Key Pair

Make sure these are set for **~/.ssh/config**

```
Host *
UseKeychain yes
```

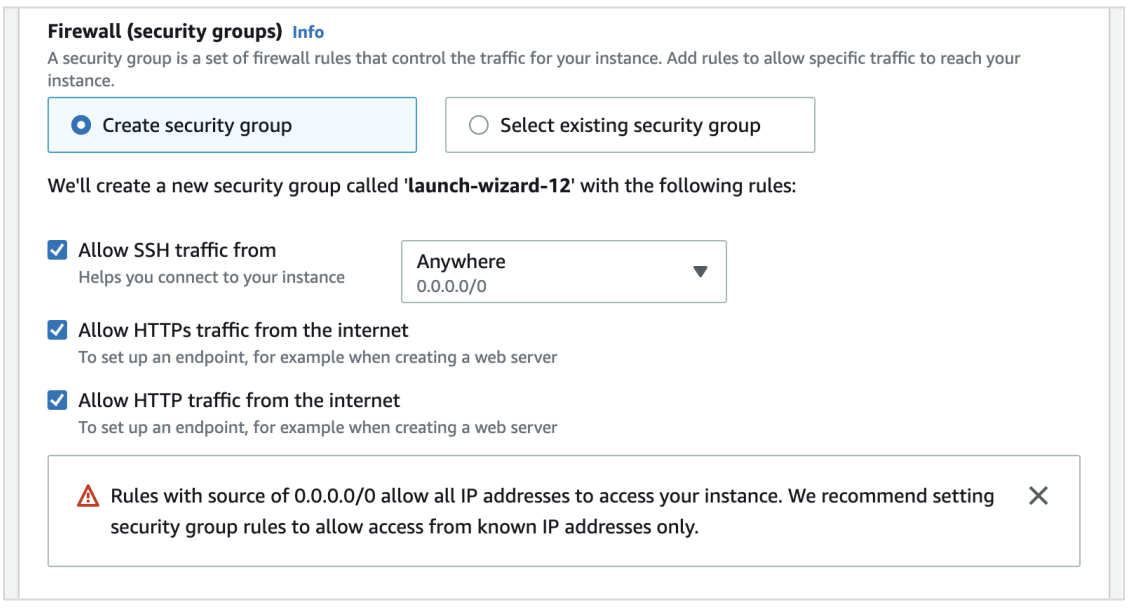
Create new keypair via AWS UI

```
# update permissions to be read only
chmod 400 ~/Downloads/FILE_NAME.pem

# add the key to your agent
ssh-add -K ~/Downloads/FILE_NAME.pem
```

Create an EC2 instance

- t2.micro
- Ubuntu 20.04 64-bit (x86)




```
ssh ubuntu@IP_ADDRESS
```

https://docs.docker.com/engine/install/ubuntu/

```
# uninstall any existing versions of Docker (it's OK if apt reports that none of these packages are installed)
sudo apt remove docker docker-engine docker.io containerd runc
```

Set up the repository

Before you install Docker Engine for the first time on a new host machine, you need to set up the Docker repository. Afterward, you can install and update Docker from the repository.

```
# update the package cache and upgrade packages
sudo apt -y update && sudo apt -y upgrade

# install packages to allow apt to use a repository over HTTPS
sudo apt -y install ca-certificates curl gnupg lsb-release

# add Dockers official GPG key (allows for the secure transmission of data)
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

# use the following command to set up the repository
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Install Docker Engine

```
# update the apt package index, and install the latest version of Docker Engine, containerd, and Docker Compose
sudo apt -y update
sudo apt -y install docker-ce docker-ce-cli containerd.io docker-compose-plugin

# verify that Docker Engine is installed correctly by running the hello-world image
sudo service docker start
sudo docker run hello-world
```

Docker Engine is installed and running. The **docker** group is created but no users are added to it.

```
# add your user to docker group
sudo usermod -aG docker $USER
exit
```

Pull the images from Docker Hub

```
ssh ubuntu@IP_ADDRESS
```

```
# pull the images
docker pull thenewboston/core
docker pull thenewboston/core-reverse-proxy
```

Associate Elastic IP with EC2 instance

- An Elastic IP is a reserved public IP address that you can assign to any EC2 instance
- Why not just use the instance's public IP address?
    - a Public IP address associated with an instance is not static and is lost when the instance is stopped
    - an Elastic IP address is a static public address
- —
- Navigate to EC2 > Network and Security > Elastic IPs
- click the “Allocate Elastic IP address” button
    - click the Allocate button
- give that Elastic IP a more memorable name
- select it and choose click on “Associate Elastic IP address”
    - associate it with your core instance
    - check “Allow this Elastic IP address to be reassociated”
    - click on “Associate”

Test by SSHing into the instance using the new IP.

Update your domain's DNS records to point to your elastic IP

- Create type A DNS record for your domain, which points to your elastic IP
    - A record stands for "address"
        - A records only supports IPV4 addresses
    - *Nameserver (NS) record specifies the authoritative DNS server for a domain*
    - *SOA record stands for "start of authority"*
        - *stores admin information about a domain including the email address of the admin and when the domain was last updated*

https://dnschecker.org/

Test by SSHing into the instance using the domain name

Run the Deployment Script

https://github.com/thenewboston-developers/Core/blob/master/DEPLOY.rst

- sets your environment variables
- sets up your SSL certificate to enable HTTPS
- start all your services

```
# run the deployment script
bash <(wget -qO- https://raw.githubusercontent.com/thenewboston-developers/Core/master/scripts/deploy.sh)

# create a superuser
docker compose exec -it core poetry run python -m core.manage createsuperuser
```

https://example.com/admin/