Table of Contents

1.Kubernetes:

Figure 1/ Kubernetes Logo

1.1 Introduction to Kubernetes:

Kubernetes is an open-source platform It allows you to easily manage clusters of containers and helps you to abstract away the underlying infrastructure, making it easier to deploy and manage applications in a consistent and scalable way.

The basic concepts:

- Nodes (Minions):

Is a machine, physical or virtual on which Kubernetes is installed. A node is a worker machine and that is where containers will be launched by Kubernetes.

Cluster:

A Cluster is a set of nodes grouped together. This way even if one node fails you have your application still accessible from the other nodes.

- Master:

The Master is another node with Kubernetes installed in it and is configured as a Master. The master watches over the nodes in the cluster and is responsible for the actual orchestration of containers on the worker nodes. Use it When a node fails to move the workload of the field node to another worker node.

1.2 Kubernetes components:

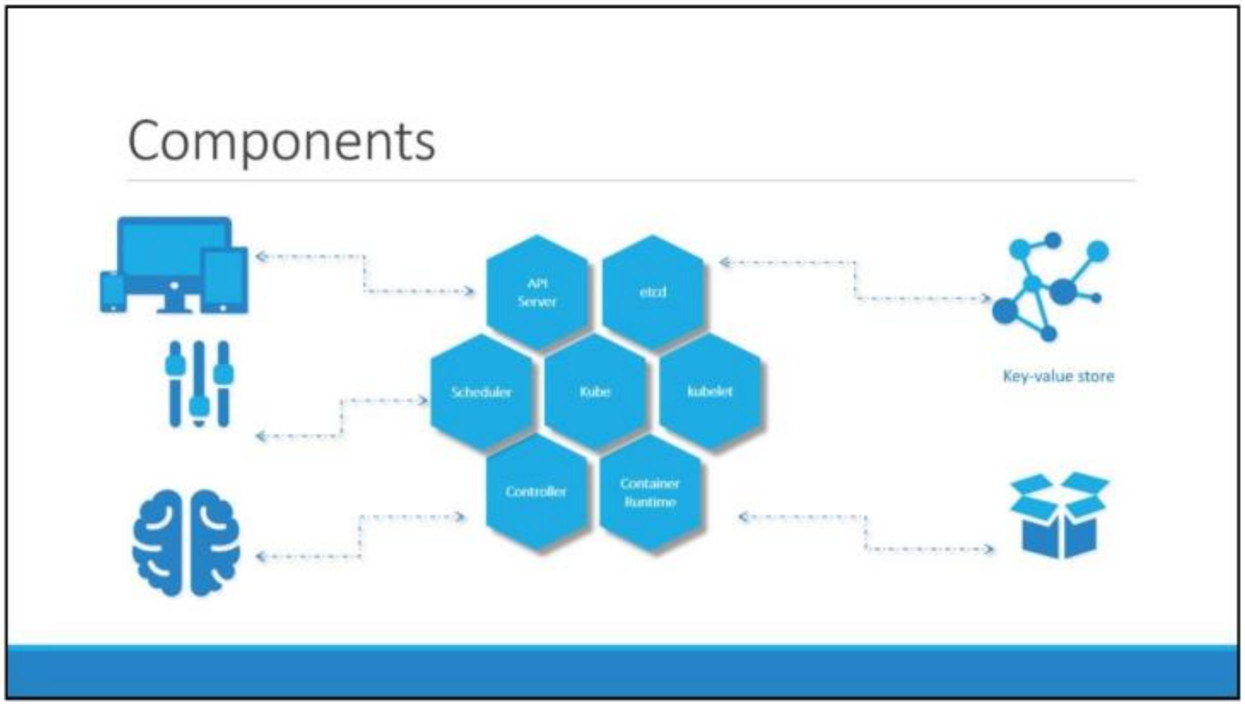


Figure 2/ Kubernetes Componets

1. The API server:

Acts as the front end for Kubernetes. The users, management devices, command line interfaces, all talk to the API server to interact with Kubernetes cluster.

2. ETCD:

Is a distributed reliable key value store used by Kubernetes to store all data used to manage the cluster.

3. The Scheduler:

Is responsible for distributing work or containers across multiple nodes. It looks for newly created containers and assigns them to nodes.

4. The controllers:

This is responsible for noticing and responding when nodes, containers, or end points go down. The controllers make decisions.

5. The Container Runtime:

Is the underlying software that is used to run containers like Docker.

6. Kubelet:

Is the agent that runs on each node in the cluster. The agent is responsible for making sure that the containers are running on the nodes as expected.

1.3 Master and Worker Nodes:

How does one server become a master and the other Worker?

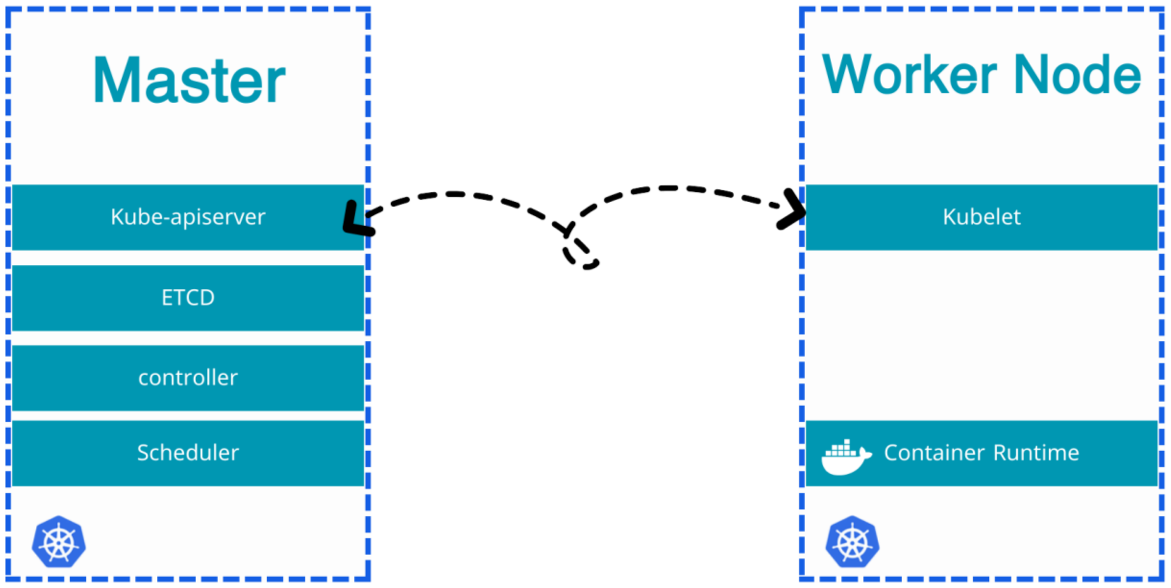


Figure 3/ Difference between Master and Worker Nodes

The worker node is where the containers are hosted. For example, Docker containers. So, to run docker containers we need container runtime installed and that's where the container runtime falls. the worker nodes have the kubelet agent that is responsible for interacting with a master, while The master server has the kube API server and that is what makes it a master.

1.4 Network Services in k8s:

- NodePort:

This type of service exposes the service on each Node’s IP at a static port (the NodePort). A ClusterIP service, to which the NodePort service routes, is automatically created, It makes the service accessible from outside the Kubernetes cluster by requesting <NodeIP>:<NodePort>
use for simple setups or development environments where you need to expose a service externally without using a LoadBalancer.

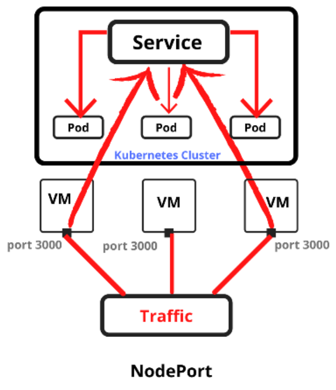


Figure 4/ NodePort service

- LoadBalancer:

This type of service is used to expose the service externally using a cloud provider's load balancer, It automatically creates a LoadBalancer on the cloud provider (e.g., AWS, GCP, Azure) and assigns a public IP to the service, making it accessible from outside the cluster.
 uses to production environments where you need to expose services to the internet with built-in load balancing and redundancy.

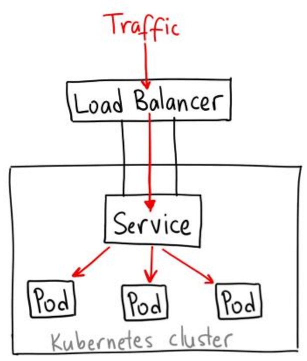


Figure 5/ LoadBalancer Service

- Cluster IP:

This is the default type of service in Kubernetes. It exposes the service on a cluster-internal IP, this type of service can only be accessed within the Kubernetes cluster. It is used for internal communication between services within the cluster. We can use it for microservices architectures where one service needs to communicate with another service within the same cluster.

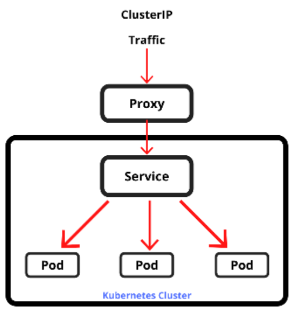


Figure 6/ ClusterIP Service

2. Kubernetes on AWS:

Involves deploying and managing Kubernetes clusters using cloud services provided by various vendors like Elastic Kubernetes Service (EKS) for AWS.

EKS is a fully-managed, certified Kubernetes conformant service that simplifies the process of building, securing, operating, and maintaining Kubernetes clusters on AWS. Amazon EKS integrates with core AWS services such as CloudWatch, Auto Scaling Groups, and IAM to provide a seamless experience for monitoring, scaling, and load balancing my containerized applications.

Managed Control Plane: AWS manages the Kubernetes control plane, ensuring it is highly available and scalable.

Integration with AWS Services: EKS integrates seamlessly with various AWS services, including IAM for security, CloudWatch for monitoring, and more.

Compliance and Security: EKS is compliant with various security standards and provides out-of-the-box security features to protect your applications.

2.1 Type of Cluster in EKS:

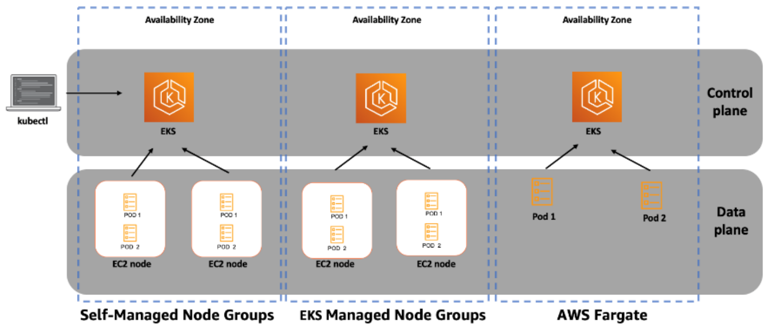


Figure 7/ Type of cluster

- Managed node groups:

Managed node groups in EKS allow users to create and manage groups of EC2 instances to run their containerized applications. AWS manages the lifecycle of these nodes, including updates and scaling, making it easier to maintain the infrastructure.

- Self-Managed Nodes:

have complete control over the EC2 instances running in their EKS clusters. This option provides more flexibility but requires users to manage the lifecycle of the nodes themselves.

- AWS Fargate:

AWS Fargate is a serverless compute engine for containers that works with EKS. Fargate eliminates the need to manage EC2 instances, allowing users to focus solely on deploying and managing applications.

2.1 Benefits of Using AWS Fargate with EKS:

Serverless Operation and This is the most important point so there is no need to provision, configure, or scale clusters of virtual machines to run containers. AWS handles the infrastructure management, allowing users to focus on their applications.

Some other benefits:

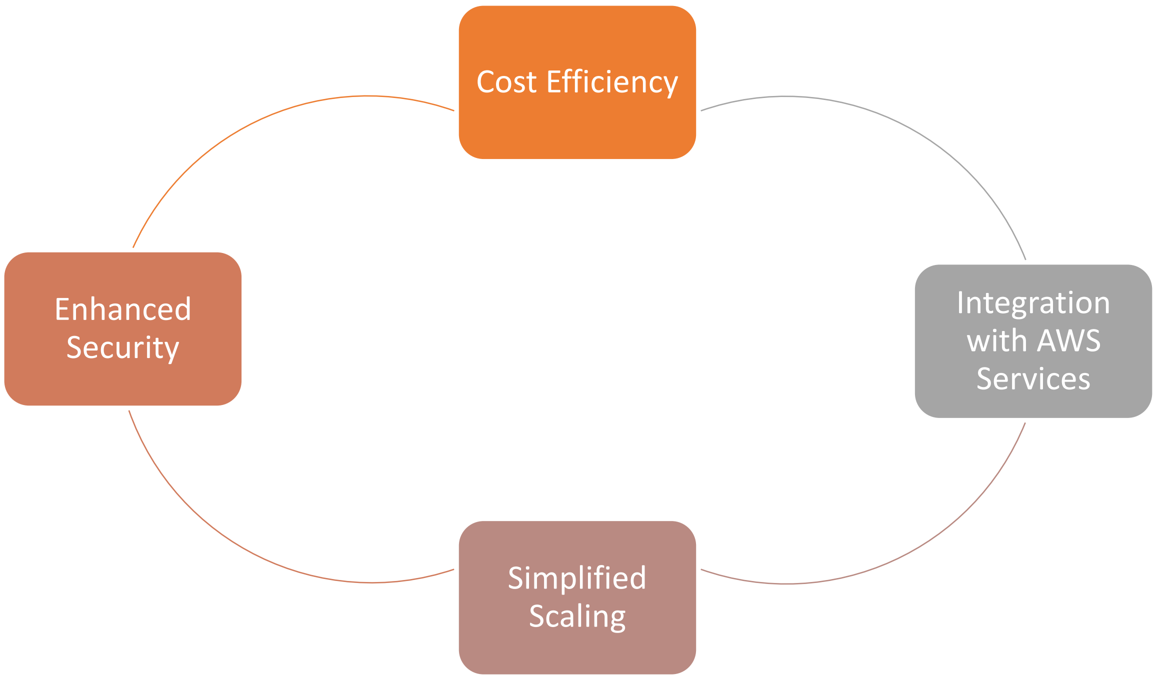


Figure 8/ Benefits of Using AWS Fargate with EKS

Fargate is ideal for running microservices architectures, allowing for independent scaling and management of each service.

3.Task: Create Microservice and run It in fargate cluster and access it via URL:

I will divide the task into three parts: the first part related to creating the microservice, the second part related to creating the cluster, and the third part related to deploying and running the microservice.

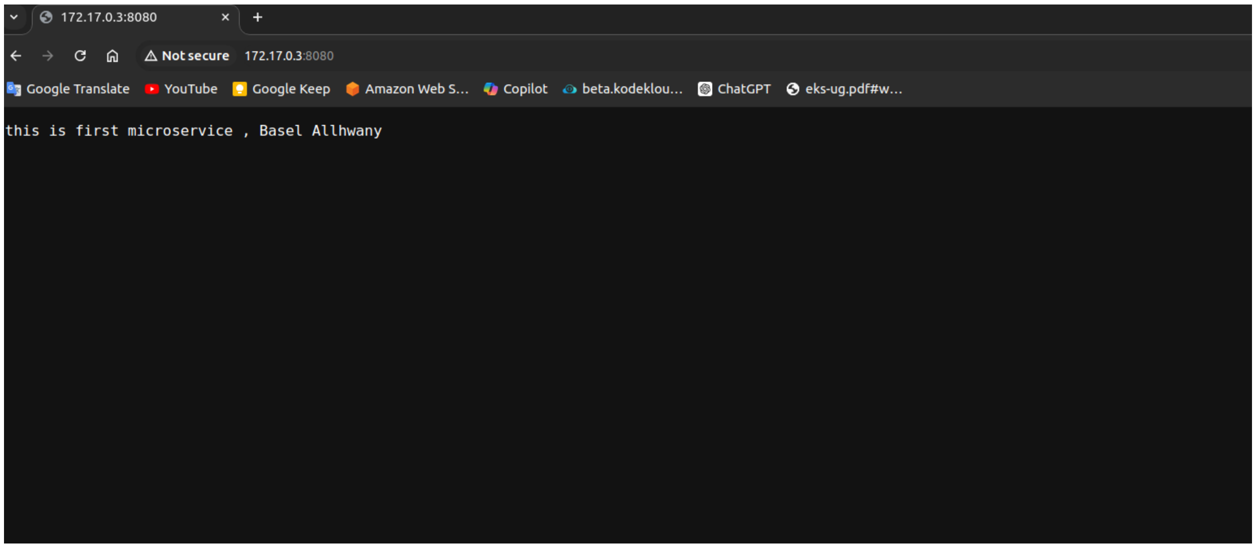
3.1 Part 1: Create Microservice:

I will create code .NET for application display sentence “this is first microservice, Basel Allhwany” and then containerized the app using Docker
The steps:
1) Install the .NET SDK

2) Install helloworld .NET Application “**dotnet new console -n HelloWorldApp**”

3) Edit the Program.cs File to include this sentence.
4) Create Dockerfile to containerized the app
5) build image and run the container
6) using docker inspect CONTAINER ID To find out its IP address

6) check my container from browser using Ip address:port



```
this is first microservice , Basel Allhwany
```

7) Deploy me image to ECR to use it in AWS to do it I need create ECR Repository and make authenticate Docker to me ECR Registry then tag docker image using finaly I will push it using those command:

# Create an ECR repository

```
aws ecr create-repository --repository-name helloworldapp --region <your-region>
```

# Authenticate Docker to your ECR registry

```
aws ecr get-login-password --region <your-region> | docker login --username AWS --password-stdin
<aws_account_id>.dkr.ecr.<your-region>.amazonaws.com
```

# Tag your Docker image

```
docker tag helloworldapp:latest <aws_account_id>.dkr.ecr.<your-
region>.amazonaws.com/helloworldapp:latest
```

# Push your Docker image to ECR

```
docker push <aws_account_id>.dkr.ecr.<your-region>.amazonaws.com/helloworldapp:latest
```

3.2 Part 2: Creating the Cluster:

Step 1: Create a VPC using the CloudFormation template:

1- Download Template using

```
$ https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-
private-subnets.yaml
```

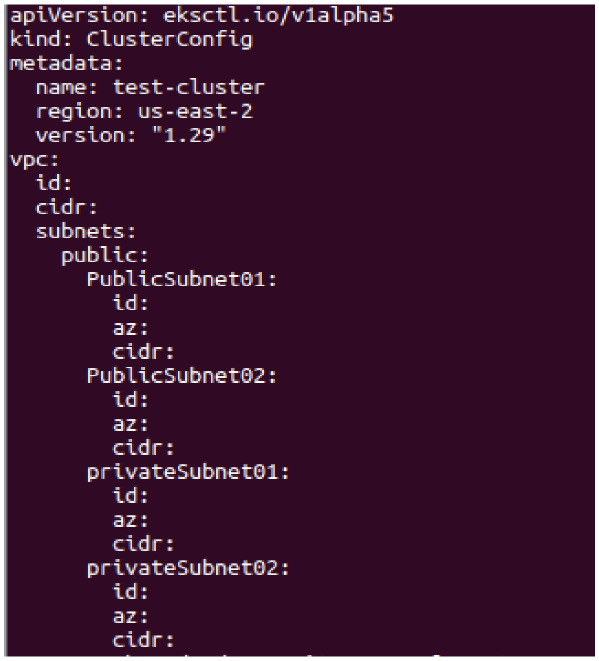
2- Create Stack in CloudFormation:

```
$ aws cloudformation create-stack --region us-east-2 --stack-name vpc-test --template-body
file:///home/basel/amazon-eks-vpc-private-subnets.yaml
```

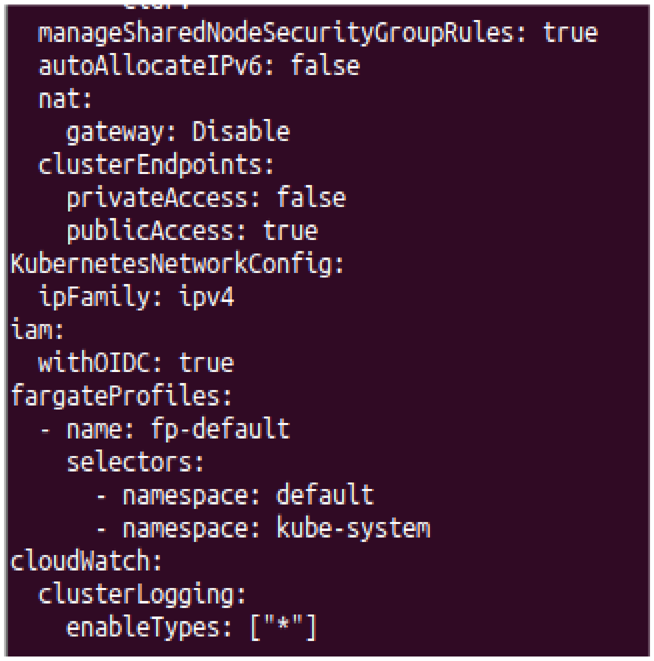
3- Wait for Completion

4- Verify Stack Status

Step 2: Create a fargate cluster using eksctl create cluster:
1- create file yaml file like it :



```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: test-cluster
  region: us-east-2
  version: "1.29"
vpc:
  id:
  cidr:
  subnets:
    public:
      PublicSubnet01:
        id:
        az:
        cidr:
      PublicSubnet02:
        id:
        az:
        cidr:
      privateSubnet01:
        id:
        az:
        cidr:
      privateSubnet02:
        id:
        az:
        cidr:
```



```
  manageSharedNodeSecurityGroupRules: true
  autoAllocateIPv6: false
  nat:
    gateway: Disable
  clusterEndpoints:
    privateAccess: false
    publicAccess: true
KubernetesNetworkConfig:
  ipFamily: ipv4
iam:
  withOIDC: true
fargateProfiles:
  - name: fp-default
    selectors:
      - namespace: default
      - namespace: kube-system
cloudWatch:
  clusterLogging:
    enableTypes: ["*"]
```

2-Deploy the file to create cluster using:

$ eksctl create cluster -f cluster-fargate.yaml -v 5

3- Wait for Completion

4- Verify cluster event

3.3 part 3: Deploy microservice to a new cluster:

Firstly I want to install AWS Load Balancer Controller ALB using helm
steps to do it:
step 0: Install the necessary CRDs:

kubectl apply -k "github.com/aws/eks-charts/stable/aws-load-balancer-controller//crds?ref=master"

now I need Associate an IAM OIDC provider with my EKS cluster:

eksctl utils associate-iam-oidc-provider --region <region-code> --cluster <cluster-name> --approve

step 1: Create an IAM policy:

# Download an IAM policy for the AWS Load Balancer Controller that allows it to make calls to AWS APIs on your behalf.

$ **curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.7.2/docs/install/iam_policy.json**

# Create an IAM policy using the policy downloaded in the previous step

```
$ aws iam create-policy \

   --policy-name AWSLoadBalancerControllerIAMPolicy \

   --policy-document file://iam_policy.json
```

step 2: Create serviceaccount :

```
eksctl create iamserviceaccount \

  --cluster=my-cluster \
  --namespace=kube-system \
  --name=aws-load-balancer-controller \
  --role-name AmazonEKSLoadBalancerControllerRole \
  --attach-policy-arn=arn:aws:iam::ID_account:policy/AWSLoadBalancerControllerIAMPolicy \
  --approve
```

step 3: Install AWS Load Balancer Controller:

# Add the eks-charts Helm chart repository

```
$ helm repo add eks https://aws.github.io/eks-charts
```

# Update your local repo to make sure that you have the most recent charts.

```
$ helm repo update eks
```

# Install the AWS Load Balancer Controller.

```
$ helm install aws-load-balancer-controller eks/aws-load-balancer-controller \

  -n kube-system \
  --set clusterName=my-cluster \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller
```

- `--set region=region-code`
- `--set vpcId=vpc-xxxxxxxx`

  Verify that the controller is installed using this command:

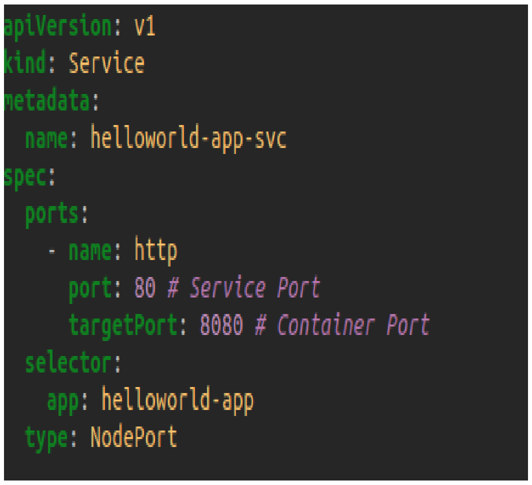
  ```
  $ kubectl get deployment -n kube-system aws-load-balancer-controller
  ```

Create new namespace farget profile

```
eksctl create fargateprofile \
```

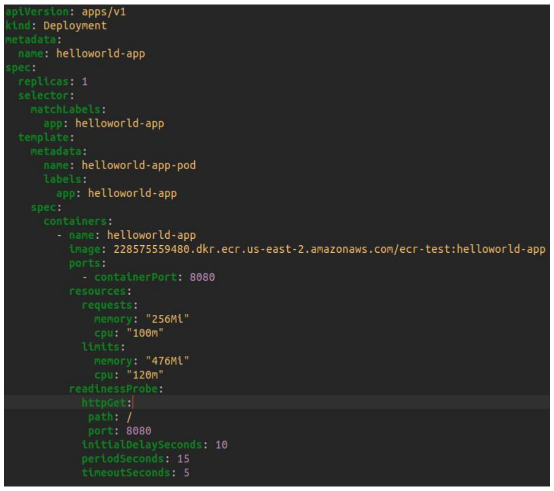
```
--cluster test-cluster \
--name fp-test \
--namespace '*'
```

And here we have three files that need to be created: deployment, service, and ingress:



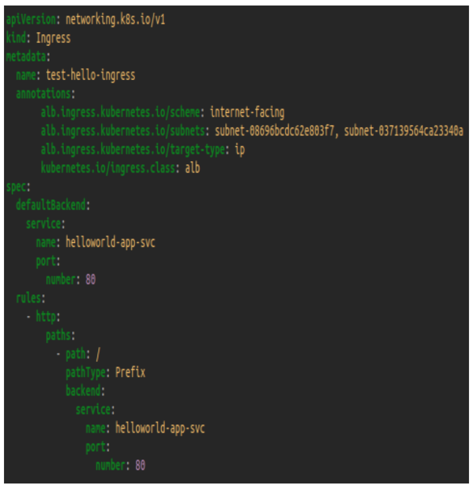
```yaml
apiVersion: v1
kind: Service
metadata:
  name: helloworld-app-svc
spec:
  ports:
    - name: http
      port: 80 # Service Port
      targetPort: 8080 # Container Port
  selector:
    app: helloworld-app
  type: NodePort
```

Figure 11/ NodePort service



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: helloworld-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: helloworld-app
  template:
    metadata:
      name: helloworld-app-pod
      labels:
        app: helloworld-app
    spec:
      containers:
        - name: helloworld-app
          image: 228575559480.dkr.ecr.us-east-2.amazonaws.com/ecr-test:helloworld-app
          ports:
            - containerPort: 8080
          resources:
            requests:
              memory: "256Mi"
              cpu: "100m"
            limits:
              memory: "476Mi"
              cpu: "120m"
          readinessProbe:
            httpGet:
             path: /
             port: 8080
            initialDelaySeconds: 10
            periodSeconds: 15
            timeoutSeconds: 5
```

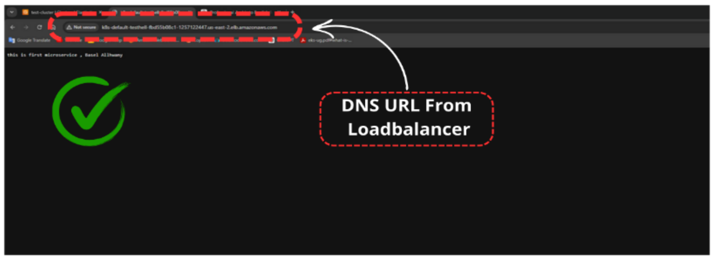
Figure 10/ Deployment.yaml



```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: test-hello-ingress
  annotations:
      alb.ingress.kubernetes.io/scheme: internet-facing
      alb.ingress.kubernetes.io/subnets: subnet-08696bcdc62e803f7, subnet-037139564ca23340a
      alb.ingress.kubernetes.io/target-type: ip
      kubernetes.io/ingress.class: alb
spec:
  defaultBackend:
    service:
      name: helloworld-app-svc
      port:
        number: 80
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: helloworld-app-svc
                port:
                  number: 80
```

Figure 9/ ingress.yaml

Using **Kubectl apply -f path** to
file to deploy this file in cluster then we must wait until everything is created successfully.
Now we can go to ec2 > LoadBalancer and access the DNS URL to Access the service.
note: must edit the security groups to allow to all traffic inbound



```
k8s-default-testhell-fbd55b08c1-1257122447.us-east-2.elb.amazonaws.com
this is first microservice , Basel Allhwany
```

4.Challenges Faced and Their Solutions

4.1 Error: failed to create cluster "test-cluster":

Full error :

```
creating CloudFormation stack "eksctl-test-cluster-cluster": operation error CloudFormation:
CreateStack, https response error StatusCode: 403, RequestID: 411f917d-93a7-4974-ab64-
4004b36915b1, api error AccessDenied: User:
arn:aws:iam::XXXXXXX:user/b.lhwany@sanadak.sa is not authorized to perform:
cloudformation:CreateStack on resource: arn:aws:cloudformation:us-east-2:XXXXX:stack/eksctl-
test-cluster-cluster/* because no identity-based policy allows the cloudformation:CreateStack
action
Error: failed to create cluster "test-cluster"
```

The Solution:
The error comes because I don't have the necessary permissions to execute Cloudformation:CreateStack so I need add this role to ma iam user :

```
{
          "Effect": "Allow",
          "Action": "cloudformation:CreateStack",
          "Resource": "*"
       }
   ]
}
```

4.2 Subnets specified must be in at least two different AZs:

Full error:

```
  AWS::EKS::Cluster/ControlPlane: CREATE_FAILED – "Resource handler returned message:
\"Subnets specified must be in at least two different AZs (Service: Eks, Status Code: 400,
Request ID: bf11031e-4fbe-425c-985d-f99844ab4cc9)\" (RequestToken: 98c0bb69-f65f-ae92-
a1b7-91ecc7af1573, HandlerErrorCode: InvalidRequest)"
```

The Solution:
The error comes because I have all subnet in the same AZs
so I need to update the vpc to create subnet in at least two different AZs

4.3 0/2 nodes are available:

Full Error:

```
0/2 nodes are available: 2 node(s) had untolerated taint {eks.amazonaws.com/compute-type: fargate}. preemption: 0/2 nodes are available: 2 Preemption is not helpful for scheduling.
```

The Solution:

I need create a new `fargateprofile`

4.4 helloworld-app-ingress not have address:

Full error:

```
$ kubectl get ingress/helloworld-app-ingress -n helloworld
```

```
NAME                    CLASS   HOSTS   ADDRESS   PORTS   AGE
helloworld-app-ingress   alb      *                 80      71s
```

The Solution:

There are several reasons for this problem, one of which could be a defect in AWS Load Balancer Controller installation so reinstall it as we explained earlier.