

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
```

Prepare S3 Bucket for Image Uploads

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>
2. In the navigation pane, choose **Users**, and then choose **Add user**.
3. Type the user name for the new user and click **Next**.
4. On the **Set permissions** page, you can attach existing policies directly to give the necessary permissions. For the S3 related permissions, you can select the **AmazonS3FullAccess** policy and click **Next**.
5. **Create user**
6. Click the user > Security credentials > Application running on an AWS compute service
 - a. Description tag value: access-key
 - b. Create access key

Create a new S3 bucket

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>
2. Choose **Create bucket**.
3. In the **Bucket name** box, type a unique DNS-compliant name for your new bucket. Remember that S3 bucket names must be globally unique.
4. In the **Region** box, choose the AWS Region where you want the bucket to reside.
5. In the Bucket settings for **Block Public Access**, make sure you **uncheck** the boxes to allow public access to the bucket.
6. Choose **Create bucket**.

Configure CORS in your S3 bucket

1. Go to your bucket and choose the **Permissions** tab.
2. Scroll down to **Cross-origin resource sharing (CORS)** and choose **Edit**.
3. Paste the following configuration and **Save**.

```
[
  {
    "AllowedHeaders": ["*"],
    "AllowedMethods": ["GET"],
    "AllowedOrigins": ["*"],
    "ExposeHeaders": []
  }
]
```

Create an EC2 Instance

- t2.micro
- Ubuntu Server 20.04 LTS (HVM), SSD Volume Type | 64-bit (x86)
- Create a new security group or choose an existing one, ensuring that it allows inbound traffic on ports 22 (SSH), 80 (HTTP), and 8000 (Django server). Click "Review and Launch."

Firewall (security groups) Info

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

Create security group

Select existing security group

We'll create a new security group called 'launch-wizard-12' with the following rules:

☒ Allow SSH traffic from

Helps you connect to your instance

Anywhere0.0.0.0/0

☒ Allow HTTPs traffic from the internet

To set up an endpoint, for example when creating a web server

☒ Allow HTTP traffic from the internet

To set up an endpoint, for example when creating a web server

Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

ssh ubuntu@IP_ADDRESS

Install Dependencies

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

# install Docker, git, NGINX, and certbot
sudo apt install -y apt-transport-https ca-certificates curl software-properties-common git nginx certbot python3-certbot-nginx

# download the Docker GPG key (used to verify the authenticity and integrity of Docker packages during installation)
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

# ONE COMMAND | adds the Docker repository entry to the system's software sources list
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" |
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# install Docker related tools
sudo apt update -y
sudo apt install -y docker-ce docker-ce-cli containerd.io

# install Docker Compose
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

# make it executable so we can run it directly from the command line
sudo chmod +x /usr/local/bin/docker-compose

# add your user to the Docker group
sudo usermod -aG docker $USER

# exit then ssh back in to pick up new permissions
exit
ssh ubuntu@IP_ADDRESS
```

Run the Application

```
# clone your GitHub repository and deploy the application
git clone https://github.com/thenewboston-developers/Cooking-Core.git
cd Cooking-Core/
test -f .env || touch .env

# create your local settings file
mkdir -p local
sudo nano ./local/settings.prod.py

DEBUG = False
SECRET_KEY = 'django-insecure-!=%dlpgnc+4mk7+2h!l4&x^3)r+q4+=nz@y3k&e)8gg^gtgqkn'
CSRF_TRUSTED_ORIGINS = ['https://penguinchess.net', 'https://www.penguinchess.net']
AWS_ACCESS_KEY_ID = 'your-aws-access-key-id'
AWS_SECRET_ACCESS_KEY = 'your-aws-secret-access-key'
AWS_STORAGE_BUCKET_NAME = 'penguinchess-backend'

# start the containers
docker-compose build
docker-compose up -d
```

Create Superuser and Config Object

```
# get container ID of app
docker ps

# shell into the container
docker exec -it CONTAINER_ID /bin/bash

make superuser
...

# enter Django shell
make shell

from cooking_core.config.models import Config
Config.objects.create(owner=None, transaction_fee=1)

# exit Django shell
exit()

# exit Docker container
exit
```

Setup NGINX

Create a new NGINX configuration file (temporary config needed for certbot so the certificate authority can communicate with our server)

```
sudo rm /etc/nginx/sites-available/default
sudo nano /etc/nginx/sites-available/default
```

```
server {
    listen 80;
    server_name localhost;

    location / {
        proxy_pass http://localhost:8000;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

```
}
```

Save the file and exit the editor

Test the NGINX configuration

```
sudo nginx -t
```

If the test is successful, restart NGINX to load in the new configuration

```
sudo systemctl restart nginx
```

Allow NGINX through firewall

```
sudo ufw allow 'Nginx Full'
```

Associate Elastic IP with EC2 instance

- 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

<https://dnschecker.org/>

Test by SSHing into the instance using the domain name

Install Certbot and Request an SSL certificate

follow the prompts to obtain an SSL certificate for your domain name. Certbot will automatically configure NGINX to use the SSL certificate.

```
sudo certbot --nginx
```

...

make sure the certificate auto-renewal is set up by running the following command

```
sudo certbot renew --dry-run
```

create a new NGINX configuration file

```
sudo rm /etc/nginx/sites-available/default
sudo nano /etc/nginx/sites-available/default
```

```
server {
    listen 80 default_server;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl default_server;
    ssl_certificate /etc/letsencrypt/live/penguinchess.net/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/penguinchess.net/privkey.pem;

    ssl_session_cache shared:le_nginx_SSL:10m;
    ssl_session_timeout 1440m;
    ssl_session_tickets off;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers off;

    client_max_body_size 20M;

    location / {
        proxy_pass http://localhost:8000/;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        proxy_redirect off;
        proxy_buffering off;

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Host $http_host;
    }
}
```

Save the file and exit the editor

Test the NGINX configuration

```
sudo nginx -t
```

If the test is successful, restart NGINX

```
sudo systemctl restart nginx
```

Updating

```
# stop the containers
docker-compose down

# fetch the updated source code
git pull

# rebuild and restart the containers
docker-compose build
docker-compose up -d --force-recreate
```

Troubleshooting

```
docker exec -it CONTAINER_ID /bin/bash
docker-compose up -d --force-recreate
sudo systemctl restart nginx
```