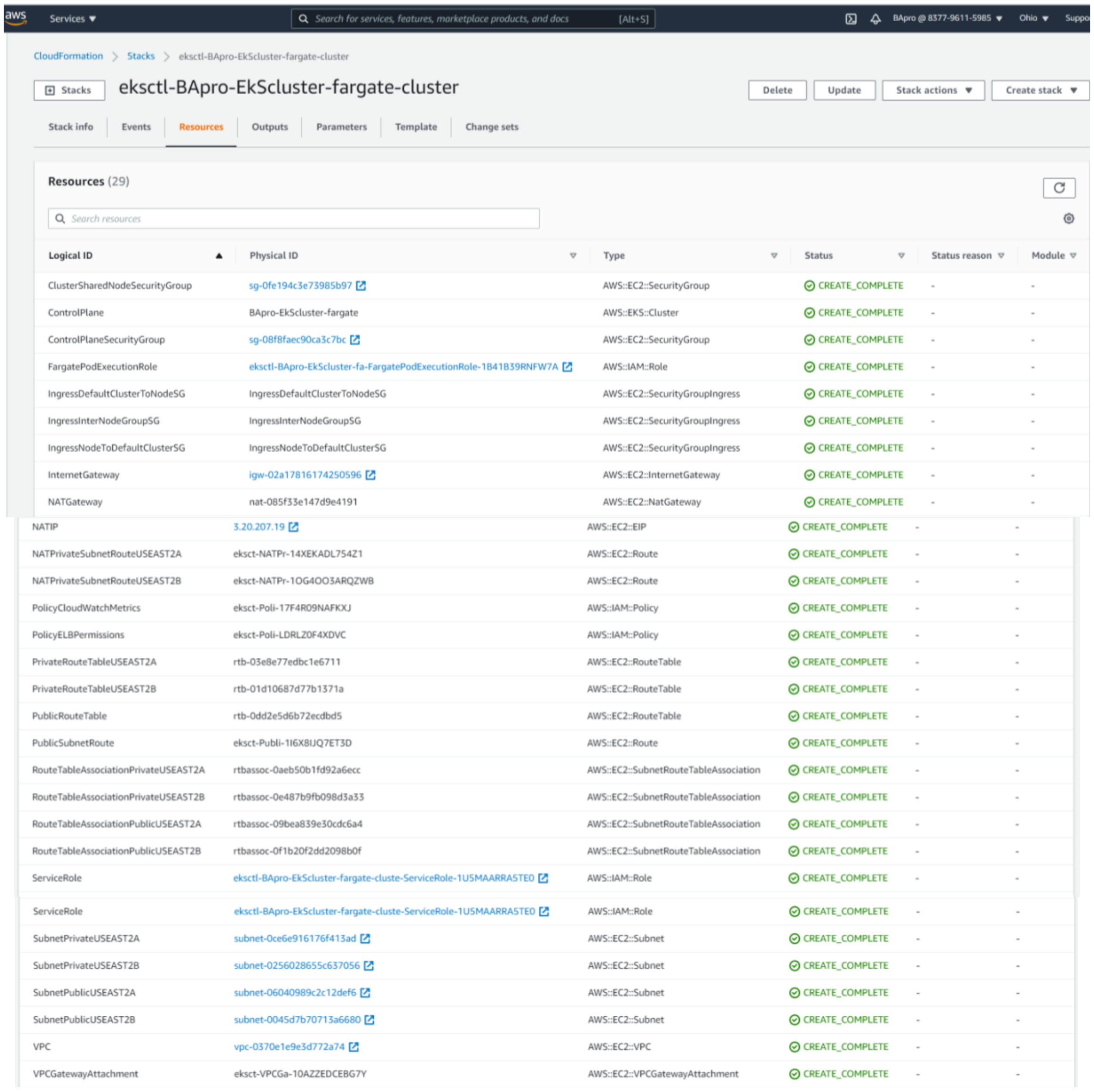
Figure 20: EKS cluster with Fargate profile creating using eksctl command



Stacks (9)

Stack name	Status	Created time	Description
eksctl-BApro-EkScluster-fargate-addon-iamserviceaccount-kube-system-aws-load-balancer-controller-1	CREATE_COMPLETE	2021-10-25 20:58:02 UTC-0400	IAM role for serviceaccount "kube-system/aws-load-balancer-controller-1" [created and managed by eksctl]
eksctl-BApro-EkScluster-fargate-cluster	CREATE_COMPLETE	2021-10-24 21:12:56 UTC-0400	EKS cluster (dedicated VPC: true, dedicated IAM: true) [created and managed by eksctl]

Figure 21: Cloud formation stacks



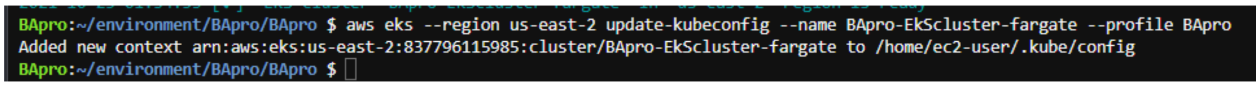
eksctl-BApro-EkScluster-fargate-cluster

Resources (29)

Logical ID	Physical ID	Type	Status	Status reason	Module
ClusterSharedNodeSecurityGroup	sg-0fe194c3e73985b97	AWS::EC2::SecurityGroup	CREATE_COMPLETE	-	-
ControlPlane	BApro-EkScluster-fargate	AWS::EKS::Cluster	CREATE_COMPLETE	-	-
ControlPlaneSecurityGroup	sg-08f8faec90ca3c7bc	AWS::EC2::SecurityGroup	CREATE_COMPLETE	-	-
FargatePodExecutionRole	eksctl-BApro-EkScluster-fa-FargatePodExecutionRole-1B41B39RNFW7A	AWS::IAM::Role	CREATE_COMPLETE	-	-
IngressDefaultClusterToNodeSG	IngressDefaultClusterToNodeSG	AWS::EC2::SecurityGroupIngress	CREATE_COMPLETE	-	-
IngressInterNodeGroupSG	IngressInterNodeGroupSG	AWS::EC2::SecurityGroupIngress	CREATE_COMPLETE	-	-
IngressNodeToDefaultClusterSG	IngressNodeToDefaultClusterSG	AWS::EC2::SecurityGroupIngress	CREATE_COMPLETE	-	-
InternetGateway	igw-02a17816174250596	AWS::EC2::InternetGateway	CREATE_COMPLETE	-	-
NATGateway	nat-085f33e147d9e4191	AWS::EC2::NatGateway	CREATE_COMPLETE	-	-
NATIP	3.20.207.19	AWS::EC2::EIP	CREATE_COMPLETE	-	-
NATPrivateSubnetRouteUSEAST2A	eksct-NATPr-14XEKADL754Z1	AWS::EC2::Route	CREATE_COMPLETE	-	-
NATPrivateSubnetRouteUSEAST2B	eksct-NATPr-1OG4OO3ARQZWB	AWS::EC2::Route	CREATE_COMPLETE	-	-
PolicyCloudWatchMetrics	eksct-Poli-17F4R09NAFKXJ	AWS::IAM::Policy	CREATE_COMPLETE	-	-
PolicyELBPermissions	eksct-Poli-LDRLZ0F4XDVC	AWS::IAM::Policy	CREATE_COMPLETE	-	-
PrivateRouteTableUSEAST2A	rtb-03e8e77edbc1e6711	AWS::EC2::RouteTable	CREATE_COMPLETE	-	-
PrivateRouteTableUSEAST2B	rtb-01d10687d77b1371a	AWS::EC2::RouteTable	CREATE_COMPLETE	-	-
PublicRouteTable	rtb-0dd2e5d6b72ecdbd5	AWS::EC2::RouteTable	CREATE_COMPLETE	-	-
PublicSubnetRoute	eksct-Publi-1I6X8IJQ7ET3D	AWS::EC2::Route	CREATE_COMPLETE	-	-
RouteTableAssociationPrivateUSEAST2A	rtbassoc-0aeb50b1fd92a6ecc	AWS::EC2::SubnetRouteTableAssociation	CREATE_COMPLETE	-	-
RouteTableAssociationPrivateUSEAST2B	rtbassoc-0e487b9fb098d3a33	AWS::EC2::SubnetRouteTableAssociation	CREATE_COMPLETE	-	-
RouteTableAssociationPublicUSEAST2A	rtbassoc-09bea839e30cdc6a4	AWS::EC2::SubnetRouteTableAssociation	CREATE_COMPLETE	-	-
RouteTableAssociationPublicUSEAST2B	rtbassoc-0f1b20f2dd2098b0f	AWS::EC2::SubnetRouteTableAssociation	CREATE_COMPLETE	-	-
ServiceRole	eksctl-BApro-EkScluster-fargate-cluste-ServiceRole-1U5MAARRA5TE0	AWS::IAM::Role	CREATE_COMPLETE	-	-
ServiceRole	eksctl-BApro-EkScluster-fargate-cluste-ServiceRole-1U5MAARRA5TE0	AWS::IAM::Role	CREATE_COMPLETE	-	-
SubnetPrivateUSEAST2A	subnet-0ce6e916176f413ad	AWS::EC2::Subnet	CREATE_COMPLETE	-	-
SubnetPrivateUSEAST2B	subnet-0256028655c637056	AWS::EC2::Subnet	CREATE_COMPLETE	-	-
SubnetPublicUSEAST2A	subnet-06040989c2c12def6	AWS::EC2::Subnet	CREATE_COMPLETE	-	-
SubnetPublicUSEAST2B	subnet-0045d7b70713a6680	AWS::EC2::Subnet	CREATE_COMPLETE	-	-
VPC	vpc-0370e1e9e3d772a74	AWS::EC2::VPC	CREATE_COMPLETE	-	-
VPCGatewayAttachment	eksct-VPCGa-10AZZEDCEBG7Y	AWS::EC2::VPCGatewayAttachment	CREATE_COMPLETE	-	-

Figure 22: Cluster resources created automatically by CloudFormation

This command used to configure access to the cluster.



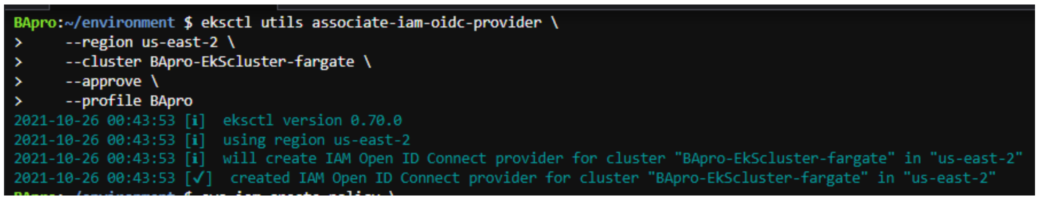
```
BApro:~/environment/BApro/BApro $ aws eks --region us-east-2 update-kubeconfig --name BApro-EkScluster-fargate --profile BApro
Added new context arn:aws:eks:us-east-2:837796115985:cluster/BApro-EkScluster-fargate to /home/ec2-user/.kube/config
BApro:~/environment/BApro/BApro $
```

Figure 23: Configure access to the EKS cluster

Step 6: Deployed the Load Balancer Controller

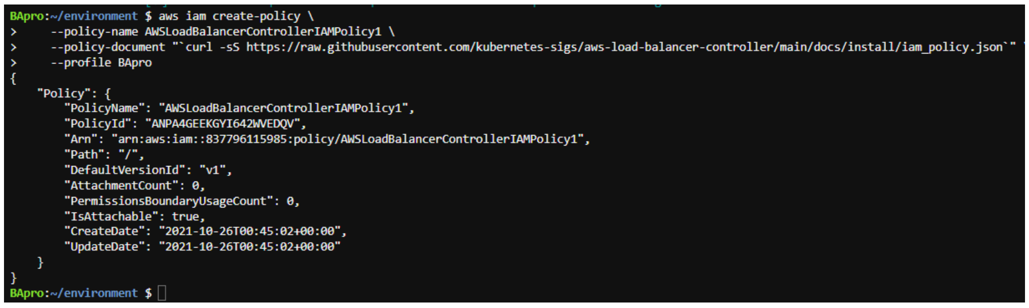
AWS Application load balancer ingress used in this project. Helm tool used to install the AWS Application load balancer (ALB) Ingress controller. Ingress controller configured by creating IAM OIDC provider, IAM policy and IAM service account.

IAM OIDC provider to provide permission to Fargate pods which are launched in the EKS cluster. IAM policy to allow controller pods to build and control the Application load balancer in the AWS account. Created IAM service account to provides AWS permission to the containers in any pods, which uses that service account. [14] Also, TargetGroupBinding CRDs installed as another important requirement for installing ingress controller.[15]



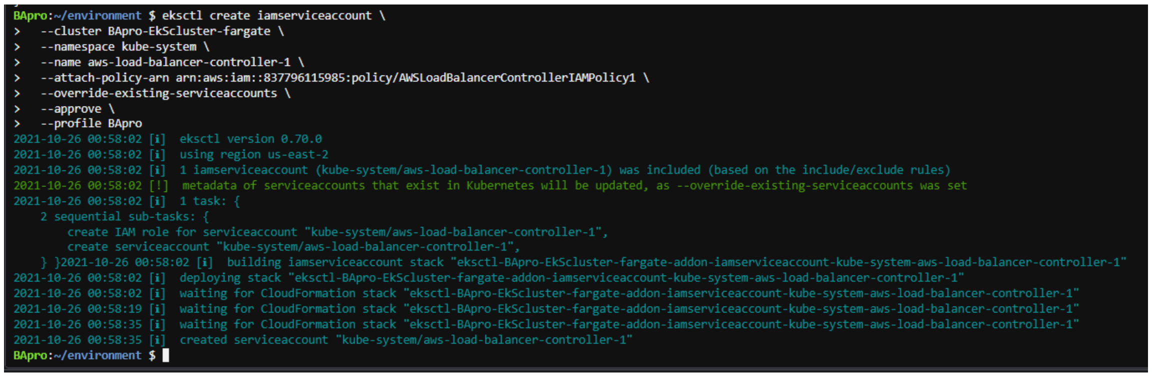
```
BApro:~/environment $ eksctl utils associate-iam-oidc-provider \
>     --region us-east-2 \
>     --cluster BApro-EkScluster-fargate \
>     --approve \
>     --profile BApro
2021-10-26 00:43:53 [i]  eksctl version 0.70.0
2021-10-26 00:43:53 [i]  using region us-east-2
2021-10-26 00:43:53 [i]  will create IAM Open ID Connect provider for cluster "BApro-EkScluster-fargate" in "us-east-2"
2021-10-26 00:43:53 [✓]  created IAM Open ID Connect provider for cluster "BApro-EkScluster-fargate" in "us-east-2"
```

Figure 24 : IAM OIDC provider



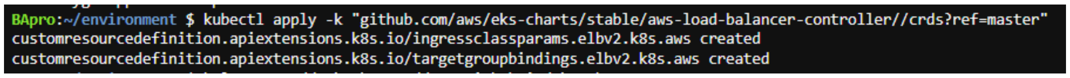
```
BApro:~/environment $ aws iam create-policy \
>     --policy-name AWSLoadBalancerControllerIAMPolicy1 \
>     --policy-document "`curl -sS https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/main/docs/install/iam_policy.json`"
>     --profile BApro
{
    "Policy": {
        "PolicyName": "AWSLoadBalancerControllerIAMPolicy1",
        "PolicyId": "ANPA4GEEKGYI642WVEDQV",
        "Arn": "arn:aws:iam::837796115985:policy/AWSLoadBalancerControllerIAMPolicy1",
        "Path": "/",
        "DefaultVersionId": "v1",
        "AttachmentCount": 0,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "CreateDate": "2021-10-26T00:45:02+00:00",
        "UpdateDate": "2021-10-26T00:45:02+00:00"
    }
}
BApro:~/environment $
```

Figure 25 : IAM policy



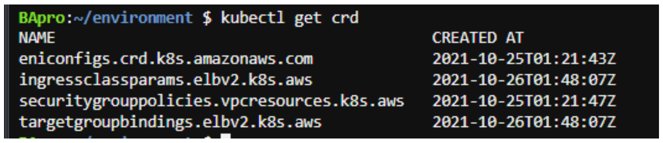
```
BApro:~/environment $ eksctl create iamserviceaccount \
>   --cluster BApro-EkScluster-fargate \
>   --namespace kube-system \
>   --name aws-load-balancer-controller-1 \
>   --attach-policy-arn arn:aws:iam::837796115985:policy/AWSLoadBalancerControllerIAMPolicy1 \
>   --override-existing-serviceaccounts \
>   --approve \
>   --profile BApro
2021-10-26 00:58:02 [i]  eksctl version 0.70.0
2021-10-26 00:58:02 [i]  using region us-east-2
2021-10-26 00:58:02 [i]  1 iamserviceaccount (kube-system/aws-load-balancer-controller-1) was included (based on the include/exclude rules)
2021-10-26 00:58:02 [!]  metadata of serviceaccounts that exist in Kubernetes will be updated, as --override-existing-serviceaccounts was set
2021-10-26 00:58:02 [i]  1 task: {
    2 sequential sub-tasks: {
        create IAM role for serviceaccount "kube-system/aws-load-balancer-controller-1",
        create serviceaccount "kube-system/aws-load-balancer-controller-1",
    } }2021-10-26 00:58:02 [i]  building iamserviceaccount stack "eksctl-BApro-EkScluster-fargate-addon-iamserviceaccount-kube-system-aws-load-balancer-controller-1"
2021-10-26 00:58:02 [i]  deploying stack "eksctl-BApro-EkScluster-fargate-addon-iamserviceaccount-kube-system-aws-load-balancer-controller-1"
2021-10-26 00:58:02 [i]  waiting for CloudFormation stack "eksctl-BApro-EkScluster-fargate-addon-iamserviceaccount-kube-system-aws-load-balancer-controller-1"
2021-10-26 00:58:19 [i]  waiting for CloudFormation stack "eksctl-BApro-EkScluster-fargate-addon-iamserviceaccount-kube-system-aws-load-balancer-controller-1"
2021-10-26 00:58:35 [i]  waiting for CloudFormation stack "eksctl-BApro-EkScluster-fargate-addon-iamserviceaccount-kube-system-aws-load-balancer-controller-1"
2021-10-26 00:58:35 [i]  created serviceaccount "kube-system/aws-load-balancer-controller-1"
BApro:~/environment $
```

Figure 26 : IAM service account



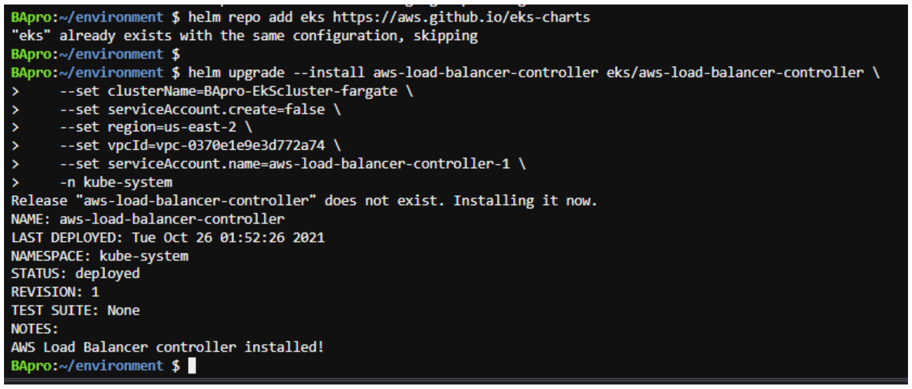
```
BApro:~/environment $ kubectl apply -k "github.com/aws/eks-charts/stable/aws-load-balancer-controller//crds?ref=master"
customresourcedefinition.apiextensions.k8s.io/ingressclassparams.elbv2.k8s.aws created
customresourcedefinition.apiextensions.k8s.io/targetgroupbindings.elbv2.k8s.aws created
```

Figure 27: Install the TargetGroupBinding CRDs



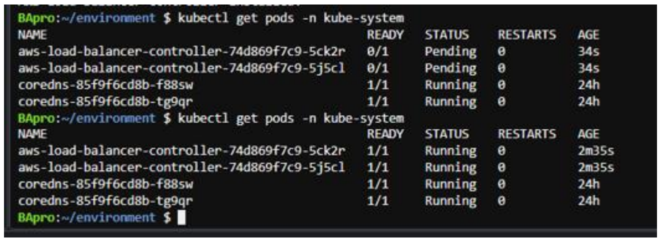
```
BApro:~/environment $ kubectl get crd
NAME                                         CREATED AT
eniconfigs.crd.k8s.amazonaws.com             2021-10-25T01:21:43Z
ingressclassparams.elbv2.k8s.aws             2021-10-26T01:48:07Z
securitygrouppolicies.vpcresources.k8s.aws   2021-10-25T01:21:47Z
targetgroupbindings.elbv2.k8s.aws            2021-10-26T01:48:07Z
```

Figure 28: Checking CRDs



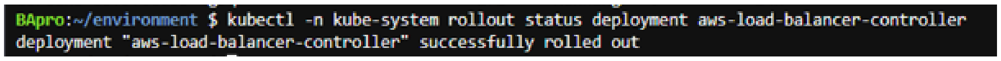
```
BApro:~/environment $ helm repo add eks https://aws.github.io/eks-charts
"eks" already exists with the same configuration, skipping
BApro:~/environment $
BApro:~/environment $ helm upgrade --install aws-load-balancer-controller eks/aws-load-balancer-controller \
>     --set clusterName=BApro-EkScluster-fargate \
>     --set serviceAccount.create=false \
>     --set region=us-east-2 \
>     --set vpcId=vpc-0370e1e9e3d772a74 \
>     --set serviceAccount.name=aws-load-balancer-controller-1 \
>     -n kube-system
Release "aws-load-balancer-controller" does not exist. Installing it now.
NAME: aws-load-balancer-controller
LAST DEPLOYED: Tue Oct 26 01:52:26 2021
NAMESPACE: kube-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
AWS Load Balancer controller installed!
BApro:~/environment $
```

Figure 29: Add the EKS repository to Helm and Install the AWS Load Balancer controller using iamserviceaccount



```
BApro:~/environment $ kubectl get pods -n kube-system
NAME                                            READY   STATUS    RESTARTS   AGE
aws-load-balancer-controller-74d869f7c9-5ck2r   0/1     Pending   0          34s
aws-load-balancer-controller-74d869f7c9-5j5cl   0/1     Pending   0          34s
coredns-85f9f6cd8b-f88sw                        1/1     Running   0          24h
coredns-85f9f6cd8b-tg9qr                        1/1     Running   0          24h
BApro:~/environment $ kubectl get pods -n kube-system
NAME                                            READY   STATUS    RESTARTS   AGE
aws-load-balancer-controller-74d869f7c9-5ck2r   1/1     Running   0          2m35s
aws-load-balancer-controller-74d869f7c9-5j5cl   1/1     Running   0          2m35s
coredns-85f9f6cd8b-f88sw                        1/1     Running   0          24h
coredns-85f9f6cd8b-tg9qr                        1/1     Running   0          24h
BApro:~/environment $
```

Figure 30: Check the pods by using kubectl get command

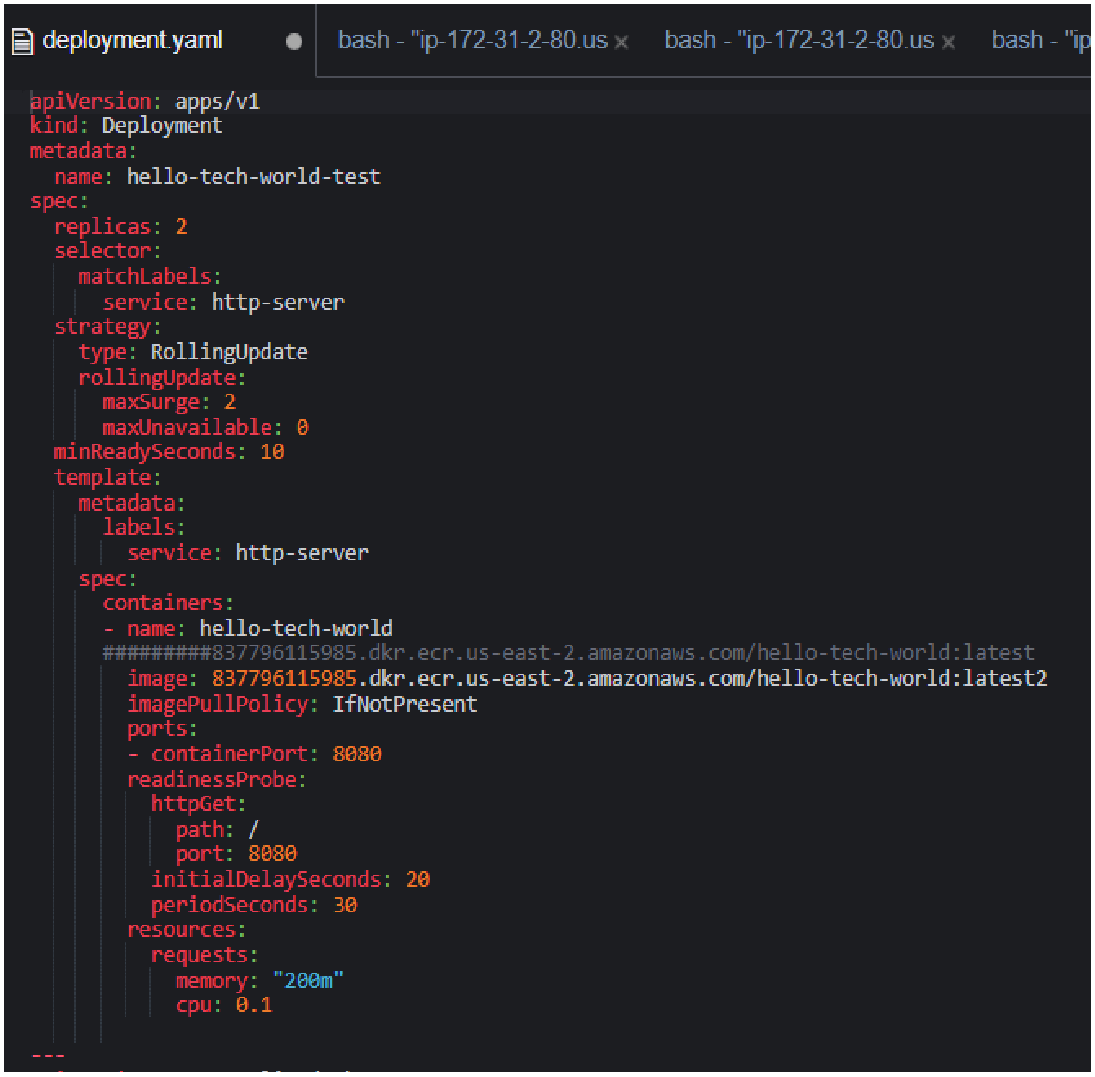


```
BApro:~/environment $ kubectl -n kube-system rollout status deployment aws-load-balancer-controller
deployment "aws-load-balancer-controller" successfully rolled out
```

Figure 31: Check the deployment completed

Step 7: Deployment prerequisites

After checked that ingress controller running successfully, two YAML files used for the deployment step. Rolling update deployment strategy implemented using kubctle. This strategy aims to successfully upgrade pods that run in the EKS cluster before bringing old versions down or terminate them. There are many benefits of using this strategy that include no high risk because of the readiness check and increases rollout without downtime which guarantees availability and scalability of the application to the users.



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-tech-world-test
spec:
  replicas: 2
  selector:
    matchLabels:
      service: http-server
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 2
      maxUnavailable: 0
  minReadySeconds: 10
  template:
    metadata:
      labels:
        service: http-server
    spec:
      containers:
      - name: hello-tech-world
      #########837796115985.dkr.ecr.us-east-2.amazonaws.com/hello-tech-world:latest
        image: 837796115985.dkr.ecr.us-east-2.amazonaws.com/hello-tech-world:latest2
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 8080
        readinessProbe:
          httpGet:
            path: /
            port: 8080
          initialDelaySeconds: 20
          periodSeconds: 30
        resources:
          requests:
            memory: "200m"
            cpu: 0.1

---
```

Figure 32: Deployment part from the deployment yaml file

Many resources that required by the application created in this project such as a deployment, a service, and a HPA. To simplify managing them, they grouped together in the same file called deployment.yaml and separated by --- .

This Deployment pulled a container image from an ECR public repository that was created in the previous step (hello-tech-world) and deployed two replicas, which means single pods, of it to the EKS cluster with Fargate profile that created previously and it called (BApro-EKScluster-fargate) .

In this deployment part of the YAML file , the.metadata.name field indicates the creation of a deployment called hello-tech-world-test and spec.replicas field indicates that this deployment created two replicated pods. The pod template is explained under spec.tamplet to define how this deployment finds the correct pods to manage. spec.selector field used and matchLabels used to tell the service:http-server to match the pods.

Rolling update strategy configured and it added under .spec.strategy.type. There are some necessary fields that should be set under the rolling update strategy, including maxSugar,maxUnavalible, and minReadySeconds.

minReadySeconds set to 10, which is how long Kubernetes should wait till creating the next pod. Once the pod is created, Kubernetes, by default, assume that the application is available. The service won't be available after updating if this field is empty because application pods are not in ready status yet.

maxSuger is how many pods should be created than the desired one. For example, in this deployment, it is set to 2, which means that there will be a maximum of four pods during the update because we have the replicas set to 2.

maxUnavailable is how many pods would be unavailable during the update; for example, in this deployment is set to 0 which there will be 0 pods unavailable during the update [16]

The template represents the pod template, pods are labeled service http-server by using metadata.labels field and the template.spec field, indicates that the Pods run one container which is hello- tech-world container which runs the docker images and used tag which allows identifying different versions[17].

One of Kubernetes benefits is that if the container crashes for any reason, it will be restarted by default. It uses many ways, and one of them is Readiness probes, which is configured in this project to ensure that traffic flows correctly and the application is in a healthy state.

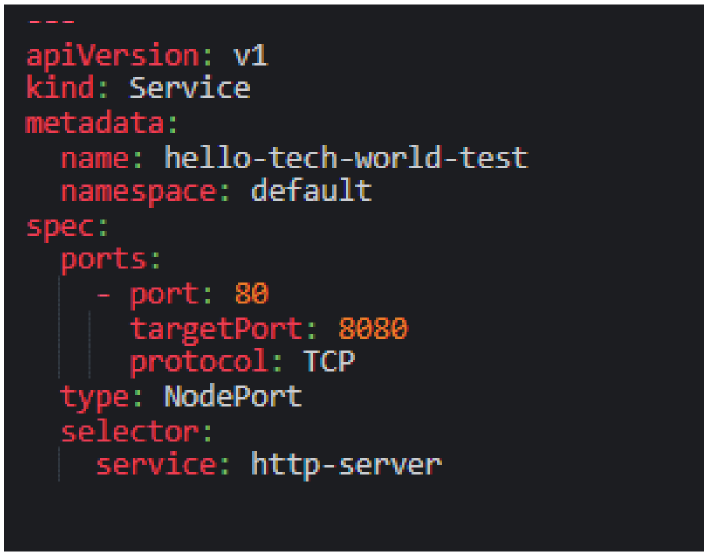
In Kubernetes, readiness probes used to know when the pod is ready to serve traffic, and the pod will receive traffic from the service when the readiness probe passes; otherwise, traffic will not be sent to the pod.

Standard configurable fields are used, including initialDelaySeconds which means that probes start running after initialDelaySeconds after a container is started and set to 20. The other field is periodSeconds which is how often the readiness probe should run, set to 30.
In this project, the readiness probe with HTTPGet handler is configured.

This handler performs an HTTP GET request against the pod's IP address on a specified port 8080 and path /.” If the response has a status code greater than or equal to 200 and less than 400, the diagnostic is considered successful”. [18]

Kubernetes uses the Requests mechanism to control resources such as CPU and memory. “The container is guaranteed to get the requests that configured. Kubernetes will only schedule container on a pod that can give it that resource that it requests for “[19]. In this deployment, the pod has one container, and it has a request of 0.1(100m) CPU and 200m of memory.

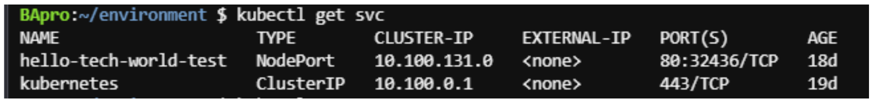
To be able to access these replicas (pods) through an IP address, Service created.



```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: hello-tech-world-test
  namespace: default
spec:
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
  type: NodePort
  selector:
    service: http-server
```

Figure 33: Service part from depolymnt.yaml file

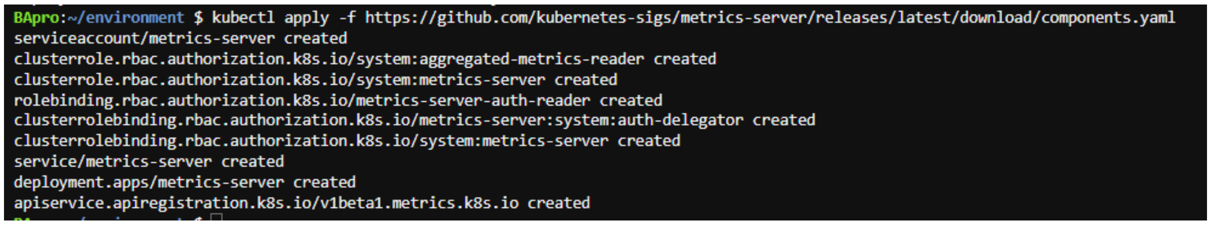
The service is named hello-tech-world-test and targets TCP, which default protocol for service, port 8080 on any pod with selector service: http-server and this service exposed as NodePort[20].



```
BApro:~/environment $ kubectl get svc
NAME                  TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
hello-tech-world-test NodePort    10.100.131.0   <none>        80:32436/TCP   18d
kubernetes            ClusterIP   10.100.0.1     <none>        443/TCP        19d
```

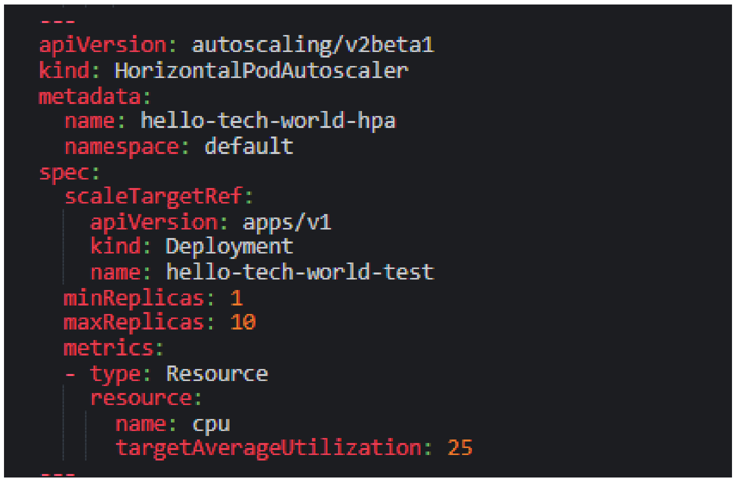
Figure 34: Service information by run get svc

Metrics server installed and it’s required for horizontal autoscaling. “Metrics Server is a scalable, efficient source of container resource metrics for Kubernetes built-in autoscaling pipelines” [21].



```
BApro:~/environment $ kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
serviceaccount/metrics-server created
clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created
clusterrole.rbac.authorization.k8s.io/system:metrics-server created
rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator created
clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
service/metrics-server created
deployment.apps/metrics-server created
apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created
```

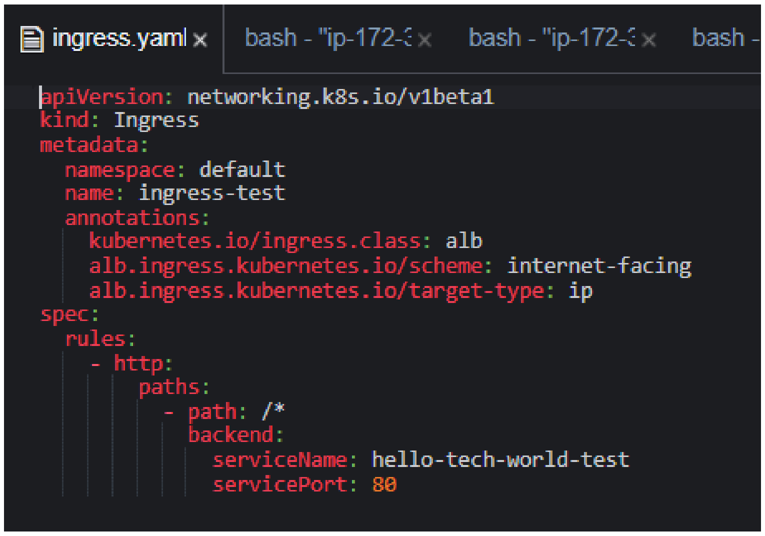
Figure 35: Installed Metrics Server



```
---
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: hello-tech-world-hpa
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: hello-tech-world-test
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 25
---
```

Figure 36: HPA part from deployment.yaml file

In this part, created Kubernetes Horizontal Pod Autoscaler (HPA) called it hello-tech-world-hpa. This HPA maintains between 1 (as the minimum) and 10 (as maximum) replicas of the pods controlled by the hello-tech-world test deployment created previously. The number of replicas increases and reduces by HPA through deployment to maintain a 25% average CPU utilization across all pods. Pod requests 100 milli-cores which means average CPU utilization would be 25 milli-cores. if 50% is the average CPU utilization, means 50 milli-cores is average CPU usage [22].



```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  namespace: default
  name: ingress-test
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
spec:
  rules:
    - http:
        paths:
          - path: /*
            backend:
              serviceName: hello-tech-world-test
              servicePort: 80
```

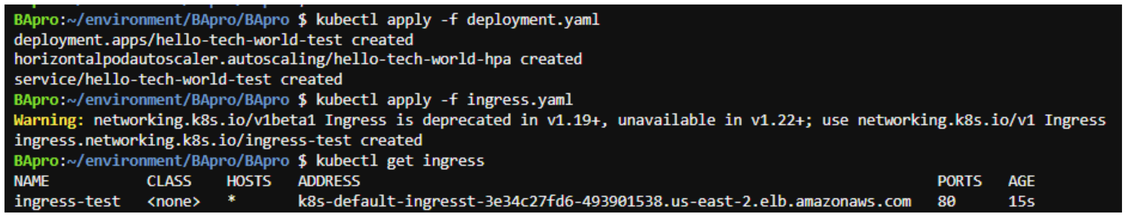
Figure 37: Ingress yaml file

Ingress with ingress-test named configured using this yaml file and some annotations are added.kubernetes.io/ingress.class: alb annotation is added to configure the application load balancer to route traffic to pods within the cluster. alb.ingress.kubernetes.io/scheme: internet-facing annotation is added to be able to access the app externally because Alb will be internal by default. Also, alb.ingress.kubernetes.io/target-type is added, and it defines how to route traffic to pods, and it is set to IP mode, which means the traffic will be routed directly to the pod IP. ingress spec includes a list of rules matched against all incoming requests. Every HTTP rule includes information such as the path which is set to /* and the backend configured with the service name and port that was created." HTTP requests to the Ingress that matches the path of the rule are sent to the listed backend". [23][24][25]

Phase2: Implantation and Testing

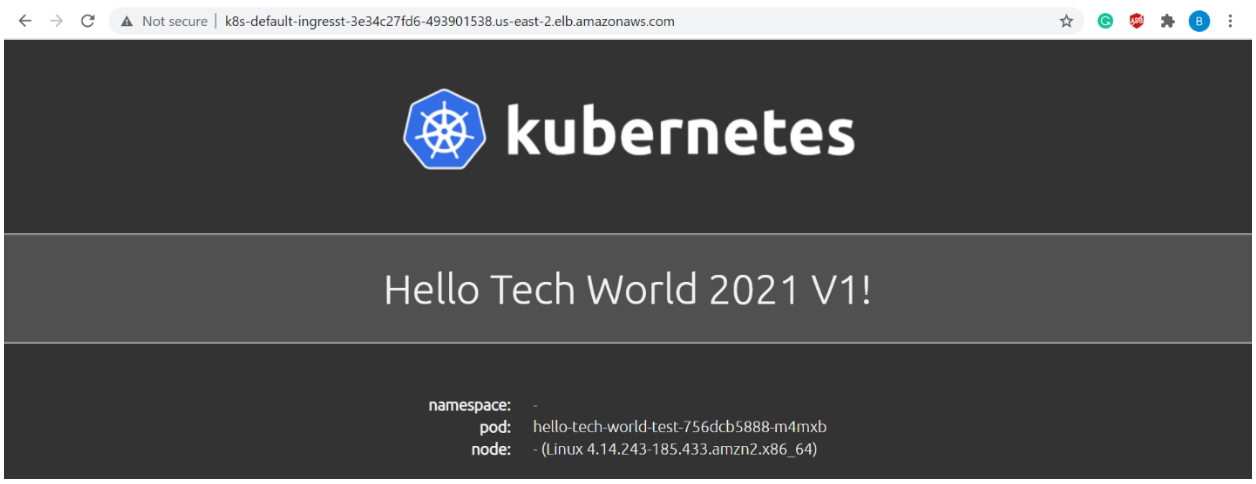
Created deployment and ingress by run the kubectl command and use get ingress to get the IP address. There are many ways to perform rolling update, set image and edit container image in deployment.yaml file are the two ways that used in this project.

Command Format : kubectl set image deployment <deployment> <container>=<image> --record



```
BApro:~/environment/BApro/BApro $ kubectl apply -f deployment.yaml
deployment.apps/hello-tech-world-test created
horizontalpodautoscaler.autoscaling/hello-tech-world-hpa created
service/hello-tech-world-test created
BApro:~/environment/BApro/BApro $ kubectl apply -f ingress.yaml
Warning: networking.k8s.io/v1beta1 Ingress is deprecated in v1.19+, unavailable in v1.22+; use networking.k8s.io/v1 Ingress
ingress.networking.k8s.io/ingress-test created
BApro:~/environment/BApro/BApro $ kubectl get ingress
NAME           CLASS    HOSTS   ADDRESS                                                       PORTS   AGE
ingress-test   <none>   *       k8s-default-ingresst-3e34c27fd6-493901538.us-east-2.elb.amazonaws.com   80      15s
```

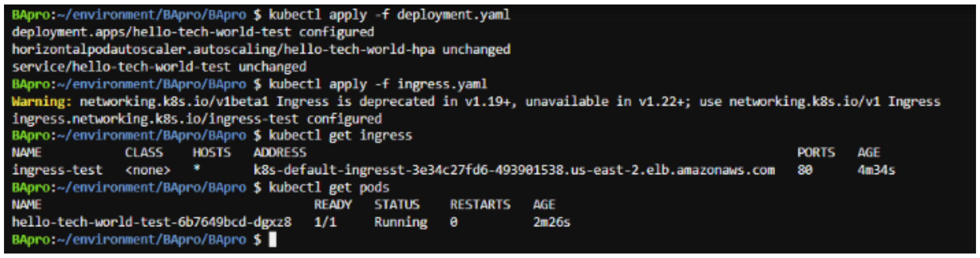
Figure 38: Deployment implemented 1



```
Not secure | k8s-default-ingresst-3e34c27fd6-493901538.us-east-2.elb.amazonaws.com

kubernetes

Hello Tech World 2021 V1!

namespace: -
      pod: hello-tech-world-test-756dcb5888-m4mxb
     node: - (Linux 4.14.243-185.433.amzn2.x86_64)
```

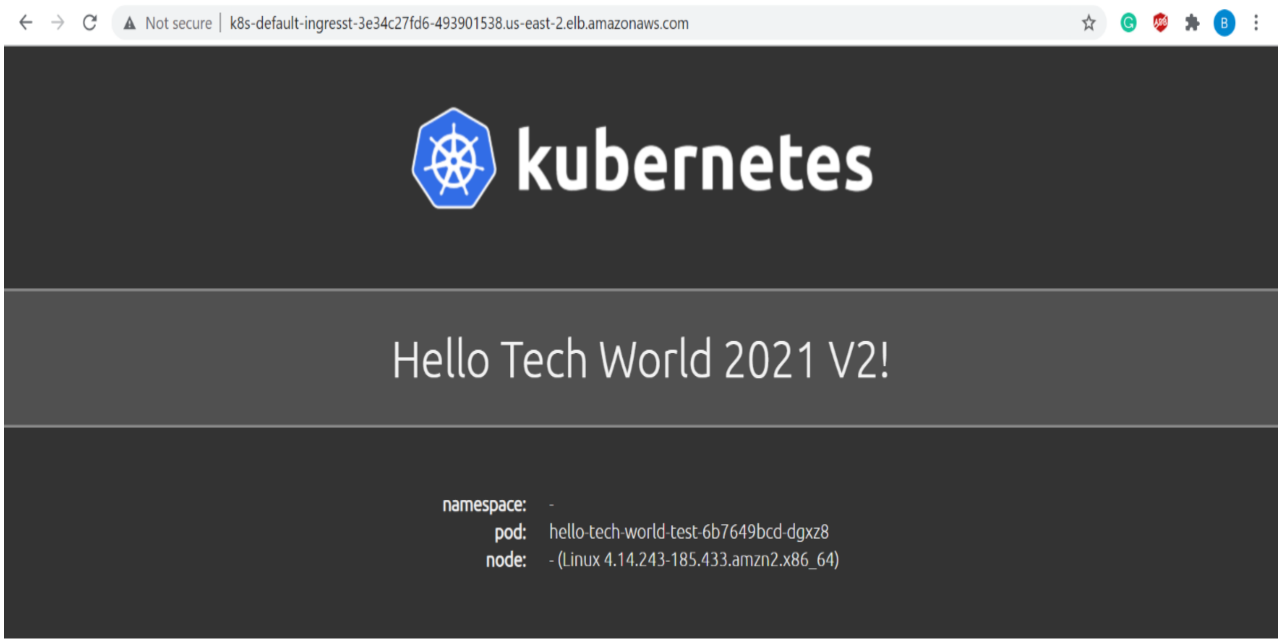
Figure 39: Check the web application Hello Tech World V1

Performed rolling update by edited container image in the container section in the deployment.yaml file.



```
BApro:~/environment/BApro/BApro $ kubectl apply -f deployment.yaml
deployment.apps/hello-tech-world-test configured
horizontalpodautoscaler.autoscaling/hello-tech-world-hpa unchanged
service/hello-tech-world-test unchanged
BApro:~/environment/BApro/BApro $ kubectl apply -f ingress.yaml
Warning: networking.k8s.io/v1beta1 Ingress is deprecated in v1.19+, unavailable in v1.22+; use networking.k8s.io/v1 Ingress
ingress.networking.k8s.io/ingress-test configured
BApro:~/environment/BApro/BApro $ kubectl get ingress
NAME           CLASS    HOSTS   ADDRESS                                                                  PORTS   AGE
ingress-test   <none>   *       k8s-default-ingresst-3e34c27fd6-493901538.us-east-2.elb.amazonaws.com   80      4m34s
BApro:~/environment/BApro/BApro $ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
hello-tech-world-test-6b7649bcd-dgxz8   1/1     Running   0          2m26s
BApro:~/environment/BApro/BApro $
```

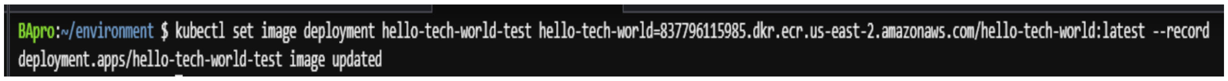
Figure 40: Deployment implemented 2



```
Not secure | k8s-default-ingresst-3e34c27fd6-493901538.us-east-2.elb.amazonaws.com

kubernetes

Hello Tech World 2021 V2!

namespace: -
      pod: hello-tech-world-test-6b7649bcd-dgxz8
     node: - (Linux 4.14.243-185.433.amzn2.x86_64)
```

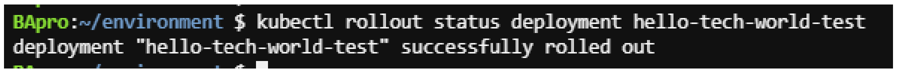
Figure 41: Check the web application Hello Tech World V2

Performed Rolling-update deployment by using set image way and tested availability of the application



```
BApro:~/environment $ kubectl set image deployment hello-tech-world-test hello-tech-world=837796115985.dkr.ecr.us-east-2.amazonaws.com/hello-tech-world:latest --record
deployment.apps/hello-tech-world-test image updated
```

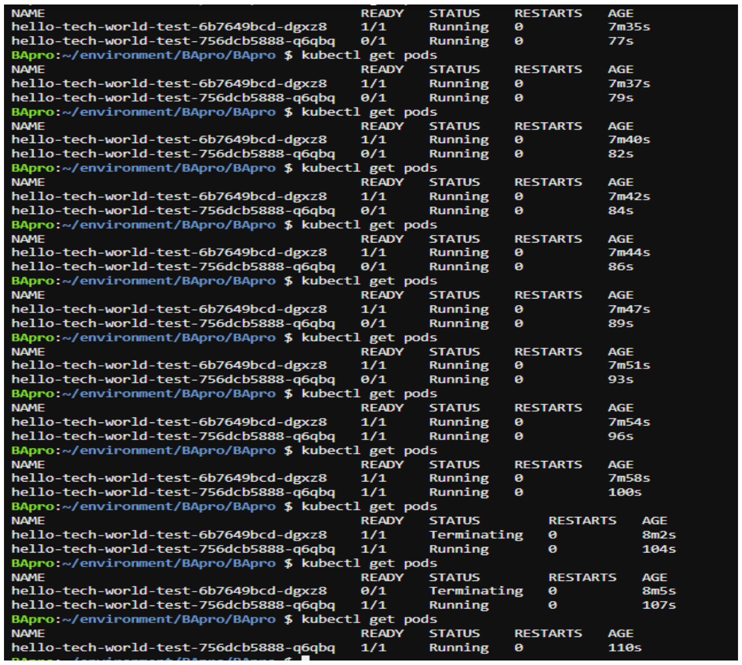
Figure 42: Rolling update preformed



```
BApro:~/environment $ kubectl rollout status deployment hello-tech-world-test
deployment "hello-tech-world-test" successfully rolled out
```

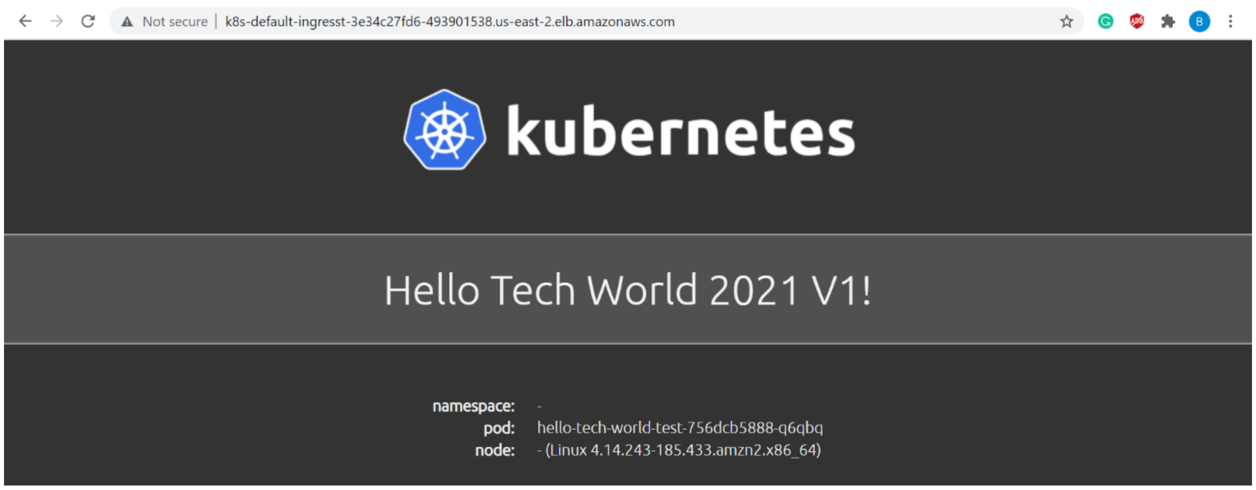
Figure 43: Rollout status

Run get pods command to see the process of running and terminating the pods.



```
NAME                                    READY   STATUS    RESTARTS   AGE
hello-tech-world-test-6b7649bcd-dgxz8    1/1     Running   0          7m35s
hello-tech-world-test-756dcb5888-q6qbq   0/1     Running   0          77s
BApro:~/environment/BApro/BApro $ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
hello-tech-world-test-6b7649bcd-dgxz8    1/1     Running   0          7m37s
hello-tech-world-test-756dcb5888-q6qbq   0/1     Running   0          79s
BApro:~/environment/BApro/BApro $ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
hello-tech-world-test-6b7649bcd-dgxz8    1/1     Running   0          7m40s
hello-tech-world-test-756dcb5888-q6qbq   0/1     Running   0          82s
BApro:~/environment/BApro/BApro $ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
hello-tech-world-test-6b7649bcd-dgxz8    1/1     Running   0          7m42s
hello-tech-world-test-756dcb5888-q6qbq   0/1     Running   0          84s
BApro:~/environment/BApro/BApro $ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
hello-tech-world-test-6b7649bcd-dgxz8    1/1     Running   0          7m44s
hello-tech-world-test-756dcb5888-q6qbq   0/1     Running   0          86s
BApro:~/environment/BApro/BApro $ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
hello-tech-world-test-6b7649bcd-dgxz8    1/1     Running   0          7m47s
hello-tech-world-test-756dcb5888-q6qbq   0/1     Running   0          89s
BApro:~/environment/BApro/BApro $ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
hello-tech-world-test-6b7649bcd-dgxz8    1/1     Running   0          7m51s
hello-tech-world-test-756dcb5888-q6qbq   0/1     Running   0          93s
BApro:~/environment/BApro/BApro $ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
hello-tech-world-test-6b7649bcd-dgxz8    1/1     Running   0          7m54s
hello-tech-world-test-756dcb5888-q6qbq   1/1     Running   0          96s
BApro:~/environment/BApro/BApro $ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
hello-tech-world-test-6b7649bcd-dgxz8    1/1     Running   0          7m58s
hello-tech-world-test-756dcb5888-q6qbq   1/1     Running   0          100s
BApro:~/environment/BApro/BApro $ kubectl get pods
NAME                                    READY   STATUS        RESTARTS   AGE
hello-tech-world-test-6b7649bcd-dgxz8    1/1     Terminating   0          8m2s
hello-tech-world-test-756dcb5888-q6qbq   1/1     Running       0          104s
BApro:~/environment/BApro/BApro $ kubectl get pods
NAME                                    READY   STATUS        RESTARTS   AGE
hello-tech-world-test-6b7649bcd-dgxz8    0/1     Terminating   0          8m5s
hello-tech-world-test-756dcb5888-q6qbq   1/1     Running       0          107s
BApro:~/environment/BApro/BApro $ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
hello-tech-world-test-756dcb5888-q6qbq   1/1     Running   0          110s
```

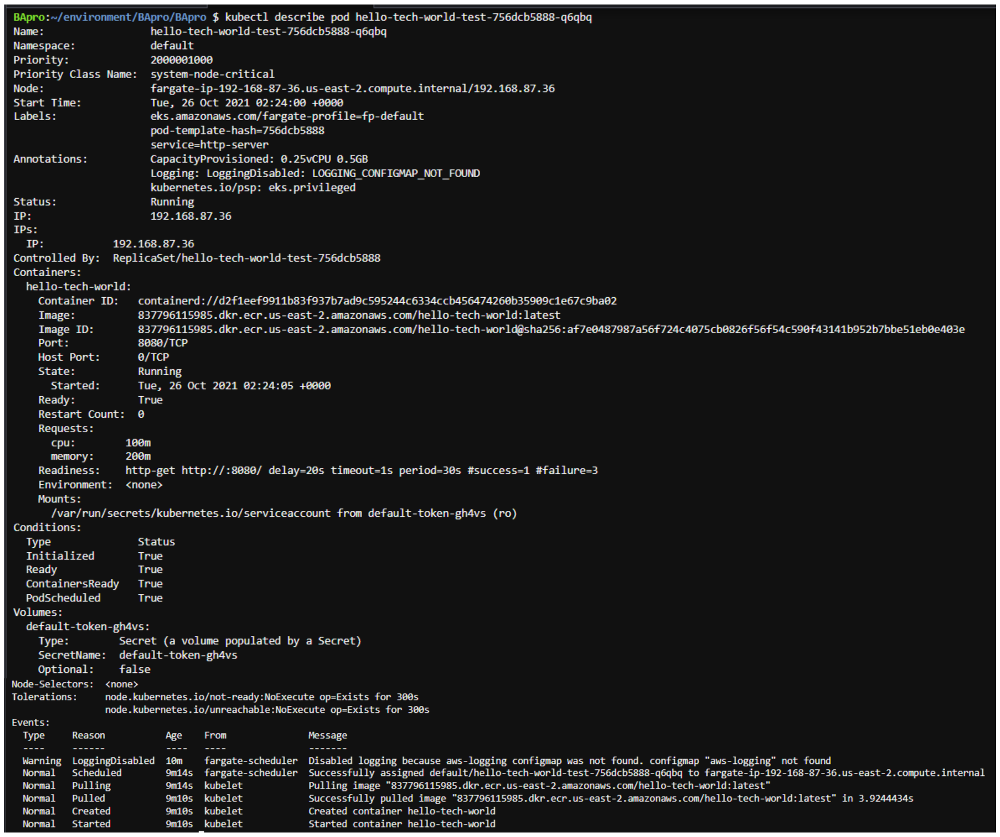
Figure 44: Check pod status


Not secure | k8s-default-ingresst-3e34c27fd6-493901538.us-east-2.elb.amazonaws.com
kubernetes

Hello Tech World 2021 V1!

namespace: -
pod: hello-tech-world-test-756dcb5888-q6qbq
node: - (Linux 4.14.243-185.433.amzn2.x86_64)

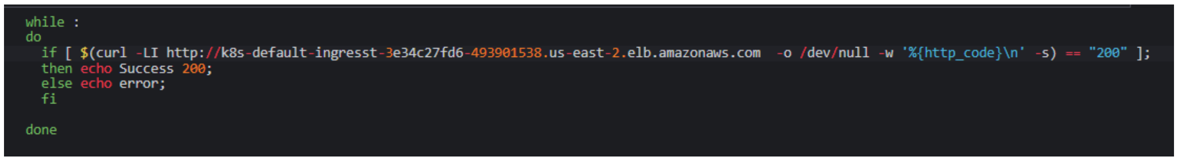
Figure 45: Check the web application Hello Tech World V1



```
BApro:~/environment/BApro/BApro $ kubectl describe pod hello-tech-world-test-756dcb5888-q6qbq
Name:                 hello-tech-world-test-756dcb5888-q6qbq
Namespace:            default
Priority:             2000001000
Priority Class Name:  system-node-critical
Node:                 fargate-ip-192-168-87-36.us-east-2.compute.internal/192.168.87.36
Start Time:           Tue, 26 Oct 2021 02:24:00 +0000
Labels:               eks.amazonaws.com/fargate-profile=fp-default
                      pod-template-hash=756dcb5888
                      service=http-server
Annotations:          CapacityProvisioned: 0.25vCPU 0.5GB
                      Logging: LoggingDisabled: LOGGING_CONFIGMAP_NOT_FOUND
                      kubernetes.io/psp: eks.privileged
Status:               Running
IP:                   192.168.87.36
IPs:
  IP:           192.168.87.36
Controlled By:  ReplicaSet/hello-tech-world-test-756dcb5888
Containers:
  hello-tech-world:
    Container ID:   containerd://d2f1eef9911b83f937b7ad9c595244c6334ccb456474260b35909c1e67c9ba02
    Image:          837796115985.dkr.ecr.us-east-2.amazonaws.com/hello-tech-world:latest
    Image ID:       837796115985.dkr.ecr.us-east-2.amazonaws.com/hello-tech-world@sha256:af7e0487987a56f724c4075cb0826f56f54c590f43141b952b7bbe51eb0e403e
    Port:           8080/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Tue, 26 Oct 2021 02:24:05 +0000
    Ready:          True
    Restart Count:  0
    Requests:
      cpu:        100m
      memory:     200m
    Readiness:    http-get http://:8080/ delay=20s timeout=1s period=30s #success=1 #failure=3
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-gh4vs (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-gh4vs:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-gh4vs
    Optional:    false
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                 node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type     Reason           Age    From               Message
  ----     ------           ----   ----               -------
  Warning  LoggingDisabled  10m    fargate-scheduler  Disabled logging because aws-logging configmap was not found. configmap "aws-logging" not found
  Normal   Scheduled        9m14s  fargate-scheduler  Successfully assigned default/hello-tech-world-test-756dcb5888-q6qbq to fargate-ip-192-168-87-36.us-east-2.compute.internal
  Normal   Pulling          9m14s  kubelet            Pulling image "837796115985.dkr.ecr.us-east-2.amazonaws.com/hello-tech-world:latest"
  Normal   Pulled           9m10s  kubelet            Successfully pulled image "837796115985.dkr.ecr.us-east-2.amazonaws.com/hello-tech-world:latest" in 3.9244434s
  Normal   Created          9m10s  kubelet            Created container hello-tech-world
  Normal   Started          9m10s  kubelet            Started container hello-tech-world
```

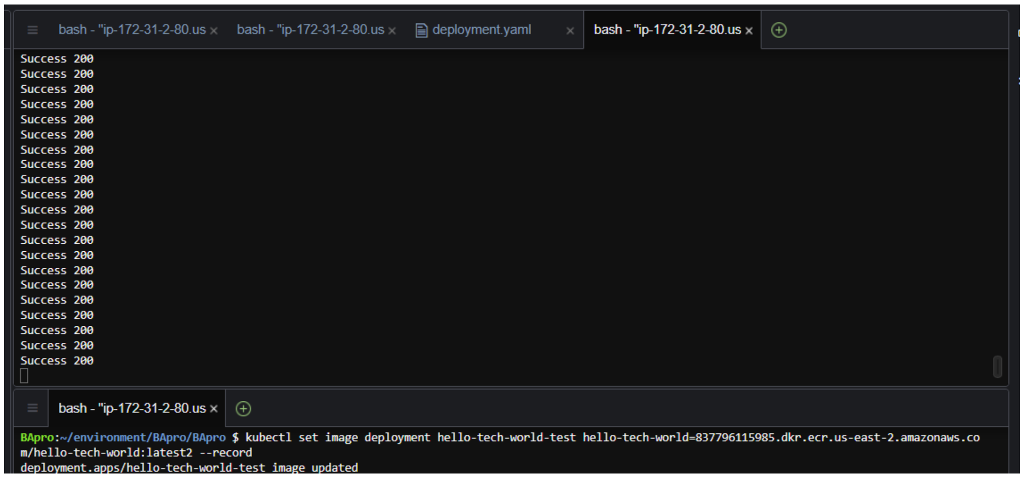
Figure 46: pod information

Test code used to check the accessibility and availability of the web application during the rolling update process and this code ran in new terminal. While in another terminal, rolling update deployment implemented.



```
while :
do
  if [ $(curl -LI http://k8s-default-ingresst-3e34c27fd6-493901538.us-east-2.elb.amazonaws.com  -o /dev/null -w '%{http_code}\n' -s) == "200" ];
  then echo Success 200;
  else echo error;
  fi

done
```

Figure 47: Testing code



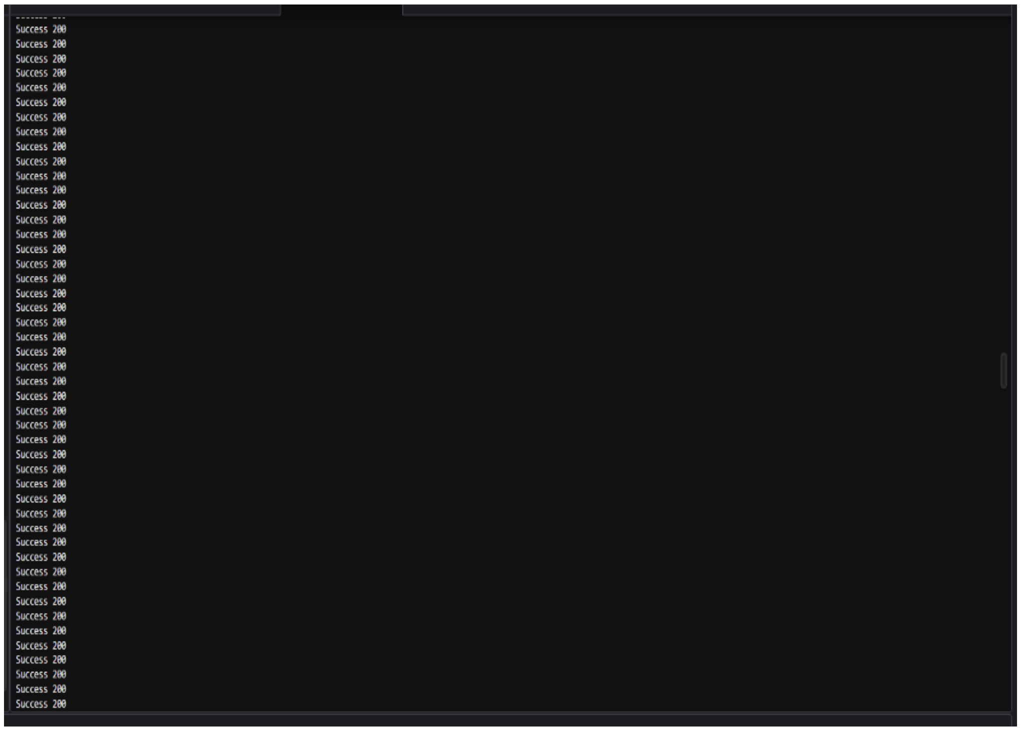
bash - "ip-172-31-2-80.us ×
bash - "ip-172-31-2-80.us ×
deployment.yaml ×
bash - "ip-172-31-2-80.us ×

```
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
```

bash - "ip-172-31-2-80.us ×

```
BApro:~/environment/BApro/BApro $ kubectl set image deployment hello-tech-world-test hello-tech-world=837796115985.dkr.ecr.us-east-2.amazonaws.co
m/hello-tech-world:latest2 --record
deployment.apps/hello-tech-world-test image updated
```

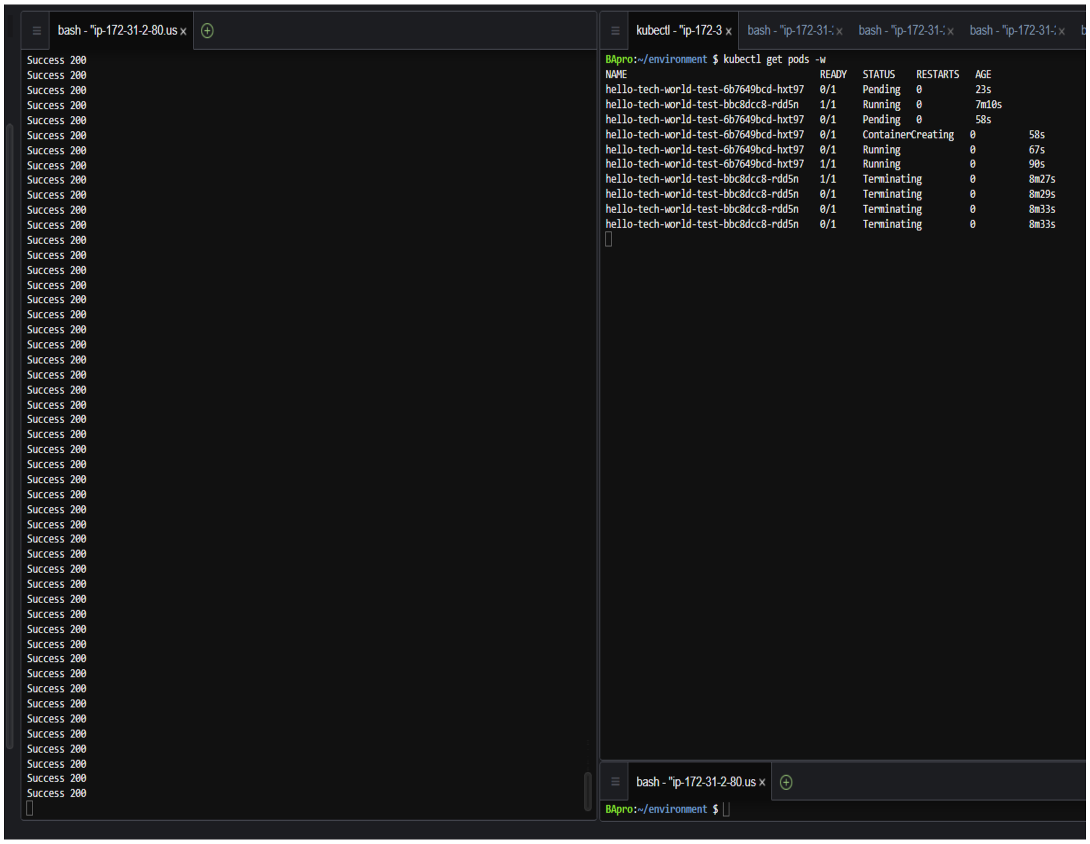
Figure 48: Rolling update preformed



```
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
```

Figure 49: Testing code result

Used get pods -w command to get clear vision of the process.

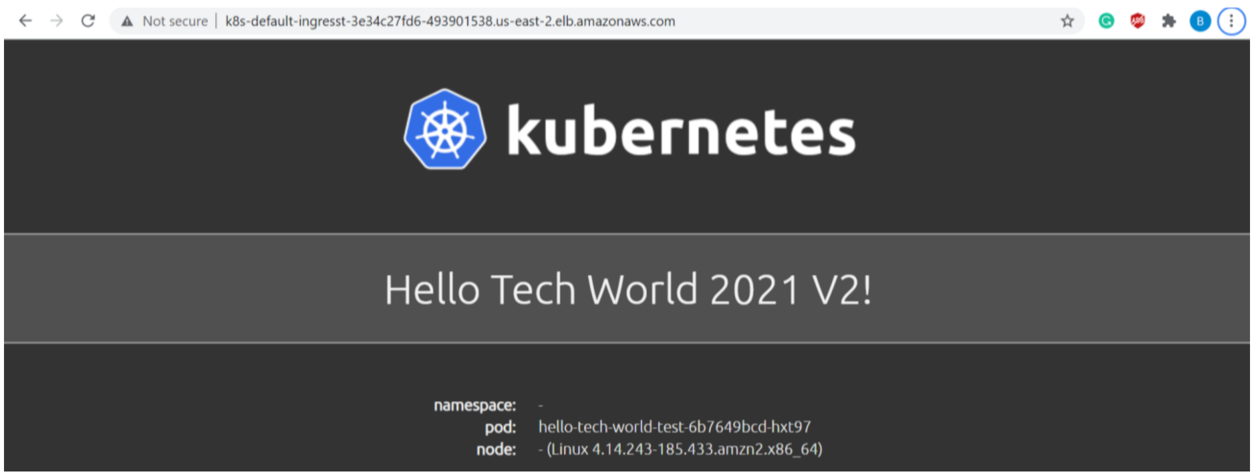


```
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
Success 200
```

```
BApro:~/environment $ kubectl get pods -w
NAME                                 READY   STATUS    RESTARTS   AGE
hello-tech-world-test-6b7649bcd-hxt97   0/1     Pending   0          23s
hello-tech-world-test-bbc8dcc8-rdd5n    1/1     Running   0          7m10s
hello-tech-world-test-6b7649bcd-hxt97   0/1     Pending   0          58s
hello-tech-world-test-6b7649bcd-hxt97   0/1     ContainerCreating   0          58s
hello-tech-world-test-6b7649bcd-hxt97   0/1     Running             0          67s
hello-tech-world-test-6b7649bcd-hxt97   1/1     Running             0          90s
hello-tech-world-test-bbc8dcc8-rdd5n    1/1     Terminating         0          8m27s
hello-tech-world-test-bbc8dcc8-rdd5n    0/1     Terminating         0          8m29s
hello-tech-world-test-bbc8dcc8-rdd5n    0/1     Terminating         0          8m33s
hello-tech-world-test-bbc8dcc8-rdd5n    0/1     Terminating         0          8m33s
```

```
BApro:~/environment $
```

Figure 50: Check pod status



Not secure | k8s-default-ingresst-3e34c27fd6-493901538.us-east-2.elb.amazonaws.com

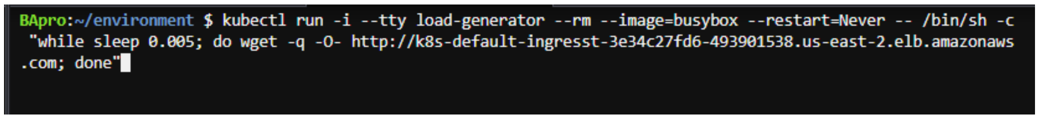
Hello Tech World 2021 V2!

namespace: -
pod: hello-tech-world-test-6b7649bcd-hxt97
node: - (Linux 4.14.243-185.433.amzn2.x86_64)

Figure 51: Check the web application Hello Tech World V2 after rolling update

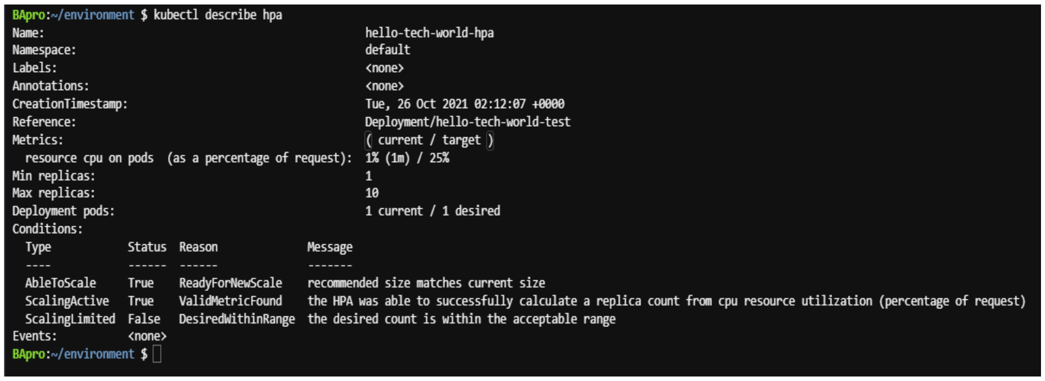
Generated load and tested Auto-Scaling using Kubernetes Horizontal Pod Autoscaler

Kubernetes Horizontal Pod Autoscaler (HAP) used in this project which automatically scale the number of pods based on CPU utilization. Load generator is used and kubectl commend is ran to see how it scale out to meet the increased demand on the application or scale in when the application is not needed. Therefore, HAP reduced the number of pods in the deployment to one, which is the minimum if the average CPU load is lower than 25% or 50%. If it is greater than that, HAP increased the number of pods in the deployment to 10, which is the maximum.



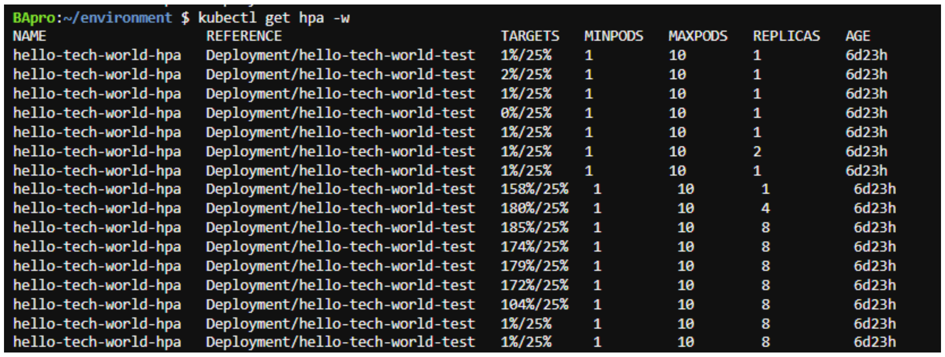
```
BApro:~/environment $ kubectl run -i --tty load-generator --rm --image=busybox --restart=Never -- /bin/sh -c
 "while sleep 0.005; do wget -q -O- http://k8s-default-ingresst-3e34c27fd6-493901538.us-east-2.elb.amazonaws
.com; done"
```

Figure 52: Generating load



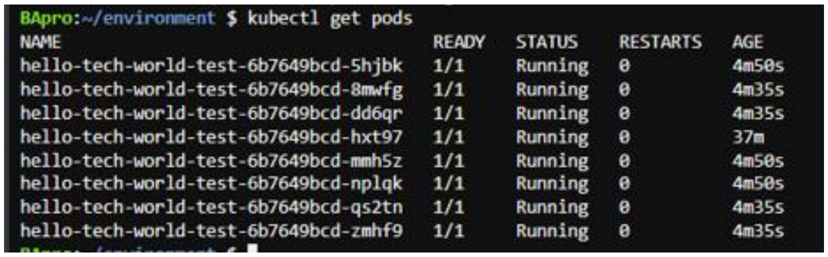
```
BApro:~/environment $ kubectl describe hpa
Name:                                                  hello-tech-world-hpa
Namespace:                                             default
Labels:                                                <none>
Annotations:                                           <none>
CreationTimestamp:                                     Tue, 26 Oct 2021 02:12:07 +0000
Reference:                                             Deployment/hello-tech-world-test
Metrics:                                               ( current / target )
  resource cpu on pods  (as a percentage of request):  1% (1m) / 25%
Min replicas:                                          1
Max replicas:                                          10
Deployment pods:                                       1 current / 1 desired
Conditions:
  Type            Status  Reason              Message
  ----            ------  ------              -------
  AbleToScale     True    ReadyForNewScale    recommended size matches current size
  ScalingActive   True    ValidMetricFound    the HPA was able to successfully calculate a replica count from cpu resource utilization (percentage of request)
  ScalingLimited  False   DesiredWithinRange  the desired count is within the acceptable range
Events:           <none>
BApro:~/environment $
```

Figure 53: Describe HPA



```
BApro:~/environment $ kubectl get hpa -w
NAME                  REFERENCE                          TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
hello-tech-world-hpa  Deployment/hello-tech-world-test   1%/25%     1         10        1          6d23h
hello-tech-world-hpa  Deployment/hello-tech-world-test   2%/25%     1         10        1          6d23h
hello-tech-world-hpa  Deployment/hello-tech-world-test   1%/25%     1         10        1          6d23h
hello-tech-world-hpa  Deployment/hello-tech-world-test   0%/25%     1         10        1          6d23h
hello-tech-world-hpa  Deployment/hello-tech-world-test   1%/25%     1         10        1          6d23h
hello-tech-world-hpa  Deployment/hello-tech-world-test   1%/25%     1         10        2          6d23h
hello-tech-world-hpa  Deployment/hello-tech-world-test   1%/25%     1         10        1          6d23h
hello-tech-world-hpa  Deployment/hello-tech-world-test   158%/25%    1         10        1          6d23h
hello-tech-world-hpa  Deployment/hello-tech-world-test   180%/25%    1         10        4          6d23h
hello-tech-world-hpa  Deployment/hello-tech-world-test   185%/25%    1         10        8          6d23h
hello-tech-world-hpa  Deployment/hello-tech-world-test   174%/25%    1         10        8          6d23h
hello-tech-world-hpa  Deployment/hello-tech-world-test   179%/25%    1         10        8          6d23h
hello-tech-world-hpa  Deployment/hello-tech-world-test   172%/25%    1         10        8          6d23h
hello-tech-world-hpa  Deployment/hello-tech-world-test   104%/25%    1         10        8          6d23h
hello-tech-world-hpa  Deployment/hello-tech-world-test   1%/25%      1         10        8          6d23h
hello-tech-world-hpa  Deployment/hello-tech-world-test   1%/25%      1         10        8          6d23h
```

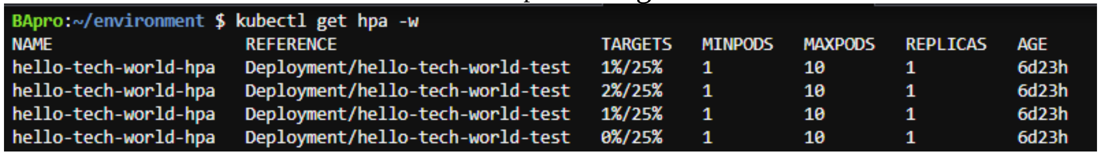
Figure 53: Auto scale out With CPU 25%



```
BApro:~/environment $ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hello-tech-world-test-6b7649bcd-5hjbk   1/1     Running   0          4m50s
hello-tech-world-test-6b7649bcd-8mwfg   1/1     Running   0          4m35s
hello-tech-world-test-6b7649bcd-dd6qr   1/1     Running   0          4m35s
hello-tech-world-test-6b7649bcd-hxt97   1/1     Running   0          37m
hello-tech-world-test-6b7649bcd-mmh5z   1/1     Running   0          4m50s
hello-tech-world-test-6b7649bcd-nplqk   1/1     Running   0          4m50s
hello-tech-world-test-6b7649bcd-qs2tn   1/1     Running   0          4m35s
hello-tech-world-test-6b7649bcd-zmhf9   1/1     Running   0          4m35s
```

Figure 54: 8 pods running

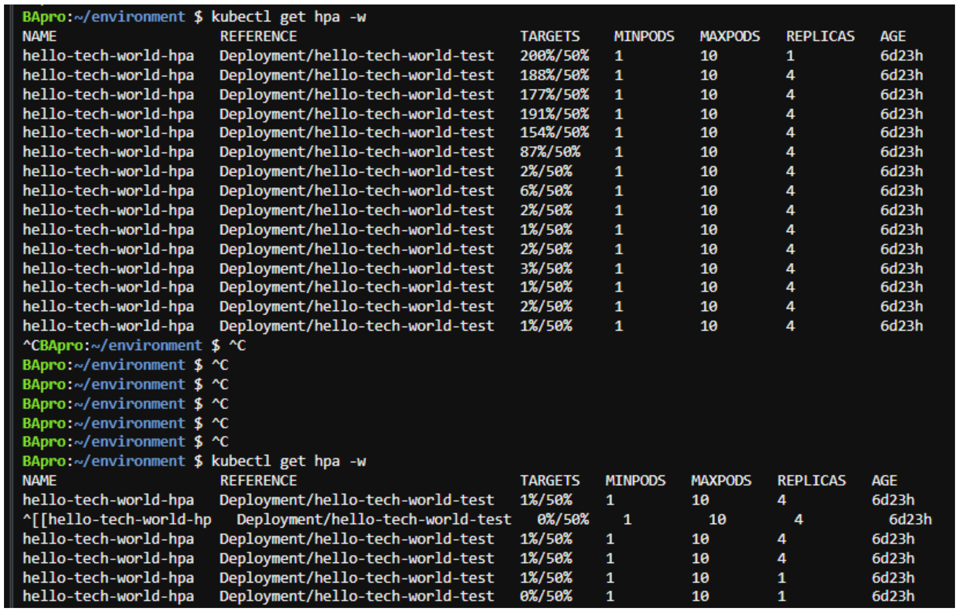
Scale in after stop the load generator



```
BApro:~/environment $ kubectl get hpa -w
NAME                   REFERENCE                            TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hello-tech-world-hpa   Deployment/hello-tech-world-test   1%/25%    1         10        1          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   2%/25%    1         10        1          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   1%/25%    1         10        1          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   0%/25%    1         10        1          6d23h
```

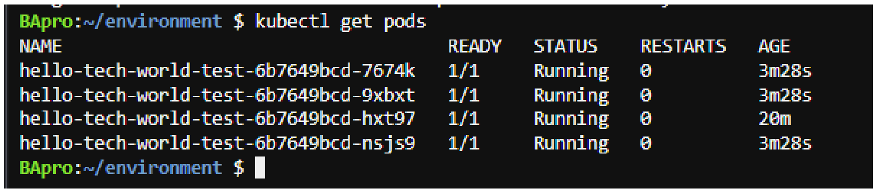
Figure 55: Auto scale in with CPU 25%

Used Ctrl+C to stop load generator.



```
BApro:~/environment $ kubectl get hpa -w
NAME                   REFERENCE                            TARGETS     MINPODS   MAXPODS   REPLICAS   AGE
hello-tech-world-hpa   Deployment/hello-tech-world-test   200%/50%    1         10        1          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   188%/50%    1         10        4          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   177%/50%    1         10        4          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   191%/50%    1         10        4          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   154%/50%    1         10        4          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   87%/50%     1         10        4          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   2%/50%      1         10        4          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   6%/50%      1         10        4          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   2%/50%      1         10        4          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   1%/50%      1         10        4          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   2%/50%      1         10        4          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   3%/50%      1         10        4          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   1%/50%      1         10        4          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   2%/50%      1         10        4          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   1%/50%      1         10        4          6d23h
^CBApro:~/environment $ ^C
BApro:~/environment $ ^C
BApro:~/environment $ ^C
BApro:~/environment $ ^C
BApro:~/environment $ ^C
BApro:~/environment $ ^C
BApro:~/environment $ kubectl get hpa -w
NAME                   REFERENCE                            TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hello-tech-world-hpa   Deployment/hello-tech-world-test   1%/50%    1         10        4          6d23h
^[[hello-tech-world-hp   Deployment/hello-tech-world-test   0%/50%    1         10        4          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   1%/50%    1         10        4          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   1%/50%    1         10        4          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   1%/50%    1         10        1          6d23h
hello-tech-world-hpa   Deployment/hello-tech-world-test   0%/50%    1         10        1          6d23h
```

Figure 56: Another try with CPU target 50 %



```
BApro:~/environment $ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hello-tech-world-test-6b7649bcd-7674k   1/1     Running   0          3m28s
hello-tech-world-test-6b7649bcd-9xbxt   1/1     Running   0          3m28s
hello-tech-world-test-6b7649bcd-hxt97   1/1     Running   0          20m
hello-tech-world-test-6b7649bcd-nsjs9   1/1     Running   0          3m28s
BApro:~/environment $
```

Figure 57: 4 pods running

CHALLENGES

There are some challenges faced to accomplish this project. The installation process for Kubernetes was kind of complex and took a long time to have it right. Many mistakes were faced during working on this project and debugging them was very challenging. Another challenge was learning about AWS EKS and Fargate in a limited time.

LIMITATIONS

Simply, scaling and rolling updated a deployed containerized web application that run on the Amazon Web Services cloud infrastructure using its Kubernetes service (EKS) and AWS Fargate is the scope of this project. In addition, the budget was a significant limitation in this project. Despite AWS offering its services at an affordable cost, the services that used in this project are costly. Lack of expertise in the AWS services especially EKS installation that used in this project was limitations at the beginning of this project. There are other limitations to using EKS with Fargate. 4 vCPU and 30 Gb memory are resource limitations per pod. It does not support stateful workloads which require file systems or persistent volumes. Also, anything run with Fargate is temporary, which means it only exists for the duration of the pod's existence. Load balancer options to use are Application Load Balancer (ALB) and Network Load Balancer (NLB)[26].

CONCLUSION

In this project, the Amazon Web Services (AWS) cloud platform and some of its services are used, such as AWS EKS and Fargate, to manage deployed containerized web applications. Rolling update strategy used in the deployment and Kubernetes Horizontal Pod Autoscaler (HPA) used for auto-scaling pods which scale-out when CPU exceeds the target and scale in when it is below the target. Achieved zero downtime when performing a rolling update deployment was the main outcome of this project.

FUTURE WORK

This project can be applied in the reality. Small business and companies can use their own web applications that include more services and concepts such as storage or monitoring and use this project methodology to provide access to their web application without facing any issue with availability and scalability. Also, this project can be enhanced by adding more services like Cloud Watch to monitor the recourses consumption of the web application.

REFERENCES:

[1]A. Cepuc, R. Botez, O. Craciun, I. -A. Ivanciu and V. Dobrota, "Implementation of a Continuous Integration and Deployment Pipeline for Containerized Applications in Amazon Web Services Using Jenkins, Ansible and Kubernetes," 2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet), 2020.

[2]R. Mode, Shubham, and R. Dangayach, "AWS Fargate - Steps To Deploy With Containerized Application". Available: https://k21academy.com/amazon-web-services/case-study-deploy-a-containerized-application-with-aws-fargate/.

[3] D. Jones, "Containers vs. Virtual Machines (VMs): What's the Difference?: NetApp Blog: Containers vs. Virtual Machines (VMs): What's the Difference?: NetApp Blog," *NetApp*, 16-Mar-2018. [Online]. Available: https://www.netapp.com/blog/containers-vs-vms/.

[4]N. A. Hansen, "Eks," *Amazon*, 1973. [Online]. Available: https://docs.aws.amazon.com/eks/latest/userguide/what-is-eks.html.

[5] Bläsi Denise and S. Enggist, "Branding through cult marketing: Fargate," *Amazon*, 2003. [Online]. Available: https://aws.amazon.com/fargate/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc&fargate-blogs.sort-by=item.additionalFields.createdDate&fargate-blogs.sort-order=desc.

[6] J. Galvis, S. James, and I. O. D. Staff, "What, Why, How: Run Serverless Kubernetes Pods Using Amazon EKS and AWS Fargate," *IOD*, 09-Mar-2020. [Online]. Available: https://iamondemand.com/blog/what-why-how-run-serverless-kubernetes-pods-using-amazon-eks-and-aws-fargate/.

[7] I. Ivanciu and V. Dobrota, "Deploying a Dockerized Application With Kubernetes on Google Cloud Platform," *2020 13th International Conference on Communications (COMM)*, 2020, pp. 471-476, doi: 10.1109/COMM48946.2020.9142014.

[8] A. Pereira Ferreira and R. Sinnott, "A Performance Evaluation of Containers Running on Managed Kubernetes Services," 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 2019, pp. 199-208, doi: 10.1109/CloudCom.2019.00038.

[9] A. Modak, S. D. Chaudhary, P. S. Paygude and S. R. Ldate, "Techniques to Secure Data on Cloud: Docker Swarm or Kubernetes?," 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), 2018, pp. 7-12, doi: 10.1109/ICICCT.2018.8473104.

[10] H. Rajavaram, V. Rajula and B. Thangaraju, "Automation of Microservices Application Deployment Made Easy By Rundeck and Kubernetes," 2019 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), 2019, pp. 1-3, doi: 10.1109/CONECCT47791.2019.9012811.

[11] Paulbouwer, “Hello-Kubernetes/Src/app at main · Paulbouwer/hello-kubernetes,” GitHub. [Online]. Available: https://github.com/paulbouwer/hello-kubernetes/tree/main/src/app

[12] “Using Amazon ECR with the AWS CLI - AWS Documentation.” [Online]. Available: https://docs.aws.amazon.com/AmazonECR/latest/userguide/getting-started-cli.html

[13] N. A. Hansen, “AWS Fargate profile,” Amazon. [Online]. Available: https://docs.aws.amazon.com/eks/latest/userguide/fargate-profile.html

[14] L. Shaat, “EKS Workshops,” Louay Shaat Workshops. [Online]. Available: https://louay-workshops.com.au/eks-immersion-day/eks/09_fargate/prerequisites-for-alb.html

[15] S. Engdahl, “introducing-aws-load-balancer-controller,” Amazon, 2008. [Online].Available:https://aws.amazon.com/blogs/containers/introducing-aws-load-balancer-controller

[16] K. Jackson, “Kubernetes Rolling Update Configuration,” Blue Matador. [Online]. Available: https://www.bluematador.com/blog/kubernetes-deployments-rolling-update-configuration

[17] “Deployments,” Kubernetes, 25-Oct-2021. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

[18] L. Ogut, “Kubernetes Probes: Startup, Liveness, readiness,” Loft Labs. [Online]. Available: https://loft.sh/blog/kubernetes-probes-startup-liveness-readiness/

[19] “Managing Resources for Containers,” Kubernetes. [Online]. Available: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/.

[20] “Service,” Kubernetes, 12-Nov-2021. [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/service/

[21] Kubernetes-Sigs, “Kubernetes-sigs/metrics-server: Scalable and efficient source of Container Resource Metrics for kubernetes built-in Autoscaling Pipelines.,” GitHub. [Online]. Available:https://github.com/kubernetes-sigs/metrics-server

[22] Horizontal pod autoscaler,” Kubernetes, 09-Sep-2021. [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/#algorithm-details

[23] “Ingress annotations,” Annotation - AWS ALB Ingress Controller. [Online]. Available: https://kubernetes-sigs.github.io/aws-load-balancer-controller/v1.1/guide/ingress/annotation/.

[24] “Ingress,” *Kubernetes*, 12-Nov-2021. [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/ingress/

[25] “Kubernetes requests vs limits: Why adding them to your pods and namespaces matters | google cloud blog,” *Google*. [Online]. Available: https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-best-practices-resource-requests-and-limits

[26] G. Chandra, “Eks + Fargate = extensibility of Kubernetes + Serverless Benefits,” Medium, 06-Dec-2019. [Online]. Available: https://itnext.io/eks-fargate-extensibility-of-kubernetes-serverless-benefits-77599ac1763

[27] “An Introduction to Containers,” *Rancher Labs*, 30-May-2019. [Online]. Available: https://rancher.com/blog/2019/an-introduction-to-containers.

[28] R. Mode, Shubham, and R. Dangayach, “AWS Fargate - Steps To Deploy With Containerized Application,” *Cloud Training Program*, 27-May-2021. [Online]. Available: https://k21academy.com/amazon-web-services/case-study-deploy-a-containerized-application-with-aws-fargate/.

[29] “Run Serverless Kubernetes Pods Using Amazon EKS and AWS Fargate,” *Amazon*, 2018. [Online]. Available: https://aws.amazon.com/about-aws/whats-new/2019/12/run-serverless-kubernetes-pods-using-amazon-eks-and-aws-fargate/.

[30] “Performing a Rolling Update,” Kubernetes, 03-Feb-2020. [Online]. Available: https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/.

[31] “Amazon EKS on AWS Fargate Now Generally Available” Amazon, 2019. [Online]. Available: https://aws.amazon.com/blogs/aws/amazon-eks-on-aws-fargate-now-generally-available/.

[32] A. Geller, “Serverless Kubernetes Cluster on AWS with EKS on Fargate,” *Medium*, 12-Dec-2020. [Online]. Available: https://betterprogramming.pub/serverless-kubernetes-cluster-on-aws-with-eks-on-fargate-a7545cf179be

[33] “Pod lifecycle,” Kubernetes, 17-Oct-2021. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#container-probes.