

Lab 3: Deploying an Application with Amazon EKS on Fargate v3.0.2

Tuesday, March 2, 2021 1:58 PM

Advanced Architecting on AWS – Lab 3: Deploying an Application with Amazon EKS on Fargate



© 2021 Amazon Web Services, Inc. and its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. All trademarks are the property of their owners.

Corrections, feedback, or other questions? Contact us at [AWS Training and Certification](#)

Corrections, feedback, or other questions? Contact us at [AWS Training and Certification](#).

Lab Overview

You are developing an online game. You are optimistic that it will be a success and expect hundreds of thousands of gamers to access this game over the internet. You want to worry less about infrastructure and focus more on game development. You don't want to allocate any upfront infrastructure capital to deploy this game. However, you want to ensure that the application is scalable for user spikes after it is deployed.

The online game is called Web 2048 and is ready to launch. Your cloud architect recommends using Amazon Elastic Kubernetes Service (Amazon EKS) on AWS Fargate.

In this lab, you deploy your web-based game as a Docker container image. After verifying that the image is successfully created, you push it to Amazon Elastic Container Registry (Amazon ECR). You then launch an Amazon EKS cluster and create a Fargate profile. Finally, you deploy your application to a Fargate cluster.

Objectives

After completing this lab, you will be able to:

- Understand the steps needed to build Docker images.
- Upload container images to an Amazon ECR repository to be used for an Amazon EKS deployment.
- Deploy containers from a repository to an Amazon EKS Fargate cluster.
- Deploy the Service type LoadBalancer to an Amazon EKS Fargate cluster.
- Deploy a sample application using Kubernetes and containers running on an

- Deploy the Service type LoadBalancer to an Amazon EKS Fargate cluster.
- Deploy a sample application using Kubernetes and containers running on an Amazon EKS Fargate cluster.

Prerequisites

This lab requires:

- Access to a notebook computer with Wi-Fi and Microsoft Windows, macOS, or Linux (Ubuntu, SuSE, or Red Hat)
- An internet browser such as Chrome, Firefox, or Microsoft Edge
- A plaintext editor

Duration

This lab takes approximately 60 minutes to complete.

AWS Services Not Used in This Lab

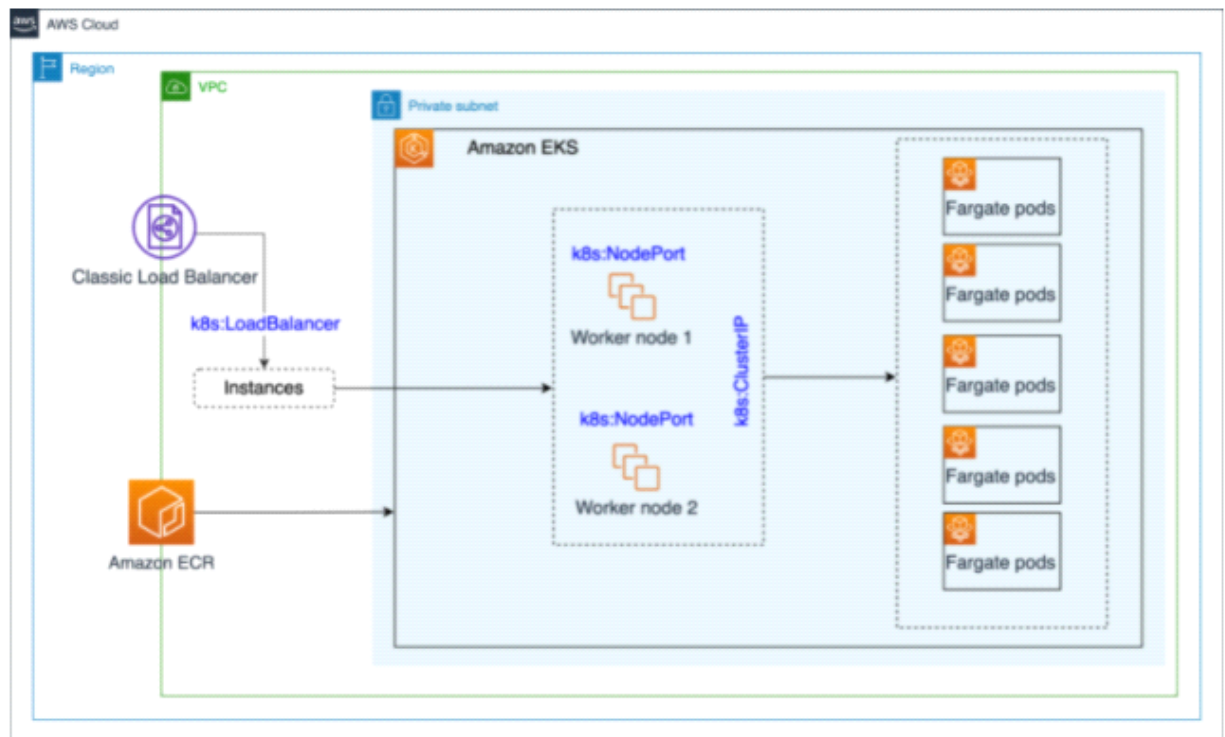
Amazon Web Services (AWS) services that are not used in this lab are deactivated in the lab environment. In addition, the capabilities of the services used in this lab are limited to what the lab requires. Expect errors when accessing other services or performing actions beyond those provided in this lab guide.

Lab Environment

As a part of the lab, a separate Amazon Virtual Private Cloud (Amazon VPC) with private and public subnets, internet gateway, NAT gateway, AWS Identity and Access Management (IAM) roles, EKS Cluster and routes were provisioned in your lab environment.

The following diagram shows the resources provisioned for this lab and how they will be connected at the end of the lab.

The following diagram shows the resources provisioned for this lab and how they will be connected at the end of the lab:



Amazon Elastic Container Service (Amazon ECS) is a highly scalable, high-performance container management service that supports Docker containers. It helps you to easily run applications on a managed cluster of Amazon Elastic Compute Cloud (Amazon EC2) instances. Amazon ECS eliminates the need for you to install, operate, and scale your own cluster management infrastructure.

Amazon Elastic Kubernetes Service (Amazon EKS) is a fully managed Kubernetes service. Amazon EKS provides a scalable and highly available control plane that runs across multiple Availability Zones to eliminate a single point of failure.

AWS Fargate is a serverless compute engine for containers that works with Amazon ECS and Amazon EKS. Fargate lets you focus on building your applications. It removes the need to provision and manage servers, lets you specify and pay for resources per application, and improves security through application isolation by design.

Start Lab

1. At the top of your screen, launch your lab by choosing [Start Lab](#)

This starts the process of provisioning your lab resources. An estimated amount of time to provision your lab resources is displayed. You must wait for your resources to be provisioned before continuing.

i If you are prompted for a token, use the one distributed to you (or credits you have purchased).

2. Open your lab by choosing [Open Console](#)

This opens an AWS Management Console sign-in page.

3. On the sign-in page, configure:

- **IAM user name:** `awsstudent`
- **Password:** Paste the value of **Password** from the left side of the lab page
- Choose [Sign In](#)

⚠ Do not change the Region unless instructed.

Common Login Errors

Error: You must first log out

Amazon Web Services Sign In

You must first log out before logging into a different AWS account.

To logout, [click here](#)

If you see the message, **You must first log out before logging into a different AWS account:**



- Choose **click here**
- Close your browser tab to return to your initial lab window
- Choose [Open Console](#) again

Task 1: Connect to the Lab Bastion Host

In this task, you connect to an Amazon EC2 instance running the Amazon Linux 2 operation system. Next, you get a local copy of the code needed for this lab and verify the supporting docker applications are present. You use this same instance throughout the lab to complete command line tasks.

4. If you have not already done so, follow the steps in the **Start Lab** section to log in to the AWS Management Console.

5. On the **Services** menu, choose **EC2**.


 **Caution** This lab is designed to use the new EC2 Console. If **New EC2 Experience** is available at the top-left of your screen, ensure  **New EC2 Experience** is selected.


6. In the left navigation pane, choose **Instances**.

7. Select the **Bastion Host** instance, and then choose [Connect](#).

The '*Connect to instance*' page is displayed.


8. Select the **Session Manager** tab.

 With Session Manager, you can connect to Amazon EC2 instances without requiring exposing the SSH port on your firewall or Amazon Virtual Private Cloud

 With Session Manager, you can connect to Amazon EC2 instances without requiring exposing the SSH port on your firewall or Amazon Virtual Private Cloud (Amazon VPC) security group. Refer to [AWS Systems Manager Session Manager](#) for more information.

9. Choose **Connect**.

A new browser tab or window opens with a connection to the bastion host instance.

10.  Enter the following command to browse to the home directory (/home/ssm-user/):

```
cd ~
```

11. In the terminal window, copy and paste the following code block, and then press ENTER:

```
mkdir environment && \  
cd ~/environment && \  
wget https://aws-tc-largeobjects.s3-us-west-2.amazonaws.com/ILT-TF-300-  
ADVARC-3/lab-3/code-lab3-v301.zip && \  
unzip code-lab3-v301.zip && \  
touch /tmp/install.log && \  
export log_file="/tmp/install.log" && \  
chmod +x init-docker.sh && \  
./init-docker.sh | tee -a "$log_file"
```

This shell script installs Docker in your environment. The script also sets up the correct credentials. kubectl and eksctl were pre-installed for this lab. In the next step, you verify the installed version of kubectl and eksctl

Learn more

- **Docker** is a command line utility for creating and managing Docker containers within your environment. For more information about Docker, refer to [Docker basics for Amazon ECS](#).
- **eksctl** is a command line utility for creating and managing Kubernetes clusters on

- **eksctl** is a command line utility for creating and managing Kubernetes clusters on Amazon EKS. For more information about eksctl, refer to [The eksctl command line utility](#).
- **kubect**l is a command line utility for communicating with the Kubernetes cluster API server. For more information about kubectl, refer to [Installing kubectl](#).

12. When the script is finished running, verify the installed versions by running each of the following commands:

```
trap 'printf "\n"' DEBUG # This command adds a blank line before the
output, which can make the output easier to read.
eksctl version
```

```
kubectl version
```

```
docker version
```

Sample output

```
user:~/environment $ eksctl version

0.44.0-rc.0

user:~/environment $ kubectl version

Client Version: version.Info{Major:"1", Minor:"16+", GitVersion:"v1.16.8-
eks-e16311", GitCommit:"e163110a04dcb2f39c3325af96d019b4925419eb",
GitTreeState:"clean", BuildDate:"2020-03-27T22:40:13Z",
GoVersion:"go1.13.8", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"19+", GitVersion:"v1.19.6-
eks-49a6c0", GitCommit:"49a6c0bf091506e7bafcdb1b142351b69363355a",
GitTreeState:"clean", BuildDate:"2020-12-23T22:10:21Z",
GoVersion:"go1.15.5", Compiler:"gc", Platform:"linux/amd64"}

user:~/environment $ docker version
```



```
user:~/environment $ docker version

Client:
 Version:           19.03.13-ce
 API version:       1.40
 Go version:        go1.13.15
 Git commit:        4484c46
 Built:             Mon Oct 12 18:51:20 2020
 OS/Arch:           linux/amd64
 Experimental:      false

Server:
 Engine:
  Version:          19.03.13-ce
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.13.15
  Git commit:       4484c46
  Built:            Mon Oct 12 18:51:50 2020
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.4
  GitCommit:        05f951a3781f4f2c1911b05e61c160e9c30eaa8e
 runc:
  Version:          1.0.0-rc93
  GitCommit:        12644e614e25b05da6fd08a38ffa0cfe1903fdec
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

Task 2: Explore the Dockerfile

In this task, you explore the Dockerfile that is used in the remaining lab tasks.

13. In the bastion host session, Run the following command.

```
cd ~/environment/Code/web2048
```

```
cd ~/environment/code/web2048
```

This directory contains a **Dockerfile** and the latest version of the application code.

14. Open the **Dockerfile** in this directory.

```
cat Dockerfile
```

Review the Dockerfile contents in the file editor. The Dockerfile contains the instructions that are needed to build the Web2048 application environment. The application pulls a base image of amazonlinux and declares some environment variables to be used in the containerized application.

Sample Dockerfile

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
...
ENV ServerName=web2048-site

RUN yum -y update && \
    yum -y install httpd unzip && \
    yum clean all
...
```

The following **COPY** command is used to copy a directory into the container. In this case, the source code is copied into the container.

```
COPY ./code/ /var/www/html/
```

Ideally, containers should be written to support dynamic assignment of values in configs at runtime. This application requires some variables to be assigned at build, rather than during runtime. You can still use Docker, but the scaling process might be more complicated.

```
# Config App
RUN echo "ServerName fargate.training" >> /etc/httpd/conf/httpd.conf
```

```
# Config App
RUN echo "ServerName fargate.training" >> /etc/httpd/conf/httpd.conf
```

Finally, the application needs to define what port the container is going to listen to for inbound requests and the process the container is going to serve requests on. You can use **CMD** or **ENTRYPOINT**. In this case, **ENTRYPOINT** is used (as shown in the following example). **ENTRYPOINT** specifies a command that is always run when the container starts. **CMD** specifies arguments fed to the **ENTRYPOINT**.

```
EXPOSE 80
ENTRYPOINT ["/usr/sbin/httpd", "-D", "FOREGROUND"]
```

Task 3: Build and Test the Web2048 Container

In this task, you build and test the Docker container as shown in the following diagram:



15. Return to the tab with the Bastion Host session.

⚠ Note If you closed the Bastion Host browser tab, refer Task 1 **Connect to the Lab Bastion Host** section.

16. To confirm that the current working directory is the web2048 directory, run the following command:

```
pwd
```

```
cd ~/environment/Code
ls -l
```

Sample output

```
drwxr-xr-x 3 ec2-user ec2-user 4096 Jul 29 2019 web2048
```

Web2048 is a simple 2048 game that runs in your web browser.

17. To build a Docker container for the website, run the following command:

Note Make sure to include the "." at the end of the code block.

```
cd web2048
docker build -t web2048/website .
```

Sample output

```
Sending build context to Docker daemon 537.6kB
Step 1/7 : FROM public.ecr.aws/amazonlinux/amazonlinux:latest
latest: Pulling from amazonlinux/amazonlinux
99a21848b482: Pull complete
Digest:
sha256:39cd6d6dde0a01b4e6883ac8de27b70e896240b731f958f55a6559d63949df2
Status: Downloaded newer image for
public.ecr.aws/amazonlinux/amazonlinux:latest
--> 3b6649ff98f5
...
```

The build might take a minute to finish. Scroll to the bottom of the terminal window until you locate a message similar to the following, which indicates a successful build:

```
...
Step 4/7 : COPY ./code/ /var/www/html/
--> cc8f908192db
```

```

...
Step 4/7 : COPY ./code/ /var/www/html/
---> cc8f908192db
Step 5/7 : RUN echo "ServerName fargate.training " >>
/etc/httpd/conf/httpd.conf
---> Running in aa1beb0d815a
Removing intermediate container aa1beb0d815a
---> d96aa8792fbc
Step 6/7 : EXPOSE 80
---> Running in fe5b5ee1cde2
Removing intermediate container fe5b5ee1cde2
---> 6ad10c1d590b
Step 7/7 : ENTRYPOINT ["/usr/sbin/httpd", "-D", "FOREGROUND"]
---> Running in a3879b09e76a
Removing intermediate container a3879b09e76a
---> a8cfe9987318
Successfully built a8cfe9987318
Successfully tagged web2048/website:latest

```

Now that the Docker image is built, test it on the command host. The **run** command starts the container. The **-p 80:80** argument in the command binds port 80 on the command host to port 80 on the Docker container. The **&** runs the container as a background process.

18. Run the following command:

```
docker run -p 80:80 web2048/website &
```

To identify the Docker containers that are currently running on this host, run the following command:

```
docker ps
```

19. Copy the **CONTAINER ID** value from the command output to a text editor to use in a later step.

20. To get the public IP address of the EC2 instance, run the following command:

```
curl http://169.254.169.254/latest/meta-data/public-ipv4
```



```
curl http://169.254.169.254/latest/meta-data/public-ipv4
```

21. Copy the public IP value from the output.
22. Open a *new* browser tab, paste in the public IP address you just copied, and press Enter.

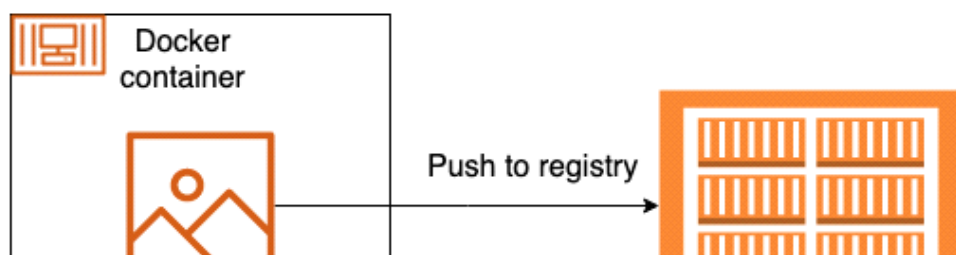
The game application running on the container is displayed.
23. Return to the terminal session
24. To stop this Docker container, run the following command, replacing **[CONTAINER ID]** with the value you copied in step 19:

```
docker stop [CONTAINER ID]
```

You have now successfully built the Web2048 container.

Task 4: Tag and Push the Docker Images to an Amazon ECR Repository

Suppose you wanted to share the Docker image with other AWS accounts and users. In such a scenario, a repository for Docker images would be useful. In this task, you use Amazon ECR to host a Docker image.





Previously, you created the Web2048 Docker image. Where was that Docker image saved? What was it named? You can use the **docker images** command to show the layers of the Docker images that are present on the local system and the repository they are from.

25. To get information about all created Docker images, run the following command:

```
docker images
```

Sample output

REPOSITORY	SIZE	TAG	IMAGE ID
web2048/website		latest	a8cfe9987318
4 minutes ago	244MB		
public.ecr.aws/amazonlinux/amazonlinux		latest	3b6649ff98f5
6 days ago	163MB		

In the output, **IMAGE ID** is a unique identifier that you can use to call out a version of a Docker image. Alternatively, you can reference the image by the repository it is associated with.

26. To create an Amazon ECR repository for each Docker image, run the following command:

```
aws ecr create-repository --repository-name web2048/website
```

27. To get the repository URI, run the following command:

```
aws ecr describe-repositories --query 'repositories[][repositoryName,
```

```
aws ecr describe-repositories --query 'repositories[][repositoryName, repositoryUri]' --output table
```

28. To export the value of the repository URI to the environment, run the following command:

```
export REPOSITORY_URI=$(aws ecr describe-repositories --query  
'repositories[][repositoryUri]' --output text)  
echo ${REPOSITORY_URI}
```

Before you can push the Docker image to the repository, you must log in to Amazon ECR. The [get-login command](http://docs.aws.amazon.com/cli/latest/reference/ecr/get-login.html) <http://docs.aws.amazon.com/cli/latest/reference/ecr/get-login.html> returns a **docker login** token, which is valid for 12 hours. You can pipe the output from this command to a shell to run the **docker login** command to complete the authentication.

29. Run the following commands:

```
export ACCOUNT_ID=$(aws sts get-caller-identity --output text --query  
Account)  
  
export AWS_REGION=$(curl -s 169.254.169.254/latest/dynamic/instance-  
identity/document | jq -r '.region')  
  
aws ecr get-login-password --region ${AWS_REGION} | docker login --  
username AWS --password-stdin  
${ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com
```

Success completion of the command is indicated when the following message returned to the terminal:

```
Login Succeeded
```

Now tag the Docker image with the URI of the repository it is stored in and indicate it

Now tag the Docker image with the URI of the repository it is stored in and indicate it is the latest build by using the **latest** keyword.

30. Run the following command:

```
docker tag web2048/website:latest ${REPOSITORY_URI}:latest
```

31. To validate that the tags have been applied, re-run the **docker images** command. The repository URI is in the output.

```
docker images
```

Use the **docker push** command to push the Docker image to the repository.

32. Run the following command:

```
docker push ${REPOSITORY_URI}:latest
```

You have successfully pushed the Docker image to an Amazon ECR repository.

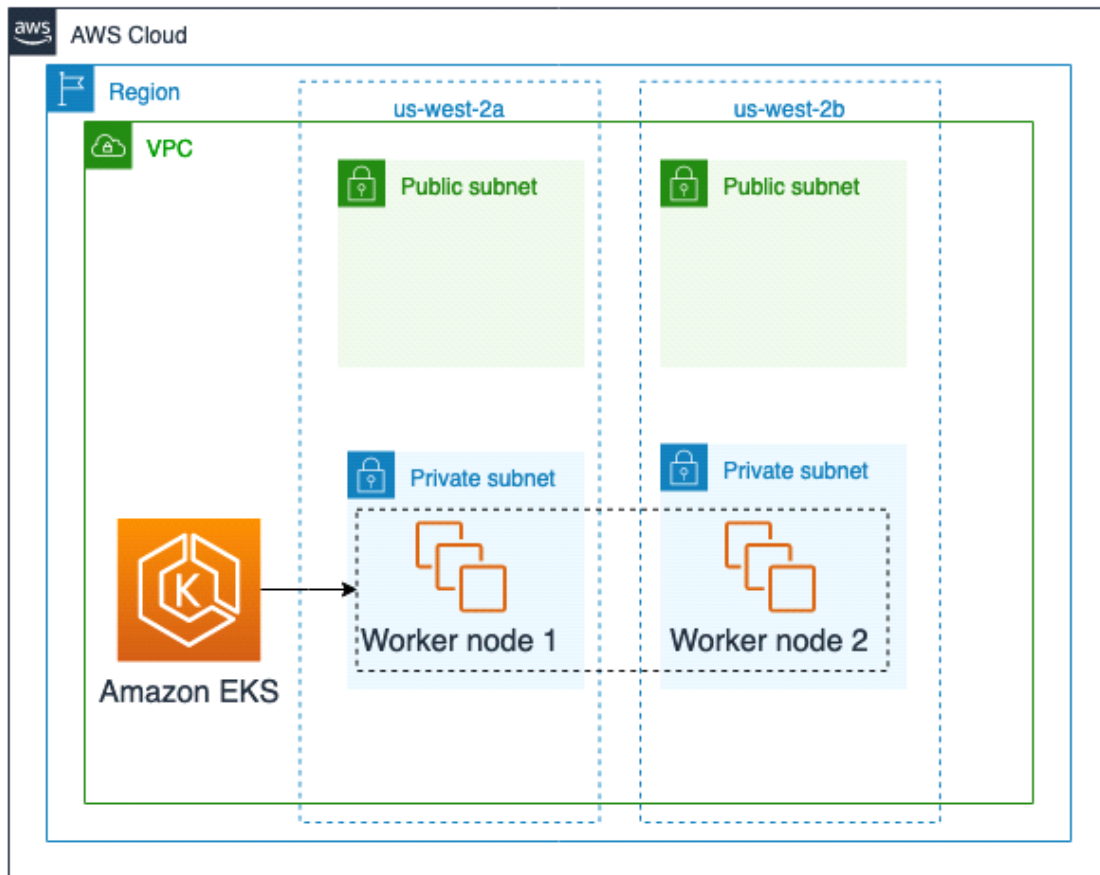
Task 5: Explore the Deployed Amazon EKS Cluster and Managed Node Group

In this task, you explore the Amazon EKS cluster with a managed node group which was provisioned for you. You also verify that you can communicate with the nodes.

The node group uses an Amazon EKS managed node group. Refer to [Managed node](#)

was provisioned for you. You also verify that you can communicate with the nodes.

The node group uses an Amazon EKS managed node group. Refer to [Managed node groups](#) for more information on this topic.



i In this lab, an Amazon EKS cluster control plane of **version 1.18** was provisioned. The version of kubectl must be within one minor version earlier or later than the Amazon EKS cluster control plane version.

For installing Amazon EKS clusters via the command line refer to [Installing kubectl](#) for more information.

i When you create an Amazon EKS cluster using the eksctl command line, a new virtual private cloud (VPC), two public subnets, and two new private subnets are created for use with the Amazon EKS control plane cluster and nodes by default. However, you can use additional command flags to specify the use of an existing VPC and subnets. Refer to [Amazon EKS networking](#) for more information.

The following list describes the configuration of the Amazon EKS cluster deployed for you in this lab:

for you in this lab:

- The name for the Amazon EKS cluster **ekscluster**
- The name for the Amazon EKS node group **dev-nodes**
- The version of the Kubernetes deployed to the Amazon EKS cluster **version 1.8**.
- The instance type is used for the worker nodes **t3.small**
- The autoscaling used for the nodes is defined as follows:
 - The minimum number of nodes to be deployed **1**
 - The desired number of nodes deployed autoscaling definition **2**
 - The maximum number of nodes to be deployed **3**

⚠ Warning The IAM role or user that creates the cluster is automatically added to the *system:masters* group in the Kubernetes RBAC configuration, making it a full administrator of the Kubernetes cluster. It is also the **only** role or user that can access the cluster until additional permissions are granted. In the case of this lab, the IAM role assumed by the bastion host is the one that created the cluster. Refer to [Managing users or IAM roles for your cluster](#) for more information about this topic.

33. Return to the AWS console.

34. On the **Services** menu, choose **Elastic Kubernetes Service**.

35. Choose **Clusters** from the left under the **Amazon EKS** section.

Notice that the status for the **ekscluster** is **Active**.

36. Choose the **ekscluster** link.

37. Choose the **Configuration** tab.

Note If the cluster is still in a creating state, all of the available information might not be populated.

38. Choose the **Compute** tab to view the details of the node groups associated with this cluster.

Notice that there is one node group named *dev-nodes*, and that the **Desired size* is two. The node group was specified with a minimum of one node and a maximum of

Notice that there is one node group named *dev-nodes*, and that the **Desired size* is two. The node group was specified with a minimum of one node and a maximum of three nodes, and a desired size of two nodes. Because these options were specified, an Amazon EC2 Auto Scaling group is utilized to manage the number of nodes in the node group.

39. On the **Services** menu, choose **EC2**.


40. In the list on the left, under **AUTO SCALING**, choose **Auto Scaling Groups**

41. Select the group with the name that beginning with **eks** to view details about it.

Notice that the minimum, maximum, and desired values are the same as what was specified for the node group. Amazon EKS uses this Amazon EC2 Auto Scaling group to automatically increase or decrease the number of nodes as the load on the node group compute resources changes.

Now, validate the provisioned cluster and managed nodes.

42. Return to the tab with the Bastion Host session.

 **Note** If you closed the Bastion Host browser tab, refer Task 1 **Connect to the Lab Bastion Host** section.

43. To display the available cluster in your Region, run the following command:

```
eksctl get clusters
```

44. To display the Kubernetes cluster, run the following command. If the cluster named **ekscluster** is present, then kubectl is authenticated correctly.

```
kubectl get svc
```

45. To display the two nodes that are available run the following command:

```
kubectl get nodes
```

Task 6: Explore the Fargate Profile

In this task, you will explore a Fargate profile on the Amazon EKS cluster. This Fargate profile has been provided to you in this lab.

Fargate profiles are immutable. However, you can create a new, updated profile to replace an existing profile and then delete the original after the new profile is created.

When the Amazon EKS cluster schedules pods on Fargate, the pods must make calls to AWS APIs to do things like pull container images from Amazon ECR. The Fargate Pod Execution Role provides the IAM permissions to do this.

46. Return to the tab with the Bastion Host session.

⚠ Note If you closed the Bastion Host browser tab, refer Task 1 **Connect to the Lab Bastion Host** section.

47. To validate the created Fargate profile which was provided to you, run the following command:

```
eksctl get fargateprofile \
  --cluster ekscluster \
  -o yaml
```

Sample output

```
- name: 2048-game
```

```
- name: 2048-game
  podExecutionRoleARN: arn:aws:iam::123456789012:role/eksctl-ekscluster-
cluster-FargatePodExecutionRole-L4Q800ICJJCV
  selectors:
  - namespace: 2048-game
  subnets:
  - subnet-05dc579c70bac08ef
  - subnet-0fe51eb8af9f26405
```

Note Notice the two private subnets in the Amazon EKS cluster. Pods running on Fargate support only private subnets (with no direct route to an internet gateway). Thus, while provisioning an Amazon EKS cluster, make sure that the VPC you create contains one or more private subnets.

Now explore the same Fargate profile from the console.

48. Return the to the AWS console.

49. On the **Services** menu, choose **Elastic Kubernetes Service**.

50. Choose **Clusters** from the left under the **Amazon EKS** section.

Notice that the status for the **ekscluster** is **Active**.

51. Choose the **ekscluster** link.

52. Choose the **Configuration** tab.

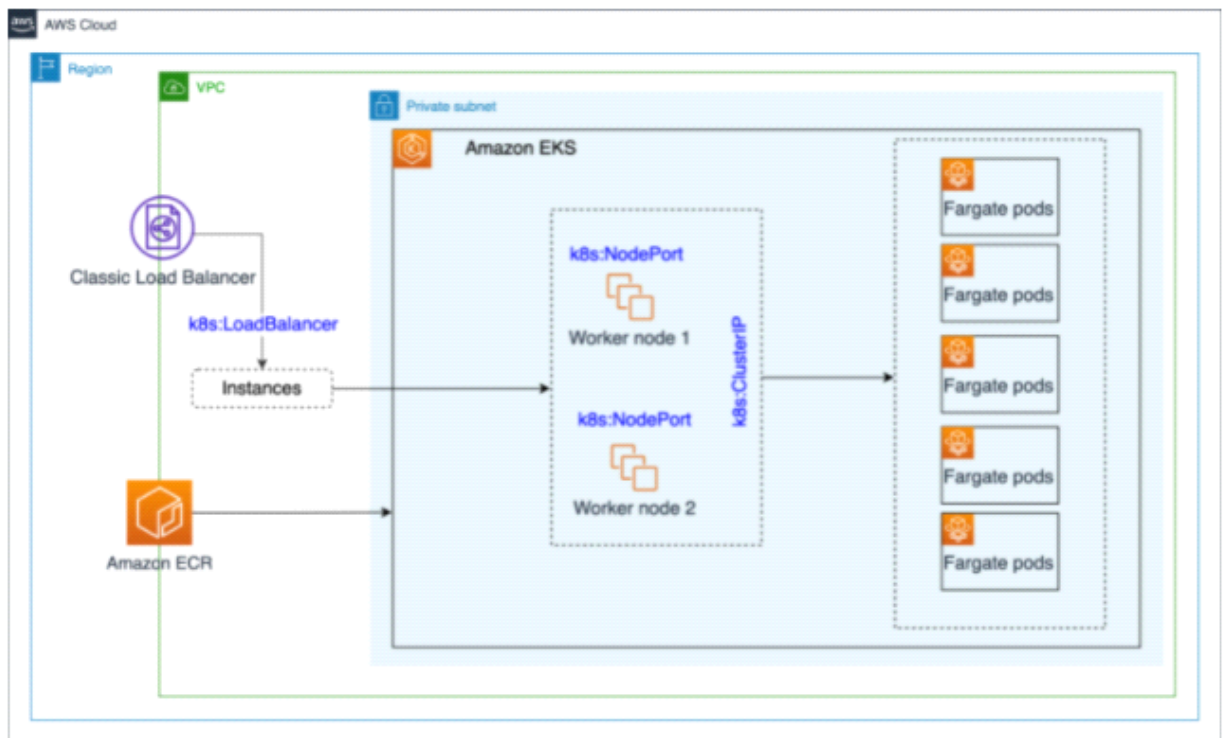
53. Choose the **Compute** tab.

54. Choose the link for the **2048-game** in the *Fargate Profiles* section.

The details page describes the information of the Fargate profile as was output by the command line earlier.

Task 7: Set Up and Deploy the Service Type Load Balancer

In this task, you provision an internet-facing Classic Load Balancer, as shown in the following diagram:



55. Return to the tab with the Bastion Host session.

⚠ Note If you closed the Bastion Host browser tab, refer Task 1 **Connect to the Lab Bastion Host** section.


56. To deploy the 2048 game as a sample application, run the following commands in the Bastion Host terminal window:

```
cd ~/environment/fargate
sed -i 's|REPOSITORY_URI|${REPOSITORY_URI}|' 2048-game.yaml
kubectl apply -f 2048-game.yaml
```


This also starts provisioning an instance of the internet-facing Classic Load Balancer.

57. To check the status of deployment, run the following command:

```
kubectl -n 2048-game rollout status deployment 2048-deployment
```

 **Note** It takes at least 5 minutes to complete this step and to show the output that is similar to *Sample output*

Sample output

```
Waiting for deployment "2048-deployment" rollout to finish: 0 of 5
updated replicas are available...
Waiting for deployment "2048-deployment" rollout to finish: 1 of 5
updated replicas are available...
Waiting for deployment "2048-deployment" rollout to finish: 2 of 5
updated replicas are available...
Waiting for deployment "2048-deployment" rollout to finish: 3 of 5
updated replicas are available...
Waiting for deployment "2048-deployment" rollout to finish: 4 of 5
updated replicas are available...
deployment "2048-deployment" successfully rolled out
```

58. When the deployment is finished, list the status of the pods and the newly provisioned Fargate worker nodes they are running on by issuing the following command:

```
kubectl get pods -n 2048-game -o=custom-
columns=NAME:.metadata.name,STATUS:.status.phase,NODE:.spec.nodeName
```

Sample output

NAME	STATUS	NODE
------	--------	------

NAME	STATUS	NODE
2048-deployment-5c5f4597c7-gwmzq west-2.compute.internal	Running	fargate-ip-10-0-3-124.us-west-2.compute.internal
2048-deployment-5c5f4597c7-km4pt west-2.compute.internal	Running	fargate-ip-10-0-4-60.us-west-2.compute.internal
2048-deployment-5c5f4597c7-q5xg8 west-2.compute.internal	Running	fargate-ip-10-0-4-213.us-west-2.compute.internal
2048-deployment-5c5f4597c7-q8zf6 west-2.compute.internal	Running	fargate-ip-10-0-3-44.us-west-2.compute.internal
2048-deployment-5c5f4597c7-qlr89 west-2.compute.internal	Running	fargate-ip-10-0-3-87.us-west-2.compute.internal

Fargate auto provisions specialized worker node instances running kubelet, and Kubernetes schedules one pod replica to each instance for pod request that match the Fargate profile.

Retrieve a list both of the managed EC2 nodes and the five Fargate nodes that are running the game.

59. Run the following command:

```
kubectl get nodes
```

Sample output

NAME	STATUS	ROLES	AGE
VERSION			
fargate-ip-10-0-3-124.us-west-2.compute.internal 2m27s v1.19.6-eks-e91815	Ready	<none>	
fargate-ip-10-0-3-44.us-west-2.compute.internal 2m33s v1.19.6-eks-e91815	Ready	<none>	
fargate-ip-10-0-3-87.us-west-2.compute.internal v1.19.6-eks-e91815	Ready	<none>	2m1s
fargate-ip-10-0-4-213.us-west-2.compute.internal 2m31s v1.19.6-eks-e91815	Ready	<none>	
fargate-ip-10-0-4-60.us-west-2.compute.internal 2m28s v1.19.6-eks-e91815	Ready	<none>	
ip-10-0-1-254.us-west-2.compute.internal v1.19.6-eks-49a6c0	Ready	<none>	138m
ip-10-0-2-206.us-west-2.compute.internal v1.19.6-eks-49a6c0	Ready	<none>	138m

ip-10-0-2-206.us-west-2.compute.internal	Ready	<none>	138m
v1.19.6-eks-49a6c0			

60. Go to the tab with the AWS Management Console open.

61. In the console, on the **Services** menu, choose **EC2**.

62. In the left navigation pane, choose **Load Balancers**.

Notice that a new Classic Load Balancer has been created.


63. Choose the Load Balancer.

64. Choose the **Instances** tab

65. Check if the status of the instances is **InService**

- You might be wondering why you are not load balancing directly from the Elastic Load Balancer to the Fargate Pods. This is because you use the Kubernetes 'In-Tree' AWS Cloud Provider deployed in Amazon EKS by default.
- The default In-Tree K8s AWS Cloud Provider in the lab, uses the following Classic Load Balancer --> NodePort --> Kube-Proxy path. Each node has a kube-proxy container process. (In the Kubernetes frame of reference, that kube-proxy container is in a pod in the kube-system namespace.) kube-proxy manages forwarding of traffic addressed to the ClusterIP addresses of the cluster's Kubernetes Service objects to the appropriate backend pods using iptables operation mode. This mode is the default for kube-proxy on most platforms. When load balancing for multiple backend pods, it uses unweighted round-robin scheduling.
- To learn how to design an Amazon EKS service to ingress directly from a AWS Application Load Balancer or Network Load Balancer to Kubernetes Pods on Fargate, please refer to [AWS LoadBalancer Controller](#).

66. Return to the tab with the Bastion Host session.

 **Note** If you closed the Bastion Host browser tab, refer Task 1 **Connect to the Lab Bastion Host** section.

67. To display the Classic Load Balancer link, run the following command:

67. To display the Classic Load Balancer link, run the following command:

```
kubectl get svc -n 2048-game
```

Sample output

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP
service-2048	LoadBalancer	172.20.8.80	a417fa6d57bf1407bbdde572246c0d31-1660622535.us-west-2.elb.amazonaws.com
80:32282/TCP	7m20s		


68. Copy the value **EXTERNAL-IP** from the table output.

69. Open a *new* browser tab, paste in the copied value and press Enter.

The page loads the 2048 game.

Note It might take 2-5 minutes for the DNS to propagate. Refresh the web page if the 2048 game is not displayed.

Conclusion

 Congratulations! You now have successfully

- Understand the steps needed to build Docker images
- Upload container images to an Amazon ECR repository to be used for an Amazon EKS deployment
- Deploy containers from a repository to an Amazon EKS Fargate cluster
- Deploy a sample application using Kubernetes and containers running on Amazon EKS Fargate

- Deploy containers from a repository to an Amazon EKS Fargate cluster
- Deploy a sample application using Kubernetes and containers running on Amazon EKS Fargate cluster

End Lab

Follow these steps to close the console, end your lab, and evaluate the experience.

70. Return to the AWS Management Console.

71. On the navigation bar, choose **awsstudent@<AccountNumber>**, and then choose **Sign Out**.

72. Choose  **End Lab**

73. Choose  **OK**

74. (Optional):

- Select the applicable number of stars ☆
- Type a comment
- Choose **Submit**
 - 1 star = Very dissatisfied
 - 2 stars = Dissatisfied
 - 3 stars = Neutral
 - 4 stars = Satisfied
 - 5 stars = Very satisfied

You may close the window if you don't want to provide feedback.

For more information about AWS Training and Certification, see <http://aws.amazon.com/training/>.

<http://aws.amazon.com/training/>.

Your feedback is welcome and appreciated.

If you would like to share any feedback, suggestions, or corrections, please provide the details in our [AWS Training and Certification Contact Form](#).

Additional Resources

- [Installing kubectl](#)
- [Installing aws-iam-authenticator](#)
- [The eksctl command line utility](#)
- [Cluster authentication](#)
- [Managing users or IAM roles for your cluster](#)
- [kubectl apply](#)
- [Scaling a Kubernetes Deployment](#)

Docker Command Dictionary

The following is a short list of Docker commands that might be helpful:

```
# Get a local shell of a container
docker run -i -t --entrypoint /bin/bash imageID

# Create an image using this directory's Dockerfile
docker build -t friendlyname .

# Run "friendlyname" mapping port 4000 to 80
docker run -p 4000:80 friendlyname

# Same thing, but in detached mode
docker run -d -p 4000:80 friendlyname

# Return a list of all running containers
docker ps

# Gracefully stop the specified container
docker stop <hash>

# Return a list of all containers, even the ones not running
```

```

# Gracefully stop the specified container
docker stop <hash>
# Return a list of all containers, even the ones not running
docker ps -a
# Force shutdown of the specified container
docker kill <hash>
# Remove the specified container from this machine
docker rm <hash>
# Remove all containers from this machine
docker rm $(docker ps -a -q)
# Show all images on this machine
docker images -a
# Remove the specified image from this machine
docker rmi <imagename>
# Remove all images from this machine
docker rmi $(docker images -q)
# Log in to this CLI session using your Docker credentials
docker login
# Tag <image> for upload to registry
docker tag <image> username/repository:tag
# Upload tagged image to registry
docker push username/repository:tag
# Run image from a registry
docker run username/repository:tag

```

kubectl Command Dictionary

The following is a short list of kubectl commands that might be helpful:

```

# Show merged kubeconfig settings
kubectl config view
# List all services in the namespace
kubectl get services
# List all pods in all namespaces
kubectl get pods --all-namespaces
# List all pods in the current namespace, with more details
kubectl get pods -o wide
# List a particular deployment
kubectl get deployment my-dep
# List all pods in the namespace
kubectl get pods
# Get a pod's YAML
kubectl get pod my-pod -o yaml
# Describe commands with verbose output
kubectl describe nodes my-node
kubectl describe pods my-pod

```

```
# Describe commands with verbose output
kubectl describe nodes my-node
kubectl describe pods my-pod
```

eksctl Command Dictionary

The following is a short list of eksctl commands that might be helpful:

```
# List the details about a cluster
eksctl get cluster [--name=<name>][--region=<region>]
# Create the same kind of basic cluster, but with a different name
eksctl create cluster --name=cluster-1 --nodes=4
# Delete a cluster
eksctl delete cluster --name=<name> [--region=<region>]
# Create an additional nodegroup
eksctl create nodegroup --cluster=<clusterName> [--name=
<nodegroupName>]
```

2048 Game

In a 2048 game, you slide numbered tiles around a grid, combining two tiles with the same number to create one tile with a larger number. You start with a **2** tile, which you combine with another **2** tile to create a **4** tile. Each time you move, a new tile appears on the grid. Combine two **4** tiles to make an **8** tile, two **8** tiles to make a **16** tile, and so on, with the goal of getting to the **2048** tile. The game ends when the grid is full and you cannot make any more combinations.

For some tips and tricks for playing 2048, refer to [2048 Game Tips & Tricks](#).