https://youtube.com/@codewithmuh?sub_confirmation=1

Deploy Django Application on EC2 with PostgreSQL, S3, Domain, and SSL Setup

Muhammad Rashid
Entrepreneur | Software Developer | AWS DevOps Guru | Python, Django, Backend…
30 articles + Suivre

31 janvier 2024

Ouvrir le lecteur immersif

Explore step-by-step instructions on deploying a Django application on EC2 with PostgreSQL, S3 integration, domain setup, and SSL configuration in this comprehensive video tutorial.

You can Watch it on YouTube as well.

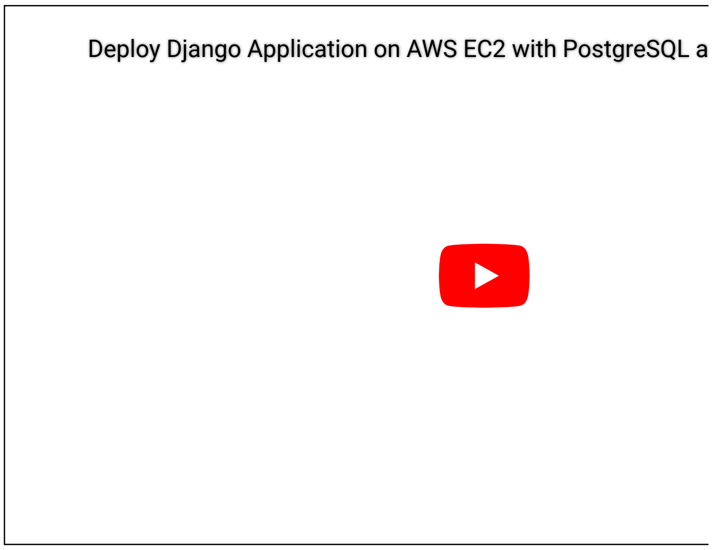


Table of Contents:

Section 01: PostgreSQL Setup

https://youtube.com/@codewithmuh?sub_confirmation=1

Amazon Relational Database Service (Amazon RDS) is a fully managed relational database service provided by Amazon Web Services (AWS). It makes it easy to set up, operate, and scale a relational database in the cloud.

To configure an Amazon Web Services (AWS) Relational Database Service (RDS) PostgreSQL database with Django, you need to follow these steps:

a. Launch an RDS instance

b. Configure Django to use the RDS instance

c. Migrate your Django models

d. Test your connection

Launch an RDS instance: Log in to your AWS account and launch an RDS instance for PostgreSQL. Select the appropriate options for your database, such as the engine version, instance type, storage, and security group.

1. **Configure Django to use the RDS instance**: In your Django settings.py file, update the DATABASES setting with the appropriate information for your RDS instance. This should include the engine, name, user, password, host, and port.

```
#Replace name, Password, Host with your RDS Database
name, user, password, host.



DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'mydatabase',
        'USER': 'mydatabaseuser',
        'PASSWORD': 'mypassword',
        'HOST': 'myrdshost.rds.amazonaws.com',
        'PORT': '5432',
    }
}
```

Where:

- database_name is the name of your database instance on Amazon RDS.
- database_user and database_password are the credentials you created when you set up the database instance.
- database_host is the host name for your database instance, which you can find in the Amazon RDS console.
- database_port is the port number for your database instance, which is typically 5432 for PostgreSQL.

3. **Migrate your Django models**: After you've updated your settings.py file, you can run the migrate command to create the necessary tables in the RDS instance.and then start using the database in your application.

```
python manage.py migrate
```

4. **Test your connection:** You can now test your connection to the RDS instance by running a query in the Django shell or by visiting a page that accesses the database.

That's it! You should now be able to use your AWS RDS PostgreSQL instance with your Django application.

Section 02: AWS S3 Bucket Setup

https://youtube.com/@codewithmuh?sub_confirmation=1

Amazon S3 (Simple Storage Service) is a highly scalable, object-based cloud storage service provided by Amazon Web Services (AWS). It allows users to store and retrieve any amount of data from anywhere on the web.

To configure Amazon S3 to store your Django project's static and media files, you can follow these steps:

1. **Create an S3 bucket**: Log in to the Amazon S3 management console, click on "Create Bucket," and follow the steps to create a new bucket.
2. **Set up an IAM user**: You need to create an IAM (Identity and Access Management) user that has permission to access your S3 bucket. Go to the IAM management console, click on "Users," and then "Add user." Give the user a name and select the "Programmatic access" option. Attach the "AmazonS3FullAccess" policy to the user, and then create the user.
3. **Store your AWS credentials**: You need to store your AWS access key ID and secret access key somewhere safe, so that your Django application can access your

S3 bucket. You can store these credentials in the settings.py file of your Django project.

4. Install the boto3 library: You'll need to install the boto3 library in order to interact with S3 from your Django project. You can install it using pip:

```
pip install boto3
```

5. **Configure your Django settings**: You'll need to update your Django settings to use S3 for storage. Add the following to your settings.py file:

```
AWS_ACCESS_KEY_ID = 'AWS_ACCESS_KEY_ID '
AWS_SECRET_ACCESS_KEY = 'AWS_SECRET_ACCESS_KEY'
AWS_STORAGE_BUCKET_NAME = 'AWS_STORAGE_BUCKET_NAME'
AWS_S3_SIGNATURE_NAME = 's3v4',
AWS_S3_REGION_NAME = 'AWS_S3_REGION_NAME'
AWS_S3_FILE_OVERWRITE = False
AWS_DEFAULT_ACL =  None
AWS_S3_VERITY = True
DEFAULT_FILE_STORAGE =
'storages.backends.s3boto3.S3Boto3Storage'
```

Finally, you will also need to add the following line at the top of your settings.py file:

6. Collect your static files: You'll need to collect your static files into one place so that they can be uploaded to S3. You can do this using the python manage.py collectstatic command.

7. Upload your static files to S3: You can upload your static files to S3 using the python manage.py collectstatic command. This command will upload your static files to your S3 bucket.

With these steps, you should be able to configure S3 to store your Django project's static and media files.

Section 03: Deploying Project on AWS EC2

Django is a popular web framework for building web applications in Python. It is fast, secure, and scalable, making it a great choice for businesses of all sizes. If you are looking to deploy your Django application on the cloud, Amazon Web Services (AWS) is a great option. AWS offers a range of services that can help you deploy and manage your application with ease.

In this article, we will walk you through the steps to deploy a Django application on AWS using the EC2 (Elastic Compute Cloud) service using Nginx with gunicorn.

This development server is not scalable and is not suited for production. Hence we need to configure gunicorn to get better scalability and Nginx can be used as a reverse proxy and a web server to serve static files. Let’s get started.

Step 1: Create an EC2 Instance

The first step in deploying a Django application on AWS is to create an EC2 instance. An EC2 instance is a virtual server that runs in the AWS cloud. To create an EC2 instance, you will need to log in to your AWS account and go to the EC2 dashboard.

Once you are in the EC2 dashboard, click on the “Launch Instance” button. From there, you will be prompted to select an Amazon Machine Image (AMI). An AMI is a pre-configured virtual machine image that includes the operating system, application server, and other software needed to run your application.

For this tutorial, we will use the Amazon Ubuntu Machine. After selecting the AMI, select the instance type that best suits your needs. For small to medium-sized applications, a t2.micro instance is a good choice.

Step 2: Configure Security Group

After creating the EC2 instance, you will need to configure the security group. The security group controls the incoming and outgoing traffic to and from your EC2 instance. To configure the security group, go to the “Security Groups” section of the EC2 dashboard and create a new security group.

Add the following rules to the security group to allow incoming traffic:

- SSH (port 22) for remote access to your instance
- HTTP (port 80) for web traffic

- HTTPS (port 443) for secure web traffic

Step 3: Connect to your EC2 Instance

Now that you have created and configured your EC2 instance, it's time to connect to it. To connect to your instance, you will need to use an SSH client. If you are using a Mac or Linux machine, you can use the terminal. If you are using Windows, you can use PuTTY.

Step 4: Installing python and Nginx

Let's update the server's package index using the command below:

```
sudo apt update
sudo apt install python3-pip python3-dev nginx
```

This will install python, pip, and Nginx server

Step 5: Creating a python virtual environment

```
sudo pip3 install virtualenv
sudo apt install python3-virtualenv
```

This will install a virtual environment package in python. Let's create a project directory to host our Django application and create a virtual environment inside that directory.

```
git clone
https://github.com/rashiddaha/blogprojectdrf.git
cd ~/blogprojectdrf

then
virtualenv env
```

A virtual environment named env will be created. Let's activate this virtual environment:

```
source env/bin/activate
pip install -r requirements.txt
```

Step 6: Installing Django and gunicorn

```
pip install django gunicorn
```

This installs Django and gunicorn in our virtual environment

Step 7: Setting up our Django project

Add your IP address or domain to the ALLOWED_HOSTS variable in settings.py.

If you have any migrations to run, perform the action:

```
python manage.py makemigrations

python manage.py migrate

python manage.py collectstatic
```

Import Thing about static files, You must make sure to add few lines in your seeting.py file.

1. add this line "**whitenoise.runserver_nostatic**", into your **Installed_apps** of setting file.
2. add 'whitenoise.middleware.WhiteNoiseMiddleware', into MiddleWare of your setting File.
3. Also, add these lines at the bottom of the *blog/urls. py file.*

```
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL,
document_root = settings.MEDIA_ROOT)
    urlpatterns += static(settings.STATIC_URL,
document_root = settings.STATIC_URL)
```

4. Also, add these imports lines at the top of the *blog/urls. py file.*

```
from django.conf import settings # new
from  django.conf.urls.static import static #new
```

5. Run this command

```
$ pip install whitenoise
```

Step 8: Configuring gunicorn

Deactivate the virtual environment by executing the command below:

```
deactivate
```

Let's create a system socket file for gunicorn now:

```
sudo vim /etc/systemd/system/gunicorn.socket
```

Paste the contents below and save the file

```
[Unit]
Description=gunicorn socket

[Socket]
ListenStream=/run/gunicorn.sock

[Install]
WantedBy=sockets.target
```

Next, we will create a service file for gunicorn

```
sudo vim /etc/systemd/system/gunicorn.service
```

Paste the contents below inside this file:

```
[Unit]
Description=gunicorn daemon
Requires=gunicorn.socket
After=network.target
```

```
[Service]
User=ubuntu
Group=www-data
WorkingDirectory=/home/ubuntu/blogprojectdrf
ExecStart=/home/ubuntu/blogprojectdrf/env/bin/gunico
rn \
          --access-logfile - \
          --workers 3 \
          --bind unix:/run/gunicorn.sock \
          blog.wsgi:application
[Install]
WantedBy=multi-user.target
```

Lets now start and enable the gunicorn socket

```
sudo systemctl start gunicorn.socket
sudo systemctl enable gunicorn.socket
```

Before You create Nginx File.

With this command, you can check if already a file exists.

cd /etc/nginx/sites-enabled

You can delete the existing default file using the command.

```
sudo rm -f FileName
```

Step 9: Configuring Nginx as a reverse proxy

Create a configuration file for Nginx using the following command

```
sudo vim /etc/nginx/sites-available/blog
```

Paste the below contents inside the file created

```
server {
    listen 80 default_server;
    server_name _;
    location = /favicon.ico { access_log off;
log_not_found off; }
    location /static/ {
        root /home/ubuntu/blogprojectdrf;
    }
    location / {
        include proxy_params;
        proxy_pass http://unix:/run/gunicorn.sock;
    }
}
```

Activate the configuration using the following command:

```
sudo ln -s /etc/nginx/sites-available/blog
/etc/nginx/sites-enabled/
```

Run this command to load a static file

$ sudo gpasswd -a www-data username

Restart nginx and allow the changes to take place.

```
sudo systemctl restart nginx

sudo service gunicorn restart

sudo service nginx restart
```

Additionally in case of errors

To check error logs

$ sudo tail -f /var/log/nginx/error.log

to check nginx working fine

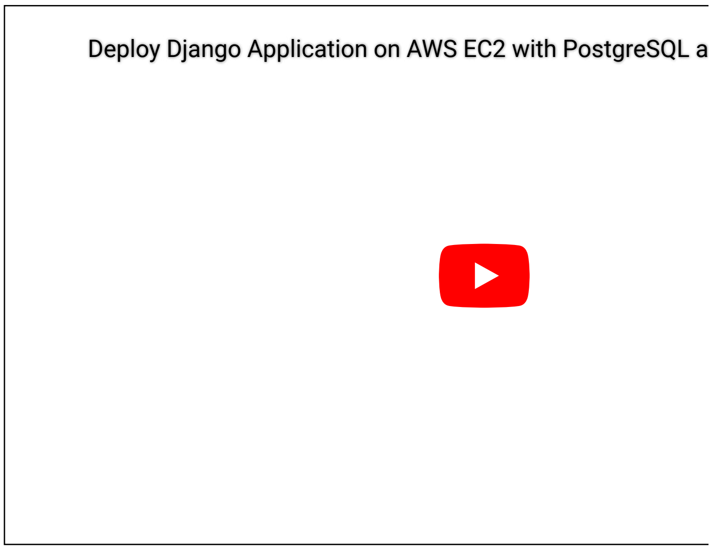
$ sudo systemctl status nginx

```
sudo fuser -k 8000/tcpsudo lsof -t -i tcp:8000 |
xargs kill -9. # to kill termianl
```

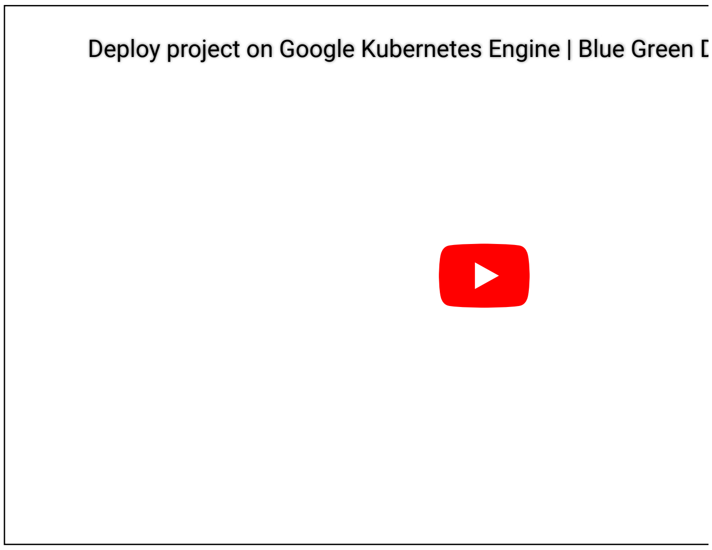
Done.

Section 04: Domain and SSL Setup:

For this Section, You can follow the Video SSL Setup Part at the end of the video.



If you want to know how to deploy applications on Google Kubernetes Engine with Gith actions, you can watch my video here.



If you enjoyed my **article**, show your appreciation with a **round of applause!** 👏 **Your support** means the world to me!

Feel free to connect with me across various social media channels! Join me on LinkedIn, YouTube, Twitter, GitHub, and Upwork. Your support means a lot. Thanks for taking the time to read this!

Signaler ceci

Publié par

Muhammad Rashid
Entrepreneur | Software Developer | AWS DevOps Guru | Python, Django, Back... 30 articles + Suivre
Publié • 2 mois

Deploying a Django application on an EC2 instance can be a challenging task, especially when you need to set up additional services like PostgreSQL, S3, domain, and SSL.

In this article, we'll walk through the step-by-step process of deploying a Django application on an EC2 instance with PostgreSQL, S3, domain, and SSL setup.

#aws #django #letsconnect #deployment #devops

J'aime Commenter Partager 8

Réactions

0 commentaire

Ajouter un commentaire...

Code With Muh

Exploring software, DevOps, freelancing, mobile, and web development with personal insights.

424 abonnés

S'abonner

En voir plus sur cette newsletter

Deploy React Application on AWS EC2 Ubuntu Server With Nginx, Domain and SSL Setup

Muhammad Rashid sur Linke...

Deploy Django app on AWS EC2 with Auto Scaling Group, Load Balancer, Route53, an...

Muhammad Rashid sur Linke...

Django vs. Node.js: A Comprehensive Guide for Backend Development

Muhammad Rashid sur Linke...