



Jenkins



AWS ECS

The goal of this assignment was to use Elastic Container Service (ECS) to host our Jenkins application that we created using a Dockerfile. This Jenkins application is our controller which essentially tells an agent hosted on an EC2 to build an image and push it to Dockerhub.

The controller application does not take a lot of resources so it's best to host it on Fargate where we are charged by the second and how much resources we use.

We are then doing all the building on an agent that is hosted on a separate EC2 instance which will save us more money than building on ECS.

Overall, installing Jenkins on Fargate is faster than installing Jenkins on an EC2 because all you need is the image. When installing on an EC2, you will need to install java, Jenkins, all the dependencies, and wait for it to start up.

Instructions

We need to create a container that has a Jenkins application using a Dockerfile. This will speed up the process of setting up a Jenkins application. We can then use these containers to host quickly on AWS using AWS Elastic Container Service. Dockerfiles is a text document that has all the commands the user uses when assembling an image. In other words, it is a text definition that defines the Docker image. It is a word to define your custom environment to be used in a docker container.

FROM jenkins/jenkins

USER root

RUN apt update && apt upgrade -y

USER jenkins

EXPOSE 8080

Once we create the Dockerfile, we have to build it. Doing this will create a docker image that we can use to create containers.

`docker build -t jenkins-app .`

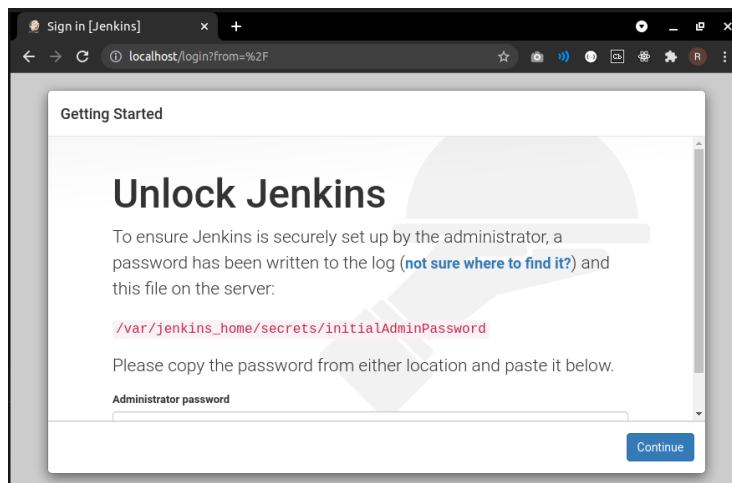
We can also test the image by creating a container. The -it flag tells docker that it should open an interactive container instance. The --name flag gives the container a name. The --rm flag tells docker that the container should automatically be removed after we close docker. The -p flag specifies which port we want to make available for docker. Port 80 is where we access and 8080 is the container.

`docker run -ti --name jenkins-app --rm -p 80:8080 jenkins-app`

We can check if the container is running by going to localhost on our browser

localhost:80

You should see an output like this...



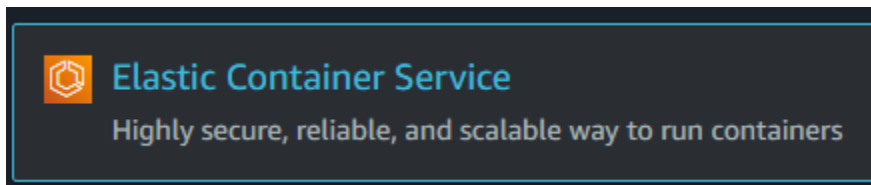
Exit the container using CTRL + C

Once we have successfully created the image and checked the image is working, we can move it on to Elastic Container Service (ECS) which will allow us to spin up multiple containers anytime.

Before we deploy to ECS, we need to make sure we have AWS CLI set up. AWS CLI is simply a tool to manage AWS services on the terminal.

Make sure you install AWS CLI and sign into it. -> [here](#)

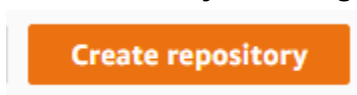
Once that is set up, Log into your AWS user and go to ECR Service



We will need to navigate to the Repositories section on the left side. This is where we will host our Dockerized Jenkins image.

Amazon ECR
Repositories

We can start by selecting create a repository.



In the options to create a repository, we will need to make the repository public for this scenario. Select a public repository

General settings

Visibility settings [Info](#)
Choose the visibility setting for the repository.

☐ Private
Access is managed by IAM and repository policy permissions.

☒ Public
Publicly visible and accessible for image pulls.

Once a repository is created, the visibility setting of the repository can't be changed.

In the details section, any name will work but follow the minimum requirements.

Detail

Repository name [Info](#)
A namespace can be included with your repository name (e.g. namespace/repo-name).

public.ecr.aws/v8g6g1g9/

7 out of 205 characters maximum (2 minimum). The name must start with a letter and can only contain lowercase letters, numbers, hyphens, underscores, and forward slashes.

Scroll down and create the repository

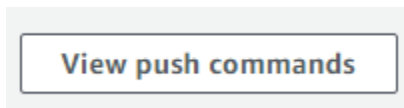
After the repository is created, copy the URI to a notepad. We will need this URI for a future step.

Public repositories (1)					View push commands	Delete	Edit	Create repository
<input type="text" value="Find repositories"/>				< 1 >				
	Repository name ▲	URI	Created at ▼					
<input type="radio"/>	jenkins	public.ecr.aws/v8g6g1g9/jenkins	October 09, 2021, 14:17:47 (UTC-04)					

We will need the helpful push commands to connect to the repository now. Select the application.



Click on the push commands in the top right.



These push commands will help us authenticate and push an image to our remote repository.

First we will need to retag our image with the URI of the repository. Run the following:
docker tag (APPNAME) (URI)

We will then use the first command to retrieve an authentication token and authentication our docker client to the ECR

1. Retrieve an authentication token and authenticate your Docker client to your registry.

Use the AWS CLI:

```
aws ecr-public get-login-password --region us-east-1 | docker login --username AWS --password-
```

You should see a Login Succeeded...

We then have to simply push the image to AWS using the uri...

docker push URI:latest

Once the image has successfully pushed, go back to ECR Repositories and click on the repository name to check if the image was successfully pushed.

Once that is successful, we can create the cluster now. Clusters allow us to group container instances which we can then run task requests on. Select clusters



Create a cluster

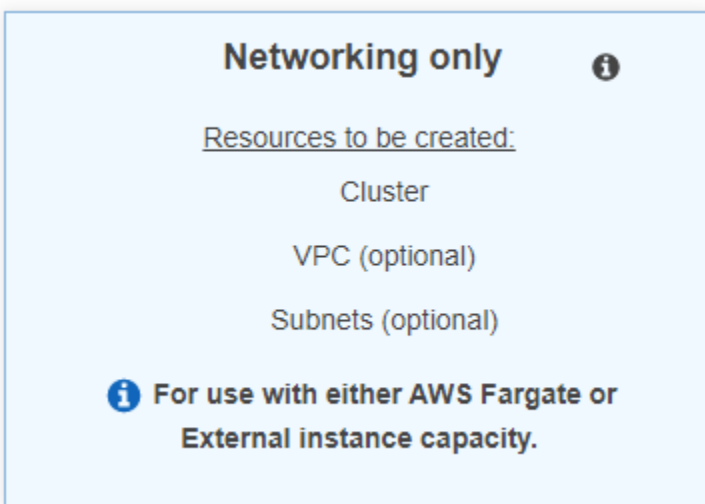
Clusters

An Amazon ECS cluster is a regional grouping of one instance type.

For more information, see the [ECS documentation](#).



For this scenario, we will be using AWS Fargate to host our application. Fargate is a serverless service compared to ECS. With Fargate, we do not have to provision resources. With ECS, we will have to manage the EC2 that is created. Select networking Only. This is used for AWS Fargate.



When configuring the cluster, we just have to worry about naming it.

Configure cluster

Cluster name*

Jenkins

Networking

Create a new VPC for your cluster to use. A VPC is an isolated portion of the AWS Cloud populated by AWS objects, such as Fargate tasks.

Create VPC

☐

Create a new VPC for this cluster

Tags

Key	Value
<div>Add key</div>	<div>Add value</div>

CloudWatch Container Insights

CloudWatch Container Insights is a monitoring and troubleshooting solution for containerized applications and microservices. It collects, aggregates, and summarizes compute utilization such as CPU, memory, disk, and network; and diagnostic information such as container restart failures to help you isolate issues with your clusters and resolve them quickly. [Learn more](#)

CloudWatch Container Insights

☐

Enable Container Insights

*Required

Cancel

Previous

Create

Once you choose a name, create it and choose the view cluster button.

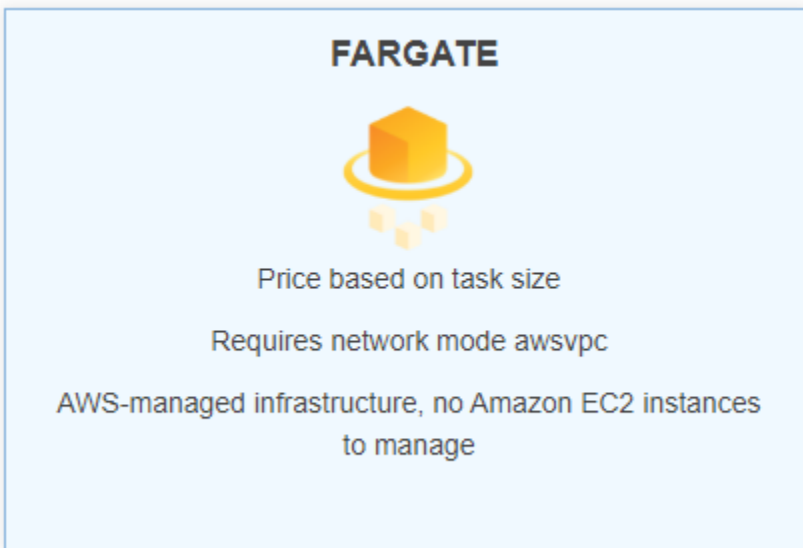
We can then move onto Task Definitions. Task definitions are required to run Docker containers in ECS. In other words, this will allow us to create our application to run on ECS. Click on Task Definitions

Amazon ECS

Clusters

Task Definitions

Select the launch type Fargate and go to the next step.



In this part, we will have to configure the task and container definitions. We will have to name it then attach a task role. If you have no task roles, then skip that.

Task definition name* ⓘ

Requires compatibilities* FARGATE


Task role ⓘ

Optional IAM role that tasks can use to make API requests to authorized AWS services. Create an Amazon Elastic Container Service Task Role in the [IAM Console](#) ↗

Network mode ⓘ

If you choose <default>, ECS will start your container using Docker's default networking mode, which is Bridge on Linux and NAT on Windows. Windows tasks support the <default> and awsvpc network modes.

If you don't have a task execution role, select the dropdown and create a new role. This role allows the tasks to pull our image and publish container logs to AWS CloudWatch for us.

Task execution role 

We will then have to give a certain task size. For this scenario, the lowest options will work.

Task memory (GB)

The valid memory range for 0.25 vCPU is: 0.5GB - 2GB.

Task CPU (vCPU)

The valid CPU for 0.5 GB memory is: 0.25 vCPU

We will need to add our application container that we pushed earlier onto the public ECR repository. Select the add container.

Container definitions

Add container

For this option we can give any name for the container name and for the image, we will need to paste the URI you copied earlier followed by :latest. We will also need to add the port mapping which will allow us to access our application on that specific port.

Add container

▼ Standard

Container name*

Jenkins

i

Image*

public.ecr.aws/v8g6g1g9/jenkins:latest

i

Private repository authentication*

☐

i

Memory Limits (MiB)

Soft limit ▼

128

i

+ Add Hard limit

Define hard and/or soft memory limits in MiB for your container. Hard and soft limits correspond to the `memory` and `memoryReservation` parameters, respectively, in task definitions.
ECS recommends 300-500 MiB as a starting point for web applications.

Port mappings

Container port	Protocol
8080	tcp ▼

+ Add port mapping

x

Once everything is configured, select add the bottom. We can then create the task definition in the bottom right.

The task definition will create. We can then select the task definition button once finished.

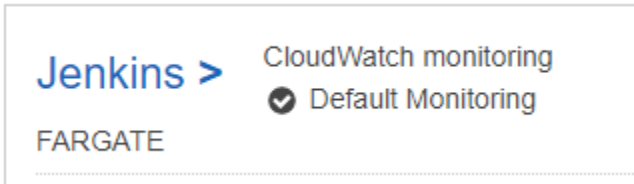
View task definition

We need to go to the clusters section to run a new task

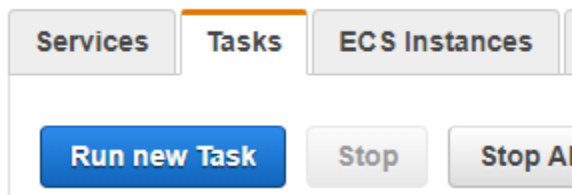
Amazon ECS

Clusters

Select the cluster that we created



Select the task tab and press Run a new task



We will have to configure this task. The following settings should be applied for our scenario.

Launch type ☒ FARGATE ☐ EC2 ☐ EXTERNAL ⓘ

[Switch to capacity provider strategy](#) ⓘ

Operating system family Linux ▼

Task Definition

Family jenkins ▼ Enter a value

Revision 1 (latest) ▼

Platform version LATEST ▼ ⓘ

Cluster Jenkins ▼

Number of tasks 1

Task Group ⓘ

Make sure the VPC is default. Remember the subnet that you choose because you will need it in future steps. (us-east-1c)

VPC and security groups

VPC and security groups are configurable when your task definition uses the awsvpc network mode.

Cluster VPC* vpc-8b9121f6 (172.31.0.0/16) ⓘ

Subnets* subnet-02d92c0eb42f7c8cf (172.31.0.0/20) - us-east-1c
assign ipv6 on creation: Disabled ⓘ

Security groups* Jenkin-426 **Edit** ⓘ

Auto-assign public IP ENABLED ⓘ

Edit the security group and add a new rule to allow port 8080 request anywhere.

Configure security groups

A security group is a set of firewall rules that control the traffic for your task. On this page, you can add rules to allow specific traffic to reach your task, or you can choose to use an existing security group. [Learn more](#).

Assigned security groups ☒ Create new security group
☐ Select existing security group

Security group name* Jenkin-426 ⓘ

Description Sat Oct 09 2021 14:30:35 GMT-0400 (Eastern Dayl ⓘ

Inbound rules for security group

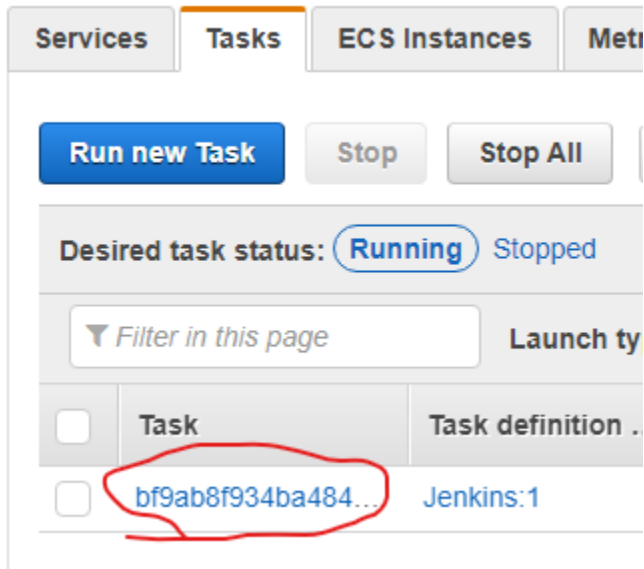
Type	Protocol	Port range	Source	
HTTP	TCP	80	Anywhere	0.0.0.0/0, ::/0 *
Custom TCP	TCP	8080	Anywhere	0.0.0.0/0, ::/0 *

[+ Add rule](#)

Once all the settings are configured, create the task.



Click on the Task that was created...



Once its running, copy the Public IP and go to it followed by the port 8080

Public IP 3.239.255.237

Example: 3.239.255.237:8080

Once the jenkins application loads and ask for your password go back to ECS and in the cluster task we created, go to the logs tab on the top

[Clusters](#) > [Jenkins](#) > Task: bf9ab8f934ba4846877a191a79094839

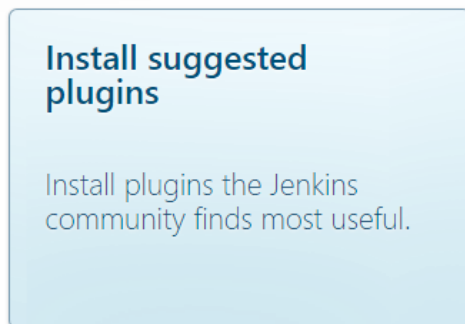
Task : bf9ab8f934ba4846877a191a79094839



The logs will contain the password for this Jenkins application. Copy it and paste it into jenkins

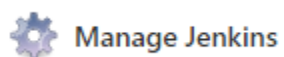
Filter logs		×	All	30s	5m
Timestamp (UTC+00:00) ▼	Message				
▶ 2021-10-09 14:40:52	2021-10-09 18:40:52.837+0000 [id=52] INFO hudson.model.AsyncPeriodicWork#lambda\$doRun\$0: Starte				
▶ 2021-10-09 14:38:29	2021-10-09 18:38:29.834+0000 [id=28] INFO jenkins.install.SetupWizard#init:				
▶ 2021-10-09 14:38:29	*****				
▶ 2021-10-09 14:38:29	*****				
▶ 2021-10-09 14:38:29	*****				
▶ 2021-10-09 14:38:29	Jenkins initial setup is required. An admin user has been created and a password generated.				
▶ 2021-10-09 14:38:29	Please use the following password to proceed to installation:				
▶ 2021-10-09 14:38:29	83b6c74f9a1b4f1e8e1fe74cf566e2e7				
▶ 2021-10-09 14:38:29	This may also be found at: /var/jenkins_home/secrets/initialAdminPassword				
▶ 2021-10-09 14:38:29	*****				

Install the suggest plugins



Create a user

Once inside Jenkins Dashboard, we will need to install two Plugins. Select Manage Jenkins in the left side of the Dashboard.



Select Manage plugins option.



Manage Plugins

Add, remove, disable or enable plugins that can extend the functionality of Jenkins.


We then need to switch to the available tab to see available plugins. Search up and install “Docker Pipeline” and “Amazon EC2” plugins.

Updates	Available	Installed	Advanced
Install ↑	Name		
<input checked="" type="checkbox"/>	Docker Pipeline Deployment DevOps docker pipeline Build and use Docker containers from pipelines.		
<input checked="" type="checkbox"/>	Amazon EC2 agent aws Cloud Providers Cluster Management spotinst This plugin integrates Jenkins with Amazon EC2 or anything implementing the EC2 API's such as an Ubuntu.		

Once both are selected, Download now and install after restart.

Download now and install after restart

Select the following option to complete all installation.

 ☐ Restart Jenkins when installation is complete and no jobs are running


Creating the Agent's EC2

While Jenkins restarts, we can create an EC2 instance in the meantime. **This EC2 instance will be our Jenkins agent. This instance will have more resources than the Jenkins Controller has. The Jenkins controller application we hosted on Fargate only gives instructions to the agents. The agents then use resources to complete the task given.**

In AWS EC2 Service, create a new Instance.

Launch instances

For this scenario we will use Ubuntu Server 20.04 LTS AMI.

**Ubuntu Server 20.04 LTS (HVM), SSD Volume Type** - ami-09e67e426f25ce0d7 (64-bit x86) / ami-00d1ab6b335f217cf (64-bit Arm)

Free tier eligible

Ubuntu Server 20.04 LTS (HVM),EBS General Purpose (SSD) Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).

Root device type: ebs

Virtualization type: hvm

ENA Enabled: Yes

Select

☒ 64-bit (x86)
☐ 64-bit (Arm)

The instance type will be a simple t2.micro

For the instance details we need to make sure to select the subnet we chose above when creating the Task. (us-east-1c)

Network ⓘ

vpc-8b9121f6 (default) ⌵

↻ Create new VPC

Subnet ⓘ

subnet-02d92c0eb42f7c8cf | Default in us-east-1c ⌵
4091 IP Addresses available

Create new subnet

Auto-assign Public IP ⓘ

Use subnet setting (Enable) ⌵

For the storage section, we can keep the defaults and continue to tags.

It's best practice to always create a tag.

Key (128 characters maximum)	Value (256 characters maximum)
Name	Docker - Jenkins Agent

For Security Groups we need to create a new security group to allow SSH access to our EC2.

Security group name:

Description:

Type <small>i</small>	Protocol <small>i</small>	Port Range <small>i</small>	Source <small>i</small>	Description
SSH <small>v</small>	TCP	22	Custom <small>v</small>	0.0.0.0/0

e.g. S

Launch the instance and select a Key Pair that you have.

While waiting for the instance to start...

Configuring the Agent's Task

Fork the Deployment 7 Repository -> https://github.com/kura-labs-org/DEPLOY07_ECS

When Forking a repository, we are simply making a copy of our own where we can edit it as we like and make a pull request to merge our changes to the original repository.

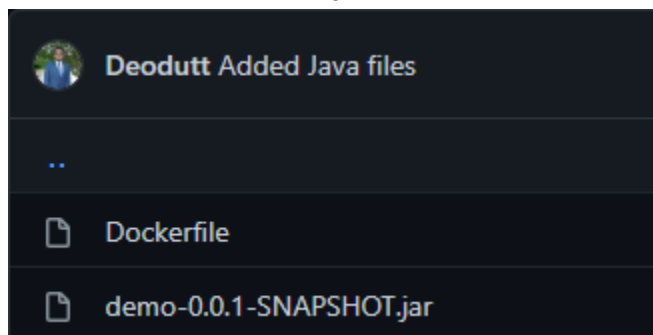
For this assignment, we need to move a java application we created into the repository. We then need to create a Dockerfile that will run a simple Java application. The following Dockerfile will pull the openjdk:11 image from DockerHub. It will then copy the local java application and move it to a new path on the container. The CMD function will then run a command once the container runs.

```
FROM openjdk:11
COPY ./demo-0.0.1-SNAPSHOT.jar app.jar
CMD ["java", "-jar", "app.jar"]
```

Inside your repository,

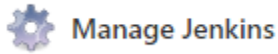
Add a the java archive file and a dockerfile with your instructions for creating the Java application to your repository.

demo-0.0.1-SNAPSHOT.jar is a simple Java application from Kura Labs Class.



Once that is set up, we will need to go back into our Jenkins application that is hosted on Fargate and set up a node. A node is simply an agent. This agent will do all the building for us.

Log back in and go to the Dashboard. Once inside, select manage jenkins



Select manage nodes.



Manage Nodes and Clouds

Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

In the left side, click create new node



New Node

We can name the agent anything. Select Permanent agent and proceed.

Node name

☒ **Permanent Agent**

Adds a plain, permanent agent to the system. Select this type if no other agent types apply.

OK

In the next steps we will need to fill out the following. Remote root directory is important. This is based on the Ubuntu 20.04 image we chose when creating the EC2. The label will also be vital when we create the pipeline script. Make sure to write down the label name.

Name	<input type="text" value="agent"/>
Description	<input type="text" value="Agent that is hosted on ec2"/>
Number of executors	<input type="text" value="1"/>
Remote root directory	<input type="text" value="/home/ubuntu/"/>
Labels	<input type="text" value="jenkins-agent"/>

In the following settings, we need to set up how we will access our agent which is hosted on an EC2. When we created our EC2, we set up SSH access from anywhere in the security group. For launch methods we need to select launch via SSH.

The Host IP is the private IPv4 of the EC2 we created. You can find yours by opening AWS EC2 service in another tab and selecting the instance you created.

Usage

Use this node as much as possible

Launch method

Launch agents via SSH

Host

172.31.15.231

Private IP

We need to add credentials to access the EC2 instance.

Credentials

- none - ▾

Add

The selected

Jenkins

For the settings, change the Kind to SSH username with a private key. The username is important. It will be the same as our AMI. Since we are using ubuntu, the username will be ubuntu.

Kind

SSH Username with private key

Scope

Global (Jenkins, nodes, items, all child items, etc)

ID

agent

Description

ssh into ec2 agent

Username

ubuntu

We then need to select Enter directly for the Private key. The following value that we enter inside this field will contain our private keypair we use to SSH into the instance. In most cases it's a .pem file. You will need to create a copy and change it to a txt file to read the value.

Private Key

☒ Enter directly

Key

add Ssh key used to Ssh into EC2

Enter New Secret Below

Once all the credentials is filled, add it

Select the credentials that you just created. You will also need to change the host key verification strategy to "Non Key Verification Strategy"

Credentials

ubuntu (ssh into ec2 agent) ▼

Add ▼

ubuntu (ssh into ec2 agent)

Host Key Verification Strategy

Non verifying Verification Strategy

Availability

Keep this agent online as much as possible

Once everything is set up, save it.

Save

We will then need to SSH manually into the EC2 and set up some requirements before the Jenkins controller can access it.

SSH into the agent using the following. The key.pem is the keypair you created. The publicIPv4 is from the EC2 we created.

```
ssh -i key.pem ubuntu@publicIPv4
```

Once successfully inside the instance, run the following commands one at a time. We will update the instance and then install Java. Java is a requirement to allow the Jenkins controller to connect to the agent.

```
sudo apt update && sudo apt upgrade -y  
sudo apt install default-jre -y
```

We will also need to install Docker on this instance because it will do all the building for us. Run the following commands one at a time.

```
curl -fsSL https://get.docker.com -o get-docker.sh && sudo bash get-docker.sh
```

```
sudo usermod -aG docker $USER  
newgrp docker  
sudo shutdown -r now
```

This will close the connection to the ec2 instance. We can then go back into the Jenkins Controller and refresh the status of the nodes. You should see the agent we created is online (in sync)



agent

Linux (amd64)

In sync

We will now need to get a docker access token which will be used in the pipeline to build and push our images to DockerHub. Go to -> <https://hub.docker.com/settings/security> and signup/signin. Once you are logged in to Dockerhub, select the security tab



We need to create an access token which will be used to access Dockerhub without a password. Create an new access token

Access Tokens

It looks like you have not created any access tokens.
Docker Hub lets you create tokens to authenticate access. Treat personal access tokens as alternatives to your password. [Learn more](#)

New Access Token

You can put anything inside the description.

New Access Token

A personal access token is similar to a password except you can have many tokens and revoke access to each one at any time. [Learn more](#)

Access Token Description *

Deployment 7 - Jenkins

Access permissions

Read, Write, Delete

Read, Write, Delete tokens allow you to manage team members. This is an admin privilege.

Cancel

Generate

Generate your token and make SURE to copy the personal access token. Navigate back to the Jenkins Controller and go to the dashboard. Once inside the dashboard, select Manage Jenkins.



Manage Jenkins

Navigate to Manage Credentials



Manage Credentials

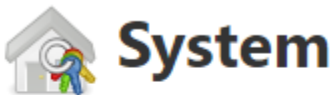
Configure credentials

Select the following Credentials.



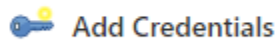
T	P	Store ↓	Domain	ID	Name
		Jenkins	(global)	agent	ubuntu (ssh into ec2 agent)

Once inside that credentials, select Global Credentials



Domain
Global credentials (unrestricted)

This is where we will configure our credentials to DockerHub. Choose Add credentials



For the following settings, the Username should be your DockerHub username. The password is the personal access token that you created previously. The ID will simply have your DockerHub username followed by -dockerhub. (ex. ricardo-dockerhub)

Kind

Username with password

Scope

Global (Jenkins, nodes, items, all child items, etc)

Username

rixardo

☐ Treat username as secret

Password

.....

ID

rixardo-dockerhub

Description

Once set up, you can save the input

Creating the Pipeline

We will have to set up a pipeline script to build our application and push it to Dockerhub.

In the Jenkins Controller, go to the dashboard and select New Item




New Item

You can name the item anything and select Pipeline.

Enter an item name

» Required field

**Pipeline**
Orchestrates long-running activities that can span multiple build a
and/or organizing complex activities that do not easily fit in free-s

In the pipeline configuration, scroll down to Pipeline and change the definition to “Pipeline scripts from SCM”.

In the SCM select “Git”

For the Repository URL paste in your repository URL that you forked previously.

We will need to add new GitHub credentials next.



In the following settings, we will create a Username with password. Inside of that the username will be your GitHub username. The password will be your GitHub personal access token that you can create -> [here](#) The ID can simple be jenkins-webhook-id.

Jenkins Credentials Provider: Jenkins

Add Credentials

Domain
Global credentials (unrestricted)

Kind
Username with password

Scope
Global (Jenkins, nodes, items, all child items, etc)

Username
Deodutt

☐ Treat username as secret

Password
.....

ID
jenkins-webhook-id

Description

Once created we need to select the credentials we just made

Credentials

- none -

- none -

Deodutt/*****

Once we created the credentials, we will need to change the branch specifier to “*/main”. Most branches have migrated from using master to main

Pipeline

Definition

Pipeline script from SCM

SCM

Git

Repositories

Repository URL

https://github.com/Deodutt/DEPLOY07_ECS

Credentials

Deodutt/*****

Add

Branches to build

Branch Specifier (blank for 'any')

*/main

Repository browser

(Auto)

Additional Behaviours

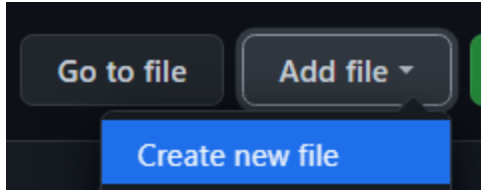
Add

Script Path

Jenkinsfile

Save the pipeline script and we can proceed.

Navigate to your GitHub and create a new file



Name is Jenkinsfile and paste the following script in it

```
pipeline {
  agent {
    label "jenkins-agent"
  }

  environment {
    DOCKERHUB_CREDENTIALS = credentials("rixardo-dockerhub")
  }

  stages {
    stage('Build') {
      steps {
        sh 'docker build -t rixardo/deploy07 .'
        sh 'echo "completed build"'
      }
    }

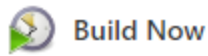
    stage('Login') {
      steps {
        sh 'echo $DOCKERHUB_CREDENTIALS_PSW | docker login -u $DOCKERHUB_CREDENTIALS_USR --password-stdin'
        sh 'echo "completed login"'
      }
    }

    stage('Push'){
      steps {
        sh 'docker push rixardo/deploy07:latest'
        sh 'echo "completed push"'
      }
    }
  }
}
```

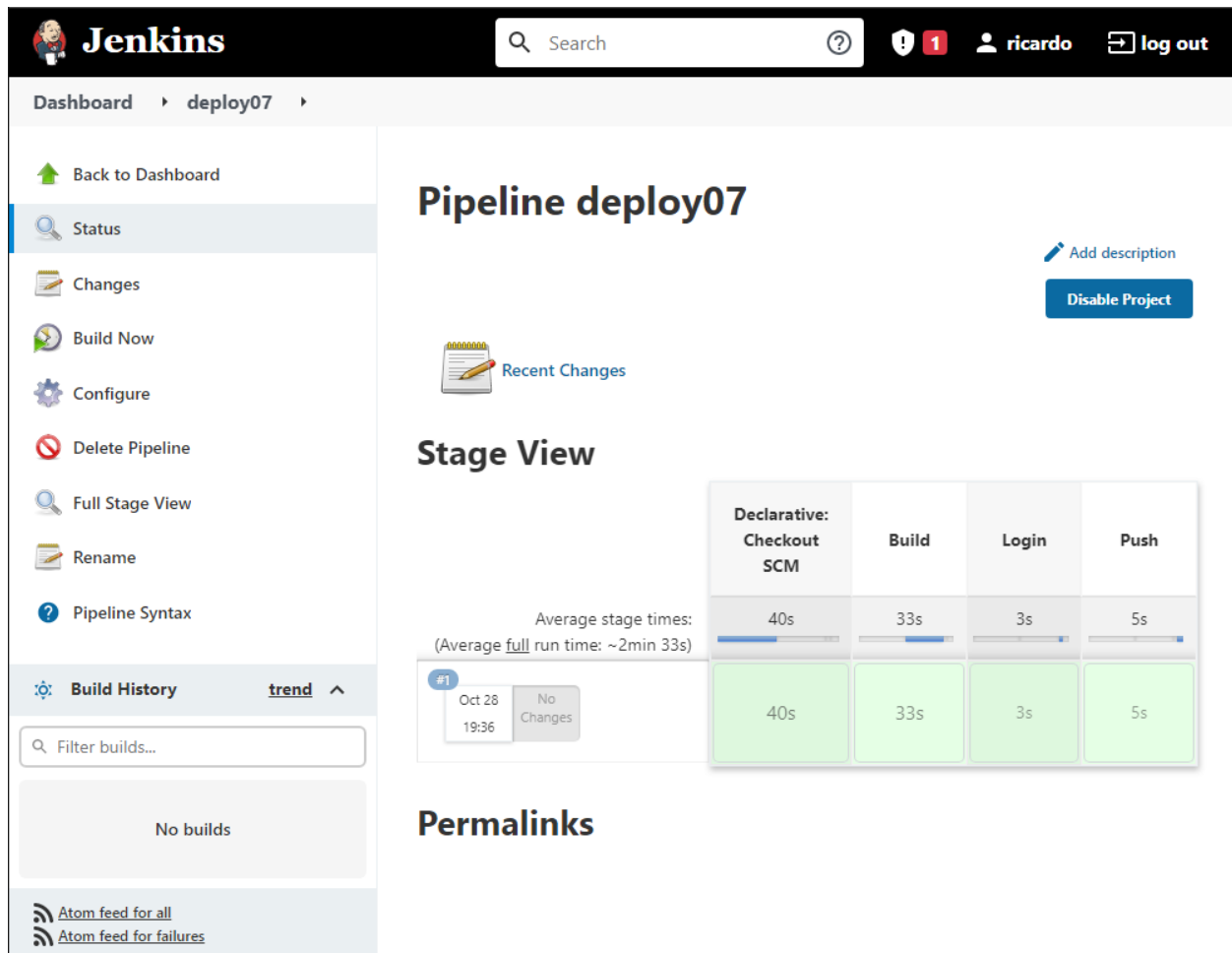
You will need to make sure to change the agent label to what you labeled your agent. You then need to change the DOCKER_CREDENTIALS to match your user (username-dockerhub). You will also have to change the docker commands which tags your username to your docker image(sh 'docker build -t user/deploy07 .'). Lastly you need to change the docker push command in the push stage to match your user (sh 'docker push

user/deploy07:latest") **Once that is set up you can commit the new file and go back to the Jenkins controller.**

Once inside the Jenkins controller, we simply have to select Build Now in the pipeline that we created

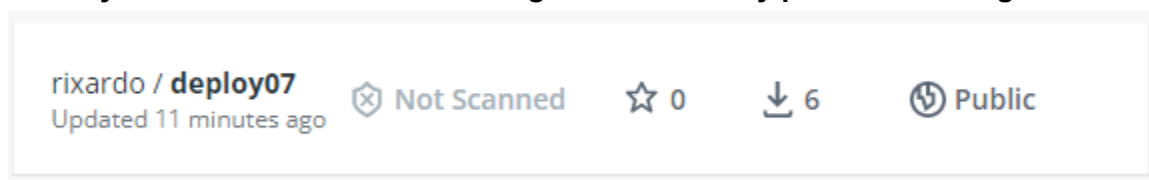


Finally, you should have a successful test build

The screenshot shows the Jenkins web interface for a pipeline named "deploy07". The left sidebar contains navigation links: Back to Dashboard, Status, Changes, Build Now, Configure, Delete Pipeline, Full Stage View, Rename, Pipeline Syntax, Build History (with a trend icon), and Atom feeds. The main content area shows the "Pipeline deploy07" view. It includes a "Recent Changes" section, a "Stage View" table, and a "Permalinks" section. The "Stage View" table shows the average stage times for four stages: Declarative: Checkout SCM (40s), Build (33s), Login (3s), and Push (5s). The "Permalinks" section is currently empty.

Stage	Declarative: Checkout SCM	Build	Login	Push
Average stage times:	40s	33s	3s	5s
(Average full run time: ~2min 33s)				

Check your DockerHub to see if the Agent successfully pushed the image.

The screenshot shows a DockerHub repository card for "rixardo / deploy07". The card displays the repository name, the update time "Updated 11 minutes ago", a "Not Scanned" status, 0 stars, 6 downloads, and a "Public" visibility setting.

rixardo / **deploy07**
Updated 11 minutes ago Not Scanned ☆ 0 ↓ 6 Public

Cleanup

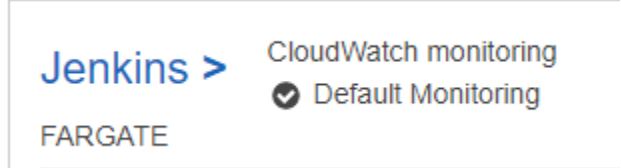
Once you have gotten all that, we should clean up to avoid AWS Fees

Terminate the AWS EC2 - Docker Agent

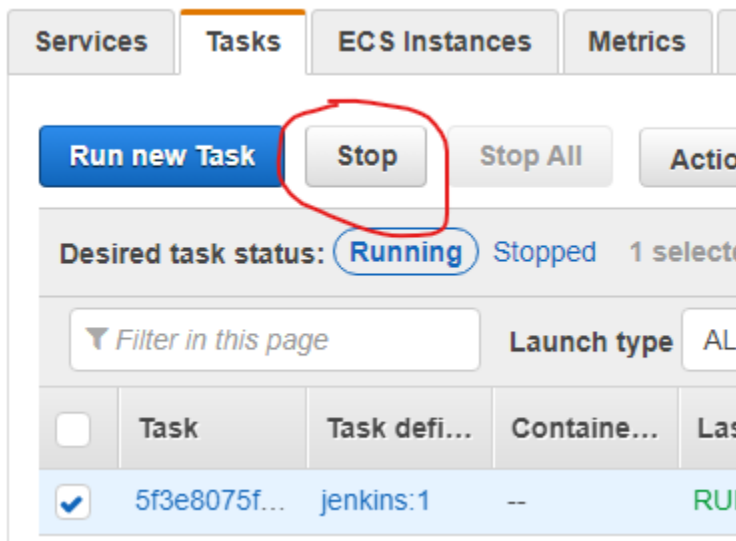
Navigate to Elastic Container Service

Amazon ECS
| Clusters

Select Cluster and pick the Cluster you created



Select the Task tab and stop the task



In the top right of the page,

[Clusters](#) > [Jenkins](#)

Cluster : Jenkins

Update Cluster

Delete Cluster

Get a detailed view of the resources on your cluster.

Go to Task Definitions

[Amazon ECS](#)

[Clusters](#)

Task Definitions

Click on your task definition you created

<input type="checkbox"/>	Task Definition
<input type="checkbox"/>	jenkins

Deregister the certain revision

Create new revision

Actions ▴

Status: **Active** Inactive

Filter in this page

<input type="checkbox"/>	Task Definition Name
<input checked="" type="checkbox"/>	jenkins:1

Run Task

Create Service

Update Service

Deregister

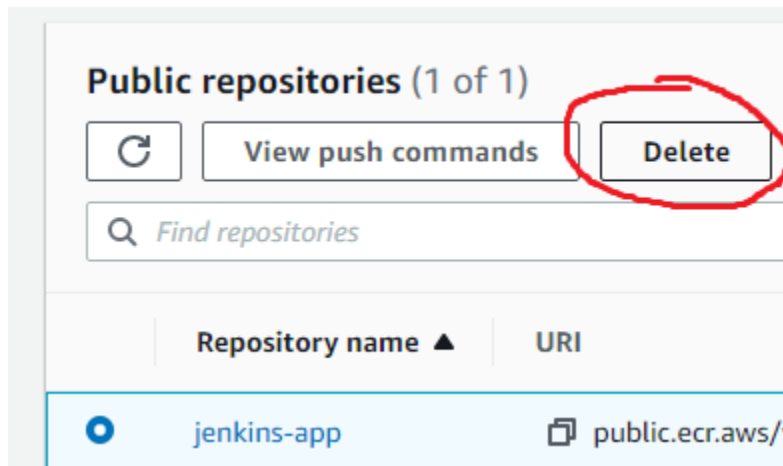
Edit tags

Navigate to repositories

[Amazon ECR](#)

[Repositories](#)

Select the repository and delete it



You are finished!