Contents

1. Introduction

- AWS Fargate is a serverless container management service (container as a service) that allows developers to focus on their application and not their infrastructure.
- AWS Fargate allows you to build and manage applications using serverless containers and works with both ECS (Elastic Container Service) and EKS (Elastic Kubernetes Service).
- AWS Fargate is an improvement on ECS because it allows to manage containers without servers.
- AWS Fargate works more efficiently because it allows you to manage your containers without creating a cluster of virtual machines.
- Fargate separates the task of running containers from the task of managing the underlying infrastructure.
- Users can simply specify the resources that each container requires, and Fargate will handle the rest.
- For example, there's no need to select the right server type, or fiddle with complicated multi-layered access rules.

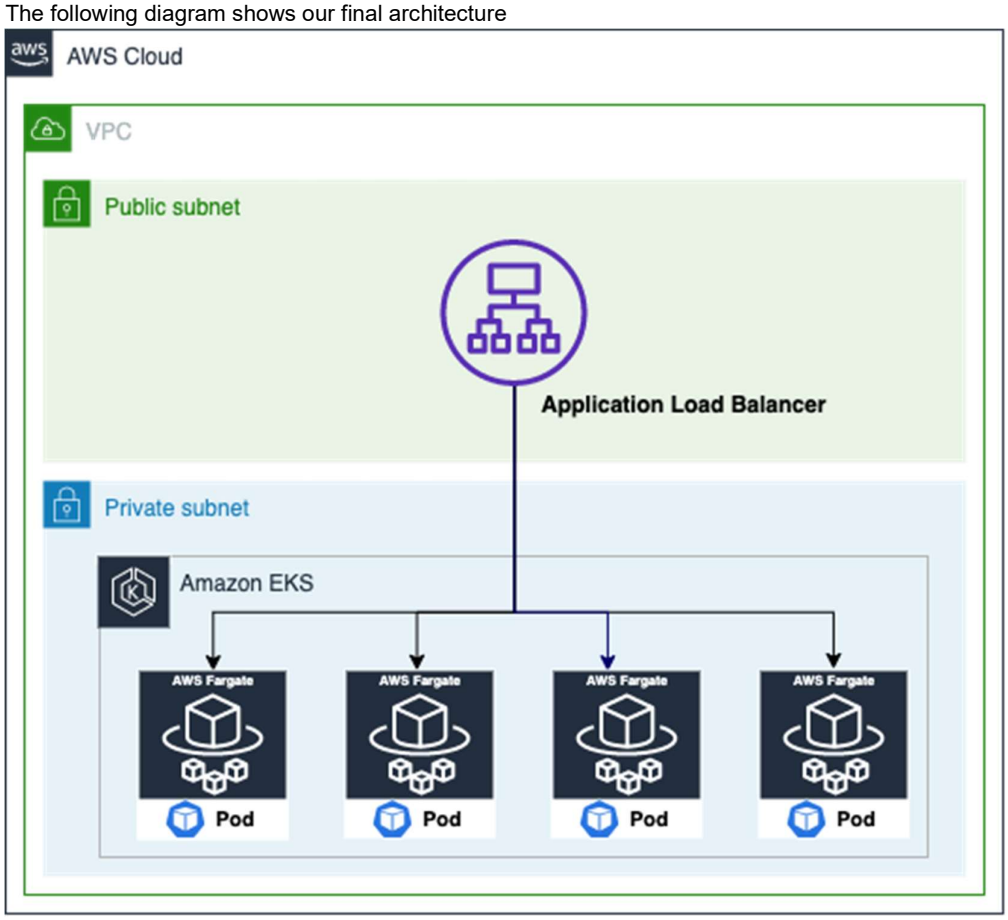
➢ Requirements.

- How to setup AWS Application Load Balancer (ALB) with your EKS cluster for ingress-based load balancing to Fargate pods using the open-source ALB Ingress Controller.

➢ Following Steps are required to build / implement.

- create an Amazon EKS cluster and a Fargate profile (which allows us to launch pods on Fargate),
- implement IAM roles for service accounts on our cluster in order to give fine-grained IAM permissions to our ingress controller pods,
- deploy a simple nginx service, and expose it to the internet using an ALB.

➢ The following diagram shows our final architecture



1.1. Prerequisites

- The EKS CLI, eksctl,
- The latest version of the AWS CLI
- The Kubernetes CLI, kubectl .
- jq

1.2. Cluster provisioning

- Setup the environment variables that we will be using

```
# AWS_REGION=<aws_region>

# CLUSTER_NAME=eks-fargate-alb-demo

# STACK_NAME=eksctl-$CLUSTER_NAME-cluster

# VPC_ID=$(aws cloudformation describe-stacks --stack-name "$STACK_NAME" |
jq -r '[.Stacks[0].Outputs[] | {key: .OutputKey, value: .OutputValue}] |
from_entries' | jq -r '.VPC')

# AWS_ACCOUNT_ID=$(aws sts get-caller-identity | jq -r '.Account')
```

- The first step is to create an Amazon EKS cluster using eksctl.

```
# eksctl create cluster --name $CLUSTER_NAME --region $AWS_REGION --fargate
```

Note: remember to replace <aws_region> with the Region that you are using (Eg.: us-east-1, us-east-2, eu-west-1, or ap-northeast-1)

- Validate the cluster status.

```
# kubectl get svc
NAME TYPE   CLUSTER-IP    EXTERNAL-IP          PORT(S)       AGE
kubernetes  ClusterIP     10.100.0.1 <none>    443/TCP       16h
```

- Setup the OIDC ID provider (IdP) in AWS. This step is needed to give IAM permissions to a Fargate pod running in the cluster using the IAM for Service Accounts feature.

```
# eksctl utils associate-iam-oidc-provider --cluster $CLUSTER_NAME --approve
```

- Create the IAM policy that will be used by the ALB Ingress Controller deployment.

```
# wget -O alb-ingress-iam-policy.json
https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/master/docs/examples/iam-policy.json

# aws iam create-policy --policy-name ALBIngressControllerIAMPolicy --policy-
document file://alb-ingress-iam-policy.json
```

- Create the Cluster Role and Role Binding:

```
# vi rbac-role.yaml
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    app.kubernetes.io/name: alb-ingress-controller
  name: alb-ingress-controller
rules:
  - apiGroups:
      - ""
```

```
      - extensions
    resources:
      - configmaps
      - endpoints
      - events
      - ingresses
      - ingresses/status
      - services
    verbs:
      - create
      - get
      - list
      - update
      - watch
      - patch
  - apiGroups:
      - ""
      - extensions
    resources:
      - nodes
      - pods
      - secrets
      - services
      - namespaces
    verbs:
      - get
      - list
      - watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    app.kubernetes.io/name: alb-ingress-controller
  name: alb-ingress-controller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: alb-ingress-controller
subjects:
  - kind: ServiceAccount
    name: alb-ingress-controller
    namespace: kube-system
```

```
# kubectl apply -f rbac-role.yaml
```

- Create Kubernetes Service Account:

```
# eksctl create iamserviceaccount \
--name alb-ingress-controller \
--namespace kube-system \
--cluster $CLUSTER_NAME \
--attach-policy-arn arn:aws:iam::$AWS_ACCOUNT_ID:policy/ALBIngressControllerIAMPolicy \
--approve
```

Example:

```
# eksctl create cluster \
--name fargate-tutorial-cluster \
--version 1.14 \
--region us-east-2 \
--fargate \
--alb-ingress-access
```

1.3. Deploy the ALB Ingress Controller

```
# vi alb-ingress-controller.yaml
  apiVersion: apps/v1
  kind: Deployment
  metadata:
    labels:
      app.kubernetes.io/name: alb-ingress-controller
    name: alb-ingress-controller
    namespace: kube-system
  spec:
    selector:
      matchLabels:
        app.kubernetes.io/name: alb-ingress-controller
    template:
      metadata:
        labels:
          app.kubernetes.io/name: alb-ingress-controller
      spec:
        containers:
        - name: alb-ingress-controller
          args:
          - --ingress-class=alb
          - --cluster-name=$CLUSTER_NAME
          - --aws-vpc-id=$VPC_ID
          - --aws-region=$AWS_REGION
          image: docker.io/amazon/aws-alb-ingress-controller:v1.1.6
        serviceAccountName: alb-ingress-controller


# kubectl apply -f alb-ingress-controller.yaml
```

1.4. Deploy sample application to the cluster

```
# vi nginx-deployment.yaml
  apiVersion: extensions/v1beta1
  kind: Deployment
  metadata:
    name: "nginx-deployment"
    namespace: "default"
  spec:
    replicas: 3
    template:
      metadata:
        labels:
          app: "nginx"
      spec:
        containers:
        - image: nginx:latest
          imagePullPolicy: Always
          name: "nginx"
          ports:
          - containerPort: 80

# kubectl apply -f nginx-deployment.yaml
```

- create a service so we can expose the NGINX pods:

```
# vi nginx-service.yaml
  apiVersion: v1
  kind: Service
  metadata:
    annotations:
```

```
    alb.ingress.kubernetes.io/target-type: ip
  name: "nginx-service"
  namespace: "default"
spec:
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
  type: NodePort
  selector:
    app: "nginx"
```

```
# kubectl apply -f nginx-service.yaml
```

- create our ingress resource:

```
# vi nginx-ingress.yaml
```

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: "nginx-ingress"
  namespace: "default"
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
  labels:
    app: nginx-ingress
spec:
  rules:
  - http:
      paths:
      - path: /*
        backend:
          serviceName: "nginx-service"
          servicePort: 80
```

```
# kubectl apply -f nginx-ingress.yaml
```

- Validation

```
# kubectl get ingress nginx-ingress
```

```
NAME          HOSTS ADDRESS                                                  PORTS AGE
nginx-ingress * 5e07dbe1-default-ngnxingr-2e9-113757324.us-east-2.elb.amazonaws.com 80 9s
```

- Check the targets status.

```
# LOADBALANCER_PREFIX=$(kubectl get ingress nginx-ingress -o json | jq -r
'.status.loadBalancer.ingress[0].hostname' | cut -d- -f1)

# TARGETGROUP_ARN=$(aws elbv2 describe-target-groups | jq -r
'.TargetGroups[].TargetGroupArn' | grep $LOADBALANCER_PREFIX)

# aws elbv2 describe-target-health --target-group-arn $TARGETGROUP_ARN | jq -r
'.TargetHealthDescriptions[].TargetHealth.State'
```

- make sure that every pod is running using AWS Fargate

```
# kubectl get pods -o wide
```