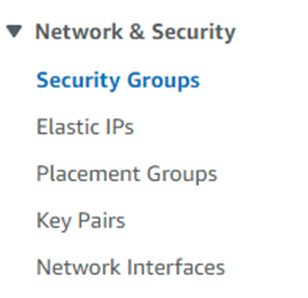
**Deployment Documentation**

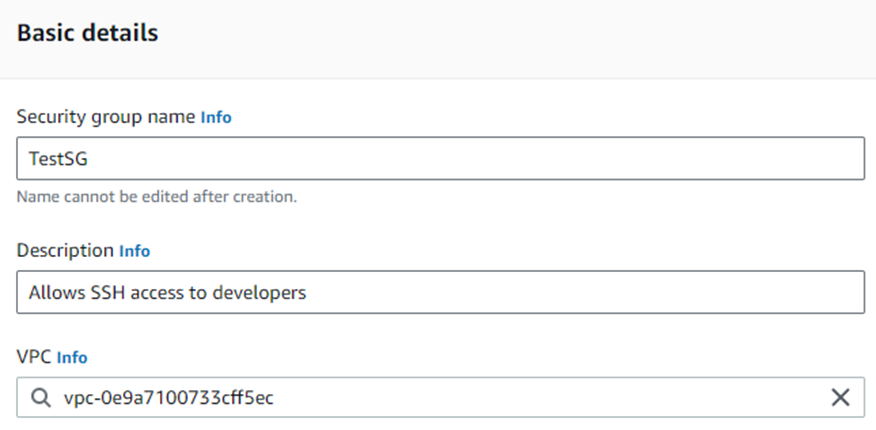
This document provides a basic setup for AWS EC2 instance. Additionally, it demonstrates the steps of connecting to the instance using PuTTY and deploying an HTML website using Docker.

**Set Up AWS EC2 Instance:**

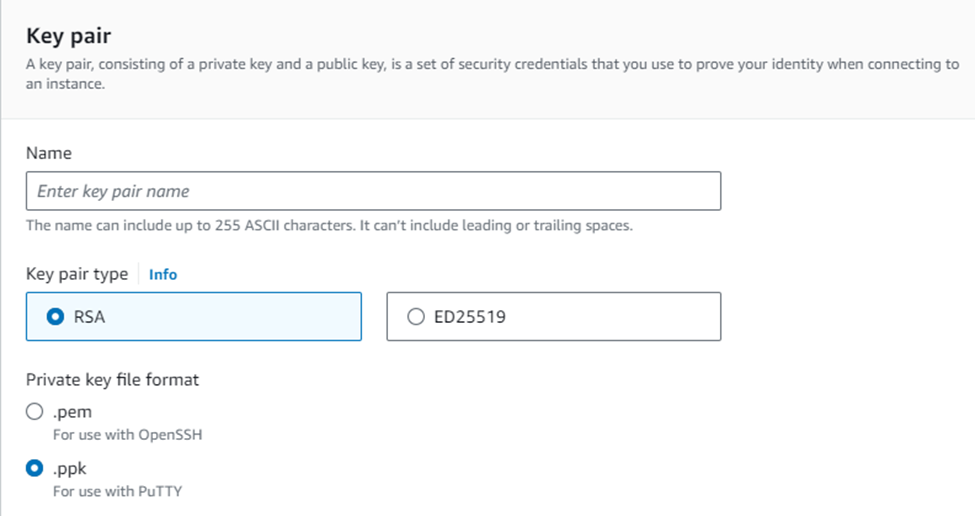
* Create a security group (Under Network & Security, click on Security Groups > Create security group).



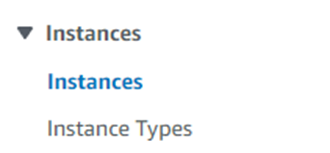
* Name it TestSG and add “Allows SSH access to developers” in the description box.



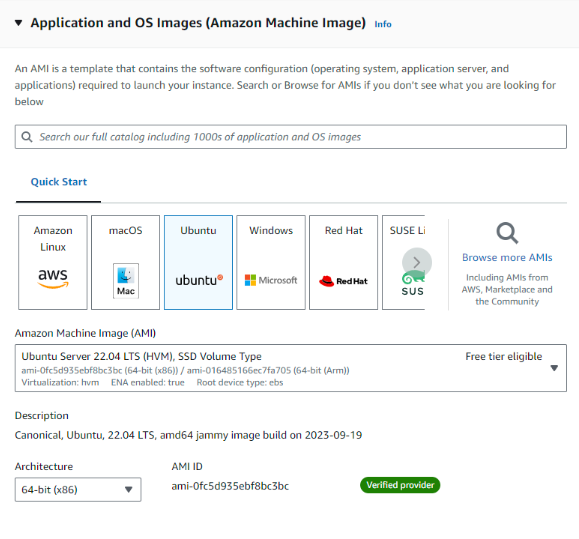
* That’s it for Security Group. Now we move to creating a keypair
* Create a keypair (Under Network & Security, click on Key Pairs > Create key pair).
* Name it TestKeyPair, select RSA type, and choose .ppk key file format. Then click Create.
* Check your browser’s Download folder (or File Explorer’s Download folder) because you will have already downloaded your TestKeyPair.ppk when you click Create (Put that file under a safe folder like TestServerKeyPair)



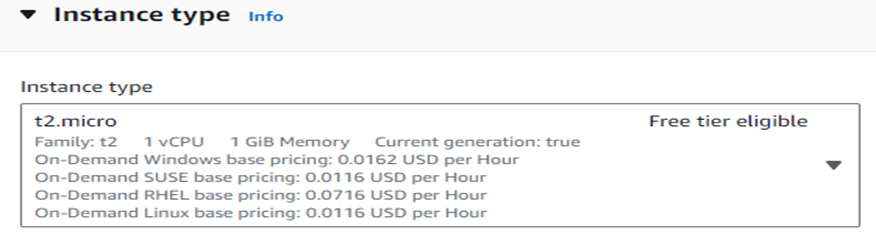
* That’s it for keypair creation. Now, we will create an instance
* Create an Instance (Under Instances, click on Instances and select Launch instances).



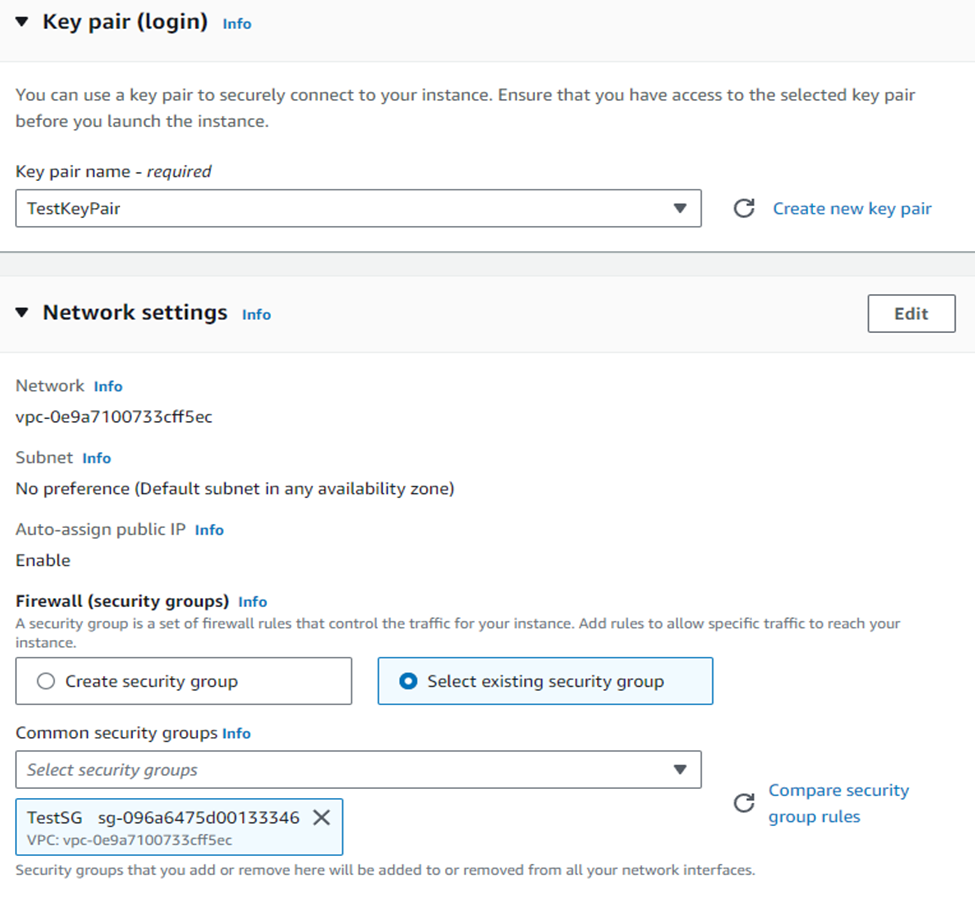
* Name it TestWebsiteServer
* Choose Ubuntu and select Ubuntu Server 22.04 LTS



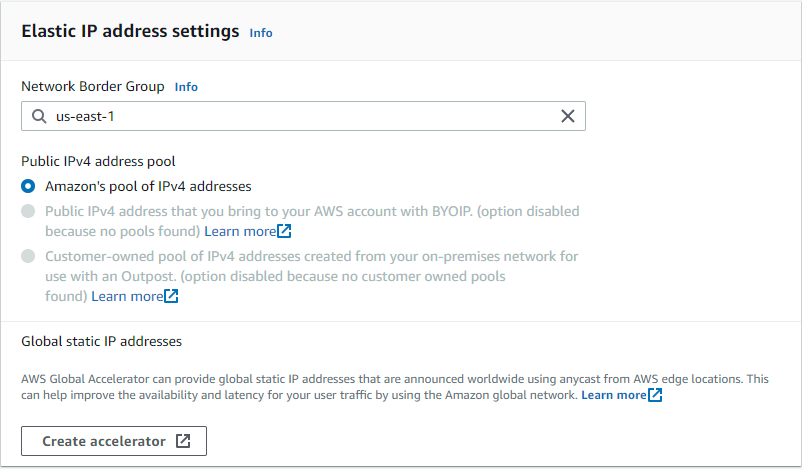
* Select t2.micro for the instance type.



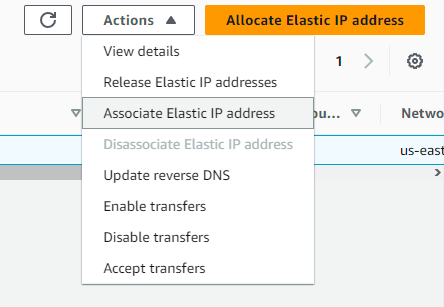
* Select TestKeyPair for Key pair (login) and select existing security group (choose TestSG for the “Common security groups” box). Then click Launch instance.



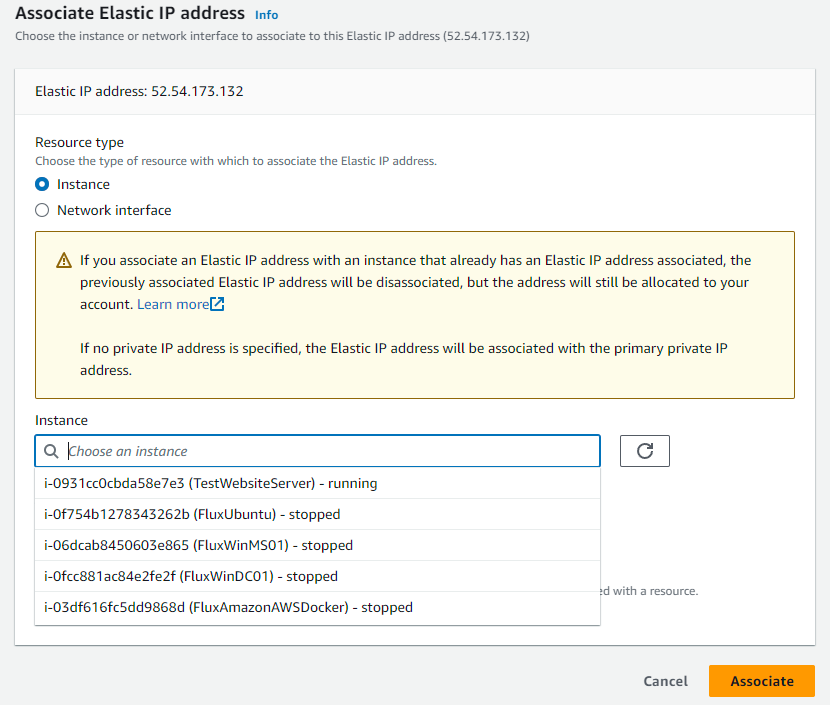
* Now, we are going to create an Elastic IP (permanent IP address) for our instance.
* Under Network & Security, go to Elastic IPs. Then click “Allocate Elastic IP address”.



* Now Click “Allocate”.
* Now select your Elastic IP address and choose Actions. Then click “Associate Elastic IP address”



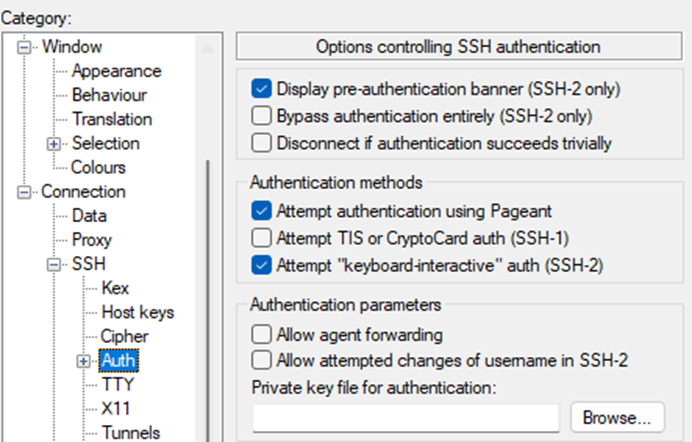
* Then, choose the instance that you just created



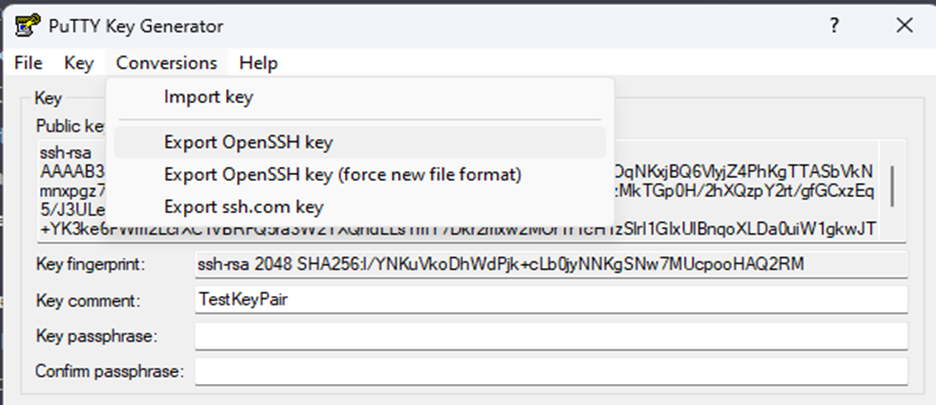
* Click Associate.
* Now your instance should have a public static IP address. (For our case, it is 52.54.173.132)

**Connect to the EC2 Instance using Putty:**

* Open PuTTY and enter your instance’s public Elastic IP address (52.54.173.132) on the “Host Name” field
* On PuTTY, go to “Connection” category, go to "SSH” and click on “Auth”. Select Browse under “Private key file for authentication: ” and select your TestKeyPair.ppk file.



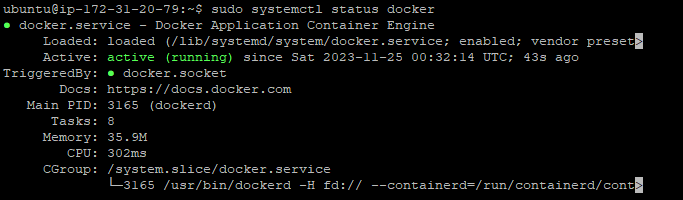
* Go back to “Session” category and give your configuration in the “Save Sessions” field. Then click Save.
* Before logging in to your EC2 Instance, we need to convert our TestKeyPair.ppk file to .pem file.
* Download PuTTYgen (<https://www.puttygen.com/>) and open it.
* Click on file and select your TestKeyPair.ppk
* Then click the “Conversions” menu at the top and select “Export OpenSSH Key” (It may give you a passphrase warning, just click Yes)
* Choose a location (We can use our TestServerKeyPair folder) to save the key and provide a file name with the .pem extension (For example: TestKeyPair.pem)
* Click Save.



* Now go back to PuTTY and click Open to connect to your EC2 instance.
* When the PuTTY terminal window opens, you should be prompted to log in. The default username for Ubuntu Image is usually **ubuntu**. Enter the username and press Enter.

**Install Docker on Ubuntu EC2 Instance:**

* Once you're connected to your EC2 instance via PuTTY, you need to update the package repository. Run the following command:  **sudo apt update**
* Install Docker dependencies: **sudo apt install apt-transport-https ca-certificates curl software-properties-common**
* Add GTG key: **curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -**
* Add docker to APT resources**: sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"**
* Since we are using the Ubuntu, we can install Docker using the following command: **sudo apt install docker-ce**
* To verify that Docker is installed and running, you can run the following command to check the Docker version: **sudo systemctl status docker**



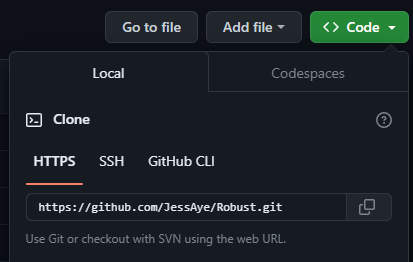
(Ctrl-C to exit this screen)

**Install Node.js and npm:**

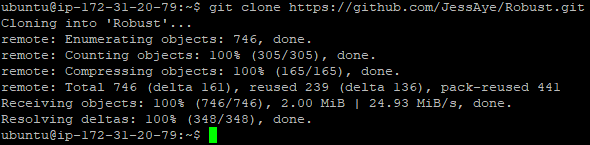
* We are installing the node.js and npm on the instance.
* sudo apt install nodejs
* sudo apt install npm

**Upload Project Folder from Github to EC2 Instance:**

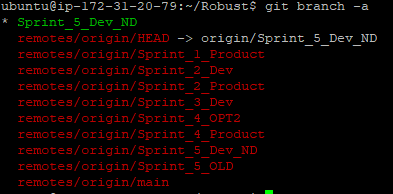
* On EC2 instance using PuTTY, use **cd ~** to go to the home directory.
* Now, you can use the **git clone** command to clone your GitHub repository.



* On the instance, type **git clone https://github.com/JessAye/Robust.git**



* Type **cd** **Robust** (You are switching to the cloned repo directory; our repo name is Robust. Yours may be different)
* Type **git branch -a**



* Type **git checkout Sprint\_5\_Dev\_ND** (This command switches your current directory to the Sprint\_5\_Dev\_ND branch, allowing you to access its contents)



* Since Sprint\_5\_Dev\_ND branch is the default branch of our repo, the command displayed a message saying “Already on Sprint\_5\_Dev\_ND”

**Create Dockerfile to deploy your website on EC2 Instance:**

* Type **ls** when you are in the desired sprint branch

Inserting image...

* Type **cd Robust**, then create a Dockerfile by using nano:



* Copy and paste the following content onto your Dockerfile:

**# Use an official NGINX runtime as a parent image**

**FROM nginx:latest**

**# Set the working directory to /usr/share/nginx/html**

**WORKDIR /usr/share/nginx/html**

**# Copy the content of your Games folder into the image**

**COPY ./FrontEnd/ .**

**# Replace the default NGINX configuration with your custom configuration**

**COPY nginx.conf /etc/nginx/nginx.conf**

**# Expose port 80 to allow external access**

**EXPOSE 80**

* Press Ctrl+X to exit and save the Dockerfile.
* In FrontEnd folder, it contains a file named LandingPage.html, and we want to use that as our default page.
* To achieve this, we will need to replace the default NGINX configuration with a custom configuration file in the “**website**” folder.
* To create a custom NGINX configuration file, we will use nano to make a nginx.conf file.
* Copy and paste the following content onto your nginx.conf file:

**events {**

**worker\_connections 1024;**

**}**

**http {**

**include /etc/nginx/mime.types;**

**sendfile on;**

**server {**

**listen 80;**

**server\_name \_;**

**location / {**

**root /usr/share/nginx/html;**

**index Games/LandingPage.html;**

**}**

**# Additional NGINX configuration can be added here if needed.**

**}**

**}**

* Press Ctrl+x to exit and save the nginx.conf
* The location / block sets the root directory to serve content from /usr/share/nginx/html and specifies LandingPage.html as the default index file.

**Build the Docker Image of the FrontEnd and run the Docker Container:**

* In the “**FrontEnd**” folder on the EC2 instance, build a Docker image using the Dockerfile and the custom NGINX configuration with the following command:
  + **sudo docker build -t my-website-image .**
  + Make sure to include the period at the end of the command.
  + Replace **my-website-image** with the name you want to give to your Docker image.
* Run a Docker container using the image you just built:
  + **sudo docker run -d -p 80:80 --name my-website-container my-website-image**
  + Replace **my-website-container** with the name you want.
* To check the status of the Docker container, run the command:
  + **sudo** **docker ps -a**
  + Here is an example image:
* The website is now deployed and set up to use LandingPage.html as the home page.

**Build the Docker Image of the OTP Microservice and run the Docker Container:**

* On the instance, type **cd Robust**
* Type **cd** OTPMicroService
* Type **nano Dockerfile** and paste the following content into the file:

# Use an official Node.js runtime as a parent image

FROM node:14

# Set the working directory in the container

WORKDIR /app

# Copy package.json and package-lock.json to install dependencies

COPY package\*.json ./

# Install application dependencies

RUN npm install

# Copy the rest of the application code to the container

COPY . .

# Expose the port on which your Node.js application will run (e.g., 80)

EXPOSE 80

# Command to start your Node.js application

CMD [ "node", "app.js" ]

* Build Docker image:
  + **sudo docker build -t my-otp-image .**
  + Make sure to include the period at the end of the command.
  + Replace **my-otp-image** with the name you want to give to your Docker image.
* Run a Docker container using the image you just built:
  + **sudo docker run -d -p 3000:80 --name my-otp-container my-otp-image**
  + Replace **my-otp-container** with the name you want.
* The microservice is now deployed.