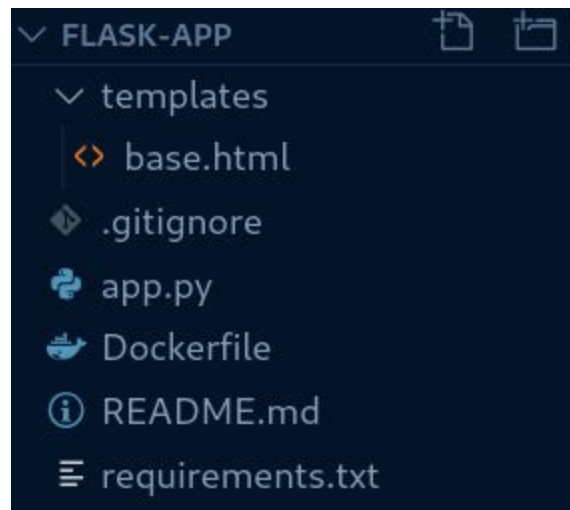


Steps:

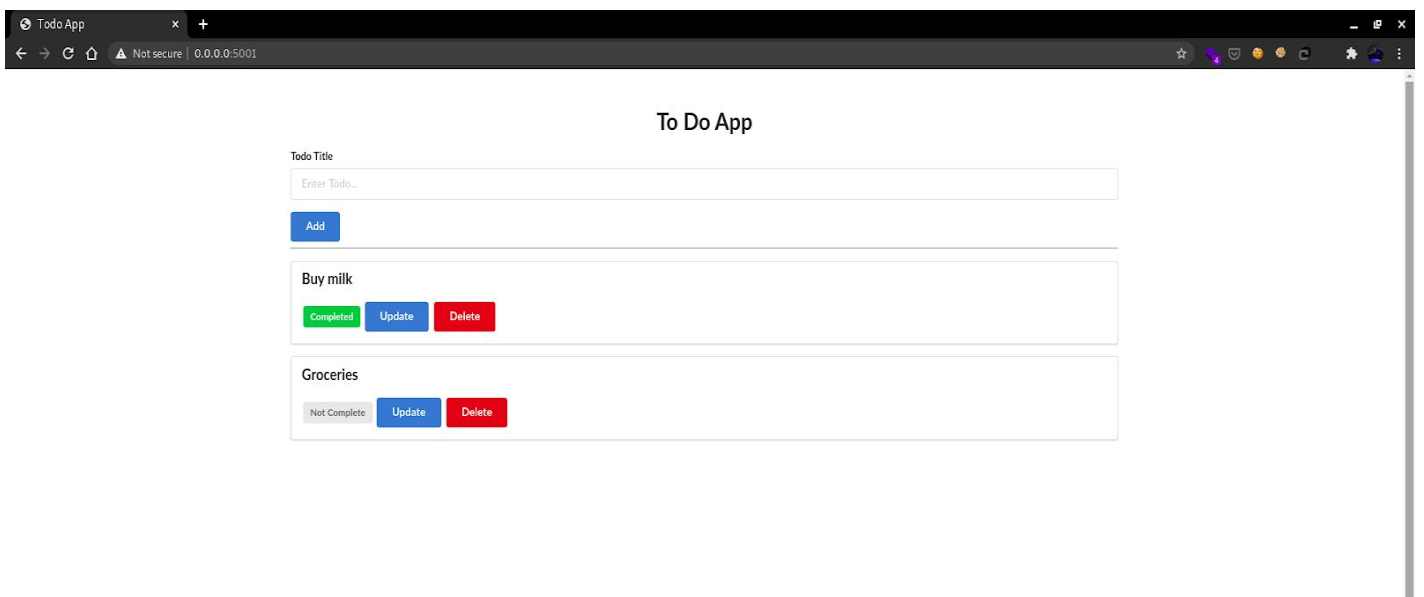
1. Build a basic Python Flask app. This will be a basic to-do task application. The structure of the files is as follows:




2. The direct command to run the application is **python3 app.py**

```
python3 app.py
/usr/lib/python3/dist-packages/sqlalchemy/orm/query.py:196: SyntaxWarning: "is not" with a literal. Did you mean "!="?
if entities is not ():
* Running on http://0.0.0.0:5001/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 492-390-025
127.0.0.1 - - [07/Oct/2020 01:54:58] "GET / HTTP/1.1" 200 -
```

Visiting the application at <http://0.0.0.0:5001/>



3. The Dockerfile for the application is as follows:

```
 Dockerfile > ...  
1 FROM python:alpine3.7  
2 COPY . /app  
3 WORKDIR /app  
4 RUN pip3 install -U setuptools wheel  
5 RUN pip3 install -r requirements.txt  
6 ENTRYPOINT [ "python3" ]  
7 CMD [ "/app/app.py" ]  
8 EXPOSE 5001  
9
```

4. Then the next step is to create a job in Jenkins and build and run the application.
The steps for that are as follows:
 - a. Create a new pipeline in Jenkins and configure the GitHub URL as the project URL.
 - b. The pipeline script for the job is as:

```
node {  
    stage('Get Source') {  
        // copy source code from local file system and test  
        // for a Dockerfile to build the Docker image  
        git ('https://github.com/potatohead11153/Flask-app')  
        if (!fileExists("Dockerfile")) {  
            error('Dockerfile missing.')  
        }  
    }  
    stage('Build Docker') {  
        // build the docker image from the source code using the BUILD_ID parameter in image name  
        sh "docker build -t flask-todo ."  
    }  
    stage("run docker container"){  
        sh "docker run -p 5001:5001 flask-todo"  
    }  
}
```

On building this job, the following console output that is seen for the job is:

```
Started by user Abhinav Sharma
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/Run Flask App
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Get Source)
[Pipeline] git
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/potatohead11153/Flask-app
> git init /var/lib/jenkins/workspace/Run Flask App # timeout=10
Fetching upstream changes from https://github.com/potatohead11153/Flask-app
> git --version # timeout=10
> git --version # 'git version 2.24.0'
> git fetch --tags --force --progress -- https://github.com/potatohead11153/Flask-app +refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url https://github.com/potatohead11153/Flask-app # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision b0367af220d8b9c47893d04fe26633c0049b6d61 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f b0367af220d8b9c47893d04fe26633c0049b6d61 # timeout=10
> git branch -a -v --no-abbrev # timeout=10
> git checkout -b master b0367af220d8b9c47893d04fe26633c0049b6d61 # timeout=10
Commit message: "Update Dockerfile"
First time build. Skipping changelog.
[Pipeline] fileExists
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Build Docker)
[Pipeline] sh
+ docker build -t flask-todo .
Sending build context to Docker daemon 95.74kB
```


```
---> 1321ef2386ae
Step 6/8 : ENTRYPOINT [ "python3" ]
---> Running in 7c15303a9ad0
Removing intermediate container 7c15303a9ad0
---> da75b58943ec
Step 7/8 : CMD [ "/app/app.py" ]
---> Running in da846abba139
Removing intermediate container da846abba139
---> 0df212f4867a
Step 8/8 : EXPOSE 5001
---> Running in 13654184f63d
Removing intermediate container 13654184f63d
---> 42457fcd916b
Successfully built 42457fcd916b
Successfully tagged flask-todo:latest
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (run docker container)
[Pipeline] sh
+ docker run -p 5001:5001 flask-todo
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://0.0.0.0:5001/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 123-772-232
172.17.0.1 - - [06/Oct/2020 19:43:43] "[37mGET / HTTP/1.1[0m" 200 -
```

5. The next step is to use Jenkins and Docker Plugin, create an image of the developed web application and push that built image to [Docker Hub](#).

The entire process for creating a job will be the same as above, the only difference will be the pipeline script where we will have an extra stage for building and pushing the image to Docker Hub.





```
pipeline {
  environment {
    registry = "abhinavs03/midsem-project"
    registryCredential = 'dockerhub'
    dockerImage = ''
  }
  agent any
  stages {
    stage('Cloning Git') {
      steps {
        git 'https://github.com/potatohead11153/Flask-app'
      }
    }
    stage('Build') {
      steps {
        sh 'docker build -t flask-todo .'
      }
    }
    stage('Building image') {
      steps{
        script {
          dockerImage = docker.build registry + ":$BUILD_NUMBER"
        }
      }
    }
    stage('Deploy Image') {
      steps{
        script {
          docker.withRegistry( '', registryCredential ) {
            dockerImage.push()
          }
        }
      }
    }
  }
}
```


Parallely configure global credentials for docker hub as follows:



Jenkins

Jenkins > Credentials > System > Global credentials (unrestricted) > abhinavs03/***** (dockerhub)

 Back to Global credentials (unrestricted)
 Update
 Delete
 Move



abhinavs03/***** (dockerhub)

dockerhub

Usage

This credential has not been recorded as used anywhere.
Note: usage tracking requires the cooperation of plugins and consequently may not track every use.

After the job runs successfully, the image should be built and pushed to <https://hub.docker.com/r/abhinavs03/midsem-project>

The console output will be as follows :

```
$ docker login -u abhinavs03 -p ***** https://index.docker.io/v1/
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /var/lib/jenkins/workspace/Flask App Docker Build/tmp/0e13ade6-1158-4a40-ad78-42c216389ecd/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[Pipeline] {
[Pipeline] isUnix
[Pipeline] sh
+ docker tag abhinavs03/midsem-project:9 abhinavs03/midsem-project:9
[Pipeline] isUnix
[Pipeline] sh
+ docker push abhinavs03/midsem-project:9
The push refers to repository [docker.io/abhinavs03/midsem-project]
3bfa6ffcfb83: Preparing
db58762bd2bd: Preparing
537da3f66ec5: Preparing
5fa31f02caa8: Preparing
88e61e328a3c: Preparing
9b77965e1d3f: Preparing
50f8b07e9421: Preparing
629164d914fc: Preparing
50f8b07e9421: Waiting
629164d914fc: Waiting
9b77965e1d3f: Waiting
88e61e328a3c: Mounted from library/python
5fa31f02caa8: Mounted from library/python
537da3f66ec5: Pushed
db58762bd2bd: Pushed
3bfa6ffcfb83: Pushed
9b77965e1d3f: Mounted from library/python
629164d914fc: Mounted from library/python
50f8b07e9421: Mounted from library/python
9: digest: sha256:70935d84d86ce0cb9b618953bc95ef85e76c3bb679813b263690459469cc0e0d size: 2000
[Pipeline] }
[Pipeline] // withDockerRegistry
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

After the job is done, the docker image was pushed successfully to Docker Hub

The screenshot shows the Docker Hub interface for the repository 'abhinavs03 / midsem-project'. The top navigation bar includes the Docker Hub logo, a search bar, and links to Explore, Repositories, Organizations, Get Help, and the user profile 'abhinavs03'. The repository page has tabs for General, Tags, Builds, Timeline, Collaborators, Webhooks, and Settings. The 'General' tab is active, showing the repository name, a note that it has no description, and the last push time of '2 hours ago'. On the right, there are 'Docker commands' with a 'Public View' button and a command box containing 'docker push abhinavs03/midsem-project:tagname'. Below the repository details, there are sections for 'Tags' (showing 1 tag) and 'Recent builds' (with a link to see build results).

The image can easily be pulled now using **docker pull abhinavs03/midsem-project:9**

6. The next and final step is to deploy our dockerized Flask application to Amazon **Elastic Container Service (ECS)**. This would be automated as Infrastructure as a Code (IaC) using **Terraform**. The following are the steps to be configured in the terraform script for the successful deployment:

- a. Create a Virtual Private Cloud on in your specific profile and region.

```
provider "aws" {
  profile = "default"
  region = "us-east-1"
}

resource "aws_vpc" "main" {
  cidr_block = "132.0.0.0/16"
  tags = {
    Name = var.vpc_name
  }
}
```

- b. Create a subnet within you VPC along with routing and destination CIDR block.


```

resource "aws_subnet" "main" {
  count = 2
  vpc_id      = aws_vpc.main.id
  cidr_block = cidrsubnet(aws_vpc.main.cidr_block, 8, count.index)
  map_public_ip_on_launch=true
  tags = {
    Name = var.subnet_name
  }
}

resource "aws_internet_gateway" "internetgateway" {
  vpc_id = aws_vpc.main.id
}

resource "aws_route" "internet_access" {
  route_table_id = aws_vpc.main.main_route_table_id
  destination_cidr_block = "0.0.0.0/0"
  gateway_id = aws_internet_gateway.internetgateway.id
}

```

- c. The next step is to implement AWS security groups for incoming traffic and to also configure permissions for AWS ECS with the help of IAM policies.

```

resource "aws_security_group" "accessgroups" {
  name = "allowinbound"
  vpc_id = aws_vpc.main.id
  ingress {
    cidr_blocks=["0.0.0.0/0"]
    from_port=0
    to_port=65535
    protocol="tcp"
  }
  ingress {
    cidr_blocks=["0.0.0.0/0"]
    from_port=0
    to_port=0
    protocol=-1
  }
  tags = {
    Name = "ECS-Access"
  }
}

data "aws_iam_policy_document" "ecs_task_execution_role" {
  version = "2012-10-17"
  statement {
    sid = ""
    effect = "Allow"
    actions = ["sts:AssumeRole"]

    principals {
      type       = "Service"
      identifiers = ["ecs-tasks.amazonaws.com"]
    }
  }
}

```

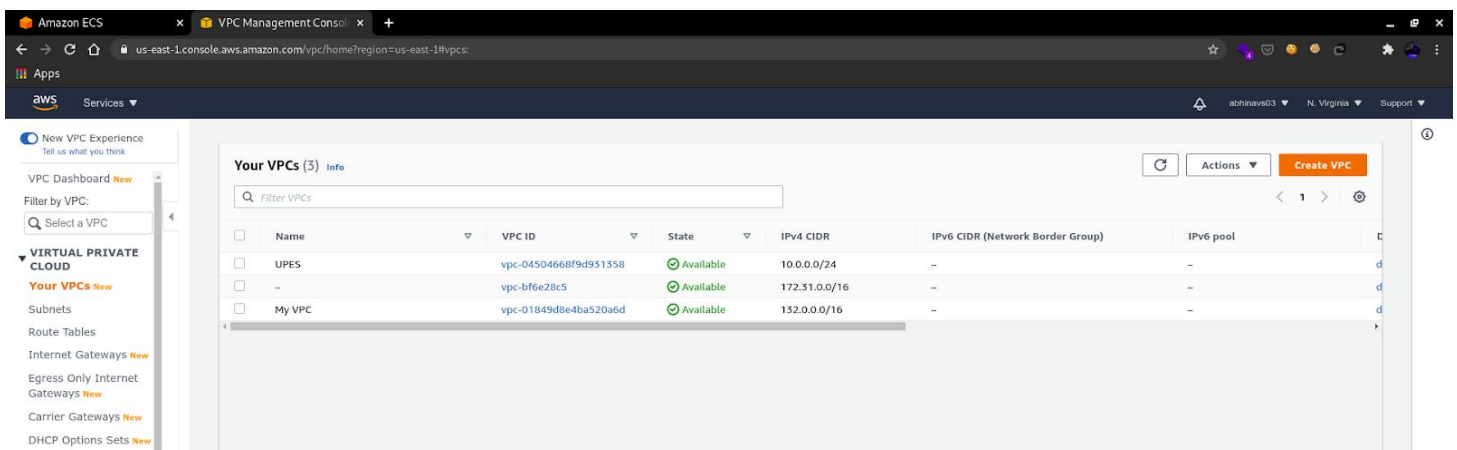
d. The other configurations are configuring memory and CPU resources.

```
resource "aws_ecs_task_definition" "flaskapp" {
  family              = "service"
  container_definitions = file("service.json")
  execution_role_arn = aws_iam_role.ecs_task_execution_role.arn
  network_mode        = "awsvpc"
  requires_compatibilities = ["FARGATE"]
  memory              = "1024"
  cpu                  = "512"
}
```

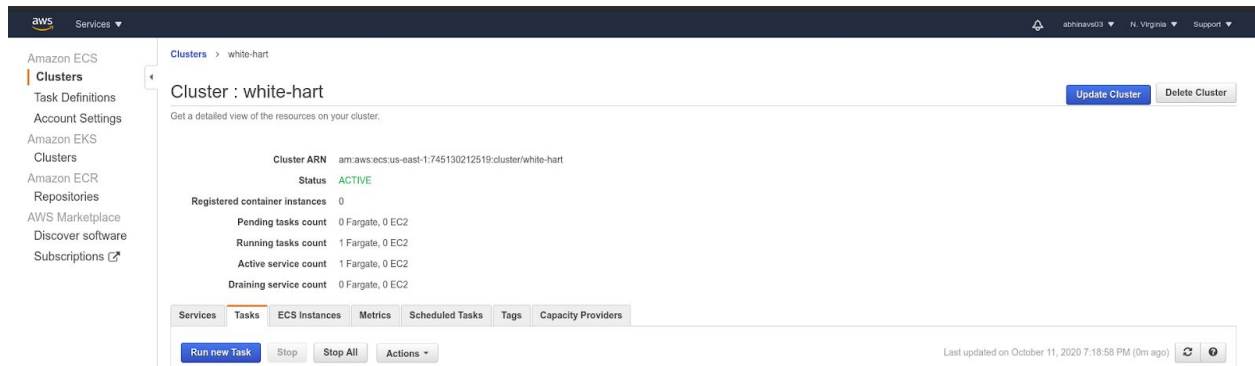
The next step is to run the following commands for the Terraform CLI:

```
$ terraform init
$ terraform plan
$ terraform apply -auto-approve
```

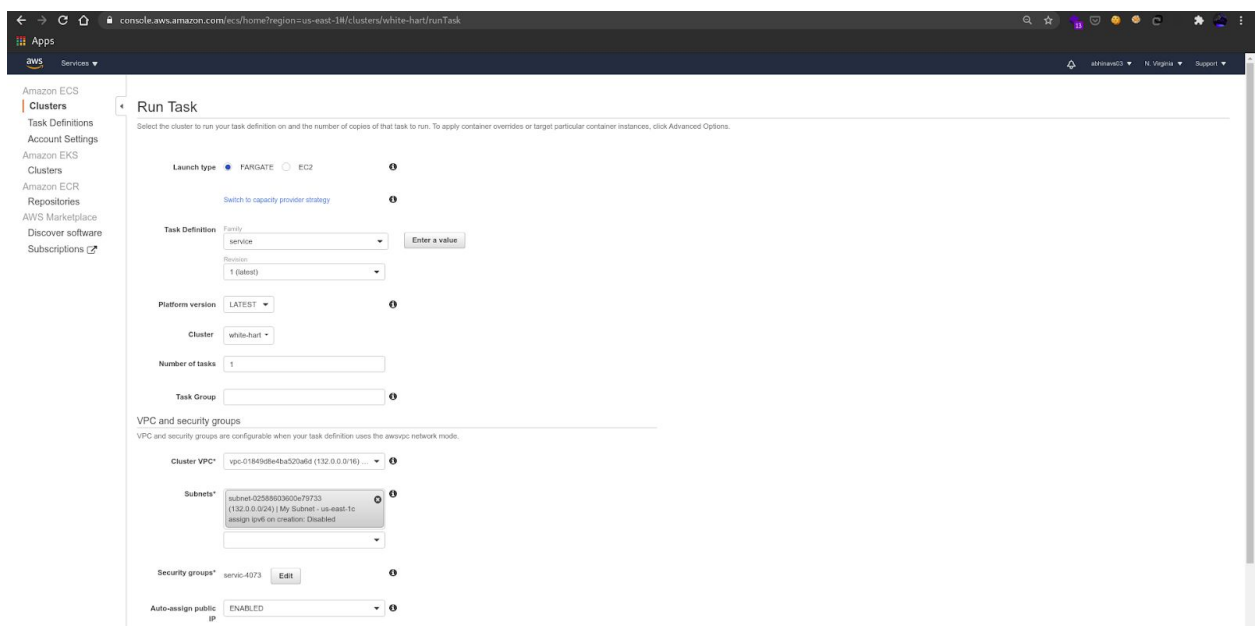
This will set up everything that we've specified as it is verified in the AWS Web interface.



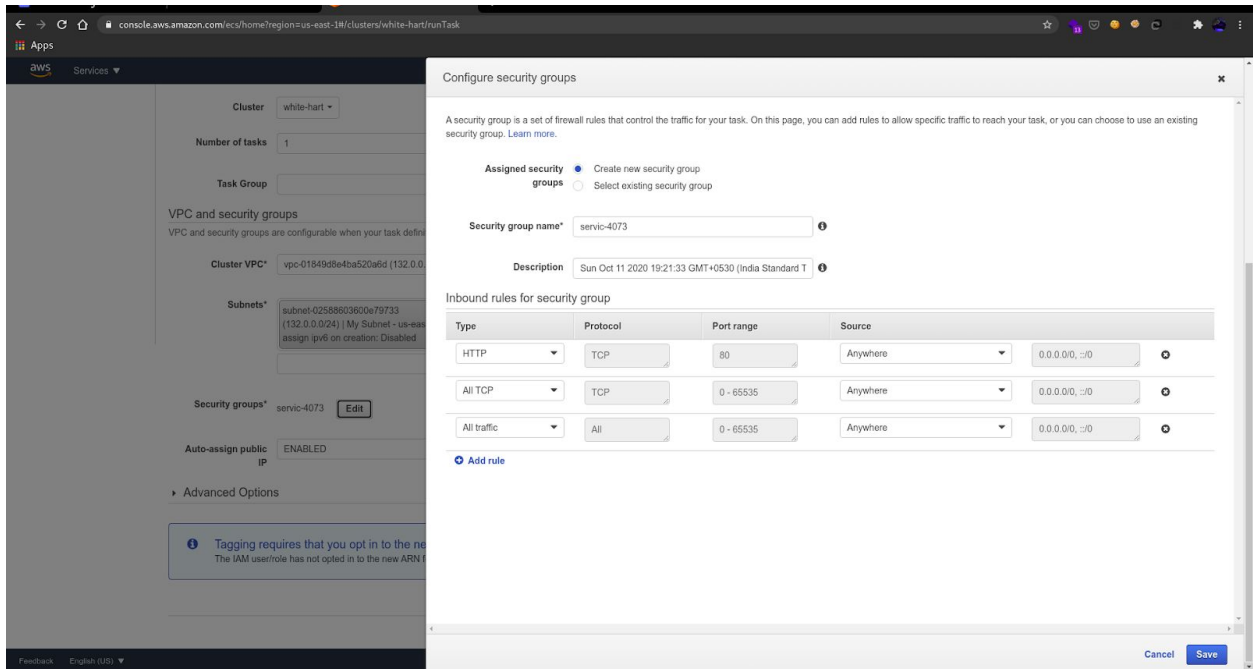
Now, we will create a new task and run it. This will fetch the Docker image and run it inside our VPC. The steps for that are as follows:



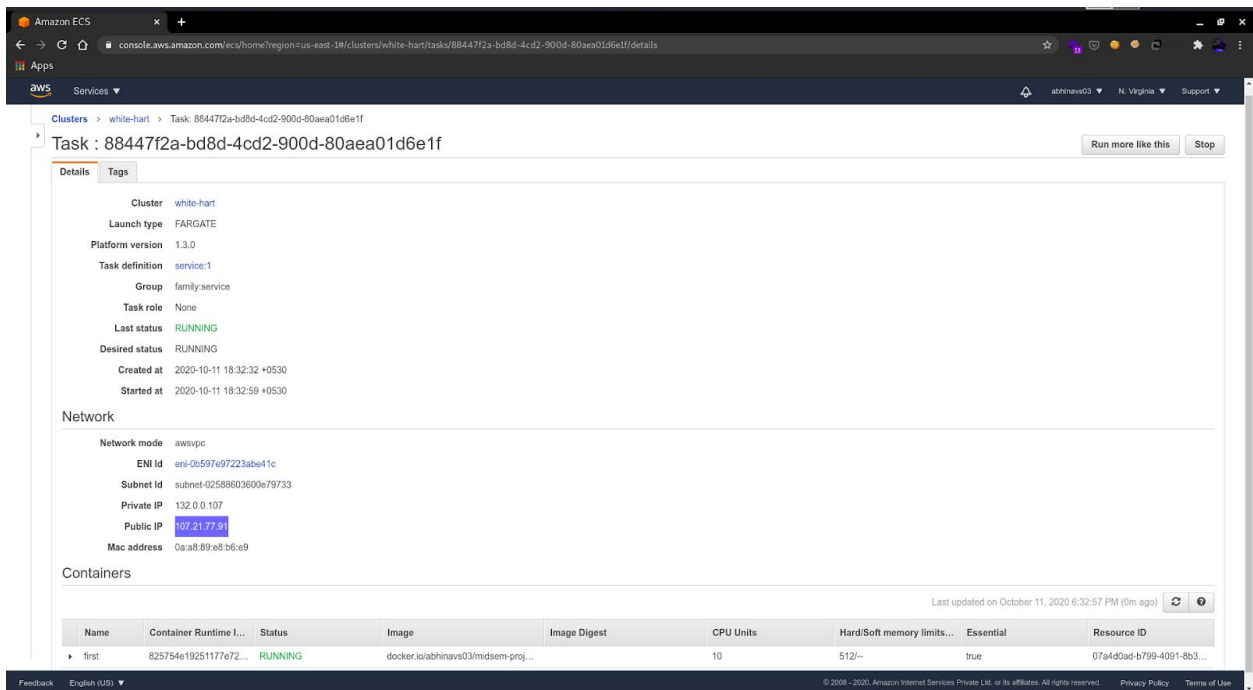
a. Run a new task



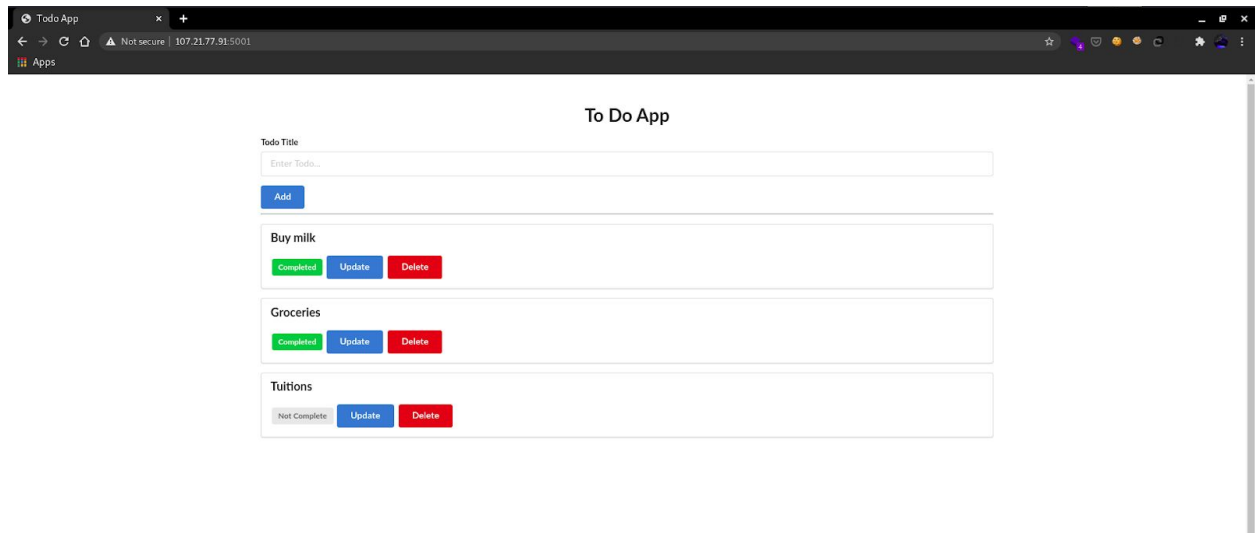
b. Configure VPC, Subnets and Security groups.



c. Once the task comes in running state it will be reachable via public IP.



Now the only task is to visit the public IP address allotted to us on the corresponding exposed port 5001.



This will run our application on the assigned public IP.