Python and Kubernetes: A match made in the cloud

Hi! I'm Jamon Camisso - jamon@digitalocean.com

- I'm a Developer Educator @ Digitalocean
- Previously an SRE/devops/sysadmin @ Canonical
- Work on Community Team designing curricula, ebooks, writing tutorials
- Teach "Intro To Sysadmin" @ YorkU

The DigitalOcean Community Team

- Community Hub
- Tutorials (Writing & Editing)
- Events, Meetups
- Support open source projects and developers
- Hacktoberfest

Workshop goals

- Get (more) familiar with containers
- Deploy and access a Kubernetes cluster
- Learn about Kubernetes architecture & objects
- Build a Docker image for a demo Flask app
- Deploy Flask app to Kubernetes cluster
- By the end (hopefully), have a load-balanced Flask app with session handling, healthcheck endpoint, and an external PostgreSQL database

Prerequisites

- K8s cluster:
- On your machine:
  - Kubectl
  - Git
  - Flask demo repo code
  - Docker
- Or: ssh lab@instructor.itec2210.ca
  - Password: `limoncelli` with no backticks
  - Type “tmux attach -r”

Lets create a Kubernetes cluster

- https://cloud.digitalocean.com/kubernetes/clusters/new

- Or build your own with Kubeadm & Ansible:

- https://www.digitalocean.com/community/tutorials/how-to-create-a-kubernetes-cluster-using-kubeadm-on-ubuntu-18-04

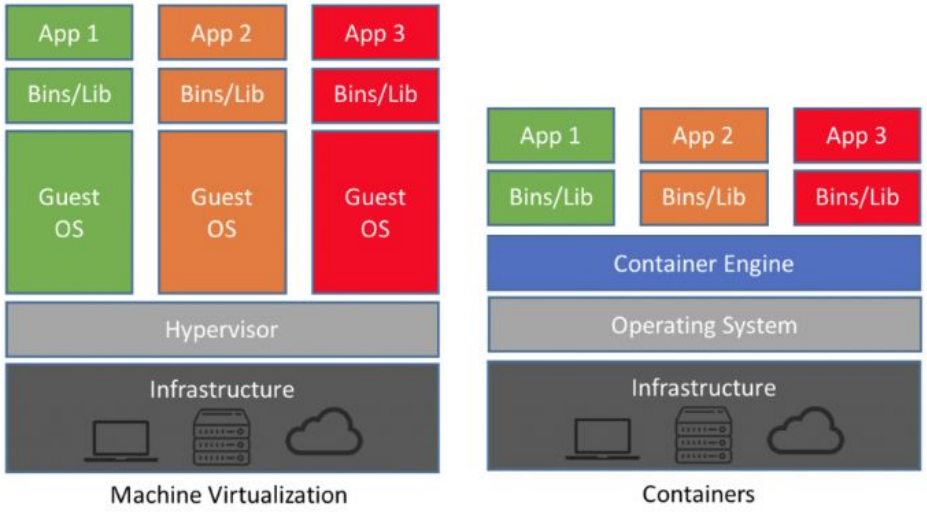
Prerequisites

- https://do.co/jamon
- Add your email to be invited to the workshop team account
- If you have your own k8s cluster then you can skip this
- Or skip if you'd prefer, I've setup a shared cluster for us

Containers

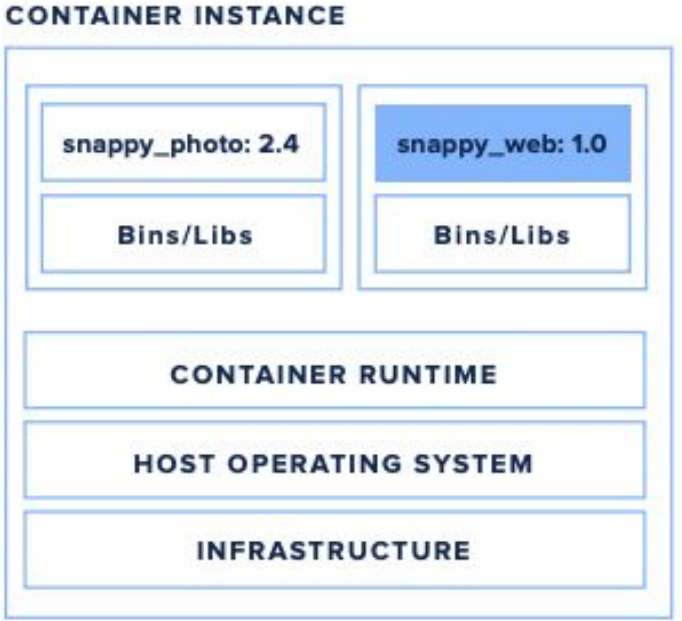
Containers

- What is a Container?
  - Namespaces: Process isolation
  - Cgroups: Resource limits
- Why are they important?
- VMs vs. Containers
  - Aside: VPS
- Key advantages
  - Lightweight
  - Portable
  - Isolated



Container Ecosystem

- Container Runtime
- Images
- Containers
- Image Registries



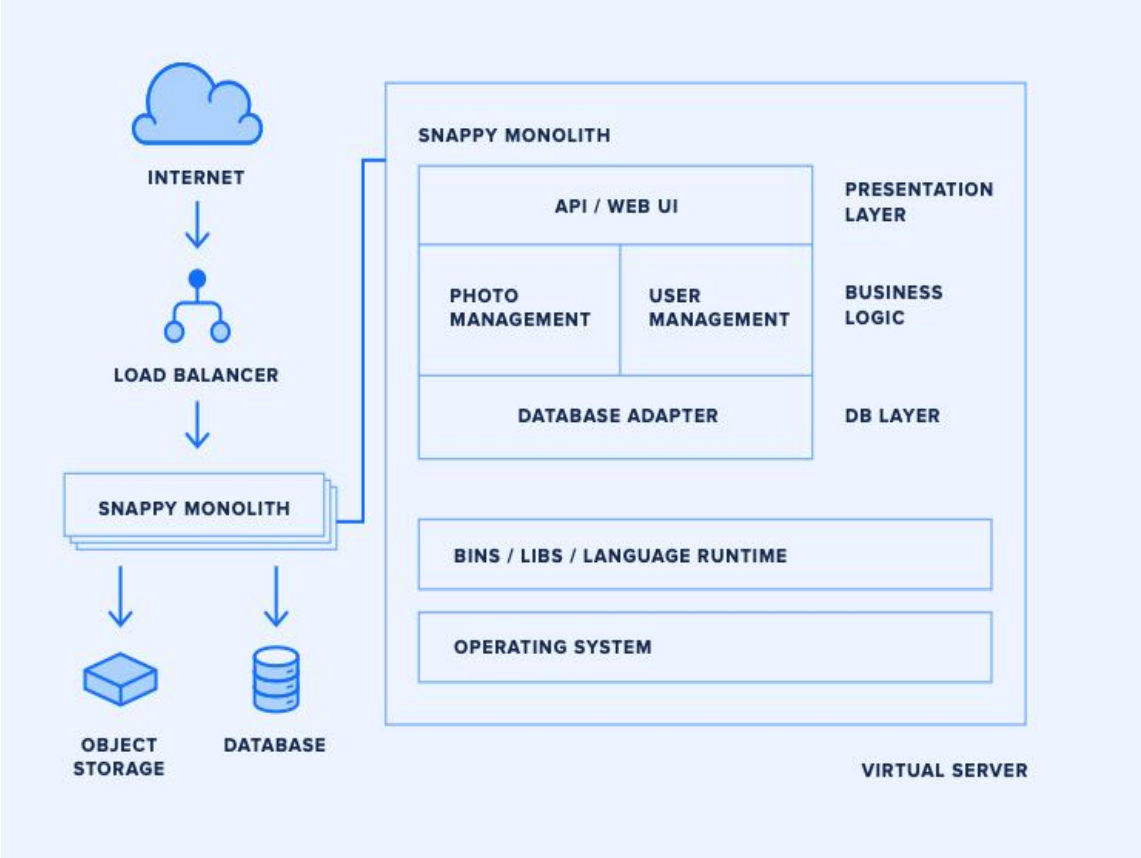
cri-o

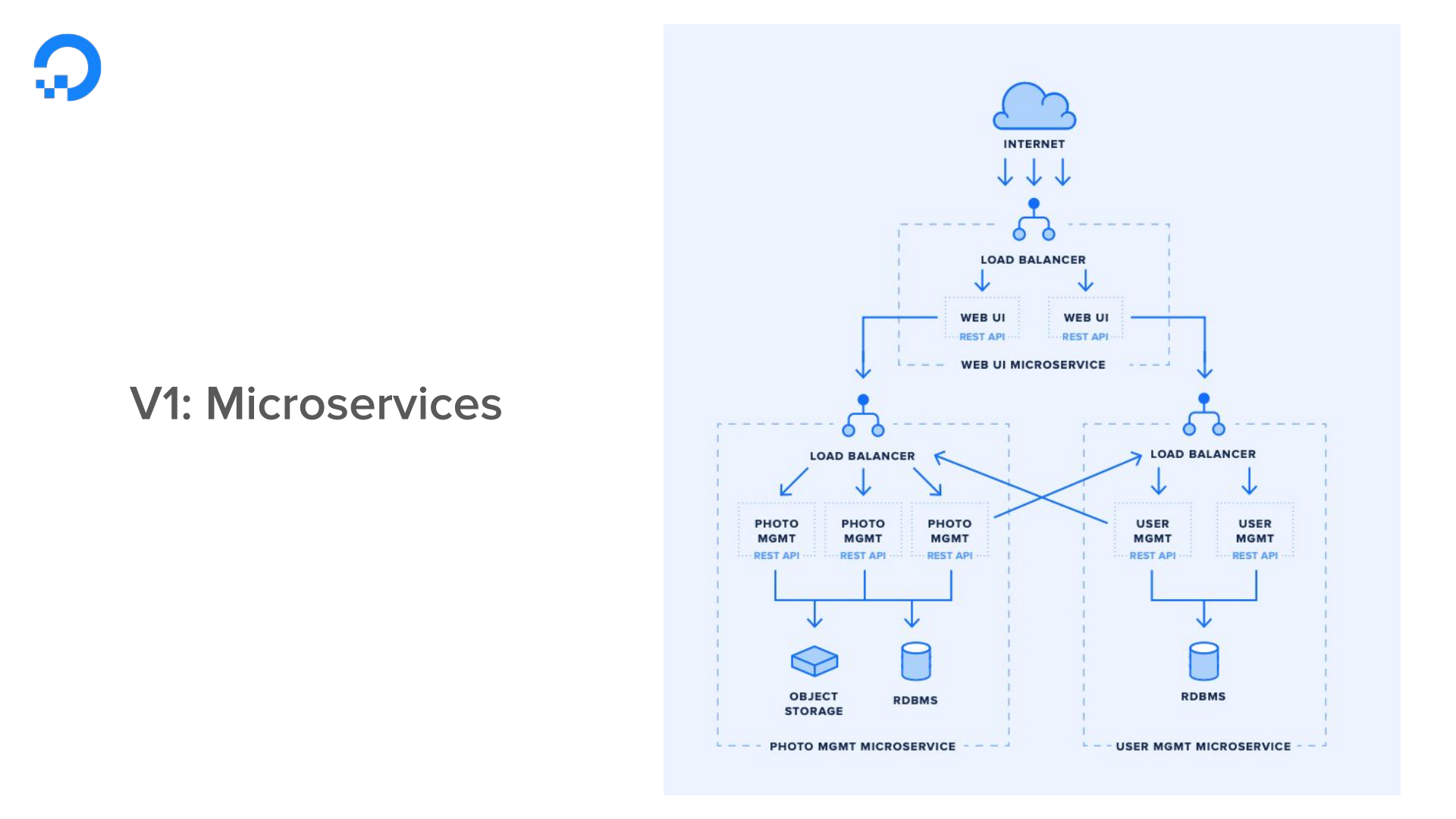
Containers

- Create a container from scratch
  - sudo unshare --fork --pid --mount-proc /bin/bash
- Find process id (outside container):
  - pgrep -af /bin/bash
  - pid=<pid of /bin/bash process under unshare parent>
- Enter into the container
  - sudo nsenter -p -m -t $pid
  - ps -ef inside and outside the container

Aside: App Modernization & Microservices

V0: The Monolith





V1: Microservices

Back to Containers...

Containers

- Are they right for me?

- Spoiler: probably, eventually, yes
  - Hard to scale the monolith
  - Hard to refactor the monolith
  - Hard to instrument & monitor all parts of the monolith
  - Continuous integration tests get longer and longer
  - No point running entire test suite for a CSS change
  - And again, containers are light, portable, isolated

Clone the workshop code

- SSH to the shared droplet
  - cd workshop/v0/app

- Or use your computer
  - git clone https://github.com/do-community/k8s-intro-meetup-kit workshop
  - cd workshop/v0/app
  - view app.py
  - view Dockerfile

So Let's Containerize a Flask App

App Code

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

@app.route('/')
def hello_world():
    return 'Hello, World! From {}'.format(gethostname())

if __name__ == "__main__":
    app.run(debug=True,host='0.0.0.0')
```

So Let's Containerize a Flask App

Dockerfile

```
FROM python:3-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["python", "app.py"]
```

- Build & tag image
  - Layers
- Run container / test
- Push to registry (Dockerhub)
- What would this look like in VM world?

Run the workshop code

- Shared droplet has a venv
  - ssh lab@lish.jamon.ca
  - tmux attach
- This should “just work”
  - python app.py
  - curl localhost:5000
- Or docker:
  - docker run --rm --net=host jamonation/flask-helloworld:v0
  - curl instructor.itec2210.ca:5000

Container Clusters

Container Clusters

- What if we have 10s, 100s, 1000s of running containers on multiple VMs?
- How to deploy, scale, restart, manage all of these containers?
- What problems do they solve?
    - Management
        - Metrics
        - Health checks
        - Security
    - Abstraction of hardware
    - Networking
    - Scheduling
    - Scaling
    - Deployment
        - Rollbacks
        - Zero-downtime / blue-green
    - Service discovery

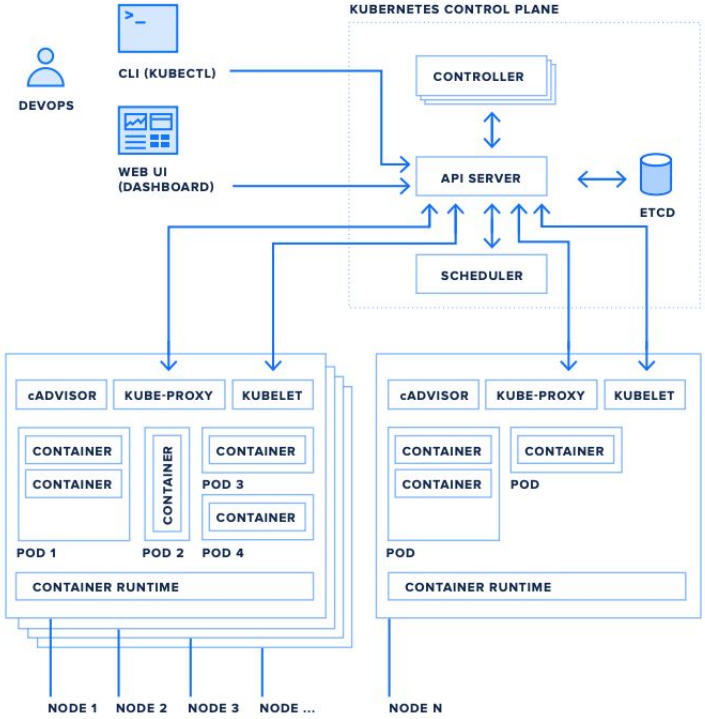
A Brief K8S History

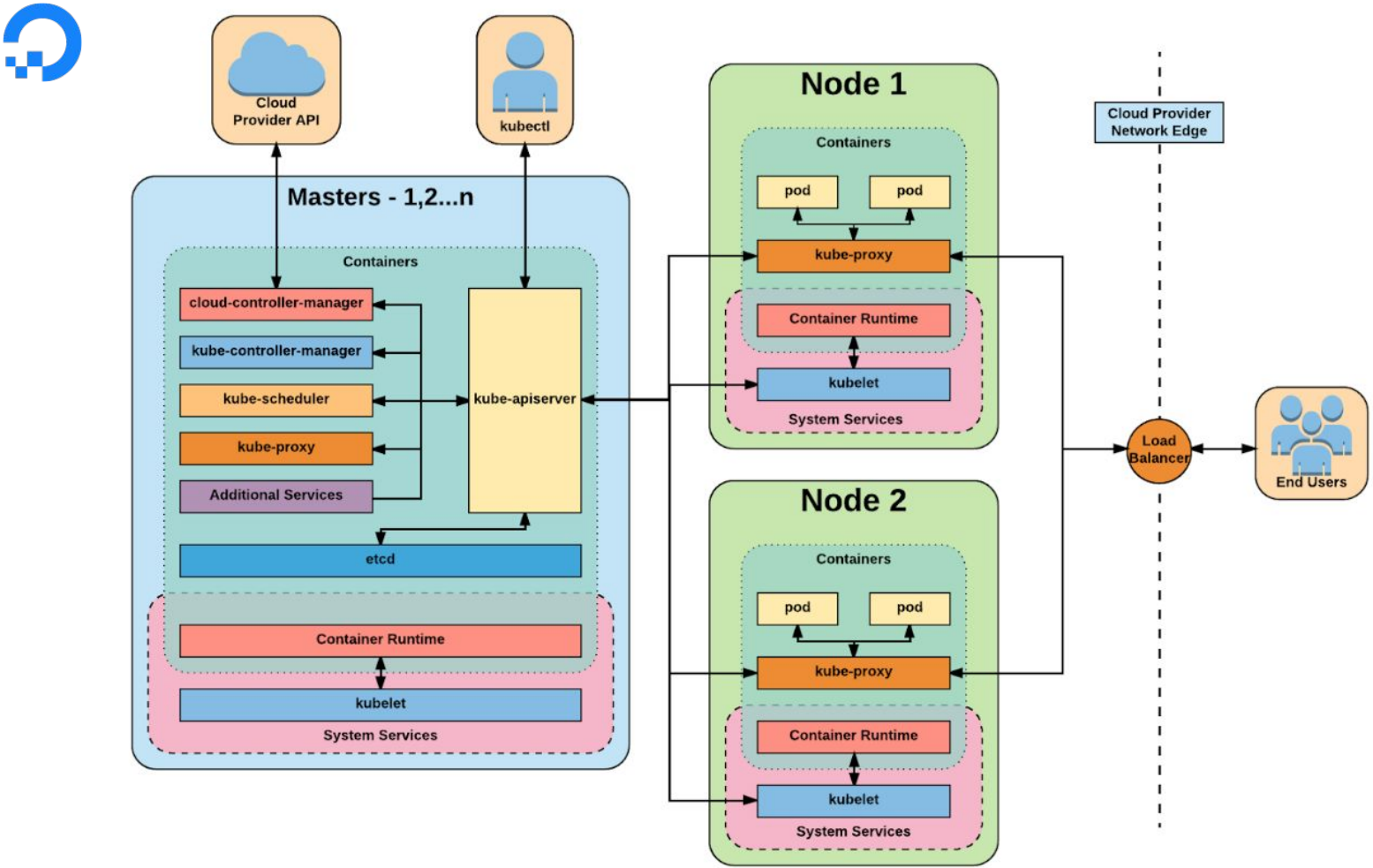
- “K8S”
- Evolved out of Borg (Google’s internal container cluster)
- Open sourced ~2014
- Grew in popularity, open source velocity increased
- Now the most popular container cluster (most cloud platforms have some sort of managed K8S offering)
- Features added regularly
- Cloud Native / CNCF - Kubernetes, Prometheus, Fluentd

CLOUD NATIVE
COMPUTING FOUNDATION

Kubernetes Architecture

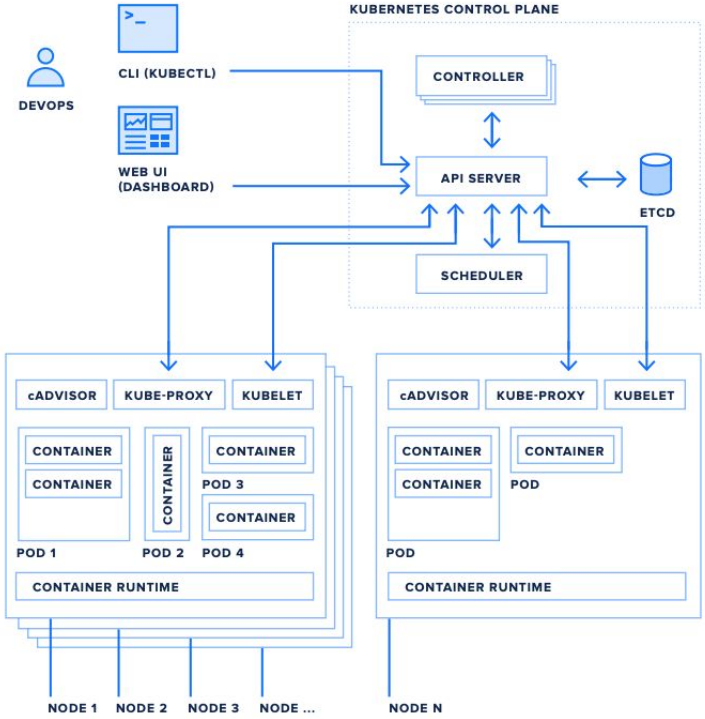
- Control Plane
  - API server
  - Scheduler
  - Controllers
    - Kubernetes
    - Cloud
  - Etcd




Cloud Provider API
kubectl
Masters - 1,2...n
Containers
cloud-controller-manager
kube-controller-manager
kube-scheduler
kube-proxy
Additional Services
kube-apiserver
etcd
Container Runtime
kubelet
System Services
Node 1
Containers
pod
pod
kube-proxy
Container Runtime
kubelet
System Services
Node 2
Containers
pod
pod
kube-proxy
Container Runtime
kubelet
System Services
Cloud Provider Network Edge
Load Balancer
End Users

Kubernetes Architecture

- Nodes
  - Kubelet
  - Kube-proxy
  - Container runtime



Is a container cluster right for me?

- Spoiler: eventually!
  - Lots of moving parts
  - Monolithic app might not be best fit
  - Larger teams, more applications, or microservices benefit

How do I interact with a Kubernetes cluster?

- Hit REST API directly
  - Can use curl, client libraries, etc.
- Kubectl
  - Command-line tool to interact with control plane
  - Abstracts away multiple REST API calls
  - Provides “get” “create” “delete” “describe”, etc. functionality
  - Filtering results

Some Kubectl Commands...

Declarative:

- kubectl get
- kubectl apply
- kubectl rollout status
- kubectl rollout undo

Imperative:

- kubectl create
- kubectl delete
- kubectl expose
- kubectl edit
- kubectl patch

Namespaces

- An abstraction that allows you to divide a cluster into multiple scoped “virtual clusters”
  - E.g. Each team gets its own Namespace with associated resource quota
- Primary mechanism for scoping and limiting access
- K8S usually starts with 3 Namespaces by default
  - default
  - kube-system
  - Kube-public

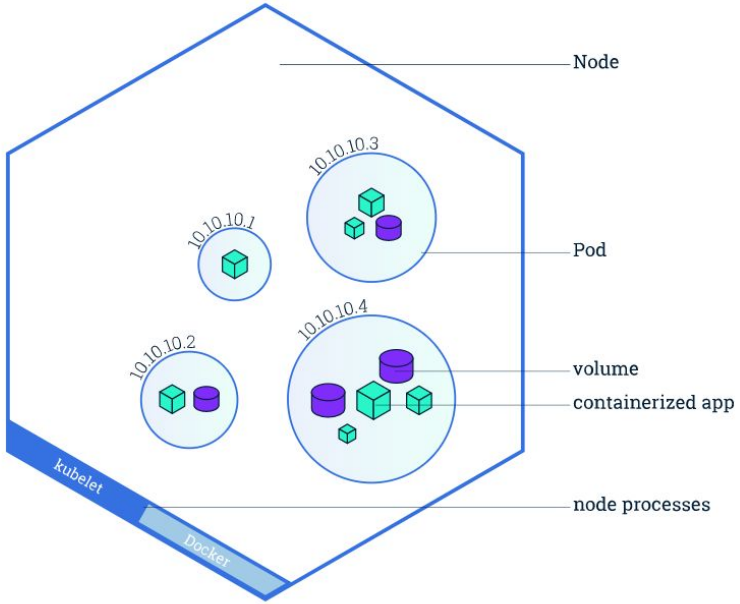
Namespaces

- List namespaces with kubectl:
  - kubectl get namespaces
  - kubectl get ns
- Create your own:
  - kubectl create ns jamon
- Specify a namespace with kubectl:
  - kubectl -n jamon get all

Kubernetes Objects: Pods and Workloads

Pods

- Fundamental Kubernetes work unit
- Can be one or more containers
  - Why more than one?
- Pod containers share resources
  - Storage
  - Network (localhost)
  - Always run on the same Node



Pod Manifest Example

```
apiVersion: v1
kind: Pod
metadata:
  name: flask-pod
  labels:
    app: flask-helloworld
spec:
  containers:
  - name: flask
    image: jamonation/flask-helloworld:v0
    ports:
    - containerPort: 5000
```

Pods

- Let’s build and deploy an image as a Pod
  - cd ~/workshop/v0/app
  - docker build . -t jamonation/flask-helloworld:v0
  - docker push jamonation/flask-helloworld:v0

- And now deploy it to Kubernetes
  - cd ../k8s
  - kubectl apply -f flask-pod.yaml

Pod Manifest Example

- Pod is visible with:
  - kubectl get pods
- But how to reach it?
  - kubectl port-forward pods/flask-pod 5000:5000
- And to test:
  - curl http://localhost:5000

Labels

- Key/value pairs: think of them as object “tags”
- Almost everything can be labeled
  - Even Nodes
- *Not* Unique
- Used to select objects with *selectors*
- Examples:
  - env: prod
  - env: staging
  - release: stable
  - release:canary

Kubernetes Workloads

- Deployments (stateless apps)
  - ReplicaSets
    - Pods
      - Containers
        - Namespaces & cgroups
          - Turtles?
- StatefulSets (stateful apps - e.g. databases)
- DaemonSets (think of these as “agents” / daemons, e.g. logger)
- Jobs & CronJobs (a job, like Django migrations)

Deployments

- How to manage multiple Pods?
- Higher-level object that “contains” the Pod object
- Pod management
  - Deployment
  - Scaling
  - Updates

Deployment v0 example

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-helloworld
  labels:
    app: flask-helloworld
spec:
  replicas: 2
  selector:
    matchLabels:
      app: flask-helloworld
  template:
    metadata:
      labels:
        app: flask-helloworld
    spec:
      containers:
      - name: flask
        image: jamonation/flask-helloworld:v0
        ports:
        - containerPort: 5000
```

<----- Dashed box looks familiar?
It is a Pod template.

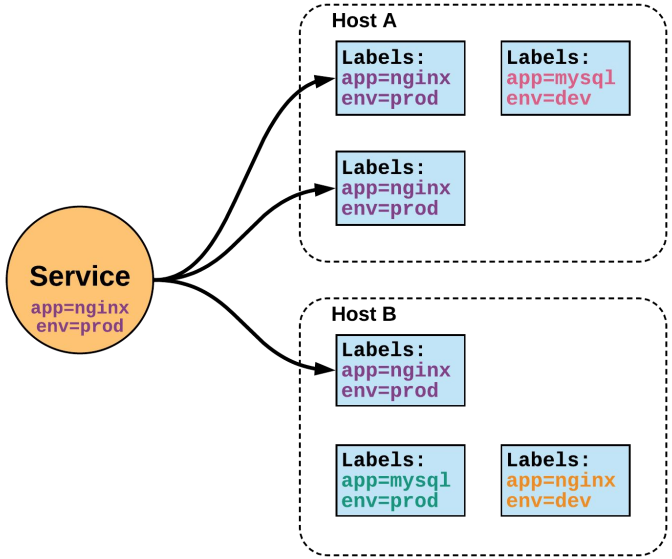
Deployments are higher level abstractions on top of replicas, on top of pods

Deployment v0 example

- Let’s do it!
  - cd ~/workshop/v0/k8s
  - view flask-deployment.yaml
  - kubectl apply -f flask-deployment.yaml
  - kubectl get deployments
  - kubectl get pods

Services: Exposing your apps to the outside world

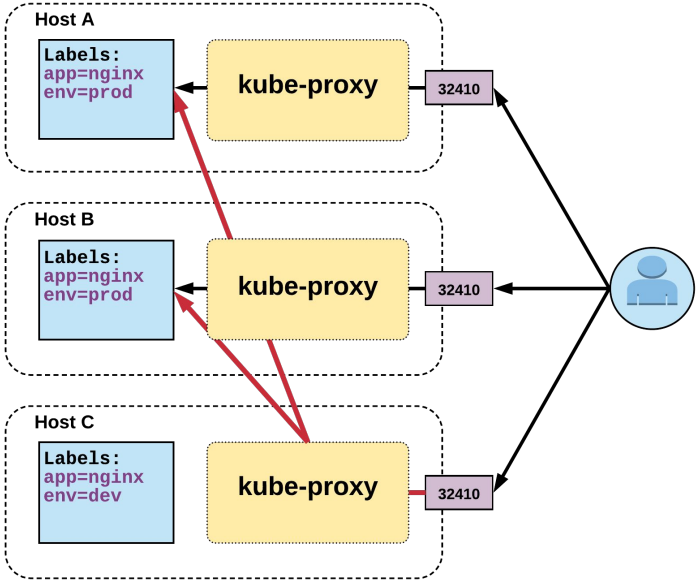
- Abstraction to expose an app as a *service* (think microservices)
- By default, every Pod will be assigned a cluster-internal IP address
- Example: Nginx web server
- Load balancing traffic
  - Routing to “healthy” / “available” Pods
- Again uses Labels + Selectors



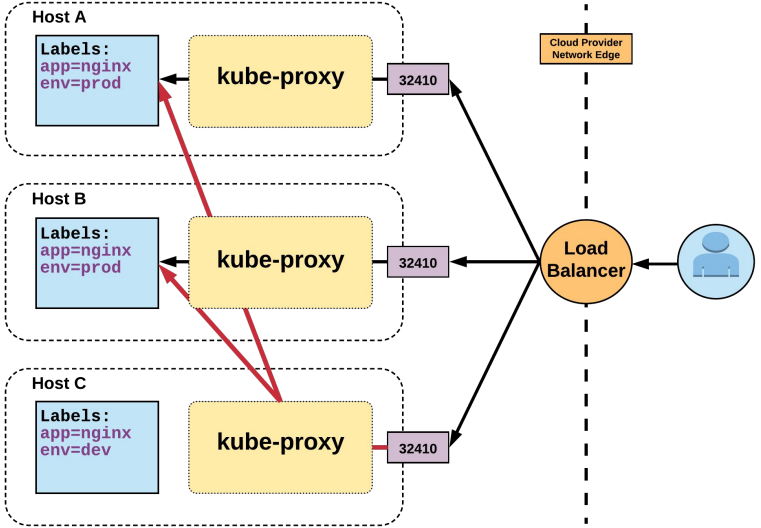
Service Types

- ClusterIP
  - Expose the service on a Cluster-internal IP
- NodePort
  - Expose the service on each Node's IP at a static port ("NodePort")
- LoadBalancer
  - Create an external LoadBalancer which routes requests to Nodeport & ClusterIP services
- Aside: Ingress Controllers

Nodeport



LoadBalancer



Let's create a LoadBalancer Service for Flask

```yaml
apiVersion: v1
kind: Service
metadata:
  name: flask-svc
  labels:
    app: flask-helloworld
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 5000
    protocol: TCP
  selector:
    app: flask-helloworld
```

Let's create a LoadBalancer Service for Flask

- Create the service:
  - cd ~/workshop/v1/k8s
  - kubectl apply -f flask-service.yaml
  - Wait.
  - Examine pods with kubectl port-forward in the meantime

- Show LoadBalancer's IP:
  - kubectl get svc
  - curl <loadbalancer IP>

Now lets update the app & deploy it

- Examine the app (note new flask_session import)
  - cd ~/workshop/v1/app
  - view ~/app.py

- Build new image:
  - docker build . -t jamonation/flask-helloworld:v1
  - docker push jamonation/flask-helloworld:v1

- Deploy it:
  - kubectl apply -f ~/workshop/v1/k8s/flask-deployment.yaml

But wait, new deploy is broken..

- Hmm, curl to the LoadBalancer still works??

- The v0 deployment is still running and reachable!
  - curl shows headers with no session header
  - And content shows pod names from v0 deploy

- This works because the flask-svc is configured to use any app labelled with ‘flask-helloworld’
  - kubectl get service flask-svc -o yaml
  - kubectl get pod flask-app -o yaml

But wait, new deploy is broken..

- Check status of new pods
  - kubectl get pods
  - kubectl describe pod/<broken pod here>
  - kubectl logs pod/<broken pod here>

- Missing Redis env variables and deployment
  - kubectl apply -f ~/workshop/v1/k8s/redis-service.yaml
  - kubectl apply -f ~/workshop/v1/k8s/redis-config.yaml
  - kubectl apply -f ~/workshop/v1/k8s/redis-deployment.yaml
  - view ~/workshop/v1/k8s/flask-deployment.yaml

Configuration: ConfigMaps & Secrets

- Kubernetes provides various features for externalizing and versioning config parameters
  - Stored in etcd
- ConfigMaps
  - Hostnames, runtime parameters for commands, config files
- Secrets
  - Base64-encoded, encrypted with tools like Vault etc.
  - Passwords, credentials, sensitive environment variables
- Versatile, can be created and used in a number of ways
  - Env vars
  - Mounted as Volumes attached to Pods

Configuration: ConfigMaps & Secrets

- ConfigMaps
  - See ~/workshop/v1/k8s/redis-config.yaml
- Secrets
  - See ~/.kube/postgres-secrets.yaml
- Used in deployment:
  - ~/workshop/v2/k8s/flask-deployment-secrets.yaml

So lets deploy again

- This time we'll add an external PostgreSQL database
  - cd ~/workshop/v2/app
  - view app.py
- We also added a healthcheck tool and endpoints
- Build and push again:
  - docker build . -t jamonation/flask-helloworld:v2
  - docker push jamonation/flask-helloworld:v2
  - kubectl edit deployment/flask-dep

Broken again!

- Roll it back:
  - kubectl rollout undo deployment/flask-dep
- Look at Pods and ReplicaSets while reverting:
  - kubectl get pods
  - kubectl get rs
- Add postgres-secrets.yaml:
  - kubectl apply -f ~/.kube/postgres-secrets.yaml
  - kubectl get pods

Working?

- Check pods & ReplicaSets
  - kubectl get pods
  - kubectl get rs

- And in a browser?
  - curl http://<service ip>/healthcheck
  - curl http://<service ip>/environment

Storage & Volumes (briefly)

- Volumes
  - Tied to the lifecycle of the Pod that requests it
  - Can be used to share data between containers in a Pod
- Persistent Volumes & PVCs
  - Abstraction that allows operators to separate storage provisioning from consumption
  - For example:
    - A PV could be a 10Gi DO block storage disk made available to the cluster
    - The PVC (defined in the workload manifest) states that this particular app needs a 10Gi disk. A controller matches the PVC with the PV
- Storage Classes

More K8S Features...

- Resource requests & limits
- Autoscaling
- Node affinity, taints, tolerations
- Dashboard
- Metrics-server

Where to go from here?

- Kubernetes For Fullstack Developers Curriculum
- DigitalOcean Kubernetes Community Tutorials
- Kubernetes Official Documentation
- Kubernetes GitHub Project
- https://k9ss.io

Any questions?

Thank you!