Kube Docs

Release 0.0.1

Shaun Cullen

Sep 24, 2020

Table of Contents

Chapter 1

Docker

Containers are the basic building block for machine independent cloud based computing. Kubernetes is a container orchestrator, allowing your application and its dependencies to be deployed independent of any given machine or architecture. Docker is the most common method for creating containers. We will go over the most useful methods for creating a Docker container for your application, which we will eventually be deploying to a Kubernetes cluster.

1.1 Dockerfile: Build Your Container Image

Dockerfiles are used to declaratively create images for Docker containers, which can be integrated into source control. This is a common theme among the dev ops topics we will be covering: finding ways to integrate configuration and deployment into the source control of your projects, so that they can be tracked and modified over time.

Example Dockerfile

```
FROM registry.cullen.io/ubuntu-base:v0.0.8

RUN apt update && apt install -y nginx

ARG GITHUB_PAT

COPY ./requirements.txt /code/requirements.txt

WORKDIR /code

RUN python3.7 -m pip install -r /code/requirements.txt

RUN python3.7 -m pip install git+https://sscullen:$GITHUB_PAT@\
github.com/sscullen/landsat_downloader.git@v0.0.4#egg=landsat_downloader
RUN python3.7 -m pip install git+https://sscullen:$GITHUB_PAT@\
github.com/sscullen/sentinel_downloader.git#egg=sentinel_downloader
RUN python3.7 -m pip install git+https://sscullen:$GITHUB_PAT@\
github.com/sscullen/spatial_ops.git@v0.0.2#egg=spatial_ops

COPY . /code

RUN cp -r /code/common/grid_files/ /usr/local/lib/python3.7/dist-packages/spatial_ops
 && \
    cp -r /code/common/data/ /usr/local/lib/python3.7/dist-packages/spatial_ops
```

```
RUN python3.7 /code/manage.py collectstatic --no-input

COPY nginx_site.txt /etc/nginx/sites-available/jobmanager

RUN ln -s /etc/nginx/sites-available/jobmanager /etc/nginx/sites-enabled/jobmanager

EXPOSE 80

EXPOSE 443

EXPOSE 5000

CMD ["gunicorn", "--bind", "0.0.0.0:5000", "--log-level=debug", "jobmanager.wsgi"]
```

Dockerfile Keywords

FROM

Source image to use as a base. The foundation for your image. Often is images of popular linux distros but can also be images you have customized with other Dockerfiles, allowing for images with common requirements to be shared among many docker files. In the example above, `ubuntu-base` has Python3 and gdal already installed.

Note Notice the URL for the docker image: this is how images are referenced and tagged for access from different docker registries, including the main docker hub. This is discussed more below in the *private registry* section.

RUN

Run a command inside the container. Each `RUN` command will add another "layer" (discrete change to the image that is tracked and transferred individually), and many layers will make the image large. `RUN` commands will often be chained with `&&` to reduce the image size.

Note You can use the experimental `--squash` feature which reduces the amount of layers in an image regardless of how many `RUN` commands there are.

ARG

Get environment variables from your machine into the image you are building. In the example this is used for Github authentication so no login or password prompts are needed. In the *build your image* section we go over the different syntaxes for Linux and Windows for using your environment variables in the image build command.

COPY

Move your source files and data from your local machine into the docker container image.

WORKDIR

Set the `root` directory for the Dockerfile, where all the `RUN` commands will be executed and where the `COPY` commands will put the files they copy from your machine.

EXPOSE

Expose the ports your application will be accessible on. For example, for an nginx docker image that is hosting web services, you will expose 443 and 80. For a Django application in development, you could use `django manage.py runserver 0.0.0.0:4000` to start your django instance, and in this case you would expose port 4000 to access your Django application.

CMD

The command your docker container will run when it starts. For a simple Python webservice, your command would be starting the webserver application such as `nginx` or `gunicorn`. If you don't

want your container to run a command here, you can use `['sleep', 'infinity']` so the container will wait for you to interact with it before shutting down. Once the `CMD` command finishes, the container shuts down.

1.1.1 Installing `pip` Packages Directly from Github

In the example Dockerfile above, we have lines that look like this:

```
RUN python3.7 -m pip install git+https://sscullen:$GITHUB_PAT@\
github.com/sscullen/landsat_downloader.git@v0.0.4#egg=landsat_downloader
```

This allows us to install pip packages directly from Github. We can specify a specific branch and tag, allowing us to make sure we are using the right package. To avoid credential issues, we use a Github Personal Access token, which can be generated from the Github general settings page for your account. This is not required for a public repo.

To be a valid pip package, the repo should be structured like so:

```
repo_root/
    landsat_downloader # <-- actual python module source code
    .gitignore
    Pipfile # <-- requirements.txt equivalent for Pipenv virtual env manager
    Readme.md
    requirements.txt
    setup.py # <-- critical file for the pip package
```

The `setup.py` file is required, and contains metadata and package information. The version number in `setup.py` should match the latest release tag for your repo.

Example `setup.py`

```
from setuptools import setup

setup(name='landsat_downloader',
    version='0.0.4',
    description='Utilities for downloading Landsat and Sentinel products from USGS',
    url='https://github.com/sscullen/landsat_downloader.git',
    author='Shaun Cullen',
    author_email='ss.cullen@uleth.ca',
    license='MIT',
    packages=['landsat_downloader'],
    zip_safe=False)
```

Pip will use `setup.py` to build the `.whl` package from your repo directly from your source code in the repo. This is important because it makes sure the version of the code we are using is up to date and correct. If we aren't doing active development on a project and only installing it as a dependency, then it is best to install in this way, rather than copying source code using a `COPY` command. If we are actively changing the code on our local machine, the `COPY` command makes more sense.

1.1.2 Exposing Your Service's Ports

Using the `EXPOSE` directive is how you make your docker container service accessible to the outside world. This is also a critical concept in how Kubernetes exposes services from containers to the outside world. While it is possible to mount local volumes and interact directly inside your container, thus bypassing the "service" model employed by exposing ports for APIs and such, it is not a good idea to use Docker containers in this way. Even modest containers should have some sort of network service exposed for interacting with your application.

Live interaction and intervention by a user in your container results in ephemeral changes to your

container that are not maintained after the container restarts. In this vein, we should strive to make containers as "stateless" as possible, and move all data "persistence" requirements into dedicated data focused containers and services, which are designed with persistence in mind. More on this will be in the *Kubernetes section* on persistence.

1.1.3 Mounting Volumes Inside the Container

1.1.4 Local Development Inside the Container

1.2 Build Your Image

1.3 Docker Private Registry

1.4 Working With Images and Running Containers

1.5 Common Docker CLI Command Reference

Optional Sidebar Title

Subsequent indented lines comprise the body of the sidebar, and are interpreted as body elements.

term 1
: Definition 1.

term 2
: Definition 2, paragraph 1.
Definition 2, paragraph 2.

term 3 : *classifier*
: Definition 3.

term 4 : *classifier one* **:** *classifier two*
: Definition 4.

Watch out for this:
: asdasdfasdf asdasd
aasdflkjasldfkj

But also this
: what the what
what what

This is an ordinary paragraph, introducing a block quote.

> "It is my business to know things. That is my trade."
>
> —Sherlock Holmes

```
def my_function():
```

```
    "just a test"
    print 8/2
```

Chapter 2

Kubernetes

Kubernetes is a container orchestration tool for deploying containers across many different physical machines organized into a cluster. We will cover only the very basics of getting a container you created up and running on Kubernetes.

2.1 Pods and Containers

2.2 Services

2.3 Ingress

2.4 Deployments

2.5 Configmaps

2.6 Secrets

2.7 Persistent Volumes and Claims

2.7.1 Storage Classes

2.7.2 NFS

2.7.3 OpenEBS

2.8 Namespaces

2.9 kubectl: Kubernetes Command Line Interface

2.10 Helm: Deployment Manager

2.11 Helm Charts

Chapter 3

Example 1 - Simple - Agricarta Preprocessing

The first example is as simple as you can get for running an application on Kubernetes. We will have one container, one service, connected to a simple wrapper Flask HTTP wrapper around our application.

3.1 Summary

3.2 Architecture

3.3 Prerequisites

3.4 Build the Container

3.5 Service

3.6 Configmaps

3.7 Ingress

3.8 API

3.9 Logs

3.10 Data Management and Access

Chapter 4

Example 2 - Simple - Landsat Downloader Service

The first example is as simple as you can get for running an application on Kubernetes. We will have one container, one service, connected to a simple wrapper Flask HTTP wrapper around our application.

4.1 Summary

4.2 Architecture

4.3 Prerequisites

4.4 Build the Container

4.5 Service

4.6 Configmaps

4.7 Ingress

4.8 API

4.9 Logs

4.10 Data Management and Access

Chapter 5

Example 3 - Complex - Datacube

This example is more complex as we have multiple containers with a database dependency, so for this example we will also integrate a helm chart.

5.1 Summary

5.2 Architecture

5.3 Prerequisites

5.4 Container

5.5 Service

5.6 Configmaps

5.7 Ingress

5.8 API

5.9 Logs

5.10 Data Management and Access

5.11 Helm Chart

Chapter 6

Example 4 - Complex - Tile Viewer API

This example brings together all of the aspects from the previous example and then adds more complexities. There are multiple pods, pods with multiple containers, and multiple service dependencies that our application needs to function.

6.1 Summary

6.2 Architecture

6.3 Prerequisites

6.4 Container

6.5 Service

6.6 Ingress

6.7 Configmaps

6.8 API

6.9 Logs

6.10 Data Management and Access

6.11 Helm Chart

6.12 Future Improvements

6.12.1 Horizontal Pod Autoscaling

6.12.2 Celery Downloader Pods as Jobs Instead of Deployments

- *Index*
- *Module Index*
- *Search Page*