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
- Phase 2: Implantation and Testing .......................................................... 27

CHALLENGES ............................................................................................ 35

LIMITATIONS ............................................................................................. 35

CONCLUSION ............................................................................................. 35

FUTURE WORK .......................................................................................... 36

REFERENCES .............................................................................................. 37
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Virtual machines vs. Containers [3]</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>Using AWS EKS to Deploy Applications [6]</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>Project Architecture</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>AWS Account</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>AWS cloud9 EC2 instance information</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>IAM role</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>Check eksct1 version</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>Check AWS CLI version</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>Check Kubect1 version</td>
<td>15</td>
</tr>
<tr>
<td>10</td>
<td>Check docker status</td>
<td>15</td>
</tr>
<tr>
<td>11</td>
<td>AWS ECR</td>
<td>16</td>
</tr>
<tr>
<td>12</td>
<td>Authenticate to the registry</td>
<td>16</td>
</tr>
<tr>
<td>13</td>
<td>Build first docker image with tag latest</td>
<td>16</td>
</tr>
<tr>
<td>14</td>
<td>First Image pushed to the ECR</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>Build second docker image with tag latest2</td>
<td>17</td>
</tr>
<tr>
<td>16</td>
<td>Second image pushed to ECR</td>
<td>17</td>
</tr>
<tr>
<td>17</td>
<td>Two images information from console</td>
<td>17</td>
</tr>
<tr>
<td>18</td>
<td>Two images information from terminal</td>
<td>17</td>
</tr>
<tr>
<td>19</td>
<td>Createcluster yaml file</td>
<td>18</td>
</tr>
<tr>
<td>20</td>
<td>EKS cluster with Fargate profile creating using eksct1 command</td>
<td>18</td>
</tr>
<tr>
<td>21</td>
<td>Cloud formation stacks</td>
<td>19</td>
</tr>
<tr>
<td>22</td>
<td>Cluster resources created automatically by CloudFormation</td>
<td>19</td>
</tr>
<tr>
<td>23</td>
<td>Configure access to the EKS cluster</td>
<td>20</td>
</tr>
<tr>
<td>24</td>
<td>IAM OIDC provider</td>
<td>20</td>
</tr>
<tr>
<td>25</td>
<td>IAM policy</td>
<td>20</td>
</tr>
<tr>
<td>26</td>
<td>IAM service account</td>
<td>21</td>
</tr>
<tr>
<td>27</td>
<td>Install the TargetGroupBinding CRDs</td>
<td>21</td>
</tr>
<tr>
<td>28</td>
<td>Checking CRDs</td>
<td>21</td>
</tr>
<tr>
<td>29</td>
<td>Add the EKS repository to Helm and Install the AWS Load Balancer controller using iamserviceaccount</td>
<td>21</td>
</tr>
<tr>
<td>30</td>
<td>Check the pods by using kubectl get command</td>
<td>21</td>
</tr>
<tr>
<td>31</td>
<td>Check the deployment completed</td>
<td>21</td>
</tr>
<tr>
<td>32</td>
<td>Deployment part from the deployment yaml file</td>
<td>22</td>
</tr>
<tr>
<td>33</td>
<td>Service part from depolymnt.yaml file</td>
<td>25</td>
</tr>
<tr>
<td>34</td>
<td>Service information by run get svc</td>
<td>25</td>
</tr>
<tr>
<td>35</td>
<td>Installed Metrics Server</td>
<td>25</td>
</tr>
<tr>
<td>36</td>
<td>HPA part from deployment.yaml file</td>
<td>26</td>
</tr>
<tr>
<td>37</td>
<td>Ingress yaml file</td>
<td>26</td>
</tr>
<tr>
<td>38</td>
<td>Deployment implemented 1</td>
<td>27</td>
</tr>
<tr>
<td>39</td>
<td>Check the web application Hello Tech World V1</td>
<td>28</td>
</tr>
<tr>
<td>40</td>
<td>Deployment implemented 2</td>
<td>28</td>
</tr>
<tr>
<td>41</td>
<td>Check the web application Hello Tech World V2</td>
<td>28</td>
</tr>
<tr>
<td>42</td>
<td>Rolling update preformed</td>
<td>29</td>
</tr>
<tr>
<td>43</td>
<td>Rollout status</td>
<td>29</td>
</tr>
<tr>
<td>44</td>
<td>Check pod status</td>
<td>29</td>
</tr>
<tr>
<td>45</td>
<td>Check the web application Hello Tech World V1</td>
<td>30</td>
</tr>
<tr>
<td>46</td>
<td>pod information</td>
<td>30</td>
</tr>
<tr>
<td>47</td>
<td>Testing code</td>
<td>31</td>
</tr>
<tr>
<td>48</td>
<td>Rolling update preformed</td>
<td>31</td>
</tr>
<tr>
<td>49</td>
<td>Testing code result</td>
<td>31</td>
</tr>
<tr>
<td>50</td>
<td>Check pod status</td>
<td>32</td>
</tr>
<tr>
<td>51</td>
<td>Check the web application Hello Tech World V2 after rolling update</td>
<td>32</td>
</tr>
<tr>
<td>52</td>
<td>Generating load</td>
<td>33</td>
</tr>
</tbody>
</table>
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](image)

**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].

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

![Project Architecture](image)

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.
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.

IAM role created for this workspace and name it as BA-admin and it attached to it.
Step 3: Installed required tools

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

![Figure 7: Check eksctl version](image)

![Figure 8: Check AWS CLI version](image)

![Figure 9: Check Kubectl version](image)

![Figure 10: Check docker status](image)

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.
It is important and required to authenticate the Docker CLI to the registry so docker command can push and pull images. [12]
Figure 15: Build second docker image with tag latest2

Figure 16: Second image pushed to ECR

Figure 17: Two images information from console

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].
Figure 21: Cloud formation stacks

Figure 22: Cluster resources created automatically by CloudFormation
This command used to configure access to the cluster.

```
Bpro:/environment/Bpro/Bpro $ aws eks --region us-east-2 update-kubeconfig --name Bpro-EKS-cluster-fargate --profile Bpro
Bpro:/environment/Bpro/Bpro/Bpro $
```

*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]

```
Bpro:/environment/Bpro $ kubectl create-policy
...
```

*Figure 24 : IAM OIDC provider*

```
Bpro:/environment/Bpro $ kubectl create-policy
...
```

*Figure 25 : IAM policy*
Figure 26: IAM service account

Figure 27: Install the TargetGroupBinding CRDs

Figure 28: Checking CRDs

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

Figure 30: Check the pods by using kubectl get command

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 kubectl. 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.

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.template 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 maxSugur,maxUnavailable, 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.

maxSugar 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.

```
---
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 depolyment.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].

```
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].

```
$ kubectl apply f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

*Figure 35: Installed Metrics Server*
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].
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

*Figure 38: Deployment implemented 1*
Performed rolling update by edited container image in the container section in the deployment.yaml file.

Figure 39: Check the web application Hello Tech World V1

Figure 40: Deployment implemented 2

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

```
```

**Figure 42:** Rolling update preformed

```
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.

```
                                 READY   STATUS     RESTARTS   AGE
hello-tech-world-test-6b76f49c-dpc8c   1/1   Running   0          7m25s
hello-tech-world-test-796e6c5688-xplqg   0/1   Terminating   0          8m5s
```

**Figure 44:** Check pod status
Figure 45: Check the web application Hello Tech World V1

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.

![Testing code](image1)

**Figure 47: Testing code**

![Rolling update performed](image2)

**Figure 48: Rolling update performed**

![Testing code result](image3)

**Figure 49: Testing code result**
Used get pods -w command to get clear vision of the process.

Figure 50: Check pod status

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.

Figure 52: Generating load

Figure 53: Describe HPA

Figure 53: Auto scale out With CPU 25%
Figure 54: 8 pods running

Scale in after stop the load generator

Figure 55: Auto scale in with CPU 25%

Used Ctrl+C to stop load generator.

Figure 56: Another try with CPU target 50%

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:


