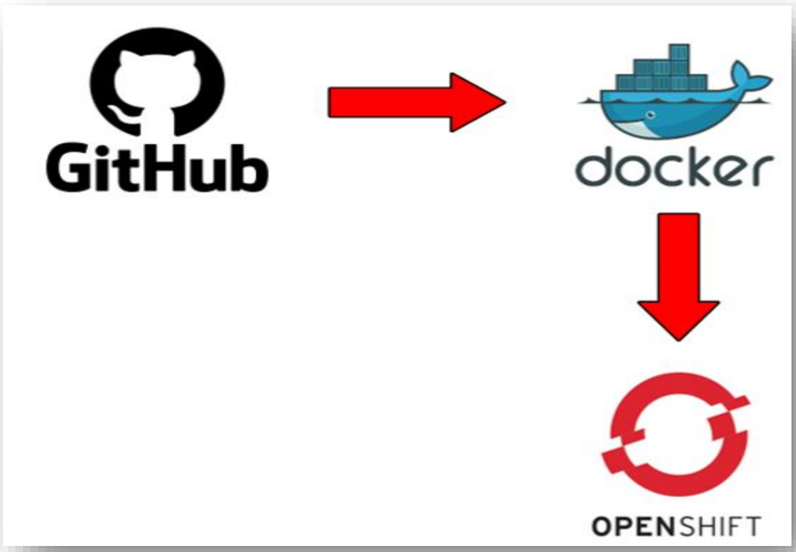
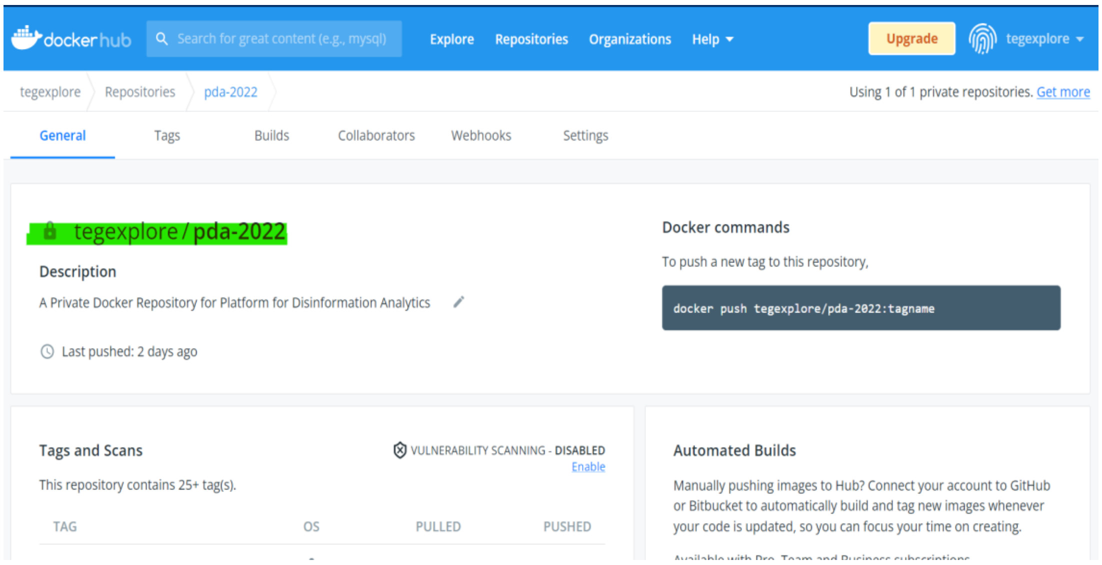
OPENSHIFT

In this documentation, we will discuss the key steps involved in the deployment of applications on Minishift - a platform for users to run OpenShift locally. In summary, the steps are as follow: the creation of docker images, building and pushing of docker images, via GitHub workflow actions, to Docker hub, and the creation of a YAML configuration to deploy Kubernetes resources on Minishift. Below, we will take a deeper look at the workflow of an application deployment.

Workflow



The general workflow for creating, pushing, and pulling docker images for deployments on Minishift, or Kubernetes, goes like this: we will first do up a Docker file, and upload it onto GitHub, and after which, we will take advantage of GitHub's CI/CD workflow, which will provide us with its amazing CI/CD automation workflow tool. This tool automates the process of building docker images from the docker file already uploaded on GitHub, and pushing the images to a private Docker Hub repository. Once the images are pushed to the private repository, Minishift will then be able to pull the images before containerizing them for deployment. Below is a sample snapshot of a private Docker Hub repository.


docker hub
Search for great content (e.g., mysql)
Explore
Repositories
Organizations
Help
Upgrade
tegexplore
tegexplore › Repositories › pda-2022
Using 1 of 1 private repositories. Get more
General
Tags
Builds
Collaborators
Webhooks
Settings

tegexplore / pda-2022

Description

A Private Docker Repository for Platform for Disinformation Analytics

Last pushed: 2 days ago

Docker commands

To push a new tag to this repository,

```
docker push tegexplore/pda-2022:tagname
```

Tags and Scans

VULNERABILITY SCANNING - DISABLED
Enable

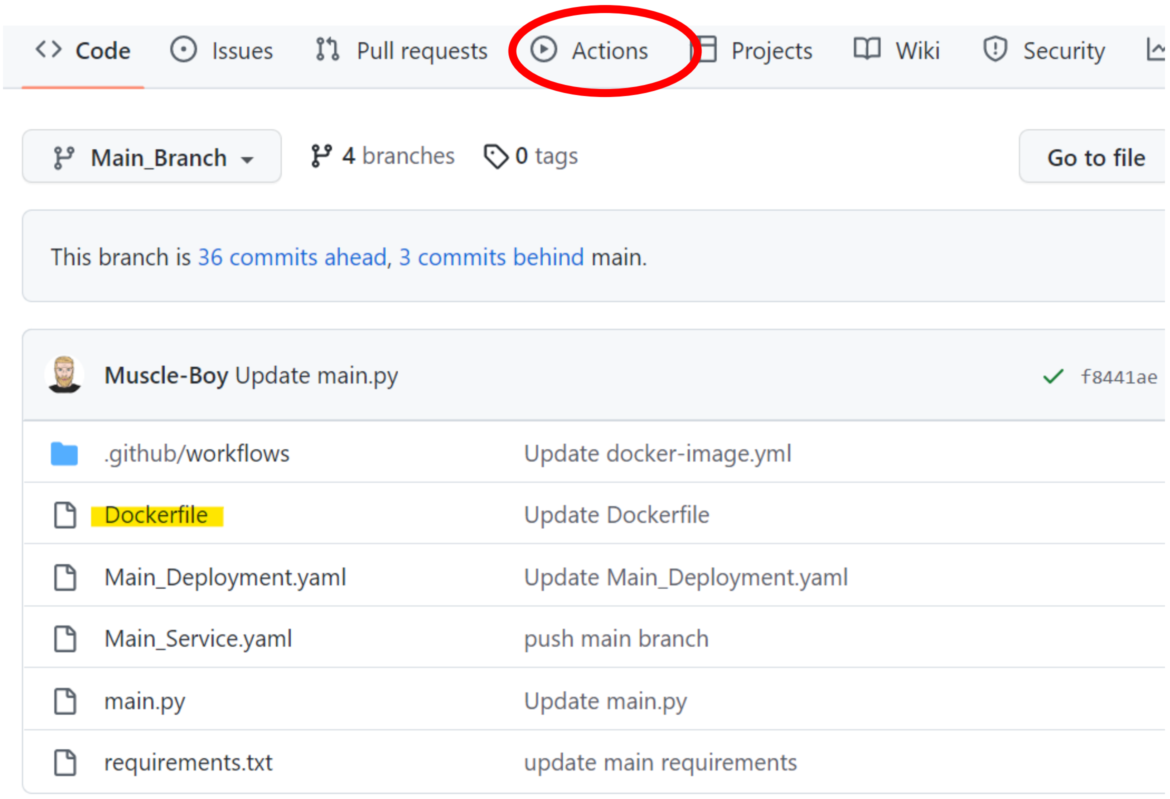
This repository contains 25+ tag(s).

TAG	OS	PULLED	PUSHED

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.

Now, before diving into the deployment aspect of TCE, let us look at how we can build a docker image using GitHub's CI/CD utility.

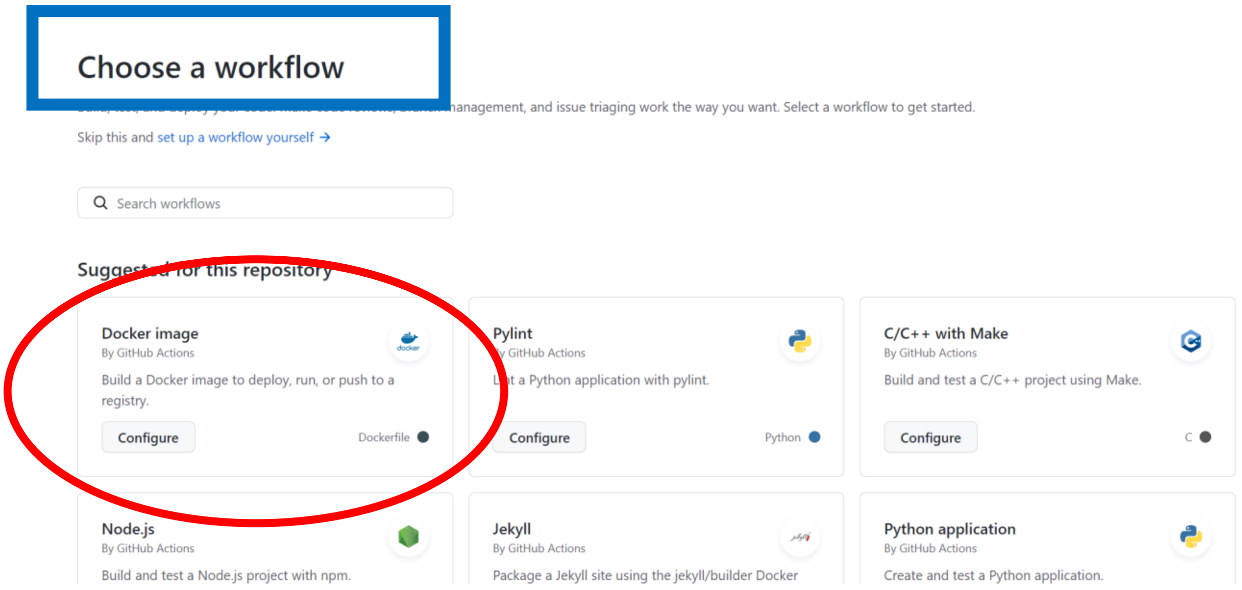


This branch is 36 commits ahead, 3 commits behind main.

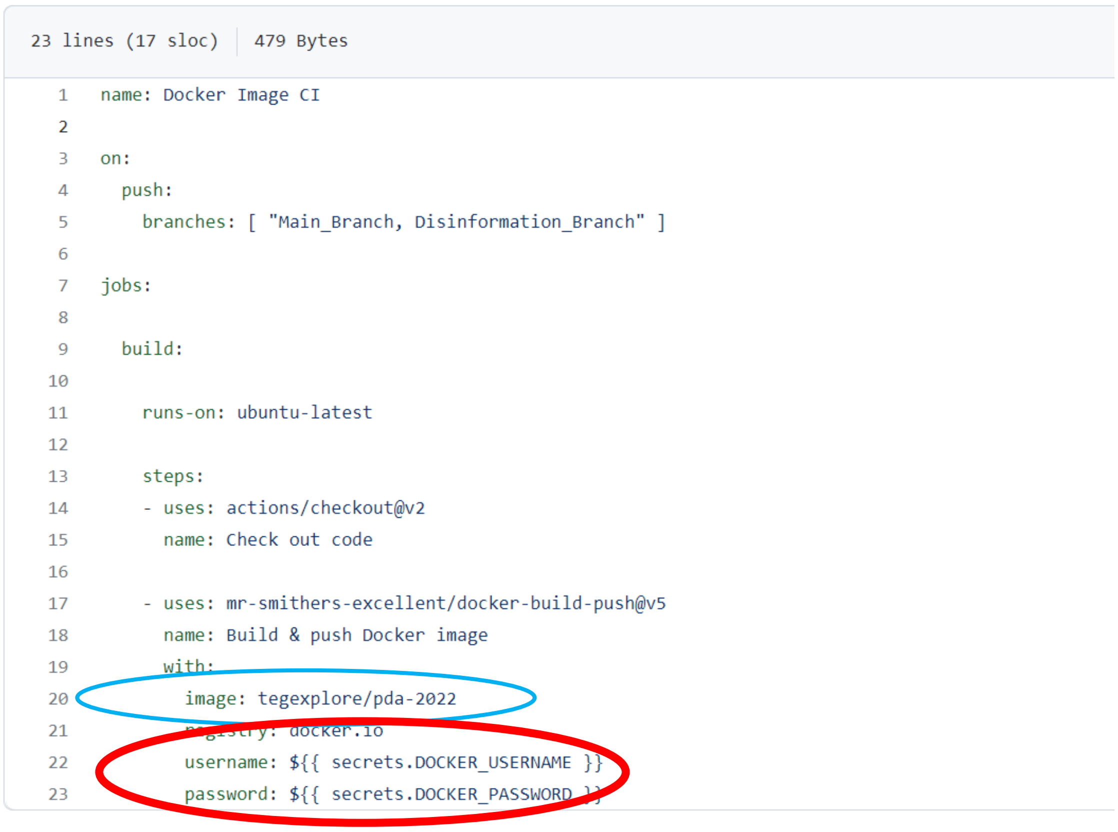
Muscle-Boy Update main.py ✓ f8441ae

.github/workflows	Update docker-image.yml
Dockerfile	Update Dockerfile
Main_Deployment.yaml	Update Main_Deployment.yaml
Main_Service.yaml	push main branch
main.py	Update main.py
requirements.txt	update main requirements

Fundamentally, it is important that we have our “Dockerfile” uploaded on GitHub as this file contains all the necessary configurations that our applications need when they are deployed into production. Without this file, the following steps will not work. Now, we will have to select the “Actions” option circled in red, and this will bring us the to the page where we can create our CI/CD workflow as shown below.


Choose a workflow
...management, and issue triaging work the way you want. Select a workflow to get started.
Skip this and set up a workflow yourself →
Search workflows
Suggested for this repository
Docker image
By GitHub Actions
Build a Docker image to deploy, run, or push to a registry.
Configure
Dockerfile
Pylint
By GitHub Actions
Lint a Python application with pylint.
Configure
Python
C/C++ with Make
By GitHub Actions
Build and test a C/C++ project using Make.
Configure
C
Node.js
By GitHub Actions
Build and test a Node.js project with npm.
Jekyll
By GitHub Actions
Package a Jekyll site using the jekyll/builder Docker
Python application
By GitHub Actions
Create and test a Python application.

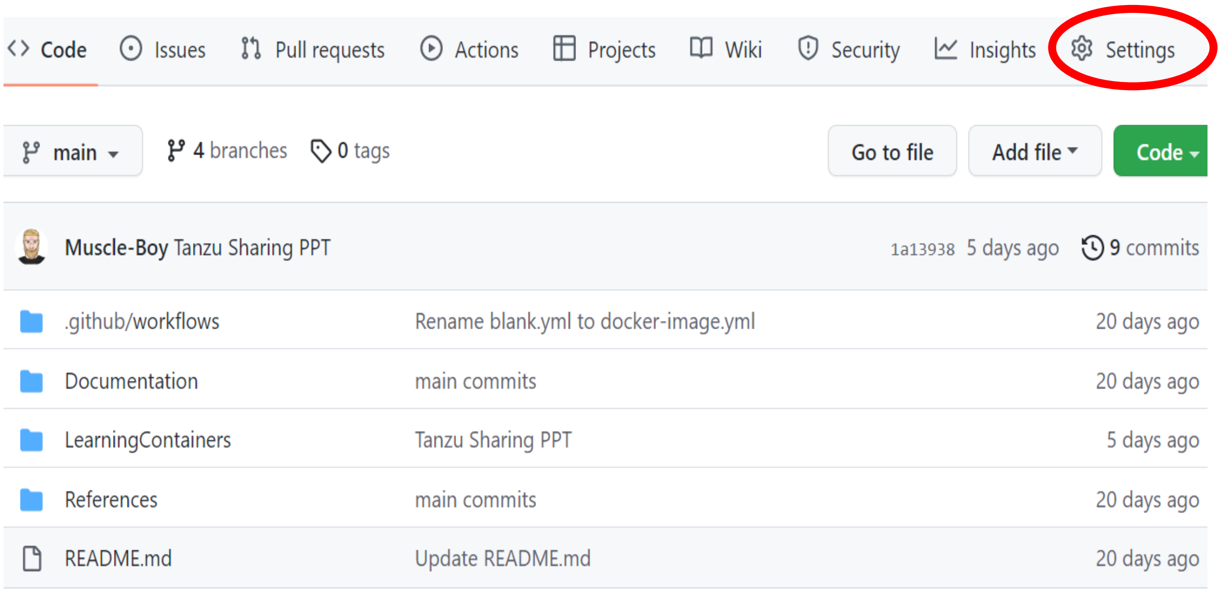
On this page, you can select from the many different “templates”, or YAML configuration files, already provided by GitHub. In our case, since we are looking to automate the building, and pushing, of Docker images, we will select the option circled in red.



```
23 lines (17 sloc)  |  479 Bytes
```

```yaml
name: Docker Image CI

on:
  push:
    branches: [ "Main_Branch, Disinformation_Branch" ]

jobs:

  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
      name: Check out code

    - uses: mr-smithers-excellent/docker-build-push@v5
      name: Build & push Docker image
      with:
        image: tegexplore/pda-2022
        registry: docker.io
        username: ${{ secrets.DOCKER_USERNAME }}
        password: ${{ secrets.DOCKER_PASSWORD }}
```

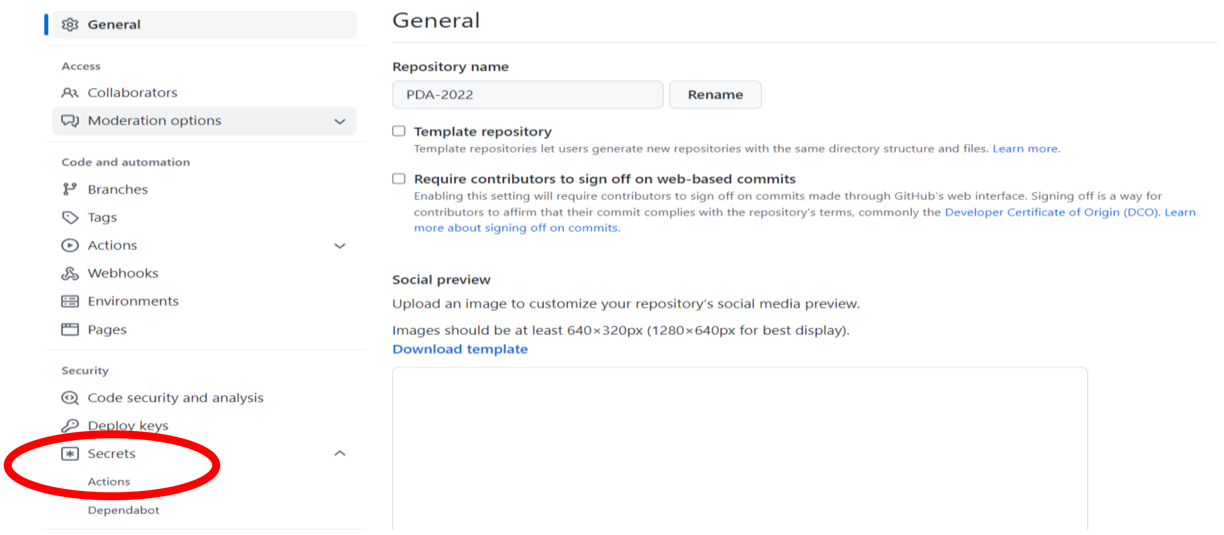
GitHub's CI/CD workflow comes in the form a YAML file. The above is a sample you can input to invoke the workflow. Pay special attention to the "image" parameter, as it will be specific to your private Docker Hub repository, and the "username" and "password" parameters, as they store your Docker Hub private repository's username and password. The values of the latter parameters, for security purposes, are recommended to only be referenced from GitHub "secret" files.



Muscle-Boy Tanzu Sharing PPT 1a13938 5 days ago 9 commits

.github/workflows	Rename blank.yml to docker-image.yml	20 days ago
Documentation	main commits	20 days ago
LearningContainers	Tanzu Sharing PPT	5 days ago
References	main commits	20 days ago
README.md	Update README.md	20 days ago

To create secret files, we have to select the "Settings" option, which will bring us to the page shown below:



General

Repository name

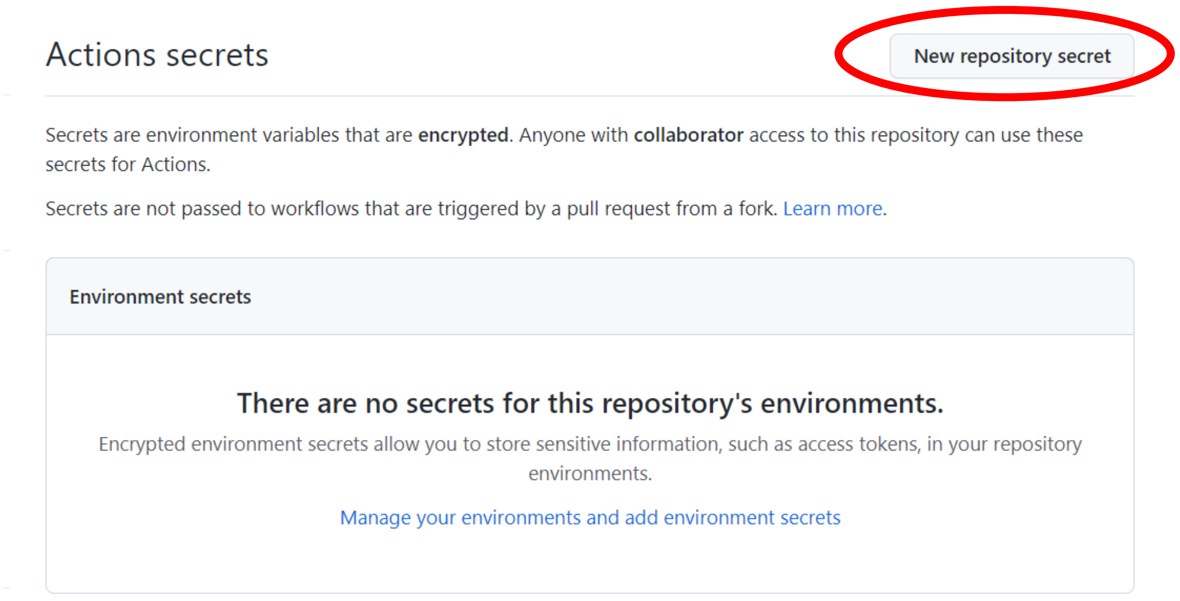
PDA-2022

Template repository
Template repositories let users generate new repositories with the same directory structure and files. Learn more.

Require contributors to sign off on web-based commits
Enabling this setting will require contributors to sign off on commits made through GitHub's web interface. Signing off is a way for contributors to affirm that their commit complies with the repository's terms, commonly the Developer Certificate of Origin (DCO). Learn more about signing off on commits.

Social preview
Upload an image to customize your repository's social media preview.
Images should be at least 640×320px (1280×640px for best display).

Next, we will have to look for the "Secrets" option, and select "Actions".


Actions secrets

Secrets are environment variables that are **encrypted**. Anyone with **collaborator** access to this repository can use these secrets for Actions.

Secrets are not passed to workflows that are triggered by a pull request from a fork. Learn more.

Environment secrets

There are no secrets for this repository's environments.

Encrypted environment secrets allow you to store sensitive information, such as access tokens, in your repository environments.

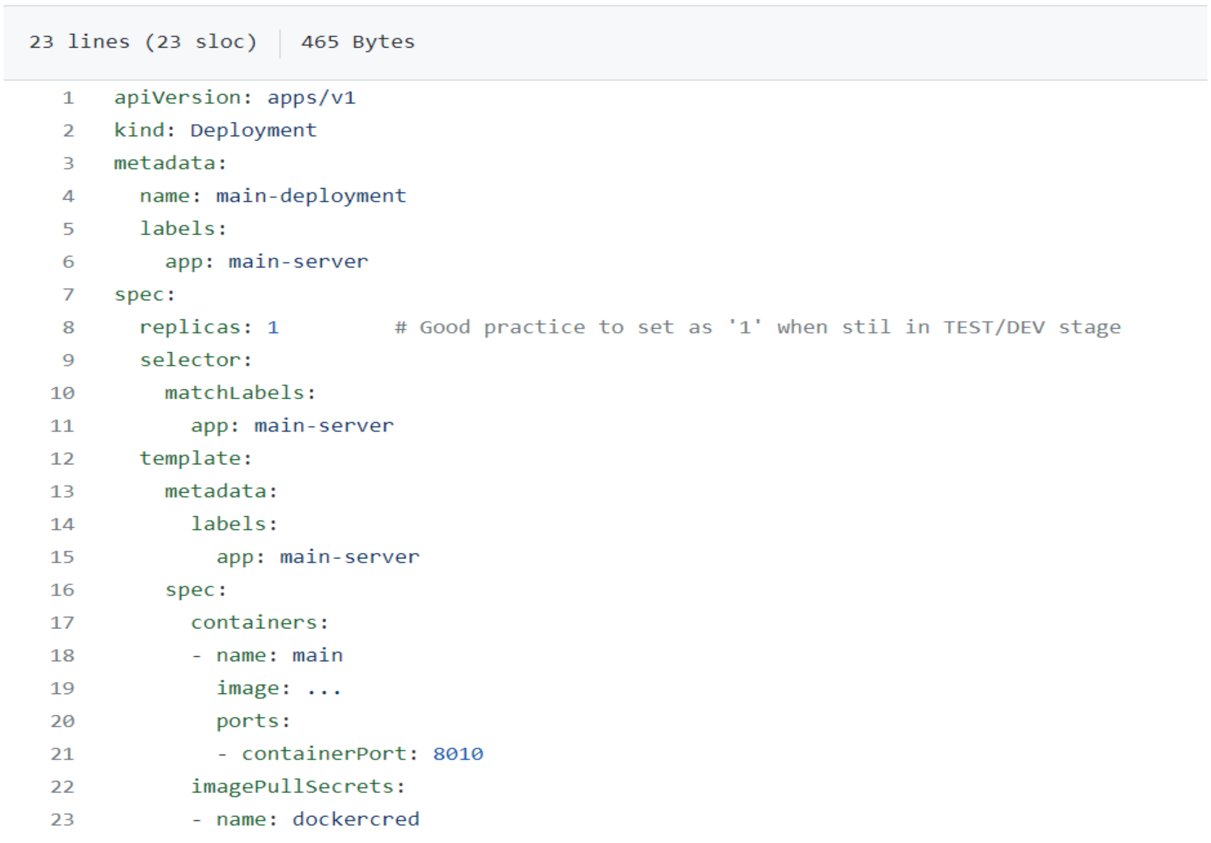
Manage your environments and add environment secrets

Finally, we will come the page above. All that is left is select the “New repository secret” option, and since GitHub secret file is key-value based, the keys should be inputted as “username” and “password” strings, and their actual values will be the values of the keys. With this, we have officially created our secret files, and in turn, our CI/CD workflow is successfully created, and should be working by now.

In summary, we will now have our docker images in our private Docker Hub repository, READY to be pulled by Minishift for deployment.

Kubernetes YAML Configuration File

For this section, we will briefly describe the parameters that make up a YAML configuration file to be used to create a Kubernetes resource, for instance, a deployment.



```
23 lines (23 sloc)  |  465 Bytes
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: main-deployment
  labels:
    app: main-server
spec:
  replicas: 1          # Good practice to set as '1' when stil in TEST/DEV stage
  selector:
    matchLabels:
      app: main-server
  template:
    metadata:
      labels:
        app: main-server
    spec:
      containers:
      - name: main
        image: ...
        ports:
        - containerPort: 8010
      imagePullSecrets:
      - name: dockercred
```

The above is a typical YAML file for deployment. While there are many lines of codes, the main things to pay attention to are:

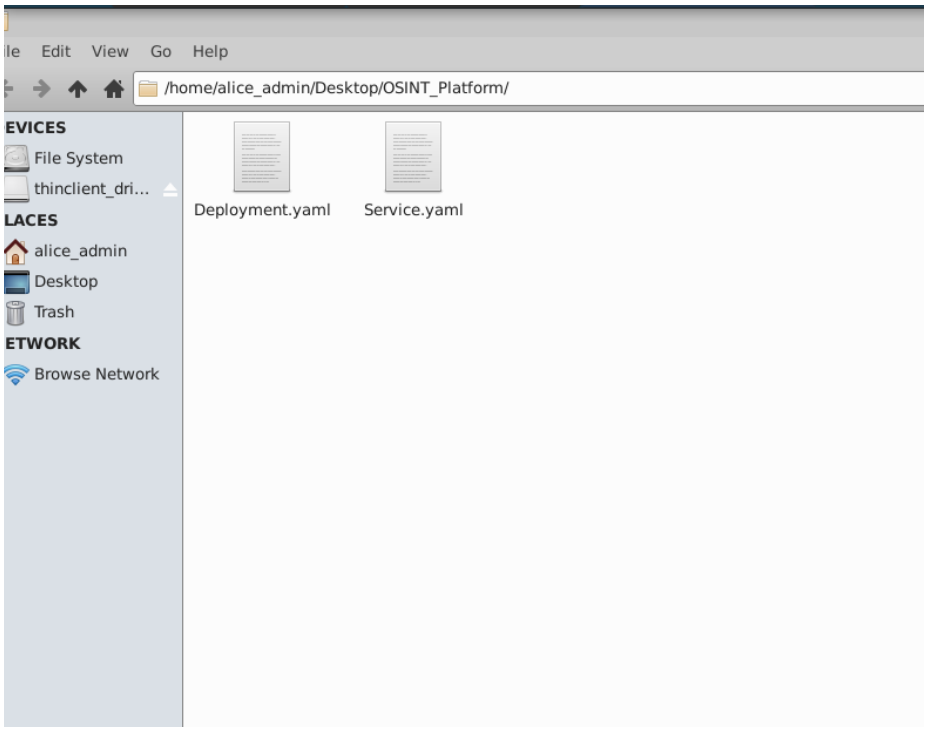
1) “kind” – the value of this parameter depends on the TYPE of Kubernetes resources we are intending to create. For our above example, since we are looking to create a deployment resource, we input “Deployment” as its value. Other types include “Service”, “Ingress”, and “Statefulset”.
2) “metadata” – The name of your resource, and its label. This information are mainly for the IDENTIFICATION of the Kubernetes resource for the Kubernetes API.
3) “replicas” – How many PODS you want to have for your application deployment. The more you have, the higher the availability of your application.
4) “image” – the Docker image NAME in your private Docker Hub repository. The format of the image should contain the registry of the private repository, and the image tag.

5) “imagePullSecrets” – an important parameter if you have created a secret resource to store the credentials of your private Docker Hub repository. With this secret resource, TCE will be AUTHENTICATED to access your private Docker Hub repository, and pull the images to containerize them. Without this, Minishift will NOT be able to access the private repository.

The above are the main parameters to look out for when creating a YAML file for Kubernetes resources.

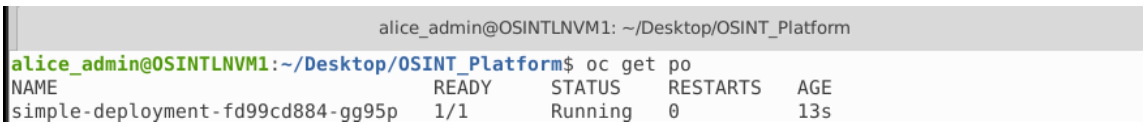
Deploying Application onto Minishift

Now that we have our YAML file(s) ready, we can carry out the actual deployment of our application. In this short section, we will look at deploying a very simple “Hello World” application. First and foremost, make sure you have all your YAML files (deployment + service +..) in your chosen directory, like in the example below:



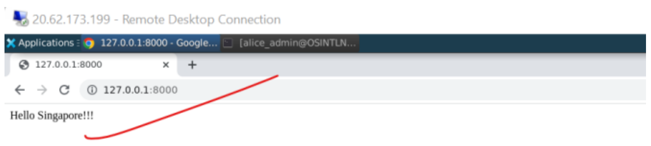
In our example, we will just have deployment and service YAML files. With these files present, we can run Minishift OC client to create the Kubernetes resources. It is important to know that Minishift, or OpenShift, is built on Kubernetes. Hence, most of Kubernetes own “kubectl” commands can be run on Minishift, just that we replace “kubectl” with “oc” in all our commands.

Traditionally, to create resources from our YAML manifests, we would execute the command “oc apply -f <filename.yaml>” for EACH YAML file. However, since we have all our YAML files (deployment + service) in one directory, we can switch our WORKING DIRECTORY to that directory by running linux “cd <directory path>”, and now that we are in the directory, we can run “oc apply -f .”. With this command, Minishift will deploy ALL the YAML files at once. To check if our deployment, and service, are deployed successfully, we can run “oc get pods” and “oc get svc”, as shown below.



```
alice_admin@OSINTLNVM1: ~/Desktop/OSINT_Platform
alice_admin@OSINTLNVM1:~/Desktop/OSINT_Platform$ oc get po
NAME                                READY     STATUS    RESTARTS   AGE
simple-deployment-fd99cd884-gg95p   1/1       Running   0          13s
```

From the above, we can see that our deployment, and, in turn, pod is running. Now, we can either head over to, or curl, the webpage to see if our application is in fact working.



```
20.62.173.199 - Remote Desktop Connection
Applications   127.0.0.1:8000 - Google...   [alice_admin@OSINTLN...
127.0.0.1:8000
127.0.0.1:8000
Hello Singapore!!!
```

Indeed, our application is running as we can see the html text “Hello Singapore!” displayed on the webpage.