Deployment
AWS Elastic Beanstalk
Deploying Django Applications with AWS Elastic Beanstalk
Tutorial

Topics

Prerequisites

- Complete this tutorial using the AWS Academy Learner Lab dedicated to our module[1].

- Note that if you are completing this tutorial using the AWS Academy Learner Lab, the AWS Academy Learner Lab states the following:

  - **Region restriction** there is a restriction on the **AWS Regions** that can be used, namely all service access is limited to the **us-east-1** and **us-west-2** AWS Regions unless mentioned otherwise;
  - **Service usage and other restrictions** for **AWS Elastic Beanstalk**
    - *"This service can assume the LabRole IAM role."*
    - *"To create an application: choose Create Application, give it an application name, choose a platform, then choose Configure more options. Scroll down to the Security panel and choose Edit. For Service role, choose LabRole. If the environment is in the us-east-1 AWS Region, for EC2 key pair, choose vockey and for IAM instance profile, choose LabInstanceProfile. Choose Save, then choose Create app."*
    - *"Supported Instance types: nano, micro, small, medium, and large. If you attempt to launch a larger instance type, it will be terminated."*

- This tutorial assumes that you are using the AWS Cloud9 Environment set up for developing in Python. Otherwise please complete the section *Setting up Cloud9 for Cloud Application Development in Python* of the Tutorial on AWS Cloud9 for Python.

- Ensure that you have installed pip for the version of Python you are using! Recall that you can upgrade the version of pip by running at the terminal the following command

```
$  python -m pip install --upgrade pip
```

[1]This assumes that you have accepted the invitation to join the AWS Academy Learner Lab received via your NCI email account.

Note In this tutorial, ensure that you use, where applicable, the AWS Region *us-east-1* i.e. *US East (N. Virginia)* as we are using the AWS Cloud services via the AWS Academy Learner Lab.

Note In this document, all the commands that you should run from the terminal are prefixed by the '$' character.

$ command.

AWS Elastic Beanstalk[2] *"is an easy-to-use service for deploying and scaling web applications and services developed with Java, .NET, .NET Core, PHP, Node.js, Python, Ruby, Go, or Docker on familiar servers such as Apache, Nginx, Passenger, and IIS."* (source: https://docs.aws.amazon.com/whitepapers/latest/overview-deployment-options/aws-elastic-beanstalk.html) AWS Elastic Beanstalk manages all infrastructure and platform tasks.

For detailed information, please consult the official AWS Developer Guide https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html.

1 Getting started with AWS Elastic Beanstalk via the AWS Management Console

When using AWS Elastic Beanstalk via the AWS Academy, please note the followings:

- This service can assume the LabRole IAM role.
- Ensure that you use a suitable *Platform* (i.e., python in this tutorial), and *Platform version*.
- Ensure that when creating the AWS Elastic Beanstalk environment to *Configure service access* for *Service role*, use the pre-created **LabRole**. If the environment is in the **us-east-1** AWS Region, for *EC2 key pair*, use **vockey** and for IAM *EC2 instance profile*, use **LabInstanceProfile**. Please note that launching the environment can take several minutes.

[2] https://aws.amazon.com/elasticbeanstalk/

2 Deploying Django applications with AWS Elastic Beanstalk via the EB CLI

2.1 Prerequisite: AWS Elastic Beanstalk CLI Installation

1. Open AWS Cloud9 Environment
2. Install the EB CLI by running the following command at the terminal

   ```
   $ pip install awsebcli --upgrade --user
   ```

3. Verify that the EB CLI has been correctly installed by running the following command at the terminal

   ```
   $ eb --version
   ```

2.2 Deploying a Django application to Elastic Beanstalk

Please follow the official AWS tutorial on deploying a Django application to Elastic Beanstalk available at `https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-python-django.html`.

In conjunction with the official AWS tutorial, please consult the sections below that have similar names with the sections from the AWS official tutorial. The sections below provide **additional information** and guidelines to be able to preview your application in Cloud9 and to deploy your application. **Ensure that you follow all the steps from the official AWS Tutorial on *Deploying a Django application to Elastic Beanstalk* provided above (i.e., do not skip sections!), rather use this document as an accompanying resource for additional information.**

Create a Django project

- Create a new directory/folder using the *mkdir* command, and then navigate into this directory using the *cd* command. That folder will contain your Django project.

```
$ mkdir simple_proj
$ cd simple_proj
```

- Recall that in Python, it's a good approach to work with virtual environments. Create a virtual environment

  ```
  $ python -m venv env
  ```

- Activate the virtual environment

  ```
  $ source env/bin/activate
  ```

- Use pip to install Django by running the following command at the terminal

  ```
  (env) ... $  pip install django==2.1.15
  ```

- Use the *django-admin startproject* command to create a Django project. Run the following command at the terminal to create a project, in this example, named *demoproj*

  ```
  (env) ... $  django-admin startproject demoproj
  ```

- *Initialize the local git repository for your Django project* (recall to first navigate to the parent directory of your project by using the *cd* command)

  ```
  (env) ... $  cd demoproj/
  (env) ... demoproj $  git init
  ```

  - When you run the *git init* command you may see the following output

    ```
    hint: Using 'master' as the name for the initial branch.
    This default branch name
    hint: is subject to change. To configure the initial branch name
    to use in all
    hint: of your new repositories, which will suppress this warning, call:
    hint:
    ```

```
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via
this command:
hint:
hint:   git branch -m <name>
```

   – Let's create the branch named main[3]. At the terminal run the following command

   ```
   (env) ... demoproj (master) $ git branch -m main
   ```

- Create a *.gitignore* file which contain all the files you don't want to track for versioning control.

  ```
  (env) ... demoproj (main) $ touch .gitignore
  ```

- For example, add the following content to your *.gitignore* file

  ```
  __pycache__/
  ```

Django – preview your application in Cloud9

To be able to preview your application in Cloud9 perform the following tasks

1. Run the server for your Django project by specifying also the port

   ```
   (env) ... demoproj $ python manage.py runserver 8080
   ```

   Use two terminals!!! i.e., in one terminal leave the server running, and in another terminal run the commands required to further develop your project. Don't forget to activate, when needed, the Python virtual environment also in the second terminal!! For details, please refer to *Appendix: Python – Working with virtual environments*, item 2. Next, ensure that you navigate to the parent directory of your project by using the *cd* command).

[3]https://education.github.com/git-cheat-sheet-education.pdf

2. Once the server has started, Cloud9 will display a window with the preview link to your application. Click on that link; a new tab should automatically open, and you may see the *Invalid HTTP_HOST header* error documented below. Next, follow the guidelines provided at item 3 to fix that error.

3. *Invalid HTTP_HOST header: '....vfs.cloud9.us-east-1.amazonaws.com'. You may need to add '....vfs.cloud9.us-east-1.amazonaws.com' to ALLOWED_HOSTS.*

   - Modify the *settings.py* file of the project to update the ALLOWED_HOSTS to allow either the recommendation provided in the error message[4] or everything i.e., '*'
     !!! Ensure that you undo the change below for the production i.e., deployed version!!!

     ```
     ALLOWED_HOSTS = ['*'] # used for development
     #ALLOWED_HOSTS = [] #revert to this version for the production
     ```

!!!Ensure that you deploy the latest stable version of your project/application!!!

- Don't forget to add to the local versioning control repository the project's/application's files running the command *git add* once per file or group of files to enable changes to be tracked.

- Next, each time you modify one or multiple files, after the implemented functionality is tested that it works according to the specifcation, you have to commit the changes using the command *git commit*.

- Recall that at any point you can check the status of your local repository, namely details about newly created/modified files using the command *git status*.

Please recall that on the module's Moodle page there is available a *Tutorial on Source Code Management with Git and GitHub*. For additional information on how to work with git[5] please consult that tutorial.

[4]For an example, please see *Appendix: Django – preview your application in Cloud9*, item 3.

[5]https://education.github.com/git-cheat-sheet-education.pdf

Configure your Django application for Elastic Beanstalk

For detailed information, please consult the official documentation available at https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-python-dja
html, section named *Configure your Django application for Elastic Beanstalk*. A summary of the steps are provided below.

- Elastic Beanstalk uses *requirements.txt* to determine which packages to install on the EC2 instances that run your application.

```
(env) ...  demoproj (main) $  pip freeze > requirements.txt
```

- Create a directory named *.ebextensions*

```
(env) ...  demoproj (main) $ mkdir .ebextensions
```

- In the *.ebextensions* directory, create a configuration file named *django.config*

```
(env) ... demoproj $ touch .ebextensions/django.config
```

  IMPORTANT Note that the configuration files must adhere to YAML[6] or JSON specifications. When using YAML, ALWAYS USE SPACES to indent keys at different nesting levels. For detailed information about the AWS Elastic Beanstalk configuration files please consult https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/ebextensions.html.

  Update the *django.config* file with the following content. Note that the WSGIPath specifies the location of the WSGI script, i.e., *wsgi.py* file, that handles the Web Server Gateway Interface (WSGI) in our Python project, in this example, the *demoproj* project. AWS Elastic Beanstalk uses that to start your application. WSGI is a Python standard interface for web servers and Python web applications or framework to support web application portability across a variety of web servers.

[6] https://yaml.org/spec/1.2.2/

```
option_settings:
  aws:elasticbeanstalk:container:python:
    WSGIPath: demoproj.wsgi:application
```

- Now, you can deactivate the Python virtual environment with the `deactivate` command. Recall that you have to reactivate the Python virtual environment when you need to add/install packages to your application or to run your application locally (i.e., on Cloud9 Environment in this example).

  ```
  (env)  ... demoproj (main) $ deactivate
  ```

Deploy your Django Web Application using AWS Elastic Beanstalk CLI

- For a list of subcommands, options and arguments of the Elastic Beanstalk Command Line Interface (EB CLI) run the following command at the terminal

  ```
  $  eb -h
  ```

- Instead of running the command from the *item 1 Initialize your EB CLI repository with the* ***eb init*** *command* of section ***To create an environment and deploy your Django application*** from the official AWS Tutorial on *Deploying a Django application to Elastic Beanstalk*, do run the following *eb init* command to also specify the region (i.e., us-east-1) in which to create the application.

  ```
  $  eb init -r us-east-1 -p <python-platform-version> <application_name>
  ```

  Note that in the previous `eb` command

  ```
      <python-platform-version>
  ```

  and

  ```
      <application_name>
  ```

are placeholders i.e., you would provide the actual values that suit the Python Platform you would like to use and the name of the AWS Elastic Beanstalk application to be created, respectively.

e.g.,

```
... demoproj (main) $  eb init -r us-east-1 -p python-3.9 simpledemo
```

For detailed information on supported Platforms please consult *https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts.platforms.html*.

- Instead of running the command from the *item 3 Create an environment and deploy your application to it with* ***eb create****.* of section ***To create an environment and deploy your Django application*** from the AWS Tutorial on *Deploying a Django application to Elastic Beanstalk*, do run the following *eb create* command to also specify *the service role* and the *IAM instance profile*. For additional information, on managing security please consult the resource on Configuring the security of the Elastic Beanstalk environment.

  ```
  eb create <env> --service-role LabRole --instance_profile LabInstanceProfile
  ```

  Note that in the previous `eb` command

  $$< env >$$

  is a placeholders i.e., you would provide the actual name for the AWS Elastic Beanstalk environment to be created,

  $$LabRole$$

  is a IAM role pre-created for us in the AWS Academy Learner Lab, and

  $$LabInstanceProfile$$

  is an IAM instance profile pre-created for us in the AWS Academy Learner Lab

Complete the development and the deployment of your application by following the remaining of the official AWS Tutorial on *Deploying a Django application to Elastic Beanstalk* from *item 4.* of subsection ***To create an environment and deploy your Django application***, and all the other sections.

2.3 Clean Up

As advised in the official AWS Tutorial, **"To save instance hours and other AWS resources between development sessions, terminate your Elastic Beanstalk environment with `eb terminate`."**

```
eb terminate <env>
```

Note that in the previous `eb` command

$$< env >$$

is a placeholders i.e., you would provide the actual name for the AWS Elastic Beanstalk environment to be terminated.

Note that the *eb terminate* command terminates the environment and all of the AWS resources that run within it! However, it will not delete the Elastic Beanstalk application, so you can create another environment with the very same configuration by running at the terminal the command `eb create` again and deploying your application (see section 2.2).

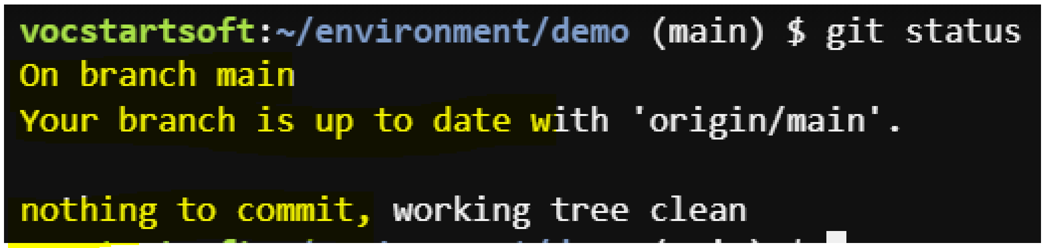
2.4 Troubleshooting

Common pitfalls that lead to deployment errors/failures.

Issue 1 Project works in the development environment (i.e., in Cloud9), but deployment fails because the deployed version **is not** the same as the latest version of your project.

Once a functionality is implemented and works as expected in the development environment, ensure that that functionality is committed to the local repository and then pushed to the private GitHub repository. Once a functionality is implemented and is working fine in Cloud 9, then we should run the following sequence of commands: 1) *git add*, 2) *git commit*, and 3) *git push*.

To double check that the latest version of your application is committed locally remember to run the command *git status*, and you should see a similar output as shown in Figure 1.



```
vocstartsoft:~/environment/demo (main) $ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Figure 1: Example output of *git status* command

Issue 2 The *django.config* file does not have the correct information and/or correct indentation (recall to use spaces NOT tabs! for indentation).

I use as an example the Django project named *demoproj* that we created in this tutorial. Then, the content of the *django.config* file should be as shown in Figure 2 (please see Figure 3 to observe the location of the *wsgi.py* file within the project).

```
option_settings:
  aws:elasticbeanstalk:container:python:
    WSGIPath: demoproj.wsgi:application
```

Figure 2: *demoproj/.ebextensions/django.config* file content

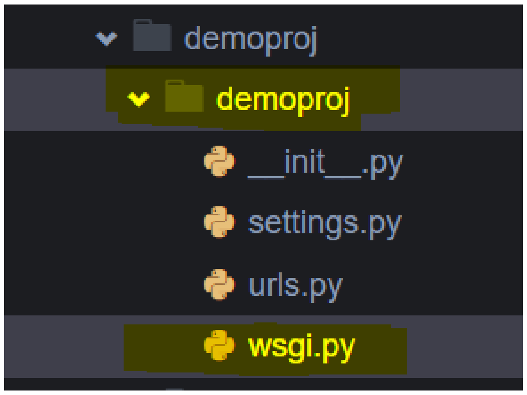


Figure 3: Directory structure of *demoproj* project with a focus on *wsgi.py*

Issue 3 Some of the libraries/packages that were installed for the project in the development environment are not installed on the instance that hosts your deployed project. Please recall that you should have used a virtual environment when developing your project (i.e., a virtual environment you created with the command *python -m venv env* **before** creating your Django project).

Ensure that before a deployment is made, if any libraries/packages have been installed since the last successful deployment, you do generate again (via *pip freeze* command) the *requirements.txt* file. Please consult the first command of the *Configure your Django application for Elastic Beanstalk* section for an example of generating the *requirements.txt* for the Django project created in this tutorial.

Once the new *requirements.txt* file has been created, as this file has been modified, do ensure that you commit it to the local repository and then push it to the private GitHub repository. Please see also *Issue 1* for additional information.

Issue 4 Models and migrations related errors

For each application one creates (i.e., via running the command *`python manage.py startapp appname`*), if that application stores information in a SQL database, then one also needs to ensure that migrations are created and run. For example, let's assume that we created a *movies* application.

```
$ python manage.py startapp movies
```

Then, each time after we create or update the models in this application, we also have to run the following two commands to:

1. Create the corresponding migration, i.e., store the changes as a migration

   ```
   $ python manage.py makemigrations movies
   ```

2. Next, run the *migrate* command to create the model table in the database

   ```
   $ python manage.py migrate
   ```

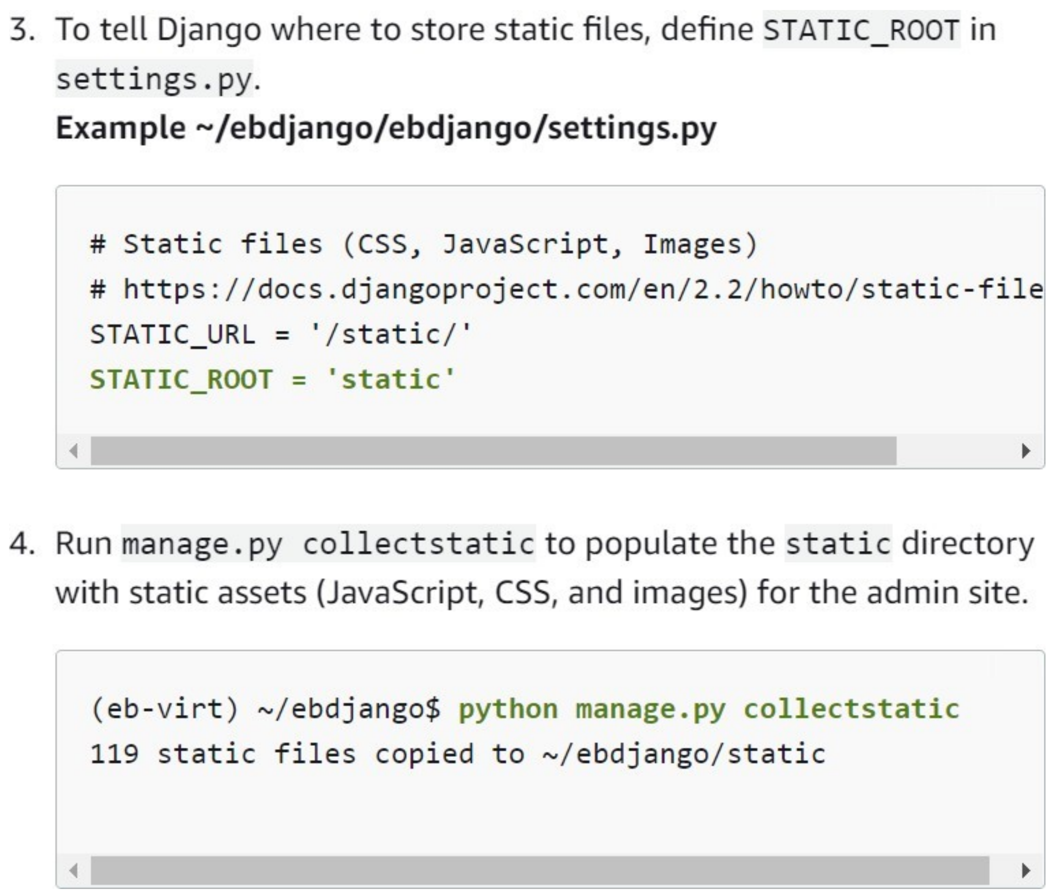
Issue 5 Static files not served for the deployed project

Please make sure that you did not skip the instructions shown in Figure 4 from the official AWS tutorial on Deploying a Django application to Elastic Beanstalk[7].

Note that when you run the command ***python manage.py collectstatic*** shown in Figure 4 a folder named ***static*** is going to be created in the parent folder i.e., root folder of the project, (parent folder named as project i.e., *ebdjango* in the official AWS tutorial, and *demoproj* in this tutorial), and then the aforementioned command is going to collect i.e., copy all the static files used by the different applications of your project (e.g., .css, .js, etc.) in that folder. <u>In the deployed project</u>, the static files are going to be served from the *static* folder (due to the configurations made in the project by following the guidelines from Figure 4).

Elastic Beanstalk configures some static file folders by default when using a Python platform, namely "by default, the proxy server in a Python

[7]https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-python-django.html


3. To tell Django where to store static files, define `STATIC_ROOT` in `settings.py`.
Example ~/ebdjango/ebdjango/settings.py

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/2.2/howto/static-file
STATIC_URL = '/static/'
STATIC_ROOT = 'static'
```

4. Run `manage.py collectstatic` to populate the `static` directory with static assets (JavaScript, CSS, and images) for the admin site.

```
(eb-virt) ~/ebdjango$ python manage.py collectstatic
119 static files copied to ~/ebdjango/static
```

Figure 4: Guidelines to be followed to ensure static files are served for the deployed project [source: AWS].

environment serves any files in a folder named ***static*** at the /***static*** path" (source: Using the Elastic Beanstalk Python platform). For an example, please consult the section *Static files* of Using the Elastic Beanstalk Python platform.

Please recall that you have to *add* and *commit* to your local repository, and then *push* to the private GitHub repository that ***static*** folder and **all** of its content. Please see also Issue 1 for additional information.

If you'd like to change the default configuration for the serving of static files please consult the AWS Elastic Beanstalk resource on Serving static files.

Appendix: Python – Working with virtual environments

Virtual environments allow us to manage separate package installations for different projects. For more information please consult https://docs.python.org/3/tutorial/venv.html and https://packaging.python.org/guides/installing-using-pip-and-virtual-environments/

1. Create a virtual environment: to create a virtual environment, navigate (by using the command *cd*) to your project's directory and run the command *venv* at the terminal. The following command will create the virtual environment named env.

   ```
   $  python3 -m venv env
   ```

2. Activate a virtual environment: before we can start installing or using packages in the virtual environment we have to activate the virtual environment we created, by running the following command at the terminal

   ```
   $  source env/bin/activate
   ```

   Note that once the virtual environment is activated the terminal's prompt is prefixed by the name of your virtual environment (i.e., (env) in this example).

3. Deactivate the virtual environment: when you want to exit the virtual environment (for example when you finish a working session, or would like to work on another project) run the following command at the terminal

   ```
   $  deactivate
   ```

4. Each time you want to enter again the virtual environment for your project follow step 2. Note that typically we run only once *item 1* per project, and as many times needed *item 2* and *item 3*.

Freezing dependencies

We can export a list off all installed packages and their versions by running the command *pip freeze* at the terminal

```
$  pip freeze
```

Appendix: Django – preview your application in Cloud9

To be able to preview your application in Cloud9 perform the following tasks

1. Run the server for your Django project by specifying also the port

   ```
   $  python manage.py runserver 8080
   ```

   Use two terminals!!! i.e., in one terminal leave the server running, and in another terminal run the commands required to further develop your project.

2. Once the server has started, Cloud9 will display a window with the preview link to your application. Click on that link; a new tab should automatically open, and you may see the *Invalid HTTP_HOST header* error documented below. Next, follow the guidelines provided at item 3 to fix that error.

3. **error:** *Invalid HTTP_HOST header: '....vfs.cloud9.eu-west-1.amazonaws.com'. You may need to add '....vfs.cloud9.eu-west-1.amazonaws.com' to ALLOWED_HOSTS.*

   - Modify the *settings.py* file of the project to update the ALLOWED_HOSTS with your Cloud9 environment's domain name provided in the error message (i.e., '....vfs.cloud9.eu-west-1.amazonaws.com')

     ```
     # used for development environment
     # NOTE you must use the Cloud 9 URL of your own environment!!!
     ALLOWED_HOSTS = ['....vfs.cloud9.eu-west-1.amazonaws.com']

     #ALLOWED_HOSTS = [] #revert to this version for the production
     ```

     !!! Ensure that you undo the changes you made above at item 3 for the production i.e., deployed version of the application!!!

Resources

- AWS Elastic Beanstalk Developer Guide https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html
- Getting Started with AWS Elastic Beanstalk https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/GettingStarted.html
- AWS Elastic Beanstalk – Working with Python https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-python-apps.html
- Using the Elastic Beanstalk Python platform https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-python-container.html
- Deploying a flask application to Elastic Beanstalk https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-python-flask.html
- Deploying a Django application to Elastic Beanstalk https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-python-django.html
- Viewing logs from Amazon EC2 instances in your Elastic Beanstalk environment https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.logging.html
- Install the EB CLI https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3-install.html
- Adding an Amazon RDS DB instance to your Python application environment https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-python-rds.html
- Serving static files https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/environment-cfg-staticfiles.html
- AWS Cloud9 https://docs.aws.amazon.com/cloud9/latest/user-guide/welcome.html

- Python Tutorial for AWS Cloud9 `https://docs.aws.amazon.com/cloud9/latest/user-guide/sample-python.html`
- Django `https://docs.djangoproject.com`
- Django Official Tutorial `https://docs.djangoproject.com/en/4.1/intro/`
- Flask `https://flask.palletsprojects.com/en/2.2.x/`