Deploying and Managing Python with Kubernetes

Joannah Nanjekye

About Me

- Software Engineer
- Aeronautical Engineer to be!
- Open Sourcerer
- Author
- Upcoming UNB Grad to work on some pypy stuff <> IBM

Agenda!!

- Containers
- Their Orchestration with Kubernetes!
- Managing a cluster with Python.

Rules

- We may not have Q and A.

</nanjekyejoannah@gmail.com>

System Admins

Meanwhile Developers

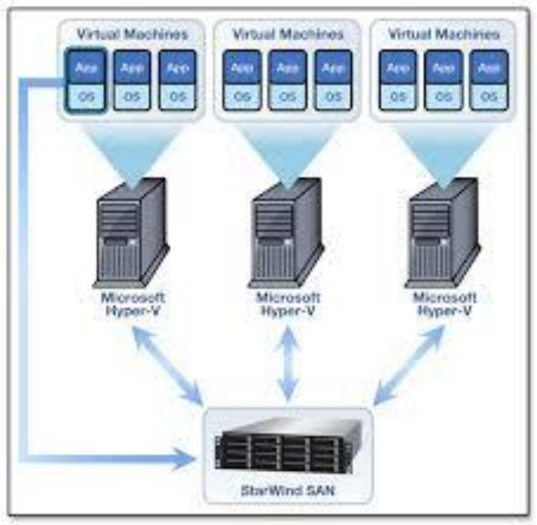
Deploying Python Apps

- Physical Machines
- Virtual Machines
- Containers

Physical Machines

Typically one application per host

Virtual Machines

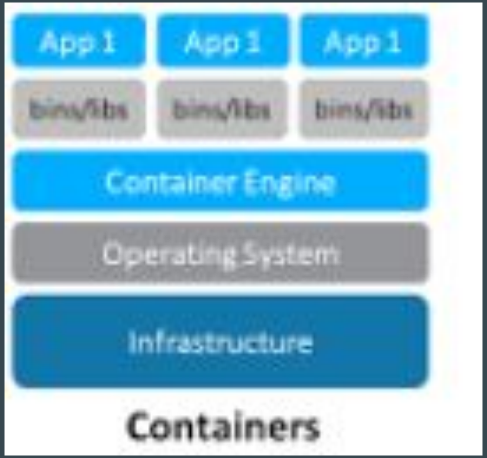


- Virtualize the hardware.
- Run multiple applications on same hardware.

Containers

- Logical Packaging for applications.
- To run abstracted from the environment.

Virtualize the OS

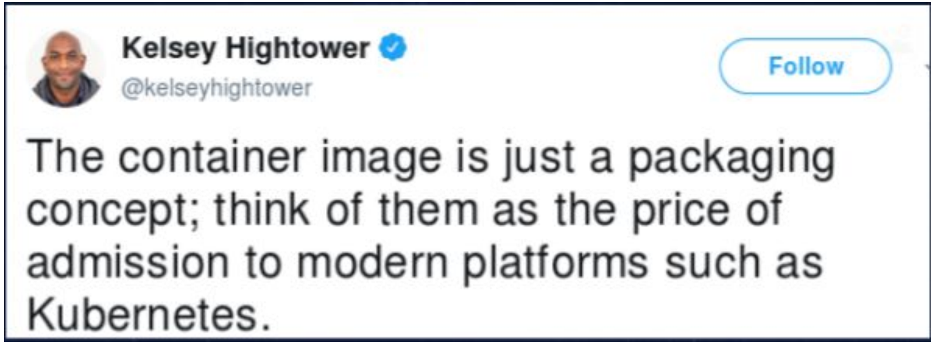


- Run a container runtime on host operating system.
- Better performance due to absence of guest operating systems

Why do we care

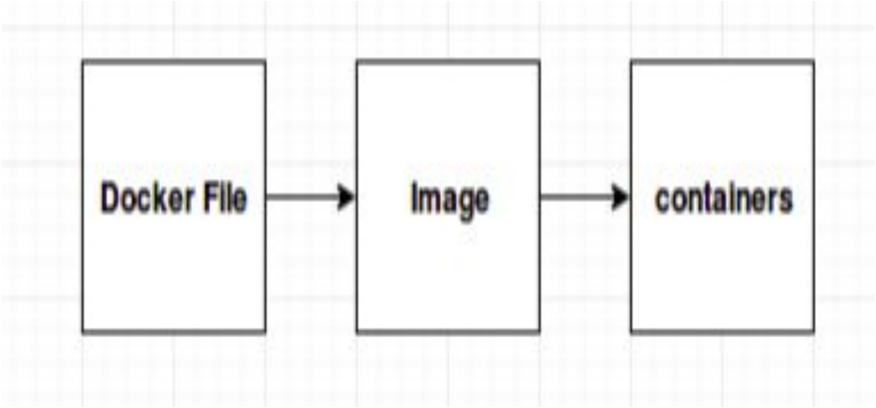
- Productivity
- Portability
- Scaling

Ushering us to robust Platforms



The container image is just a packaging concept; think of them as the price of admission to modern platforms such as Kubernetes.

Containerization



For a simple app

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
        return 'Hello  World!'

if __name__ == '__main__':
        app.run(host='0.0.0.0')
```

Dockerfile

```
FROM gliderlabs/alpine:3.2

RUN apk-install python

COPY requirements.txt .
RUN apk --update add --virtual
build-dependencies py-pip \
  && pip install -r requirements.txt \
  && apk del build-dependencies

COPY . /code
WORKDIR /code

EXPOSE 5000
ENTRYPOINT ["python", "myapp.py"]
```

Build and Run

```
docker build -t docker-sinatra .

docker run -p 4000:80 docker-sinatra
```

That's not all Folks!

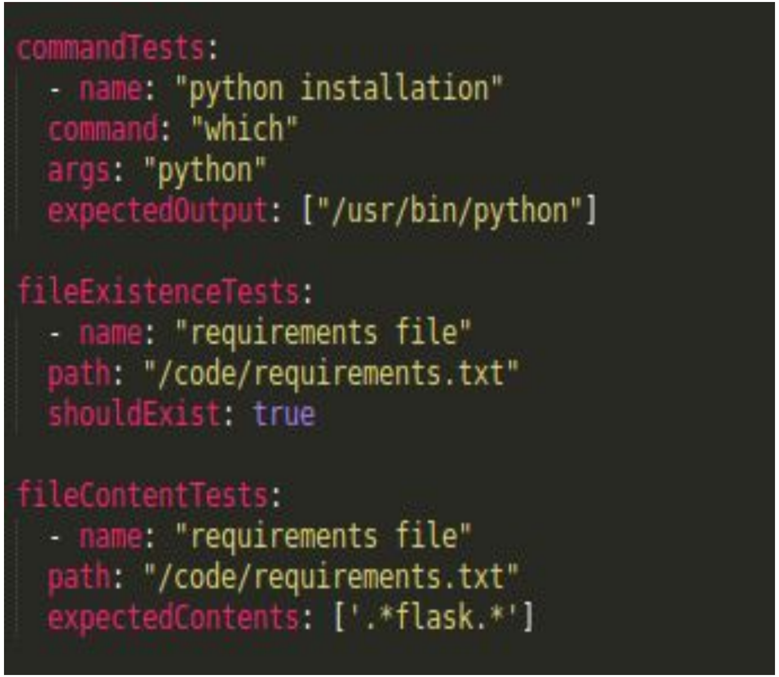
Best Practices

- Create small images
  - Performance
  - Security
- Unit test container images

Unit Test Images

With The Google Container Structure Test Framework

#config.yaml



```yaml
commandTests:
  - name: "python installation"
  command: "which"
  args: "python"
  expectedOutput: ["/usr/bin/python"]

fileExistenceTests:
  - name: "requirements file"
  path: "/code/requirements.txt"
  shouldExist: true

fileContentTests:
  - name: "requirements file"
  path: "/code/requirements.txt"
  expectedContents: ['.*flask.*']
```

Summary

- Containers are a packaging for applications.
- Giving us portability, increased productivity and scaling.
- Create a container by creating a Dockerfile, build the image using instructions from this file, and run it to start the container.

The last mile!

kubernetes
Google
Manage a cluster of Linux containers as a single system to accelerate Dev and simplify Ops.

- What happens when we have many containers?
- How do we monitor them?
- What about Scaling ?

Kubernetes comes to the rescue!

To help manage;

- Deployments
- Monitoring
- Scaling

Deployment

- Specify an image from which to create a container.
- Deployment criteria i.e RAM, CPU, storage

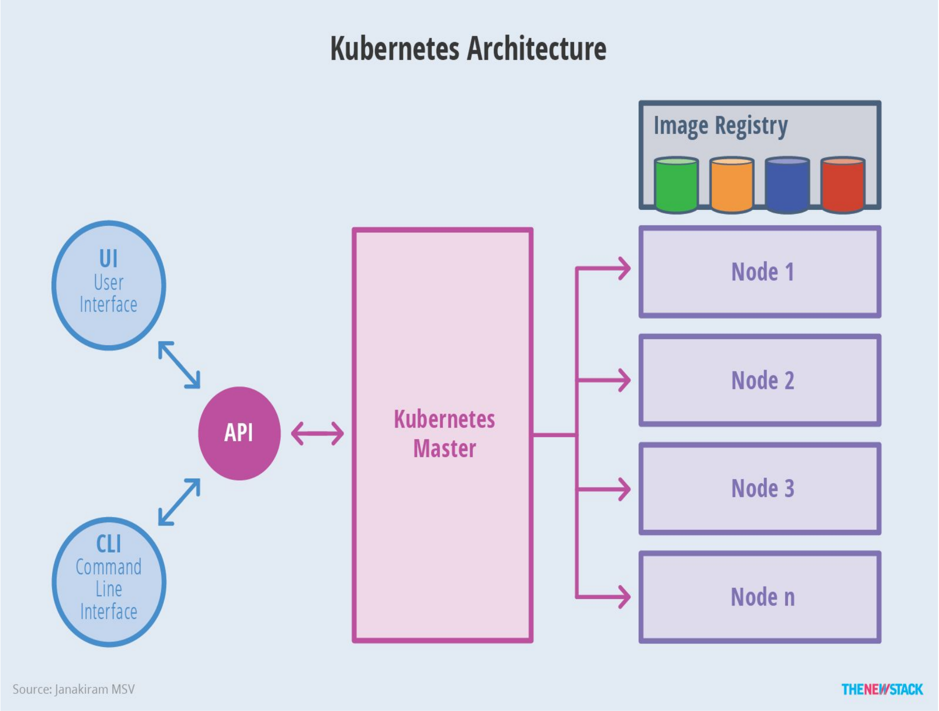
Monitoring

- Health check
- Autorecovery

Scaling

- Cluster scaling.
- Horizontal pod scaling.

Kubernetes Architecture

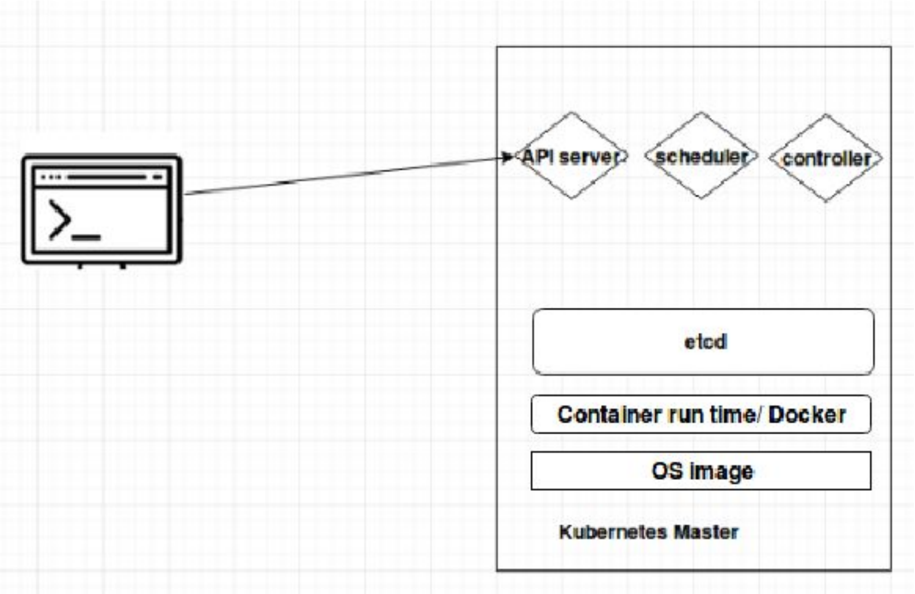


Source: Janakiram MSV

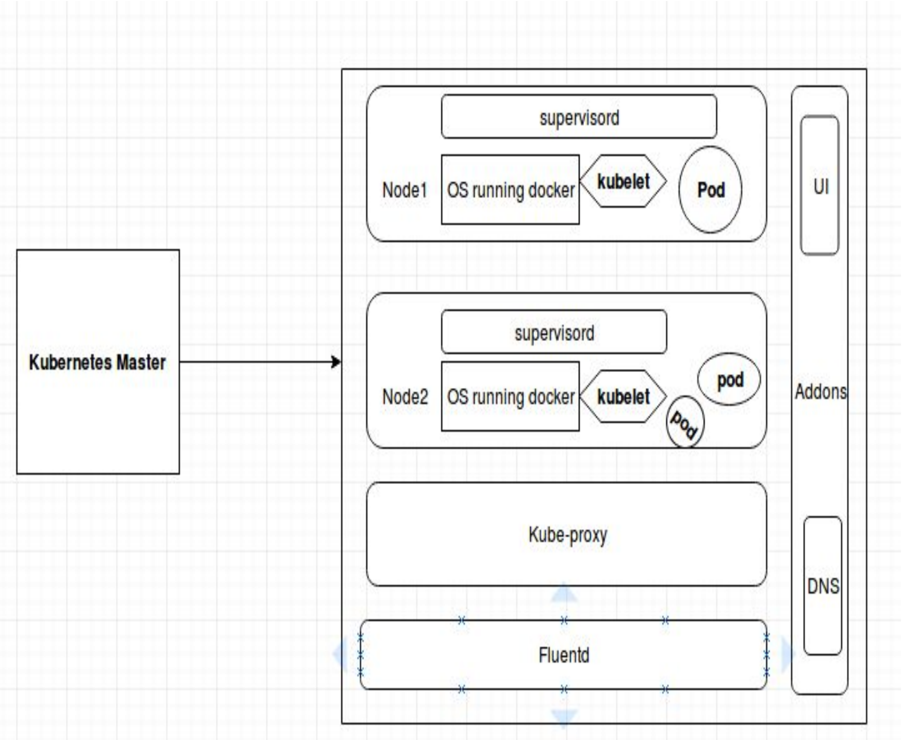
Client

Makes API calls to the master

Master


API server
scheduler
controller
etcd
Container run time/ Docker
OS image
Kubernetes Master

Nodes


Kubernetes Master
supervisord
Node1
OS running docker
kubelet
Pod
supervisord
Node2
OS running docker
kubelet
pod
pod
Kube-proxy
Fluentd
UI
Addons
DNS

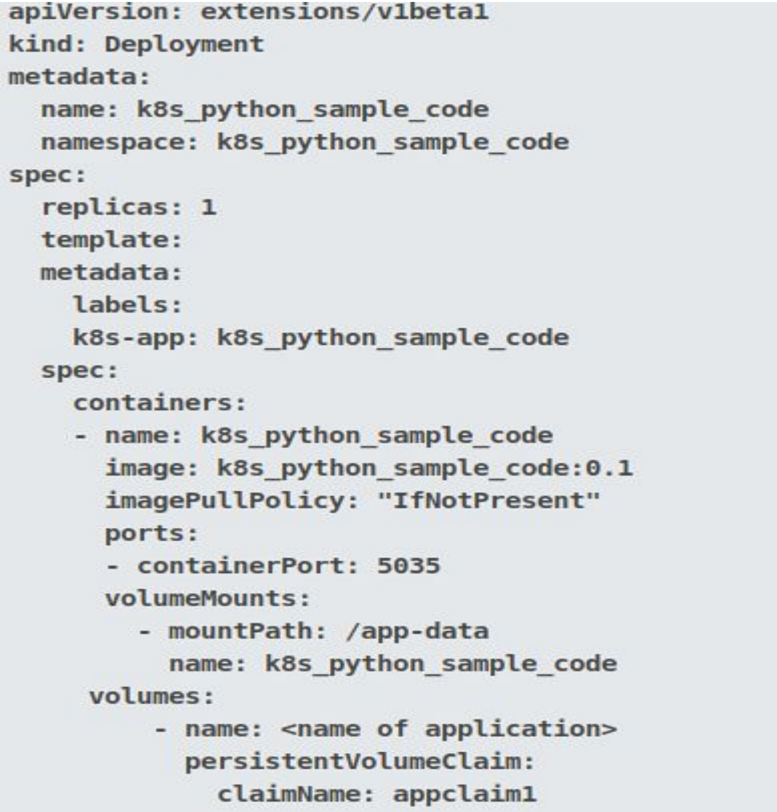
Deploying Python App to Kubernetes

Key Requirements

- A container image for your application.
- A deployment.
- A service to expose your deployment

Kubernetes Deployment

Using a .yaml file



```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: k8s_python_sample_code
  namespace: k8s_python_sample_code
spec:
  replicas: 1
  template:
  metadata:
    labels:
    k8s-app: k8s_python_sample_code
  spec:
    containers:
    - name: k8s_python_sample_code
      image: k8s_python_sample_code:0.1
      imagePullPolicy: "IfNotPresent"
      ports:
      - containerPort: 5035
      volumeMounts:
        - mountPath: /app-data
          name: k8s_python_sample_code
     volumes:
         - name: <name of application>
           persistentVolumeClaim:
             claimName: appclaim1
```

kubectl create dep.yaml

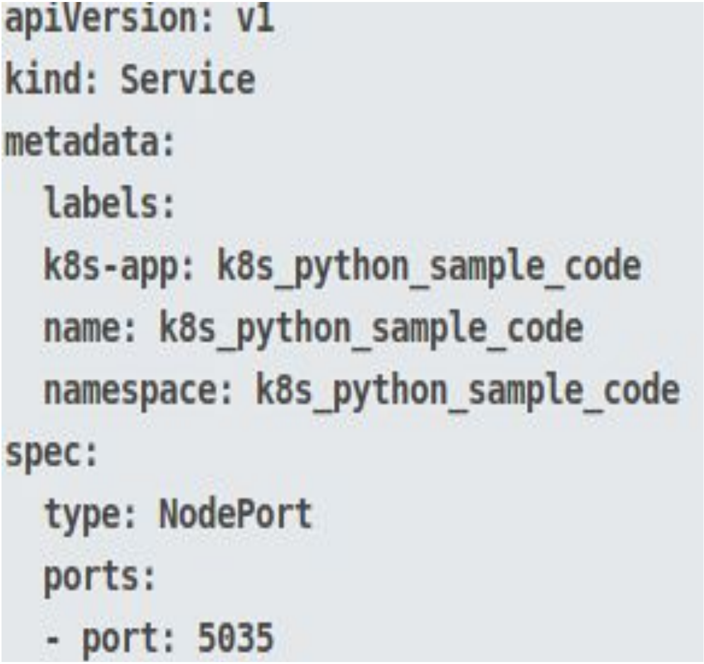
Kubernetes Deployment

Using kubectl

```
kubectl run kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1 --port=8080
```

Kubernetes service

Using a .yaml file

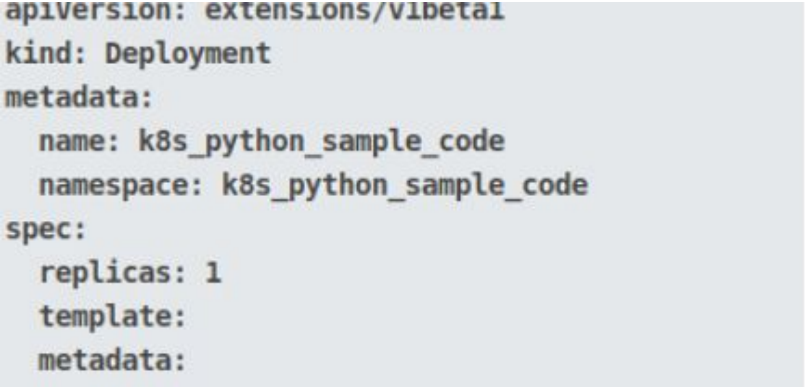


```
apiVersion: v1
kind: Service
metadata:
  labels:
  k8s-app: k8s_python_sample_code
  name: k8s_python_sample_code
  namespace: k8s_python_sample_code
spec:
  type: NodePort
  ports:
  - port: 5035
```

kubectl create service.yaml

Scaling

Increase number of replica sets in yaml or set in kubectl command.



```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: k8s_python_sample_code
  namespace: k8s_python_sample_code
spec:
  replicas: 1
  template:
  metadata:
```

AutoScaling

- Cluster scaling
- Horizontal pod scaling

Rolling Updates

For zero upgrade downtimes

Pass a new image name and tag with the --image flag and (optionally) a new controller name

```
kubectl rolling-update NAME [NEW_NAME]
--image=IMAGE:TAG
```

Summary

Kubernetes gives us;

- Scaling
- Monitoring
- Zero upgrade down times

Kubernetes Python Client

Managing a cluster using python

Write python code to access your cluster.

For Inspiration

- Watch Abby Fuller’s talk at Dockercon 2017

Thank you

@Captain-joannah

nanjekyejoannah

Thank you