

Kubernetes End-to-End Project: Deploying a Serverless 2048 Game with ALB on EKS with Fargate

Part 1: Using eksctl to Create an EKS Cluster with Fargate

This guide details how to create an Amazon EKS cluster with Fargate support and configure kubectl for access using eksctl. Fargate allows you to deploy containerized applications without managing worker nodes.

Prerequisites:

1. **AWS Account:** Ensure you have an AWS account with IAM permissions to create EKS clusters.
2. **AWS CLI (Command Line Interface):** Install and configure the AWS CLI on your machine. Refer to the official AWS documentation: <https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>
3. **eksctl Installation:** Install eksctl on your machine following the instructions from the eksctl GitHub repository: <https://docs.aws.amazon.com/eks/latest/userguide/setting-up.html>

Installation Steps:

1. **Configure AWS Credentials:**
 - o Ensure your AWS CLI is configured with proper credentials that have the necessary IAM permissions to create EKS clusters. You can use commands like `aws configure` or set environment variables to configure your credentials.
2. **Verify eksctl Installation:**
 - o Open a terminal window and run the following command to verify eksctl installation:

```
eksctl version
```

- This should display the installed eksctl version.

3. **Create a Cluster with Fargate:**

- o Run the following command to create a new EKS cluster with Fargate support:

```
eksctl create cluster --name demo-cluster --region us-east-1 --fargate
```

- Replace `<cluster-name>` with a unique name for your cluster.
- Replace `<region>` with the AWS region where you want to create the cluster (e.g., `us-east-1`).

- The `--fargate` flag specifies that you want to create a cluster with Fargate as the default execution mode for pods. `eksctl` will interact with the AWS API to create the cluster resources, including CloudFormation stacks for Fargate-related configurations.
- This process may take several minutes as `eksctl` provisions the cluster with Fargate support.

4. Validate Cluster Access:

This command updates your local kubeconfig file, which acts like a keycard for `kubectl` to access your EKS cluster. Without it, you can't manage your cluster using `kubectl` commands.

Just run:

```
aws eks update-kubeconfig --name your-cluster-name
```

- Once the cluster creation is complete, `eksctl` will configure your `kubectl` to access the newly created cluster. Verify access by running:

```
kubectl get nodes
```

- This command should list the pods running on Fargate instead of worker nodes, indicating a Fargate-enabled cluster.

By following these steps, you've successfully created an EKS cluster with Fargate support from scratch using `eksctl`. Now, you can deploy containerized applications leveraging Fargate's serverless execution model without managing worker nodes.

Part 2: Installing and Deploying the 2048 Game on an Amazon EKS Cluster with Fargate

This guide extends the previous instructions on creating an EKS cluster with Fargate support to deploy the popular 2048 game application.

Prerequisites:

- You've completed the steps in Part 1 to create an EKS cluster with Fargate.
- Basic understanding of Kubernetes deployments, services, and Ingress resources.

Deployment Steps:

1. Create Fargate Profile:

- Use `eksctl` to create a Fargate profile that defines configurations for pods running the 2048 application:

```
eksctl create fargateprofile \  
  --cluster demo-cluster \  
  --region us-east-1 \  
  --name alb-sample-app \  
  --namespace game-2048
```

- This command creates a Fargate profile named `alb-sample-app` within the `game-2048` namespace of your cluster. This profile defines the execution environment for the 2048 application pods running on Fargate.

2. Deploy the 2048 Application:

- We'll leverage a pre-configured deployment manifest for the 2048 game from the AWS Load Balancer Controller project. This manifest defines the deployment, service, and Ingress resources required to run the application.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-  
sigs/aws-load-balancer-  
controller/v2.5.4/docs/examples/2048/2048_full.yaml
```

3. Verify Deployment:

- Once you've applied the manifest, wait for the deployment to complete. You can check the deployment status using:

```
kubectl get deployments -n game-2048
```

- Look for the `2048` deployment to be in a "running" state with all pods ready.

By following these steps, you've successfully deployed the 2048 game application on your EKS cluster with Fargate support. This demonstrates how to leverage Fargate for deploying containerized applications without managing worker nodes.

Important Note:

- Before deploying the ALB Ingress Controller:

```
C:\Users\karathore\Desktop\DevOps_Journey_Kartik\AWS>kubectl get ing -n game-2048
NAME          CLASS  HOSTS  ADDRESS  PORTS  AGE
ingress-2048  alb    *      80       49s
```

- When you initially run `kubectl get ing`, the output might not show an address for your Ingress resources. This is because the Ingress resource itself doesn't create the load balancer.
- **After deploying the ALB Ingress Controller:**
 - Once you've deployed the ALB Ingress Controller, it will automatically provision and manage an Application Load Balancer (ALB) based on your Ingress definitions.
 - Running `kubectl get ing` after the controller deployment will display the address of the ALB associated with your Ingress.

Part 3: Setting Up the AWS Load Balancer Controller for EKS

This section guides you through setting up the AWS Load Balancer Controller for your EKS cluster with Fargate. The controller automates provisioning and management of Application Load Balancers (ALBs) for your deployments.

Enhanced Security with IAM OIDC Provider:

Before proceeding, it's important to enhance security by associating an IAM OpenID Connect (OIDC) provider with your cluster:

1. Associate IAM OIDC Provider:

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

- Replace `<cluster-name>` with your actual cluster name.
- The `--approve` flag automatically approves the creation of IAM resources.

This command associates a new IAM OIDC identity provider with your cluster, enabling features like:

- **IAM Roles for Service Accounts (IRSA):** Assign IAM roles directly to Kubernetes service accounts, allowing for fine-grained access control for your applications.
- **Authentication from Other AWS Services:** Use the OIDC provider to authenticate to other AWS services that support OIDC, enhancing security across your AWS environment.

2. Download and Create IAM Policy:

- Download the pre-defined IAM policy for the ALB controller:

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.5.4/docs/install/iam_policy.json
```

- Create an IAM policy named `AWSLoadBalancerControllerIAMPolicy` from the downloaded file:

```
aws iam create-policy \
  --policy-name AWSLoadBalancerControllerIAMPolicy \
  --policy-document file://iam_policy.json
```

3. Create IAM Role and Service Account (eksctl):

- Use eksctl to create an IAM role (AmazonEKSLoadBalancerControllerRole) and link it to a Kubernetes service account (aws-load-balancer-controller) in the kube-system namespace:

```
eksctl create iamserviceaccount \
  --cluster=<your-cluster-name> \
  --namespace=kube-system \
  --name=aws-load-balancer-controller \
  --role-name AmazonEKSLoadBalancerControllerRole \
  --attach-policy-arn=arn:aws:iam::<your-aws-account-id>:policy/AWSLoadBalancerControllerIAMPolicy \
  --approve
```

- Replace `<your-cluster-name>` with your actual cluster name.

4. Deploy the ALB Controller using Helm:

- Add the eks Helm repository:

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

- Update the repository:

```
helm repo update eks
```

- Install the `aws-load-balancer-controller` using Helm:

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  -n kube-system \
  --set clusterName=<your-cluster-name> \
```

```
--set serviceAccount.create=false \  
--set serviceAccount.name=aws-load-balancer-controller \  
--set region=<region> \  
--set vpcId=<your-vpc-id>
```

- Replace placeholders:
 - <your-cluster-name> with your cluster name.
 - <region> with your AWS region.
 - <your-vpc-id> with your VPC ID.
- Verify the deployment status:

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

Look for the `aws-load-balancer-controller` deployment to be in a "running" state.

Now you can access the Application:

- To find the ALB's public DNS address, run:

```
kubectl get ingress -n game-2048 -o jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}'
```

Open this address in a web browser to access the deployed 2048 game application

By following these steps, you've successfully deployed the AWS Load Balancer Controller. This enables automatic and secure management of Application Load Balancers (ALBs) for your deployments on the EKS cluster

2048

SCORE
200

BEST
200

Join the numbers and get to the **2048** tile!

New Game

4	32	4	2
8	4	16	
2			
		2	