

# Deploying a Django web app as containerized microservice using a CI/CD pipeline

## 0. Introduction

For this project, we were required to deploy a Django web app as containerized microservice using a CI/CD pipeline, making full use of what we've learned over the course of 2 months in this bootcamp.

There are 6 tasks in total:

- Writing the docker file
- Configuring an ubuntu instance as a slave for use with the CI/CD pipeline
- Integrating a slack workspace with our Jenkins instances
- Installing a plugin to create statistics about builds
- Writing Jenkinsfiles for both a master and a dev branch
- configure job in jenkins using multibranch pipeline type with our repo

I detail how I went about each task in the following sections.

## 1. Writing the docker file

I started by figuring out how to launch a Django app, since I wasn't familiar at all with the frameworks. Thankfully, the repo README provided a clear step-by-step for launching the application. This would go into the docker file.

```
pip3 install -r requirements.txt
python3.6 manage.py makemigrations
python3.6 manage.py migrate
python3.6 manage.py runserver 0.0.0.0:8000
```

It was required to use an ubuntu image as base for this project, thus I had to install python and pip on the image.

While working on the dockerfile, I was forced to learn about how docker applies layer cache. Each time a layer's hash is invalidated, all subsequent layers are re-applied, ignoring any existing cache. This made the placement COPY something to be careful about, in order not to have to install python on every build.

Aside from dealing with layer cache, the dockerfile was pretty straight forward.

## 2. Configuring an ubuntu slave

This is a step that could've been much harder without guidance from the recorded video. Even then, I was faced with some problems, most notable of which was forgetting to turn on the ssh service.

Another problem I faced is that restarting the docker container would change its assigned IP address, stopping the Jenkins master's attempts at communication. I didn't find any solution to this problem aside from manually changing the host in the nodes configurations on the Jenkins master.

```
karim@karim:~/sprints/Booster_CI_CD_Project$ docker image ls |grep django
karimelsayad247/django-app      prod      a9f12b29ef7f   15 minutes ago   445MB
karimelsayad247/django-app      dev       89169ddf2c5c   35 hours ago     445MB
django-app                      v2        73544a9b5d1a   4 days ago       445MB
karim@karim:~/sprints/Booster_CI_CD_Project$ docker image ls | grep slave
jenkins-slave                   ubuntu    bd6fbe6eafcc   2 days ago       719MB
```

Figure 1: Django app images and the Jenkins slave image

Aside from that, it was all about installing the java dependencies, openssh, and git.

From inside the slave, we can see that the pipeline worked as intended

```
root@c8b5294aefbb:/jenkins-home# ls
caches  remoting  remoting.jar  workspace
root@c8b5294aefbb:/jenkins-home# ls workspace/
django_project_dev      django_project_master      workspaces.txt
django_project_dev@tmp  django_project_master@tmp
root@c8b5294aefbb:/jenkins-home# ls workspace/django_project_master
Dockerfile  Jenkinsfile  README.md  manage.py  requirements.txt  simpleApp
root@c8b5294aefbb:/jenkins-home#
```

Figure 2: Inside the ubuntu slave container

### 3. Integrating a slack workspace

This step might be the most straightforward. After installing the Jenkins integration on the workspace's end, we are presented with simple guide on how to set everything up on the Jenkins server's end. A particular issue was that the plugin guide was outdated, and interface has changed.

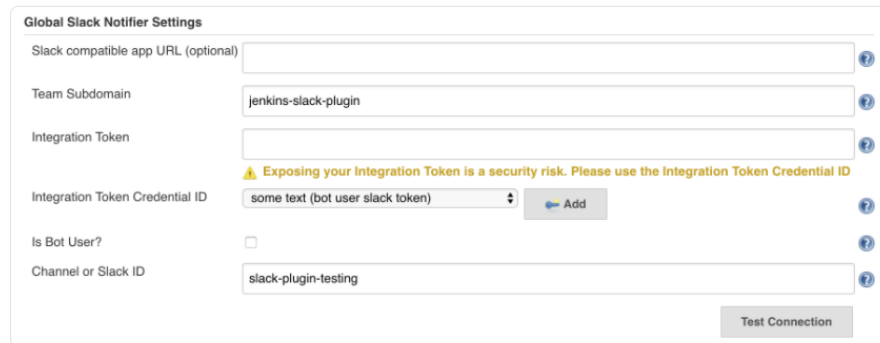
#### Step 3

After it's installed, click on **Manage Jenkins** again in the left navigation, and then go to **Configure System**. Find the **Global Slack Notifier Settings** section and add the following values:

- **Team Subdomain:** [REDACTED]
- **Integration Token Credential ID:** Create a secret text credential using [REDACTED] as the value

The other fields are optional. You can click on the question mark icons next to them for more information. Press **Save** when you're done.

**Note:** Please remember to replace the Integration Token in the screenshot below with your own.



The screenshot shows the 'Global Slack Notifier Settings' configuration page. It includes the following fields and options:

- Slack compatible app URL (optional):** An empty text input field.
- Team Subdomain:** A text input field containing 'jenkins-slack-plugin'.
- Integration Token:** An empty text input field.
- Integration Token Credential ID:** A dropdown menu showing 'some text (bot user slack token)' and an 'Add' button.
- Is Bot User?:** An unchecked checkbox.
- Channel or Slack ID:** A text input field containing 'slack-plugin-testing'.
- Test Connection:** A button at the bottom right.

A warning message is displayed: '⚠ Exposing your Integration Token is a security risk. Please use the Integration Token Credential ID'.

Figure 3: The slack plugin's guide on how to set up the Jenkins server plugin

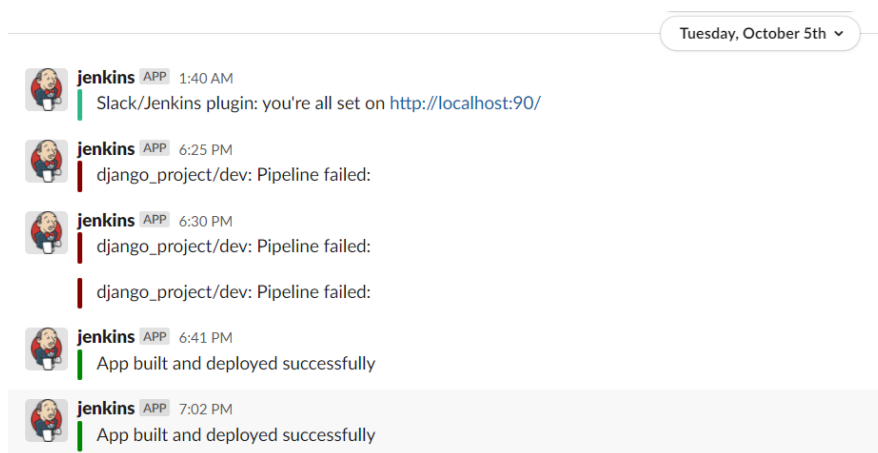


Figure 4: The plugin reporting build results

## 4. Writing Jenkinsfiles

For this step, we are required to write jenkinsfile with four stages for both dev and master branches. These steps are:

- **Preparation:** Checking out the code from the repo
- **Build image:** build image using the dockerfiles we created in first step
- **Push image:** push the built image to the docker hub registry
- **Deploy:** deploy a container from the pushed image
- **Notification:** send a slack message with the build status

This was perhaps the step where I faced most trouble. Small typos here and there would cause the pipeline to fail, and catching them would require waiting for the pipeline to reach the problematic step, making debugging rather time-consuming.

In the following subsection, I detail how I went about each of the aforementioned steps. The repo contains the full jenkinsfile, so I'll keep it to the essential commands.

### 4.1. Preparation

Using the git plugin for Jenkins makes this step an easy one liner:

```
git "https://github.com/KarimElsayad247/Booster_CI_CD_Project.git"
```

### 4.2. Building the image


The pipeline would invoke this shell script, substituting prod with dev depending on the branch

```
docker build -t karimelsayad247/django-app:prod .
```

### 4.3. Pushing the image

Using Jenkins credentials binding plugin, one could easily substitute credentials in the jenkinsfile while keeping them safe away from version control.

```
withCredentials([usernamePassword(credentialsId: 'dockerhub',
usernameVariable: 'USERNAME', passwordVariable: 'PASSWORD')])
{
    sh """
    docker login -u ${USERNAME} -p ${PASSWORD}
    docker push karimelsayad247/django-app:prod
    """
}
```


**karimelsayad247 / django-app**

This repository does not have a description

⌚ Last pushed: an hour ago

**Docker commands**

To push a new tag to this repository,

```
docker push karimelsayad247/django-app:tagname
```

[Public View](#)

**Tags and Scans**

This repository contains 2 tag(s).

TAG	OS	PULLED	PUSHED
dev	linux	a day ago	a day ago
prod	linux	In a few seconds	an hour ago

[See all](#)

**Automated Builds**

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available on Pro and Team plans.

[Upgrade to Pro](#) [Learn more](#)

Figure 5: App repository on docker hub. Notice 2 images, one for each branch

## 4.4. Deployment

One thing to be careful of is to delete existing containers before deploying new ones. This keeps the server from having way too many containers up and running at the same time, allows usage of the same port, and reusing the same container name for ease of management.

```
docker container stop djangoApp-dev || true
docker container rm djangoApp-dev || true
```

```
docker container run -d -p 8001:8000 --name djangoApp-dev
karimelsayad247/django-app:prod
```

One thing to notice is that the `stop` and `rm` commands fail if there are no containers of the specified name. A workaround is to OR them with `true`, to allow them to fail without disrupting the pipeline if there were no containers running our app.

CONTAINER ID	IMAGE	COMMAND	CREATED
80aab6d0bbad	karimelsayad247/django-app:prod	"python3 manage.py r..."	2 hours ago
c8b5294aefbb	jenkins-slave:ubuntu	"/bin/bash"	2 days ago
2cf3a1888f06	jenkins/jenkins:lts	"/sbin/tini -- /usr/..."	9 days ago

STATUS	PORTS	NAMES
Up 2 hours	0.0.0.0:8001->8000/tcp, :::8001->8000/tcp	djangoApp-dev
Up 2 hours		jenkins_slave
Up 10 hours	50000/tcp, 0.0.0.0:90->8080/tcp, :::90->8080/tcp	confident_neumann

Figure 6: Containers running at this point. Notice the `djangoApp-dev` that was run by the pipeline

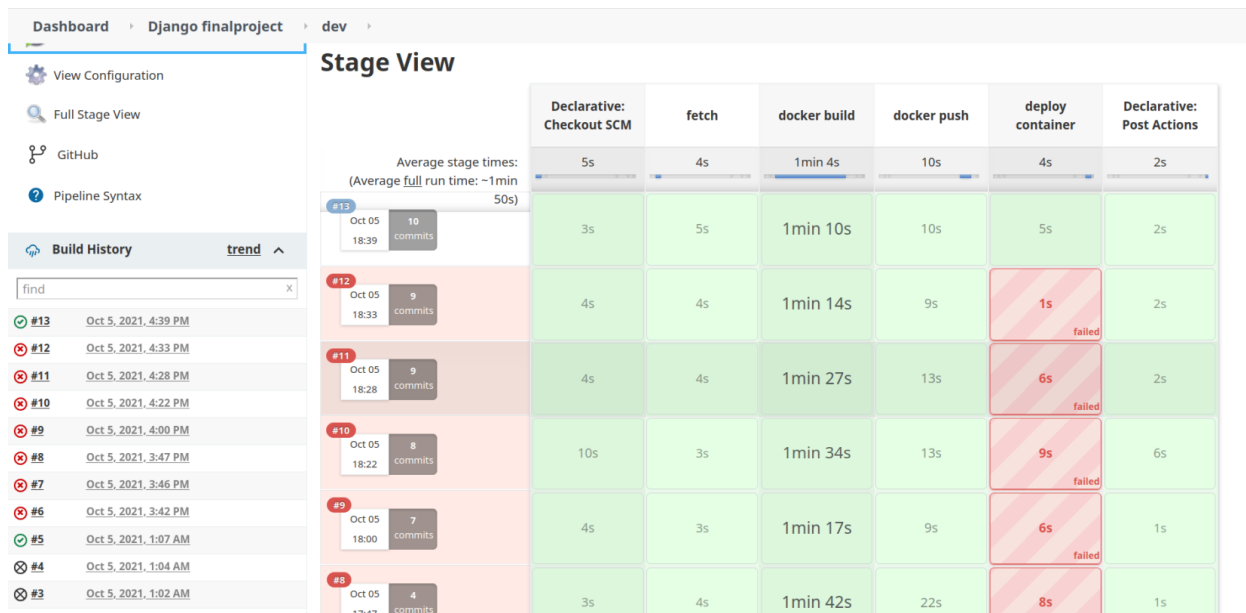


Figure 7: Multiple runs (and multiple failures!) of the pipeline

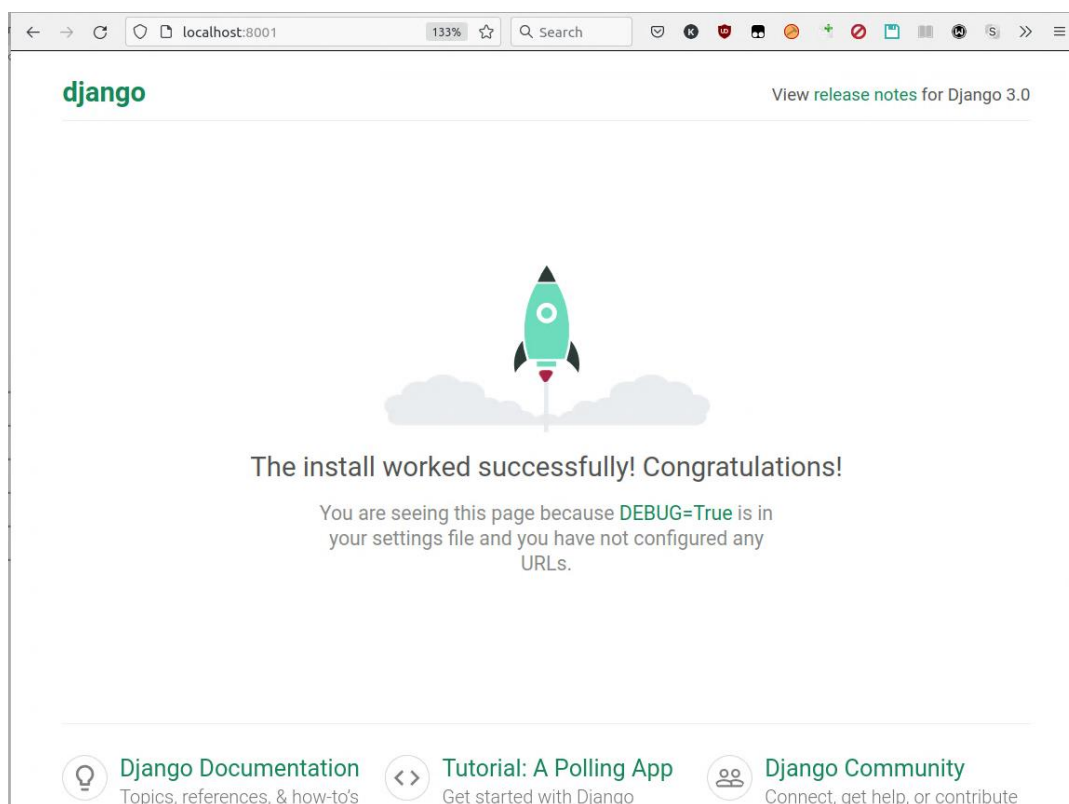


Figure 9: The Django app, up and running on port 8001

## 4.5. Sending messages to slack channel

Upon success or failure of the pipeline, appropriate messages are sent to the designated slack workspace channel. Refer to Figure 4 for a demonstration.

Example: in case of pipeline success, the following line is invoked

```
slackSend(color: "#008800", message: "App built and deployed successfully")
```

## 5. Installing a plugin to create statistics about build

The only working plugin I've managed to find is [global-build-stats](#), however it appears not to record jobs of pipeline type, making it rather useless for this particular project. It records freestyle-type jobs fine, as seen in the following figure when I ran my first job ever.

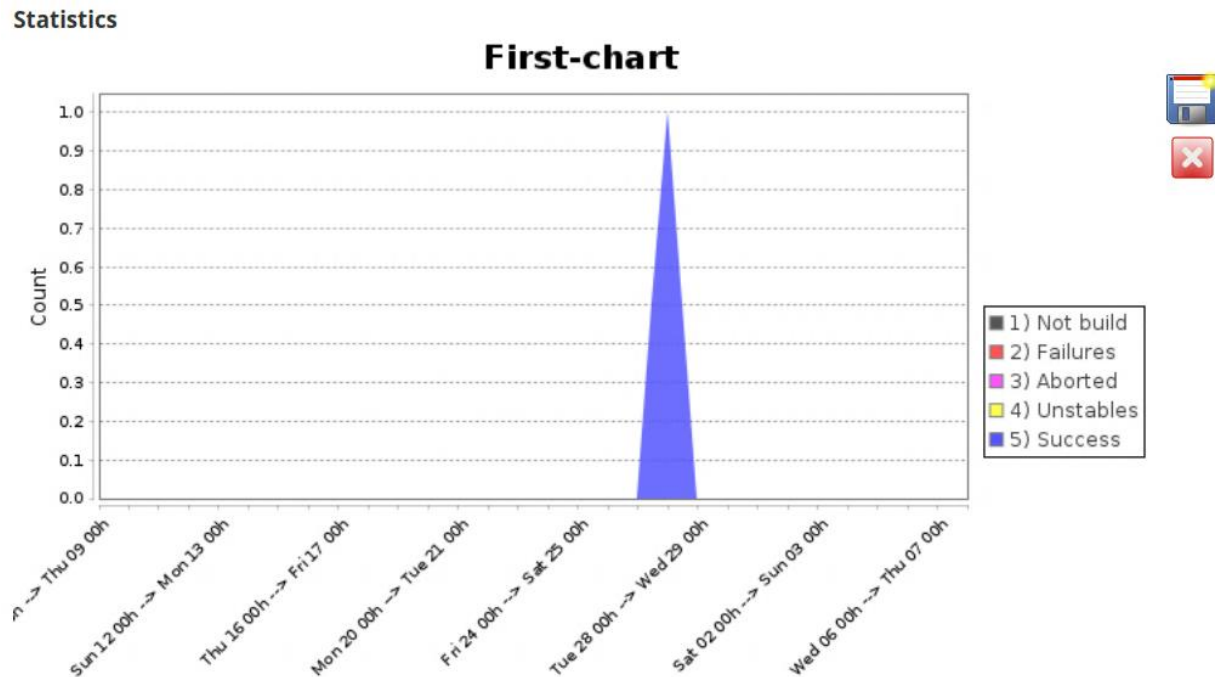


Figure 10: Chart showing a single successful freestyle job, the only one I've ran so far