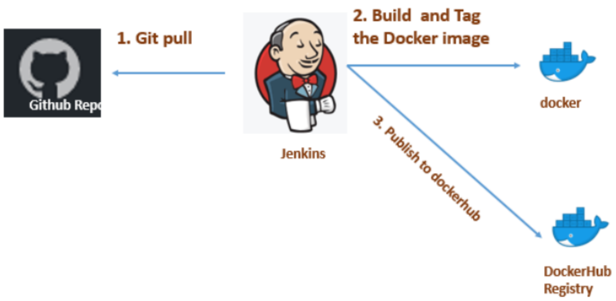
Deployment 7

Kenneth Tan

The Task:

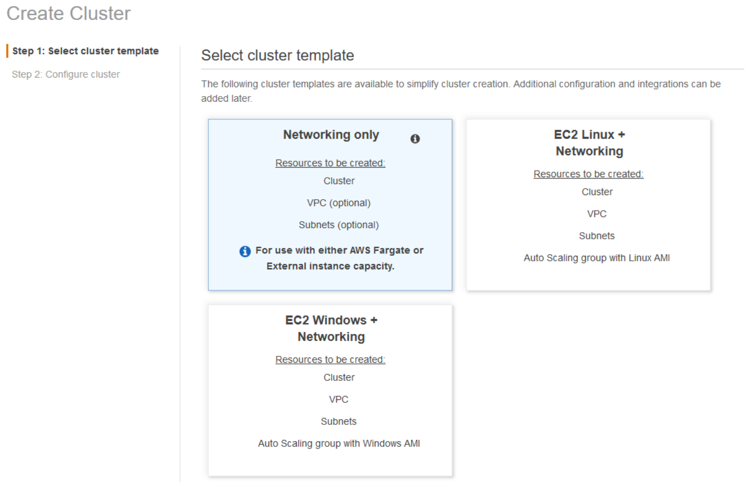
To pull a jar file from github into jenkins to both build and push that image to dockerhub.



Requirements to Accomplish this task:

ECS

- This will act as the master and host jenkins
- Created a Cluster. The container for the master will be here



Create Cluster

Select cluster template

The following cluster templates are available to simplify cluster creation. Additional configuration and integrations can be added later.

Networking only

Resources to be created:

Cluster

VPC (optional)

Subnets (optional)

For use with either AWS Fargate or External instance capacity.

EC2 Linux + Networking

Resources to be created:

Cluster

VPC

Subnets

Auto Scaling group with Linux AMI

EC2 Windows + Networking

Resources to be created:

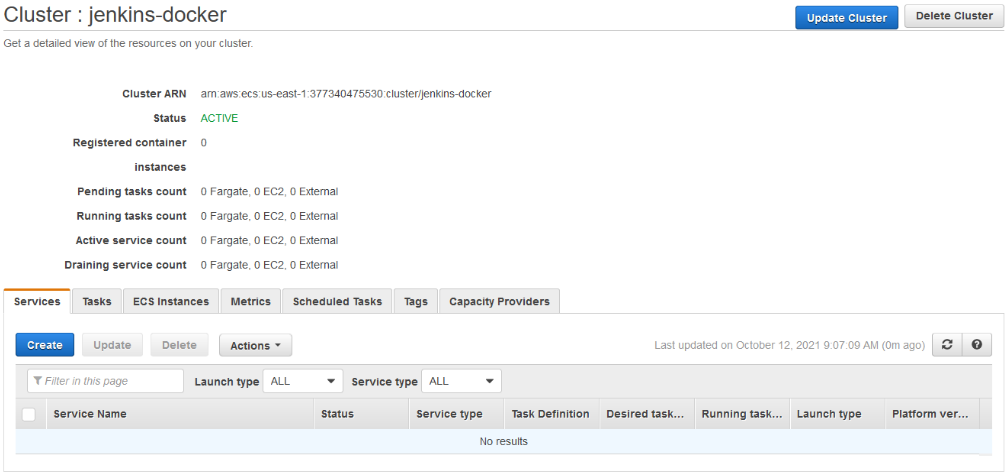
Cluster

VPC

Subnets

Auto Scaling group with Windows AMI

-



Cluster : jenkins-docker

Get a detailed view of the resources on your cluster.

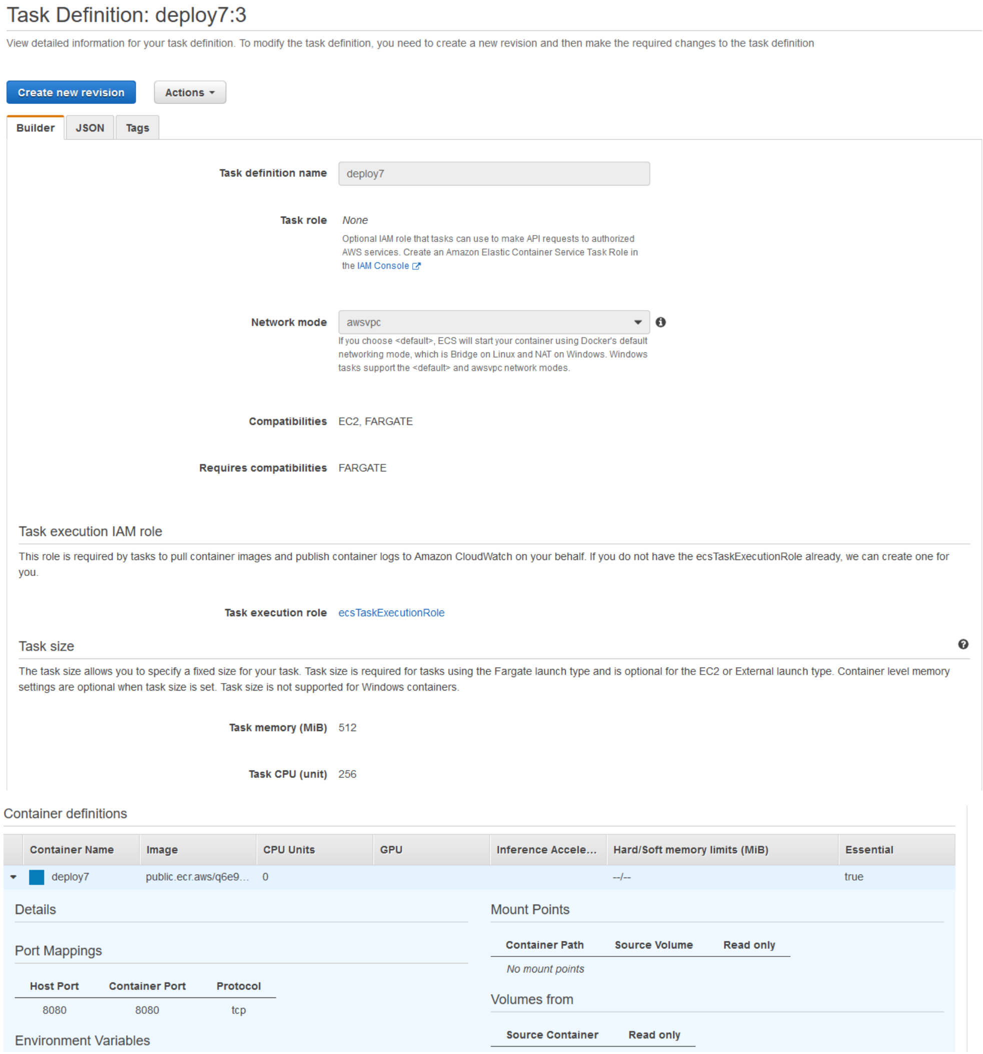
Cluster ARN	arn:aws:ecs:us-east-1:377340475530:cluster/jenkins-docker
Status	ACTIVE
Registered container instances	0
Pending tasks count	0 Fargate, 0 EC2, 0 External
Running tasks count	0 Fargate, 0 EC2, 0 External
Active service count	0 Fargate, 0 EC2, 0 External
Draining service count	0 Fargate, 0 EC2, 0 External

Last updated on October 12, 2021 9:07:09 AM (0m ago)

Launch type ALL Service type ALL

Service Name	Status	Service type	Task Definition	Desired task...	Running task...	Launch type	Platform ver...
No results							

- Created a repository in ECR. This will have the image that will create the master container. The image choses was jenkins/jenkins from docker hub. This was pushed up to ECR from my local machine. I named this image ‘deploy7’ on ECR.
- Created a Task Definition. This task definition will allow for computing and memory resources for my container. This is created based on the ‘deploy7’ image in ECR.



Task Definition: deploy7:3

View detailed information for your task definition. To modify the task definition, you need to create a new revision and then make the required changes to the task definition

Task definition name: deploy7

Task role: *None*

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: awsvpc

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.

Compatibilities: EC2, FARGATE

Requires compatibilities: FARGATE

Task execution IAM role

This role is required by tasks to pull container images and publish container logs to Amazon CloudWatch on your behalf. If you do not have the ecsTaskExecutionRole already, we can create one for you.

Task execution role: ecsTaskExecutionRole

Task size

The task size allows you to specify a fixed size for your task. Task size is required for tasks using the Fargate launch type and is optional for the EC2 or External launch type. Container level memory settings are optional when task size is set. Task size is not supported for Windows containers.

Task memory (MiB): 512

Task CPU (unit): 256

Container definitions

Container Name	Image	CPU Units	GPU	Inference Accele...	Hard/Soft memory limits (MiB)	Essential
deploy7	public.ecr.aws/q6e9...	0			--/--	true

Details

Port Mappings

Host Port	Container Port	Protocol
8080	8080	tcp

Environment Variables

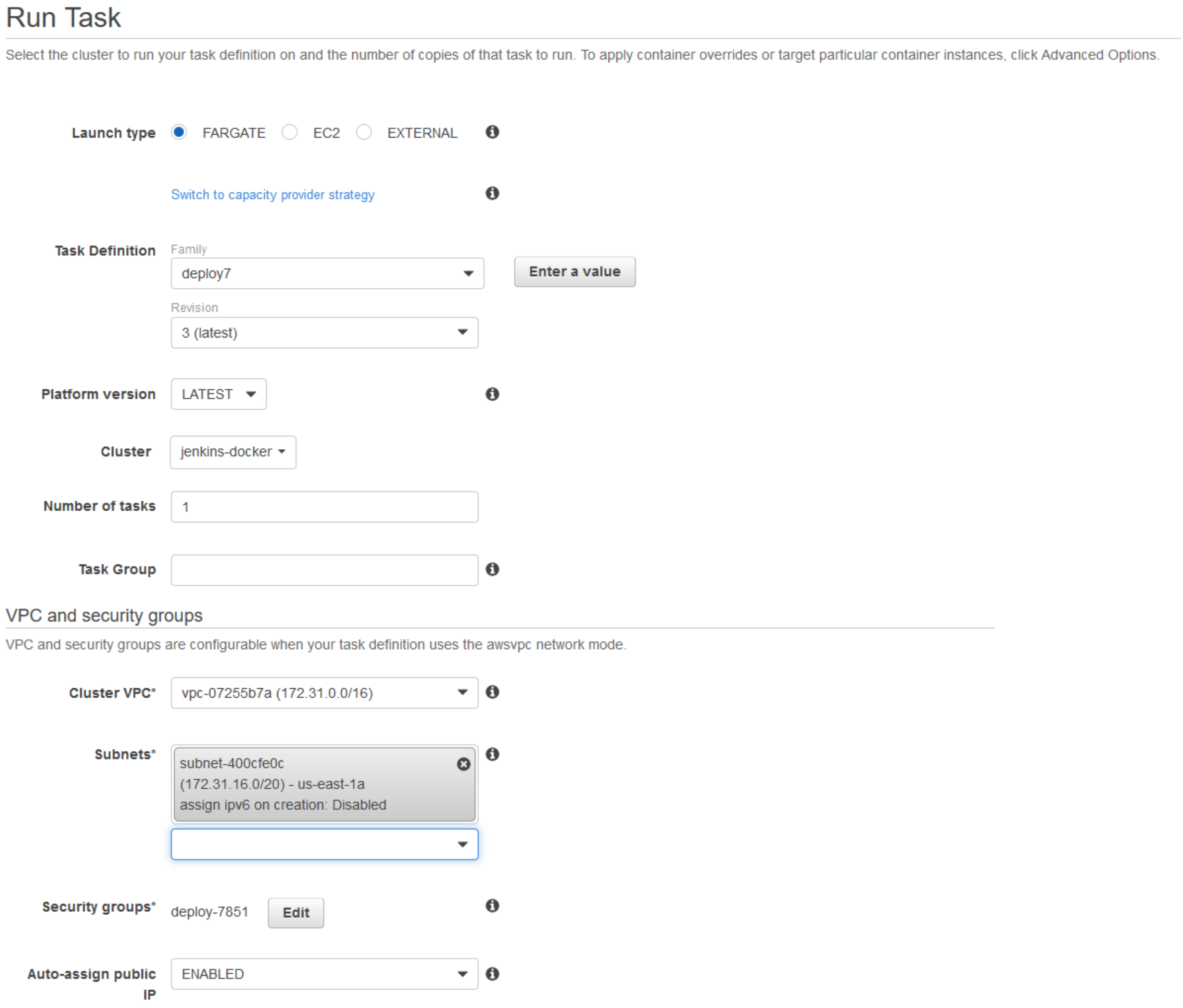
Mount Points

Container Path	Source Volume	Read only
No mount points		

Volumes from

Source Container	Read only

- Task to Run. This will launch the container on the cluster with the task definition specifications. This will also allow me to ensure that the container is in the same subnet as my EC2 agent so the 2 can connect .



Run Task

Select the cluster to run your task definition on and the number of copies of that task to run. To apply container overrides or target particular container instances, click Advanced Options.

Launch type ● FARGATE ○ EC2 ○ EXTERNAL

Switch to capacity provider strategy

Task Definition: Family deploy7 — Enter a value

Revision: 3 (latest)

Platform version: LATEST

Cluster: jenkins-docker

Number of tasks: 1

Task Group:

VPC and security groups

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

Cluster VPC*: vpc-07255b7a (172.31.0.0/16)

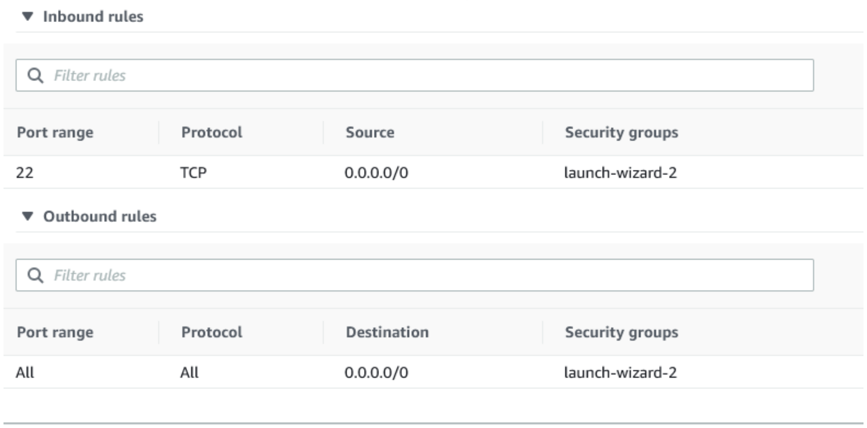
Subnets*: subnet-400cfe0c (172.31.16.0/20) - us-east-1a assign ipv6 on creation: Disabled

Security groups*: deploy-7851 Edit

Auto-assign public IP: ENABLED

EC2

- This will act as the agent for building the docker image
- Included Docker Pipeline and Amazon EC2 plugins on installation
- Created an Ubuntu Instance with below inbound and outbound permissions



▼ Inbound rules

Filter rules

Port range	Protocol	Source	Security groups
22	TCP	0.0.0.0/0	launch-wizard-2

▼ Outbound rules

Filter rules

Port range	Protocol	Destination	Security groups
All	All	0.0.0.0/0	launch-wizard-2

Dockerfile

- What will be called to build the image
- This file is in the github repository

```
# pull openjdk image from docker
FROM openjdk:latest
#copy contents from current directory(github repo) into target directory
COPY . /usr/src/myapp
#change working directory to the target directory
WORKDIR "/usr/src/myapp"
#create image from java jar file
ENTRYPOINT ["java", "-jar", "demo-0.0.1-SNAPSHOT.jar"]
```

Jenkins Pipeline file

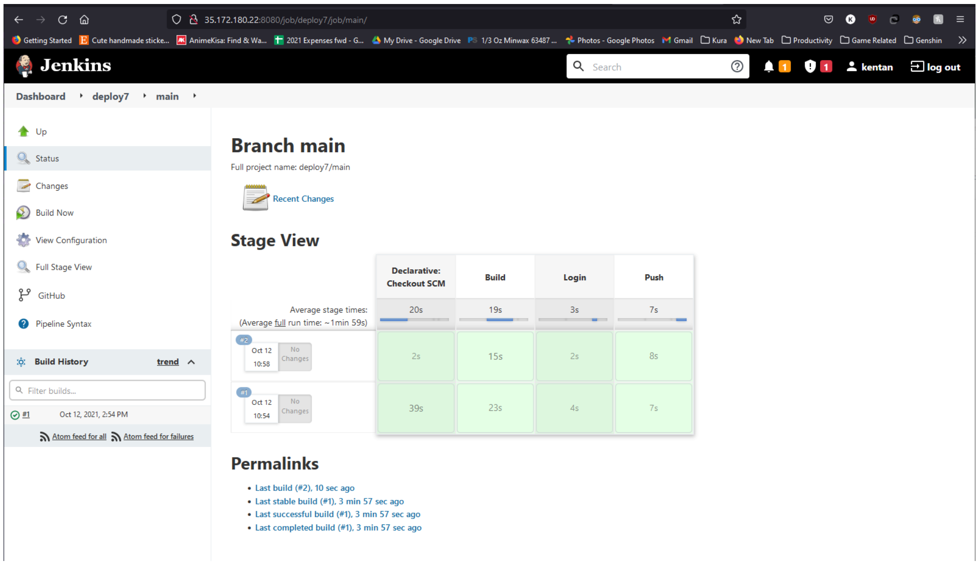
- To automate the build and push process of the docker image

```
pipeline {
  agent { label 'agent1'} /* This calls the agent to build the following  */
  environment{
        DOCKERHUB_CREDENTIALS = credentials('kentan404-dockerhub') /* references the
credential key given in jenkins */

  }
  stages {
        stage ('Build') {
        steps {
        sh 'docker build -f dockerfile .' /* */
        }
        }
        stage ('Login') {
        steps {
        sh 'echo $DOCKERHUB_CREDENTIALS_PSW | docker login -u
$DOCKERHUB_CREDENTIALS_USR --password-stdin'
        /* logs into docker with credential info */
        }
        }
        stage ('Push') {
        steps {
        sh '''
        var1=$( docker images --filter 'dangling=true' --format "{{.ID}}" )
        docker tag $var1 kentan404/deploy7repo:deploy7repo
        /* when the image is created from the dockerfile, there is no name or tag. Above logic
will search for images with no name and retag them*/
```

```
            docker push kentan404/deploy7repo:deploy7repo /* push to repository for user kentan404 with name deploy7repo*/
            docker image prune -a -f /* cleans environment of all images for next run*/
            '''
            }
            }
  }
}
```

Screenshots



#1 Oct 12, 2021, 2:54 PM

Branch main

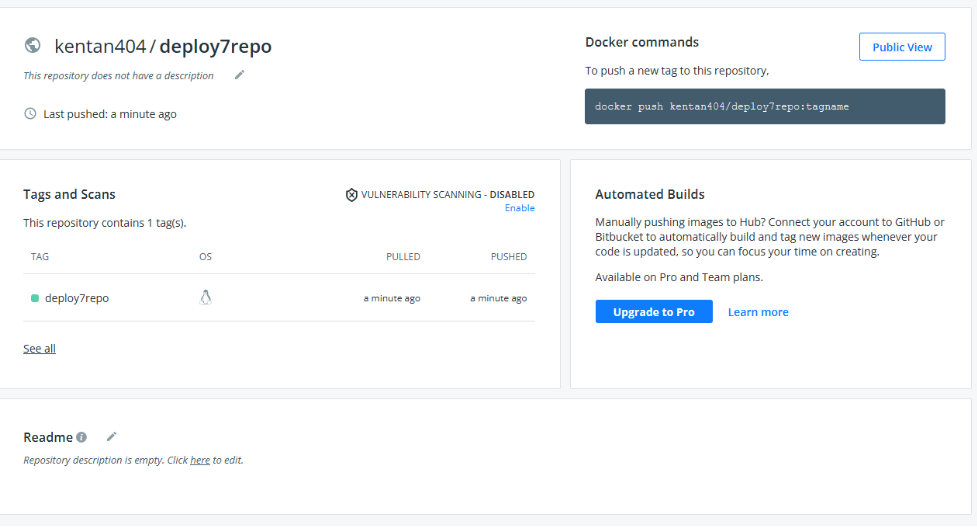
Full project name: deploy7/main

Stage View

	Declarative: Checkout SCM	Build	Login	Push
Average stage times: (Average full run time: ~1min 59s)	20s	19s	3s	7s
#2 Oct 12 10:58 No Changes	2s	15s	2s	8s
#1 Oct 12 10:54 No Changes	39s	23s	4s	7s

Permalinks

- Last build (#2), 10 sec ago
- Last stable build (#1), 3 min 57 sec ago
- Last successful build (#1), 3 min 57 sec ago
- Last completed build (#1), 3 min 57 sec ago



kentan404 / deploy7repo

This repository does not have a description

Last pushed: a minute ago

Docker commands

To push a new tag to this repository,

```
docker push kentan404/deploy7repo:tagname
```

Tags and Scans

VULNERABILITY SCANNING - DISABLED

This repository contains 1 tag(s).

TAG	OS	PULLED	PUSHED
deploy7repo		a minute ago	a minute ago

Automated Builds

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available on Pro and Team plans.

Readme

Repository description is empty. Click here to edit.

Issues/Challenges

One of the major blockers for this deployment was the permission to execute docker commands in the EC2 agent. I had received numerous errors including denied access to resources or problems logging in. I have solved this issue by running

```
sudo usermod -a -G docker ubuntu
```

This removed restrictions as it made ubuntu the primary user group for docker to ubuntu, which is the user OS. This means ubuntu users are allowed to execute docker commands now. I then executed

```
Docker login -u kentan404
```

This successfully logged me in and I attempted to rerun my build and the error was gone.

Another bug issue I had was my push block in my pipeline ran into errors because it would search for images without names indicated in the dangling = true. Because my script stores all of the images with no names into the venv variable $var1, after a few runs, it would accumulate multiple IDs and passing them all as arguments in

```
Docker tag <image name/id> <new tag name>
```

The error arose when multiple images were passed to have their tags changed. I rectified this by including docker prune -a at the end of the the block, which removes all images from the machine. The dockerfile will always create a new set of images so I can rerun this pipeline without worrying about creating duplicates or having errors when running it as it will start each build with a fresh environment.

I would improve this further if I could automate tagging and making uploads to the dockerhub repo more modular- you can choose to push to different repos if desired. However, this current iteration does a good job building and pushing onto the dockerhub repo.

One last issue I had run into was the task that was running on my cluster would sometimes stop running on its own causing my Jenkins to freeze and disconnect. In order for me to continue working on the deployment I would need to create a new running task identical to the original but I would need to reinstall plugins and recreate the agent and credentials. This is because a new task is another instance of a container being run so it is a new environment when it starts up. I would consider this to be the biggest slow down as i would need to wait for plugins to load.