

# **Report: Deploying a 2040 game Application using Amazon EKS with Fargate and ALB Controller**

**Name:** Chaithanya M

**Date:** July 10, 2025

**Topic:** 2040 game Deployment on AWS EKS (with Fargate and ALB Controller)

## **1. Introduction**

Kubernetes is the leading platform for orchestrating containerized applications. Amazon Elastic Kubernetes Service (EKS) provides a fully managed Kubernetes control plane on AWS, allowing developers to focus on deploying and managing workloads rather than managing infrastructure.

This report outlines the process of deploying a sample application using Amazon EKS with AWS Fargate, which allows serverless pod execution, and the AWS Load Balancer Controller to manage application traffic through an Application Load Balancer (ALB).

## **2. Objectives**

- Set up an EKS cluster using eksctl with Fargate as the compute backend.
- Deploy a sample application in a Kubernetes namespace.
- Configure IAM and security to support ALB integration.
- Set up the AWS Load Balancer Controller.
- Expose the application using an ALB Ingress and validate deployment.

## **3. Prerequisites**

- AWS CLI installed and configured with required credentials.
- kubectl and eksctl installed and added to system path.
- IAM user with sufficient privileges to create EKS clusters, IAM roles, and policies.
- Basic understanding of Kubernetes, AWS networking, and container deployment.

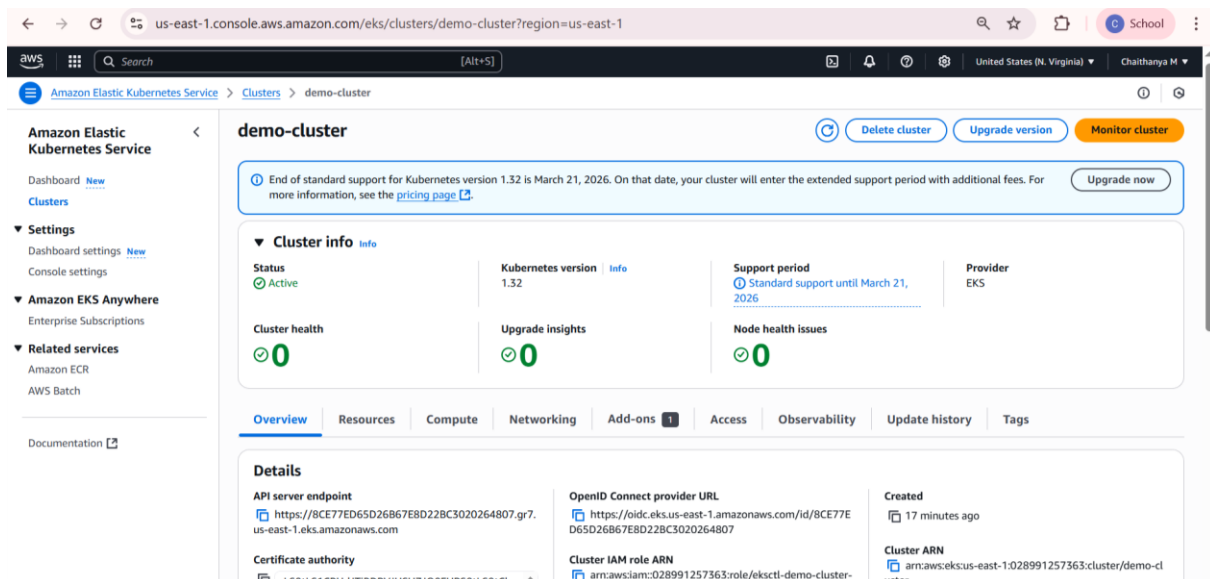
## 4. Step-by-Step Deployment Process

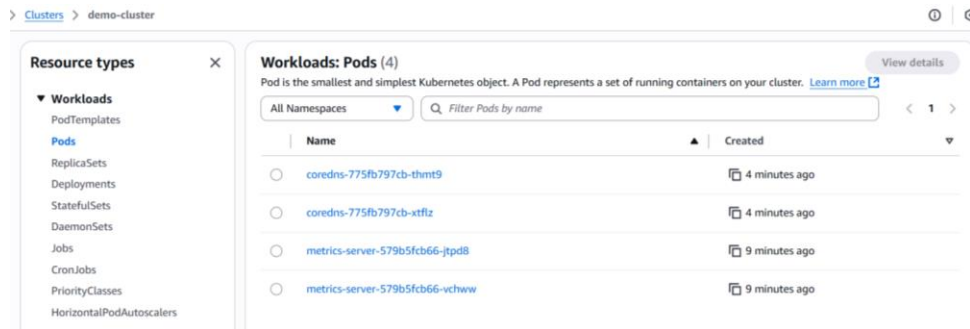
### Step 1: Create the EKS Cluster Using Fargate

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

This command creates an EKS cluster named demo-cluster in the us-east-1 region using AWS Fargate, allowing pods to run serverlessly without provisioning EC2 worker nodes.

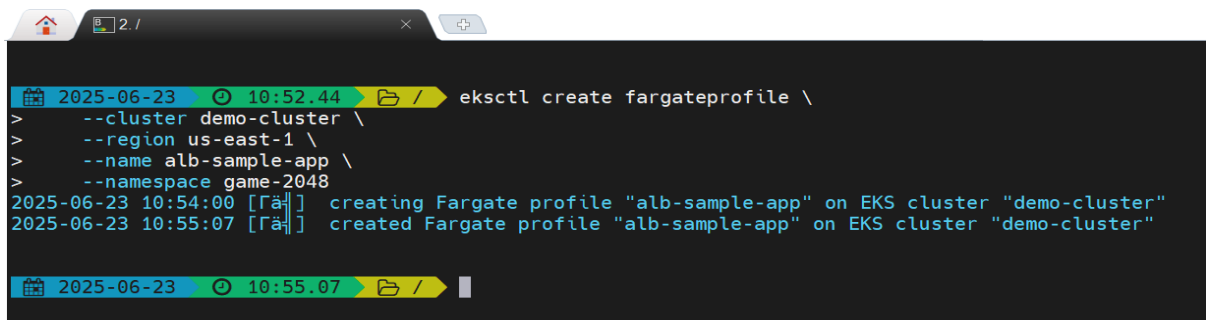
```
2025-06-23 10:29:08 / eksctl create cluster --name demo-cluster --region us-east-1 --fargate
2025-06-23 10:29:12 [ra] eksctl version 0.210.0
2025-06-23 10:29:12 [ra] using region us-east-1
2025-06-23 10:29:13 [ra] setting availability zones to [us-east-1d us-east-1a]
2025-06-23 10:29:13 [ra] subnets for us-east-1d - public:192.168.0.0/19 private:192.168.64.0/19
2025-06-23 10:29:13 [ra] subnets for us-east-1a - public:192.168.32.0/19 private:192.168.96.0/19
2025-06-23 10:29:13 [ra] using Kubernetes version 1.32
2025-06-23 10:29:13 [ra] creating EKS cluster "demo-cluster" in "us-east-1" region with Fargate profile
2025-06-23 10:29:13 [ra] if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --re
region=us-east-1 --cluster=demo-cluster'
2025-06-23 10:29:13 [ra] Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for clu
ster "demo-cluster" in "us-east-1"
2025-06-23 10:29:13 [ra] CloudWatch logging will not be enabled for cluster "demo-cluster" in "us-east-1"
2025-06-23 10:29:13 [ra] you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-
HERE (e.g. all)} --region=us-east-1 --cluster=demo-cluster'
2025-06-23 10:29:13 [ra] default addons kube-proxy, coredns, metrics-server, vpc-cni were not specified, will install them a
s EKS addons
2025-06-23 10:29:13 [ra]
2 sequential tasks: { create cluster control plane "demo-cluster",
  3 sequential sub-tasks: {
    1 task: { create addons },
    wait for control plane to become ready,
    create fargate profiles,
  }
}
2025-06-23 10:29:13 [ra] building cluster stack "eksctl-demo-cluster-cluster"
2025-06-23 10:29:14 [!] 1 error(s) occurred and cluster hasn't been created properly, you may wish to check CloudFormation co
nsole
```





## Step 2: Create a Fargate Profile

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

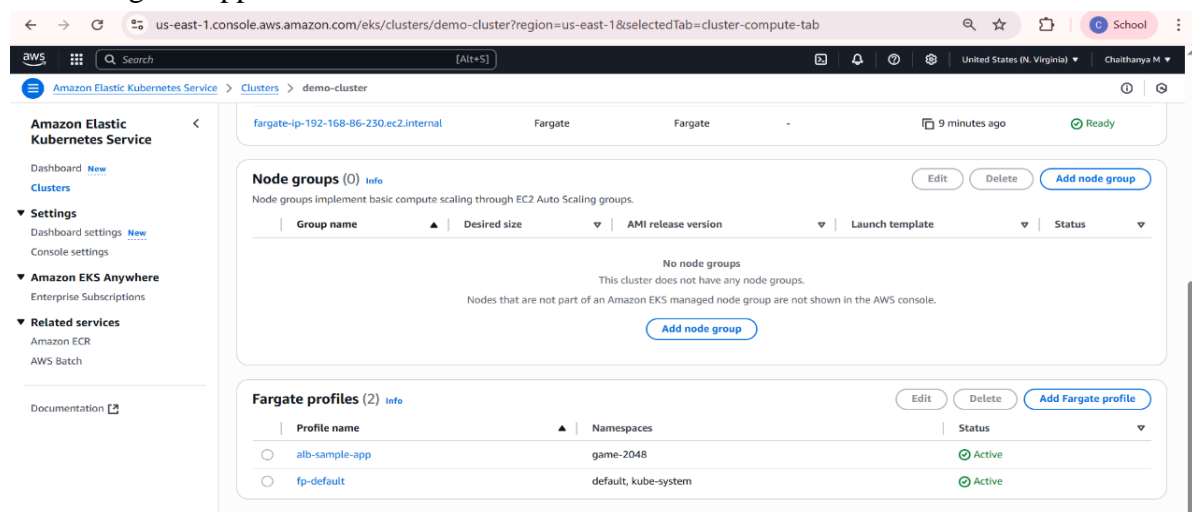


A Fargate profile enables Kubernetes pods in the game-2048 namespace to run on Fargate.

## Step 3: Deploy the Sample Application

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

This command applies all necessary Kubernetes manifests (deployment, service, ingress) for the 2048 game application.



```

2025-06-23 10:56:40 kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.5.4/docs/examples/2048/2048_full.yaml
namespace/game-2048 created
deployment.apps/deployment-2048 created
service/service-2048 created
ingress.networking.k8s.io/ingress-2048 created

```

```

2025-06-23 10:59:33 kubectl get pods -n game-2048
NAME                                READY   STATUS    RESTARTS   AGE
deployment-2048-bdbddc878-2vdtv    1/1     Running   0           54s
deployment-2048-bdbddc878-4fp2j    1/1     Running   0           54s
deployment-2048-bdbddc878-fjwc8    1/1     Running   0           54s
deployment-2048-bdbddc878-gvtjs    1/1     Running   0           54s
deployment-2048-bdbddc878-xvb6z    1/1     Running   0           54s

2025-06-23 10:59:50 kubectl get svc -n game-2048
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
service-2048  NodePort    10.100.37.15 <none>        80:31721/TCP    80s

```

## Step 4: Associate IAM OIDC Provider

```
$ eksctl utils associate-iam-oidc-provider --cluster demo-cluster --approve
```

This step ensures that the EKS cluster is associated with an OIDC provider for secure IAM role assumption by Kubernetes service accounts.

```

2025-06-23 11:00:25 kubectl get ingress -n game-2048
NAME          CLASS  HOSTS  ADDRESS  PORTS  AGE
ingress-2048  alb    *      80       107s

2025-06-23 11:00:44 eksctl utils associate-iam-oidc-provider --cluster $cluster_name --approve
2025-06-23 11:01:50 [F6] will create IAM Open ID Connect provider for cluster "demo-cluster" in "us-east-1"
2025-06-23 11:01:52 [F6] created IAM Open ID Connect provider for cluster "demo-cluster" in "us-east-1"

2025-06-23 11:01:53

```

## Step 5: Set Up IAM for ALB Controller

- **Download the IAM Policy:**

```
$ curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.11.0/docs/install/iam_policy.json
```

- **Create IAM Policy**

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

```
2025-06-23 11:01:53 curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.11.0/docs/install/iam_policy.json
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 8759 100 8759 0 0 16823 0 --:--:-- --:--:-- --:--:-- 16941

2025-06-23 11:02:38 aws iam create-policy \
--policy-name AWSLoadBalancerControllerIAMPolicy \
--policy-document file://iam_policy.json
{
  "Policy": {
    "PolicyName": "AWSLoadBalancerControllerIAMPolicy",
    "PolicyId": "ANPAQNQADPM12GXS5DHBV",
    "Arn": "arn:aws:iam::028991257363:policy/AWSLoadBalancerControllerIAMPolicy",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2025-06-23T05:32:56+00:00",
    "UpdateDate": "2025-06-23T05:32:56+00:00"
  }
}
```

- **Create IAM Role with Service Account**

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

```
2025-06-23 11:04:01 eksctl create iamserviceaccount \
--cluster=demo-cluster \
--namespace=kube-system \
--name=aws-load-balancer-controller \
--role-name AmazonEKSLoadBalancerControllerRole \
--attach-policy-arn=arn:aws:iam::028991257363:policy/AWSLoadBalancerControllerIAMPolicy \
--approve
2025-06-23 11:06:04 [FÄ] 1 iamserviceaccount (kube-system/aws-load-balancer-controller) was included (based on the include/exclude rules)
2025-06-23 11:06:04 [!] serviceaccounts that exist in Kubernetes will be excluded, use --override-existing-serviceaccounts to override
2025-06-23 11:06:04 [FÄ] 1 task: {
  2 sequential sub-tasks: {
    create IAM role for serviceaccount "kube-system/aws-load-balancer-controller",
    create serviceaccount "kube-system/aws-load-balancer-controller",
  } }
2025-06-23 11:06:04 [FÄ] building iamserviceaccount stack "eksctl-demo-cluster-addon-iamserviceaccount-kube-system-aws-load-balancer-controller"
2025-06-23 11:06:04 [FÄ] deploying stack "eksctl-demo-cluster-addon-iamserviceaccount-kube-system-aws-load-balancer-controller"
2025-06-23 11:06:04 [FÄ] waiting for CloudFormation stack "eksctl-demo-cluster-addon-iamserviceaccount-kube-system-aws-load-balancer-controller"
2025-06-23 11:06:36 [FÄ] waiting for CloudFormation stack "eksctl-demo-cluster-addon-iamserviceaccount-kube-system-aws-load-balancer-controller"
2025-06-23 11:06:38 [FÄ] created serviceaccount "kube-system/aws-load-balancer-controller"
```

## Step 6: Deploy AWS Load Balancer Controller

- Add Helm Repository

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

```
2025-06-23 11:07.00 helm repo add eks https://aws.github.io/eks-charts
"eks" has been added to your repositories

2025-06-23 11:08.12
```

- Install the Controller

```
$ helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
-n kube-system \
--set clusterName=demo-cluster \
--set serviceAccount.create=false \
--set serviceAccount.name=aws-load-balancer-controller \
--set region=us-east-1 \
--set vpcId=<your-vpc-id>
```

```
2025-06-23 11:12.22 helm install aws-load-balancer-controller eks/aws-load-balancer-controller -n kube-system --set clusterName=demo-cluster --set serviceAccount.create=false --set serviceAccount.name=aws-load-balancer-controller --set region=us-east-1 --set vpcId=vpc-0e11ed1661ce2d143
NAME: aws-load-balancer-controller
LAST DEPLOYED: Mon Jun 23 11:13:10 2025
NAMESPACE: kube-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
AWS Load Balancer controller installed!

2025-06-23 11:13.20
```

## Step 7: Verify the Controller Deployment

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

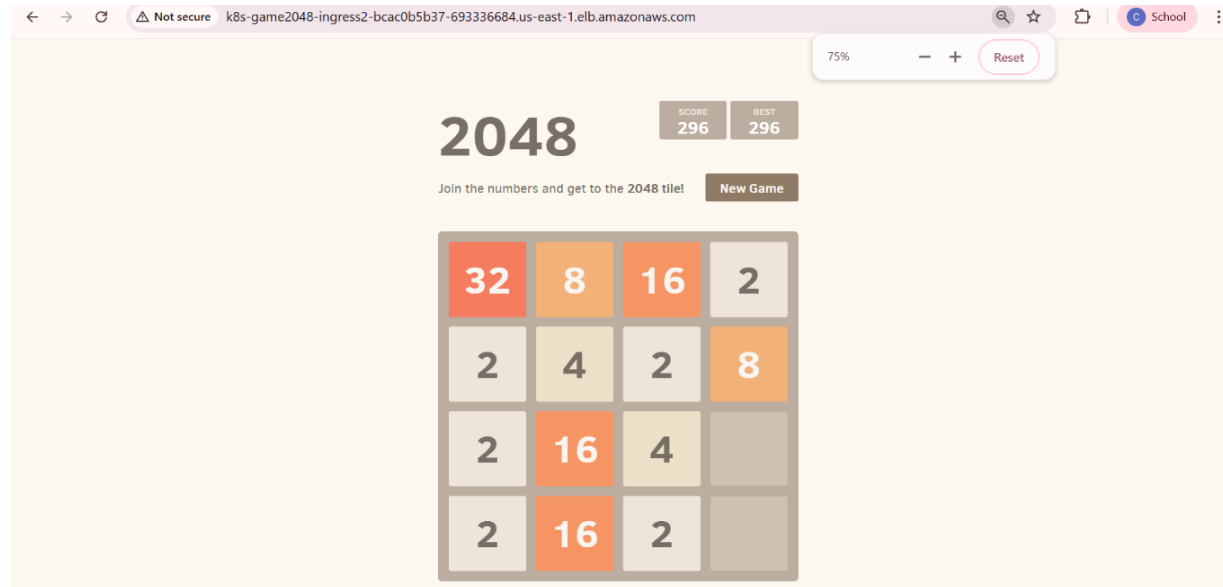
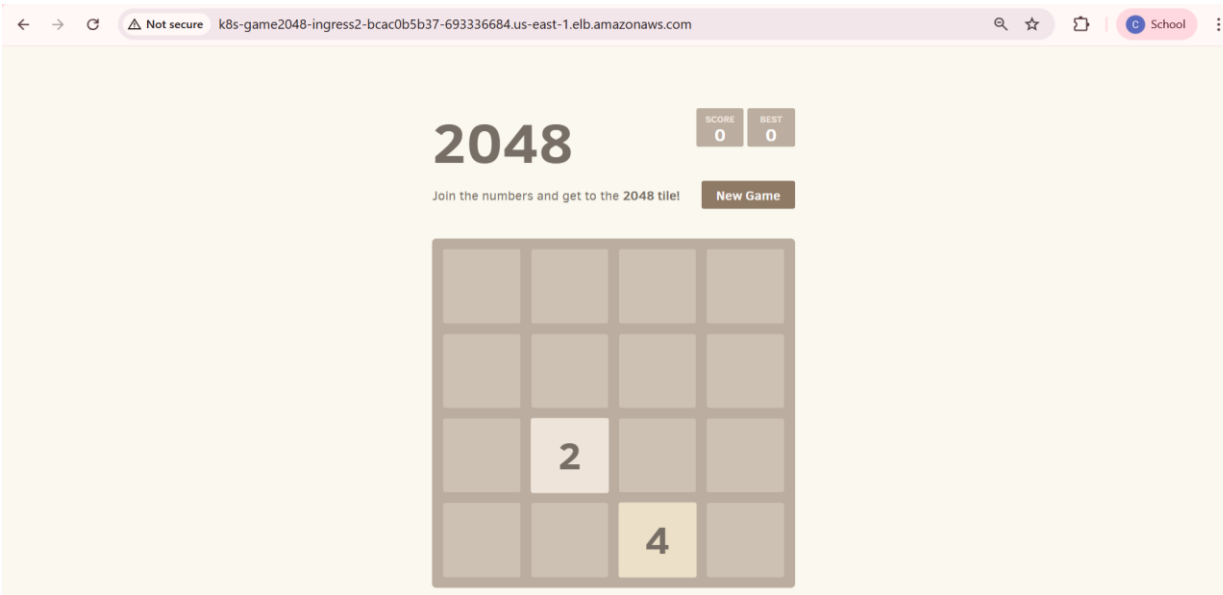
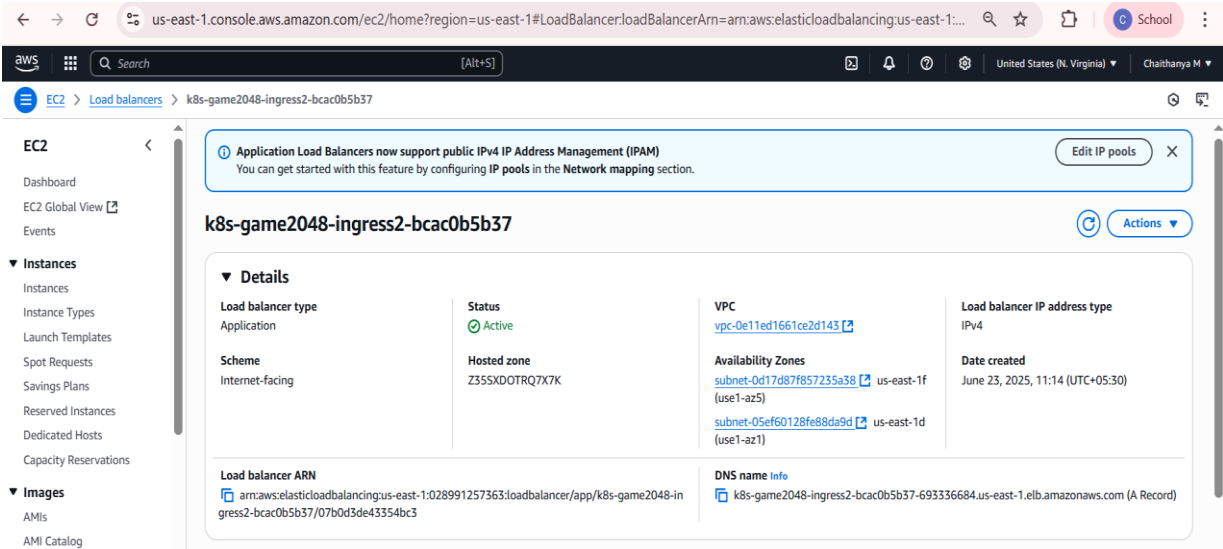
Ensure the deployment is running successfully without any errors.

```
2025-06-23 11:13.44 kubectl get deployment -n kube-system aws-load-balancer-controller
NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
aws-load-balancer-controller        0/2      2              0            31s

2025-06-23 11:13.51
```

```
2025-06-23 11:13.51 kubectl get deploy -n kube-system
NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
aws-load-balancer-controller        2/2      2              2            2m15s
coredns                             2/2      2              2            34m
metrics-server                      0/2      2              0            34m
```

# Deployed 2048 game Application Screenshot



## 5. Conclusion

This project provided comprehensive hands-on experience with deploying a containerized application using Amazon Elastic Kubernetes Service (EKS) backed by AWS Fargate. By following a step-by-step approach to cluster provisioning, Fargate profile creation, application deployment, and load balancer integration, I was able to gain a deeper understanding of Kubernetes operations in a cloud-native environment.

Using Fargate significantly simplified the infrastructure layer by eliminating the need to manage EC2 instances. This allowed me to focus purely on deploying and managing the application rather than configuring and maintaining the underlying compute resources. The integration of AWS Load Balancer Controller demonstrated how Kubernetes Ingress can be used in conjunction with AWS ALB to provide scalable, secure, and resilient access to services.

Key takeaways from this experience include:

- Proficiency in using `eksctl`, `kubectl`, and `helm` for EKS and Kubernetes management.
- Understanding the role of **IAM policies**, **OIDC providers**, and **service accounts** in securely granting access to AWS services from within Kubernetes.
- The importance of namespace-based resource isolation, especially when using Fargate profiles.
- Real-world exposure to deploying applications using declarative Kubernetes manifests.

This project also highlighted the **serverless nature of Fargate**, showing how it can help reduce operational complexity and scale containerized workloads efficiently. In a production environment, this approach can lower infrastructure overhead, improve security posture, and speed up deployment cycles.