Created by: Nicolas Melillo
4/2/2017

Elastic Beanstalk Free Tier Deployment Instructions 2017

Detailed herein is a step by step process (and explanation) of how to prepare a project to be deployed to Amazon Web Services (AWS) in an EC2 instance, using Elastic Beanstalk.

Foreword:

Elastic Beanstalk is a part of the AWS feature set that allows the creation of load-balanced virtual machines. Notoriously, Elastic Beanstalk is known for being expensive if not configured correctly as it will spawn multiple EC2 instances to take care of demand concerns! Follow this step by step process very closely to make sure that the configured EC2 instance is set to "Single-Instance" mode (more about this later) or charges could add up quickly.

The reason Elastic Beanstalk is used in this process is because of its incredible ease of deployment with Django environments. An alternative to this tutorial is to deploy directly to a singular instance of an EC2 Linux machine. This requires lots of configuration in both the virtual machine and routing in the AWS Web Console to get a Django app to run. Elastic Beanstalk's Command Line Interface abstracts this difficult configuration process.

Lastly, please note that these instructions were created in Q1 of 2017. Since AWS is widely used but always improving, some of these instructions could become out of date. Modification to these instructions in the event the deployment process is changed is suggested and admired.

Required Prerequisites:

Note: These instructions were completed on a Linux environment. This is the preferred development and deployment environment. While deployment on a Windows machine is possible, *some* modification to the Deployment Instructions below may be required.

- Python >= 3.4 (you can deploy 2.7 but these instructions were completed with 3.4)
- Pip utility
- Awsebcli – Instructions on installation of this package are below.
- Command line interface of your choosing (Terminal, Terminator, etc.)

Optional Prerequisites

While not required it is advised to have these packages installed.

- SSH – to communicate with the created EC2 environment.

Deployment Instructions:

1) **Install awsebcli:** This package is supplied by Amazon to help facilitate the initiation, creation and installation of an Elastic Beanstalk instance.

- To install this, run the following command in your terminal:

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

For more information on the exact installation steps of this package can be found at: http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3-install.html

2) **Create a working directory:** A directory for pulling the data from the Git repository will need to be created so that we can initialize it for use with AWS.

- Create a directory as usual using this command in your terminal:
  ```
  mkdir ./{name of folder}
  ```
  The exact name of the directory is not important, call it whatever you please.
- Once you have done this, move into that directory using:
  ```
  cd ./{name of folder}
  ```

3) **Clone the Source-Code from the Git Repository (repo):**

- Using standard Git commands, pull the applicable branch of your codebase into the current directory. An example of this would be:
  ```
  git clone https://{address of repo} --branch {applicable branch}
  ```
  After this command completes, a new directory named after your project should be created and populated with data from the selected branch.
- Move into this directory by using:
  ```
  cd ./project-name
  ```

4) **Create a requirements.txt File:** To tell AWS what packages need to be installed to the EC2 instance, we must create a requirements.txt file that will contain the packages used in the project.

- Create a file titled “requirements.txt” using either a text editor, vim, nano, etc.
- Add in the required packages from the Git repository’s requirements section in the following format (always check the repository’s README for up-to-date requirements):
  Django==1.10.6
  django-filter==1.0.1
  djangorestframework==3.5.3
  Markdown==2.6.8
  Pillow==4.0.0
  pycryptodome==3.4.5
  pycryptodomex==3.4.5
  pyjwkest==1.3.2
  social-auth-app-django==1.1.0

5) **Create a django.config:** To configure AWS to run the Django app, a configuration file must be created that points to the wsgi.py file in the project folder. This is a **very important** step or your program will not be deployed correctly and the site will not be accessible!

- Create a new directory by running the following command in your terminal:
  ```
  mkdir .ebextensions
  ```
- Move into this directory like before, using:

```
cd .ebextensions
```

- Now, using a text editor like before, create a file named django.config.
- Add the following text into the file to configure the project:

```
option_settings:
  aws:elasticbeanstalk:container:python:
    WSGIPath: ApplicationBackend/wsgi.py
```

In this example above, the wsgi.py file is in a folder called ApplicationBackend. The file hierarchy may be different for your project. If this is the case, make sure the file path points to the wsgi.py file in your project while ignoring the root folder for your project! For example: ApplicationBackend/wsgi.py **NOT** MyProjectFolder/ApplicationBackend/wsgi.py

- When your done editing, save the file and move back to the root project directory for the next step using:

```
cd ..
```

6) **Initialize Project Folder for Elastic Beanstalk:** Now that our files are local, we need to initialize the folder (and the instance settings) using the AWS Command Line Utility.

- Running the following command will start the Elastic Beanstalk CLI tool:

```
eb init
```

- Select the applicable region to deploy your app. In this example, deployment will be on us-west-2: US West (Oregon).
- Enter an application name. To leave the default (which is satisfactory) just press enter.
- A prompt for using Python will appear, select 'Y' and choose the option for Python 3.4.
- **Optional:** You can enable AWS CodeCommit at this point, this documentation does not cover this and recommends selecting 'n'.
- **Optional:** Lastly, you can setup your instance to support SSH. If you have not created a key-pair through the AWS console via a web browser, the process will guide you through creating one after selecting 'Y'.

  If you have a key-pair previously configured, you will have the option to select it or create a new one.

7) **Commit Your Local Changes:** For some reason, certain config files will not be detected on deployment unless you commit your changes. While you don't need to push these changes to your repo, it is suggested to have a "aws deployment ready" branch. This simplifies the process later.

- Be sure to add the files we created, including the config folder using the standard:

```
git add {file(s)}
```

  It is important that you add the .ebextentions folder! In my experience I have always added the .elasticbeanstalk folder created as a result from the step above, as well.
- Commit the changes using:

```
git commit "comment"
```

8) **Create Environment and Deploy:** Once the eb init command is completed a directory

titled .elasticbeanstalk should have been created (this validates the current directory has been initialized for deployment).

Make sure you are ready to deploy!

- This step will create a virtual environment and deploy the code **all in one go** using the following command:

```
eb create project-name-env
```

Optionally, you can replace “project-name-env” with any title for your EC2 instance that you would like. Additionally, you can add any modifications here to request specific EC2 instances or settings. Above is an example of using the basic settings, more info on modifiers can be found on the AWS developer pages.

9) **Optional (but highly recommended) Modify Elastic Beanstalk Configuration:** To prevent any outrageous costs, you can edit the created Elastic Beanstalk project to be a “single instance”.

- Log into the AWS web console by typing the following command into your terminal

```
eb open
```

- Click on the project we just created.
- Maneuver to the configuration page (on the left side).
- Select the cog on the top right of the “Scaling” box.
- Select the drop-down that says “Environment Type” and select “Single Instance”
- Now select an Availability Zone in the box below (this is up to your preference and you can leave it unselected for AWS to decide automatically).
- Apply your settings and Elastic Beanstalk will make the changes automatically.

Troubleshooting:

This section details a few issues I experienced personally with deploying a project. Because the nature of “anything can happen”, not every issue can be specified in this section. Below are some basic issues that I feel should be noted and more importantly, these solutions could be a huge help in other projects! I advise you read these sub sections even if you aren’t experiencing any issues.

Problem: I cannot get Pillow package dependencies to install (jpeglib)
Solution:

- Create a file titled 01_packages.config in your .ebextensions folder.
- Add the following to the file:

```
packages:
  yum:
    libjpeg-turbo-devel: []
container_commands:
  05_uninstall_pil:
    command: "source /opt/python/run/venv/bin/activate && yes | pip uninstall Pillow"
  06_reinstall_pil:
```

command: "source /opt/python/run/venv/bin/activate && yes | pip install Pillow --no-cache-dir"

Pillow requires a set of dependencies to be installed before pip will install the package successfully. The above config file tells AWS you want to install libjped-turbo-devel (the main required dependency for Pillow) **before** installing the requirements.txt items. This is a great place to put any further dependencies needed for your project (note they must be supported by the yum repo system) that cannot be obtained through pip.

Note: AWS has a multitude of virtual machines! In this example (and the default created by the Elastic Beanstalk deployment process [unless otherwise specified]) is an AMI virtual machine which is RPM based. You will need to change yum to apt-get in the event you use an Ubuntu virtual machine!

Problem: Backgrounds or static files are not loading [correctly]
Solution: If this is happening the static files directory needs to be changed. The easiest way to do this is to add the following to the django.config (in line with "aws:elasicbeanstalk:container:python: but under the WSGIPath line) created in the steps above:

```
aws:elasticbeanstalk:container:python:staticfiles
  /static/ = Static/Folder/Path/static/
```

Change Static/Folder/Path with the path to your static files in your project. This will link any reference to /static/ (which should be used often in your Django app) to that directory.

Problem: "My app isn't working and I don't know what's wrong"
Solution: Awsebcli gives us a powerful way to view how the process of deployment went without logging into the AWS console via a browser.

- Run the following command in a terminal (be sure you are still in your project directory):

```
eb logs
```

This will gather the multitude of logs that are accumulated during deployment and runtime. In my experience these logs are much more informative than the AWS web console logs and can help decipher a better reason as to why your app is not running.

Conclusion:

The main purpose of this document was to aid a developer in simply deploying a Django app to an EC2 instance. AWS supplies a very large feature set for developers to take advantage of. If you successfully followed the basic steps in this document (especially step 9) than your app should remain within the free tier with average usage (obviously, this is also based on your specific implementation). Note that this document does not touch on the permissions system, or other in depth features of AWS. It is your duty to make sure your app is functional, and secure! Further information on specific features and or configurations should be researched on your own time.

If there are any errors in the process described above, or if you have any specific questions please feel free to email me at nkm38@nau.edu.

Helpful Links:

During my personal attempts at deploying a Django app to AWS, I consulted a multitude of links and resources online to narrow down the process to the steps above. Below are a few helpful links to resources I followed (apart from trial and error) and the ones I feel will remain most up to date. In the event that this deployment process changes, the Amazon resources are likely the best place to go for reference material on how to deploy.

- http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-python-django.html
  This is a great walk through for deploying a Django app in Python 2.7
- http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3.html
  Here is basic information (along with installation links for multiple OS's) for awsebcli
- https://realpython.com/blog/python/deploying-a-django-app-and-postgresql-to-aws-elastic-beanstalk/
  This resource is very useful if you intend on using AWS's database systems for your Django app. Be careful as some of the instructions on this page do not remind you to commit your changes!
- https://www.sayonetech.com/blog/deploying-django-aws-elastic-beanstalk/#.WOcO3ojyuUk
  This is a nice resource with images on how to deploy to EB by first creating a prebuilt project. This process is unnecessary if you follow the steps above, but it could still be helpful with getting comfortable with the AWS Web Console. This also has some instructions on how to implement a database for your app through AWS.