

Getting Started with Containers and Kubernetes





Hi! I'm...

- Wayne Warren
 - I am a Software Engineer
 - Based out of Chicago
 - Working on DigitalOcean Kubernetes (DOKS)



Webinar Goals

- Discuss trends in app design and deployment
- High-level overview of and motivation for containers
- Learn about Kubernetes architecture and objects
- Demo
 - Build a container image for a demo Flask app
 - Deploy Flask app to Kubernetes cluster
 - Create a public load-balancer to access that app



Prerequisites

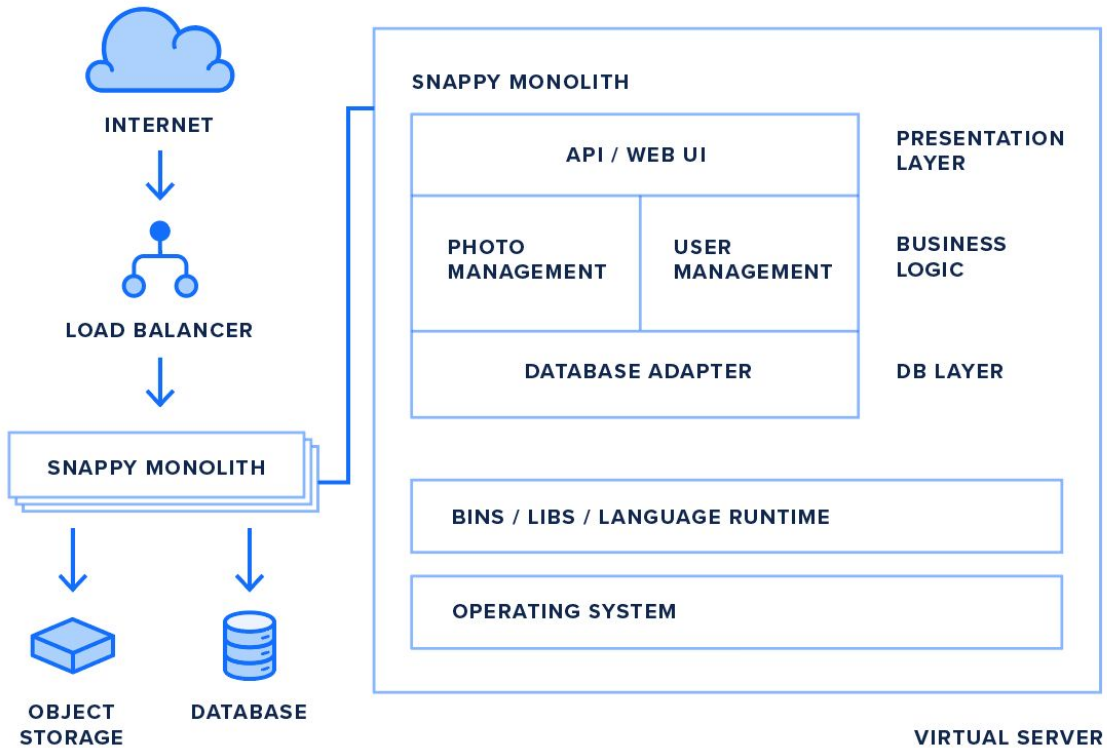
- [Kubernetes cluster](#) you have access to (we'll use DigitalOcean Kubernetes throughout this talk)
- On your machine:
 - [Kubectl](#) configured to access your cluster
 - Git
 - Docker
- Clone the [Flask demo code](#)
 - `git clone https://github.com/do-community/k8s-intro-meetup-kit.git`
 - `cd k8s-intro-meetup-kit`



App Modernization: Monoliths vs Microservices

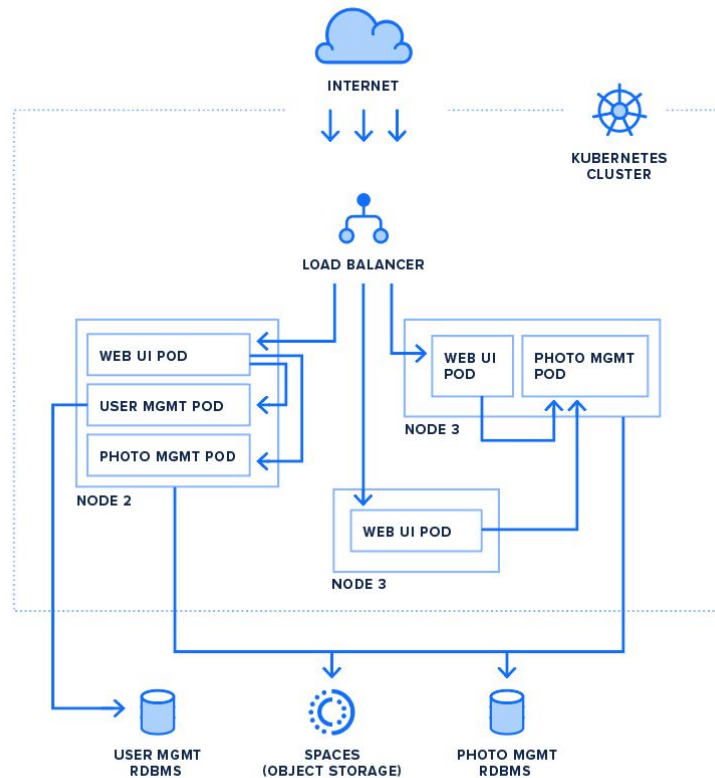
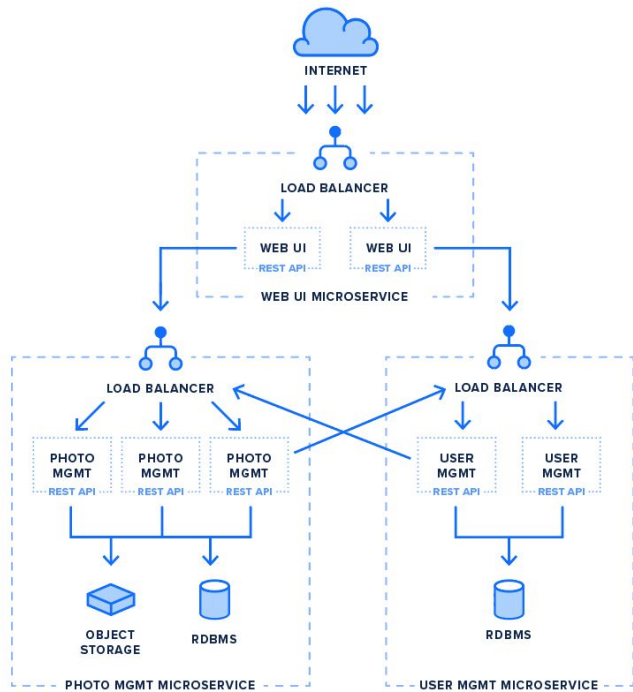


The Monolith





Breaking the Monolith



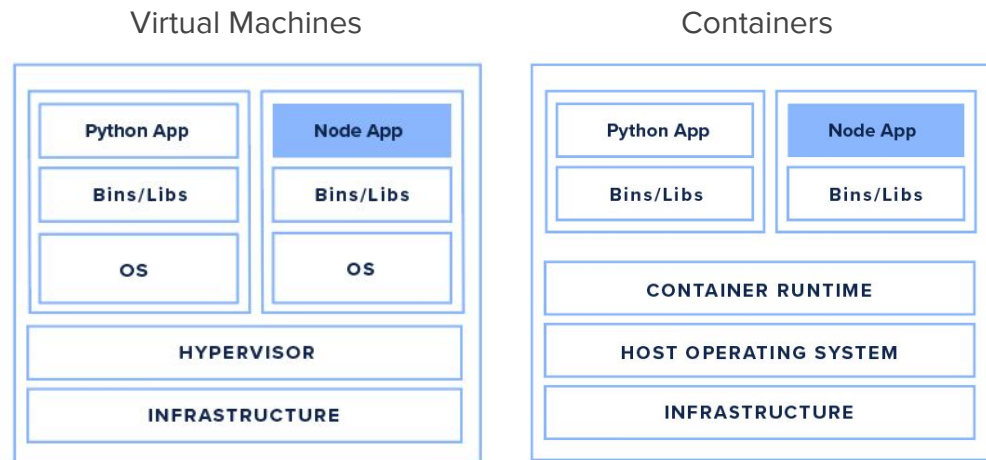


Revisiting Containers



What is a Container?

- VMs vs. Containers
- Container features
 - Lightweight
 - Portable
 - Isolated





But what are they, really?

- A package of application code and all of its dependencies
 - Includes everything needed to run the application
- Built around two Linux kernel features
 - Namespaces: Process isolation
 - Cgroups: Resource limits



Let's try it out!

- Create a PID namespace from scratch
 - `ps aux`
 - `sudo unshare --fork --pid --mount-proc /bin/bash`
 - `ps aux`
- In a **new shell**, find process id of PID namespace we created:
 - `pgrep -af /bin/bash`
 - `pid=<pid of /bin/bash process under unshare parent>`
- Enter into the PID namespace
 - `ps aux`
 - `sudo nsenter -a -t $pid`
 - `ps aux`



Container Ecosystem

- Container
- Container Images
- Container Runtime
- Container Registries



DEV



PUSH CODE



VERSION CONTROL REPOSITORY

TRIGGER BUILD



BUILD TEST



PUSH IMAGE

snappy_web: 1.0

snappy_photo: 2.4	snappy_web: 1.0
snappy_photo: 2.3	snappy_web: 0.9
snappy_photo: 2.2	snappy_web: 0.8
...	...

IMAGE REGISTRY





Example: Containerized Flask App

App Code (cat app/app.py)

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

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

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





Example: Containerized Flask App

Dockerfile (cat app/Dockerfile)

```
FROM python:3-alpine

WORKDIR /app

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

COPY . .

EXPOSE 5000

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

- Build & tag image
 - `docker build -t flask:v0 .`
 - `docker images`
- Run container / test
 - `docker run -p 5000:5000 flask:v0`
 - `docker ps`
 - `curl http://localhost:5000`
- Push to Docker Hub repo (optional)
- What would this look like in VM world?



Container Clusters

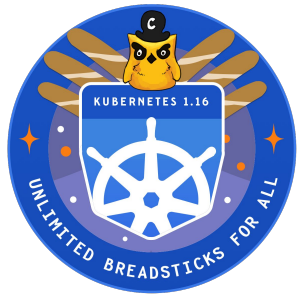


Container Clusters

- 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 Kubernetes 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 and frequently
- Cloud Native / CNCF - Kubernetes, Prometheus, Fluentd

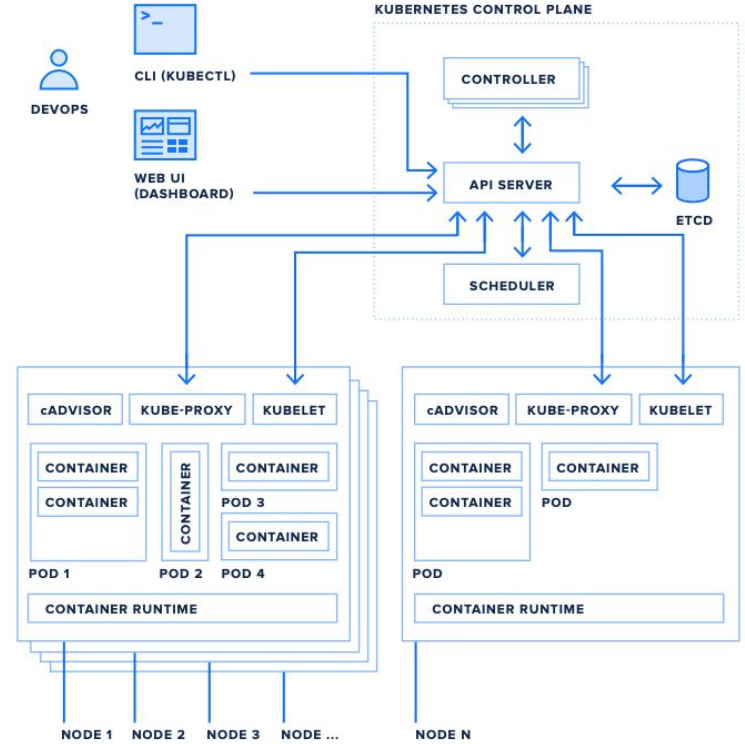


CLOUD NATIVE
COMPUTING FOUNDATION



Kubernetes Architecture

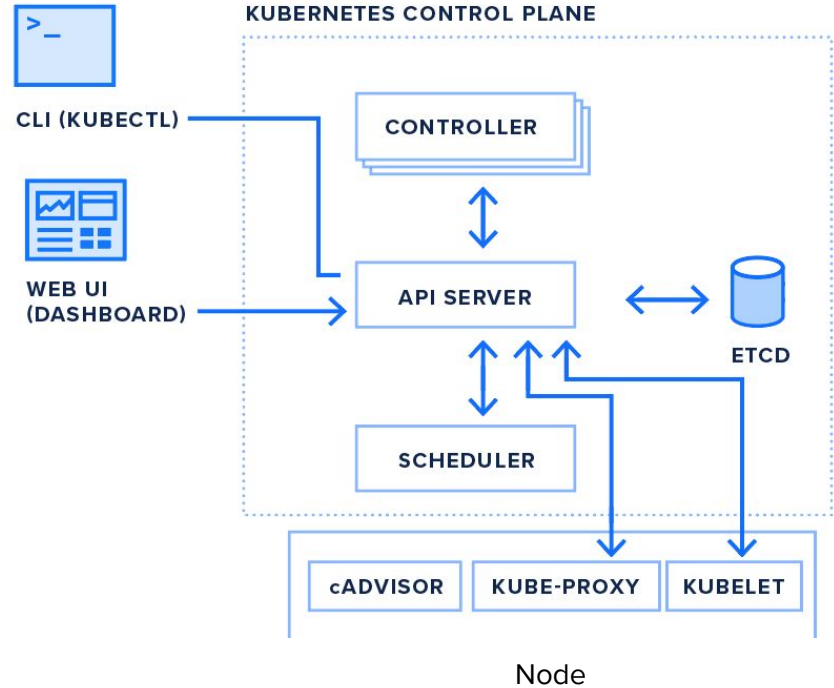
- Client-Server architecture
 - Server: Control Plane
 - Clients: Nodes





Kubernetes Architecture

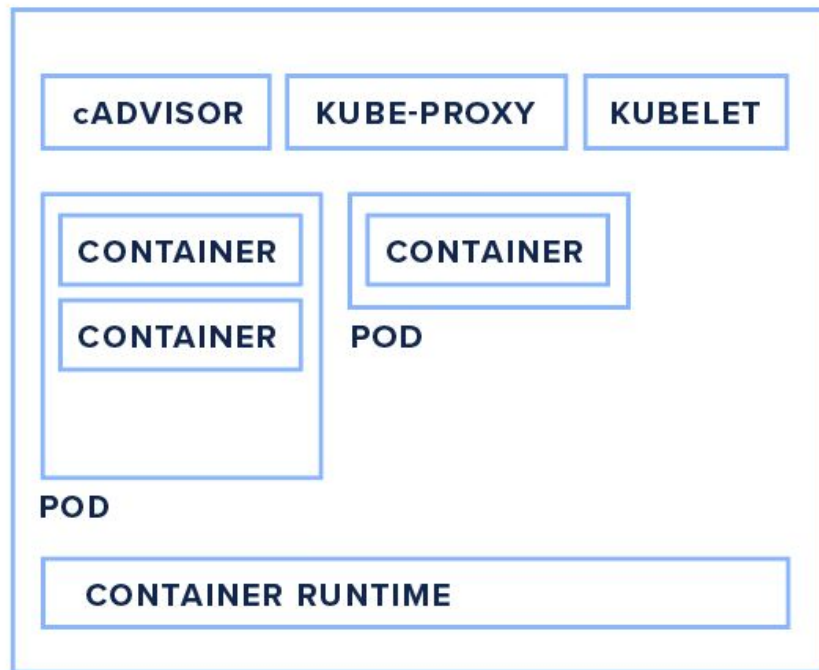
- Control Plane
 - API server
 - Scheduler
 - Controllers
 - Kubernetes
 - Cloud
 - Etcd





Kubernetes Architecture

- Nodes
 - Kubelet
 - Kube-proxy
 - cAdvisor
 - Container runtime





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
- [Set up kubectl](#)
 - `cp k8s_config_file ~/.kube/config`
 - May need to create this directory, depending on your OS
 - `kubectl cluster-info`



Some Kubectl Commands...

- kubectl get
- kubectl apply
- kubectl rollout status
- kubectl rollout undo
- kubectl create
- kubectl delete
- kubectl expose
- kubectl edit
- kubectl patch



Kubernetes Objects: Pods and Workloads



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
- Kubernetes usually starts with 3 Namespaces by default
 - default
 - kube-system
 - kube-public



Creating a Namespace

- List namespaces with kubectl:
 - `kubectl get namespaces`
 - `kubectl get ns`
- Create your own:
 - `kubectl create ns flask`
- Specify a namespace with kubectl:
 - `kubectl -n flask get all`
- If you don't want to use the `-n` flag with every command: **contexts**
 - `kubectl config current-context`
 - `kubectl config set-context --current --namespace=flask`
 - `kubectl config get-contexts`
 - `kubectl get all`



Pods

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

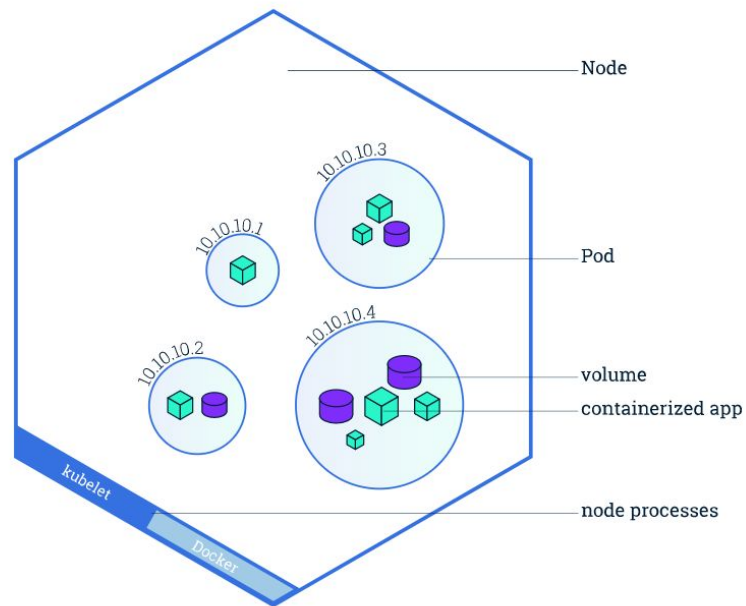


Image Attribution: [K8s Official Docs](#)



Pod Manifest Example

Pod Manifest (cat k8s/flask-pod.yaml)

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

- [Deploy the Flask Pod](#)
 - `kubectl apply -f flask_pod.yaml -n flask`
- Check that it's up
 - `kubectl get pod -n flask`
- Forward a local port into the cluster so that we can access it
 - `kubectl port-forward -n flask pods/flask-pod 5000:5000`
 - `curl http://localhost:5000`
- Delete the Pod
 - `kubectl delete pod flask-pod -n flask`



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
- StatefulSets (stateful apps - e.g. databases)
- DaemonSets (think of these as “agents” / daemons)
- Jobs & CronJobs



Deployments

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



Deployment example

Deployment Manifest (cat k8s/flask-deployment.yaml)

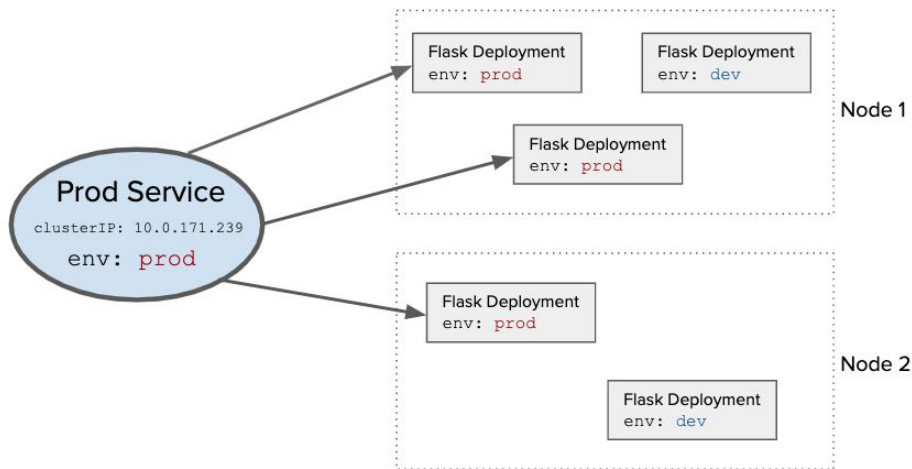
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-dep
  labels:
    app: flask-helloworld
spec:
  replicas: 2
  selector:
    matchLabels:
      app: flask-helloworld
  template:
    metadata:
      labels:
        app: flask-helloworld
    spec:
      containers:
        - name: flask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

- [Roll out the Flask Deployment](#)
 - `kubectl apply -f flask_deployment.yaml -n flask`
- Check that it's up
 - `kubectl get deploy -n flask`
 - `kubectl get pods -n flask`
- Forward a local port into the cluster so that we can access it
 - `kubectl port-forward -n flask deployment/flask-dep 5000:5000`
 - `curl http://localhost:5000`



Services: Exposing your apps to the outside world

- By default, every Pod will be assigned an ephemeral cluster-internal IP address
- If you have a set of Pod replicas (Deployment), how to create a stable endpoint?
- Services: Abstraction to expose an app as a *service* (think microservices)
- Load balancing traffic
 - Routing to “healthy” / “available” Pods
- Again uses Labels + Selectors
- Example: “Prod Service”
 - ClusterIP
 - Stable network endpoint
 - Load-balances traffic to prod Deployment Pods



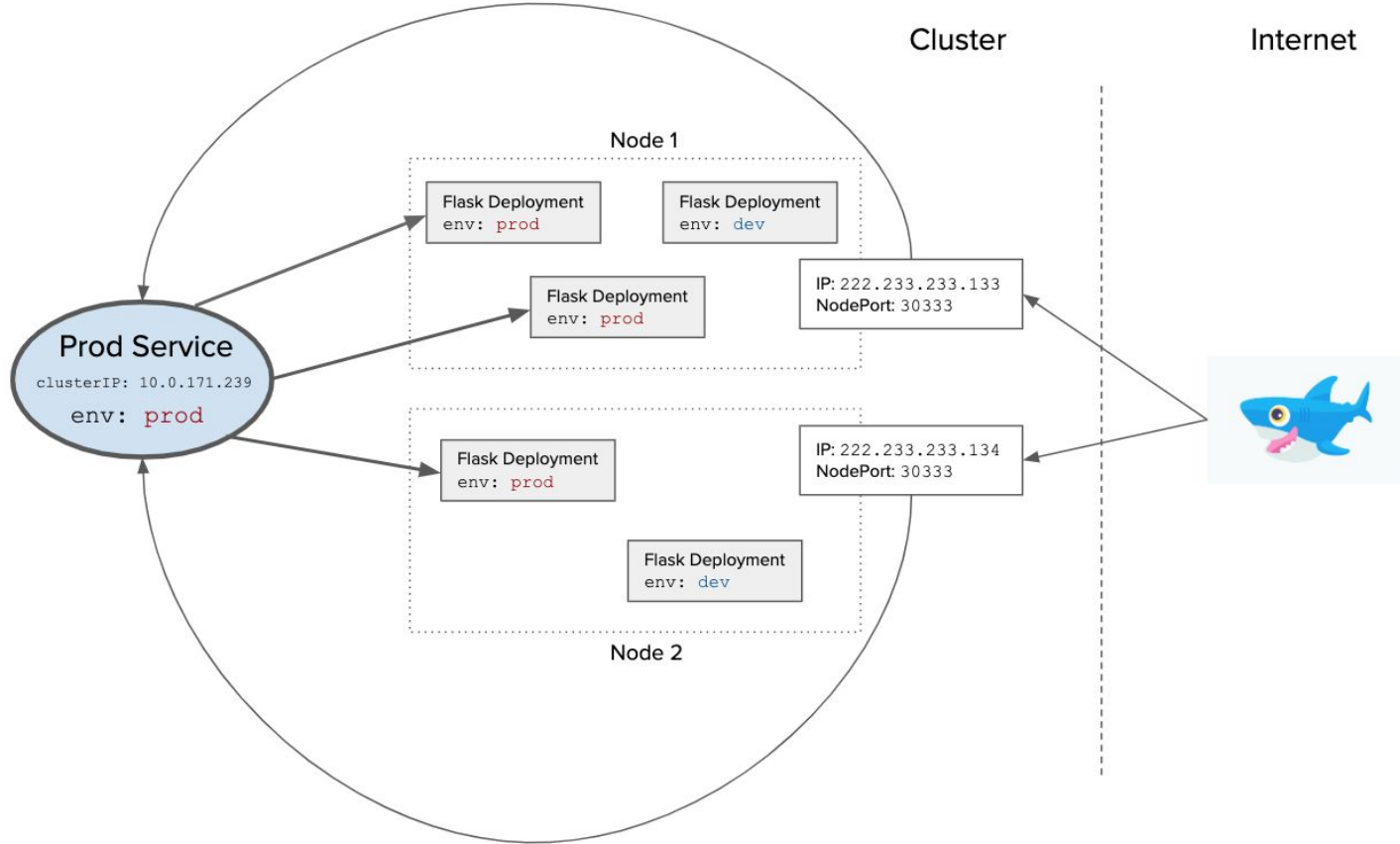


Service Types

- 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

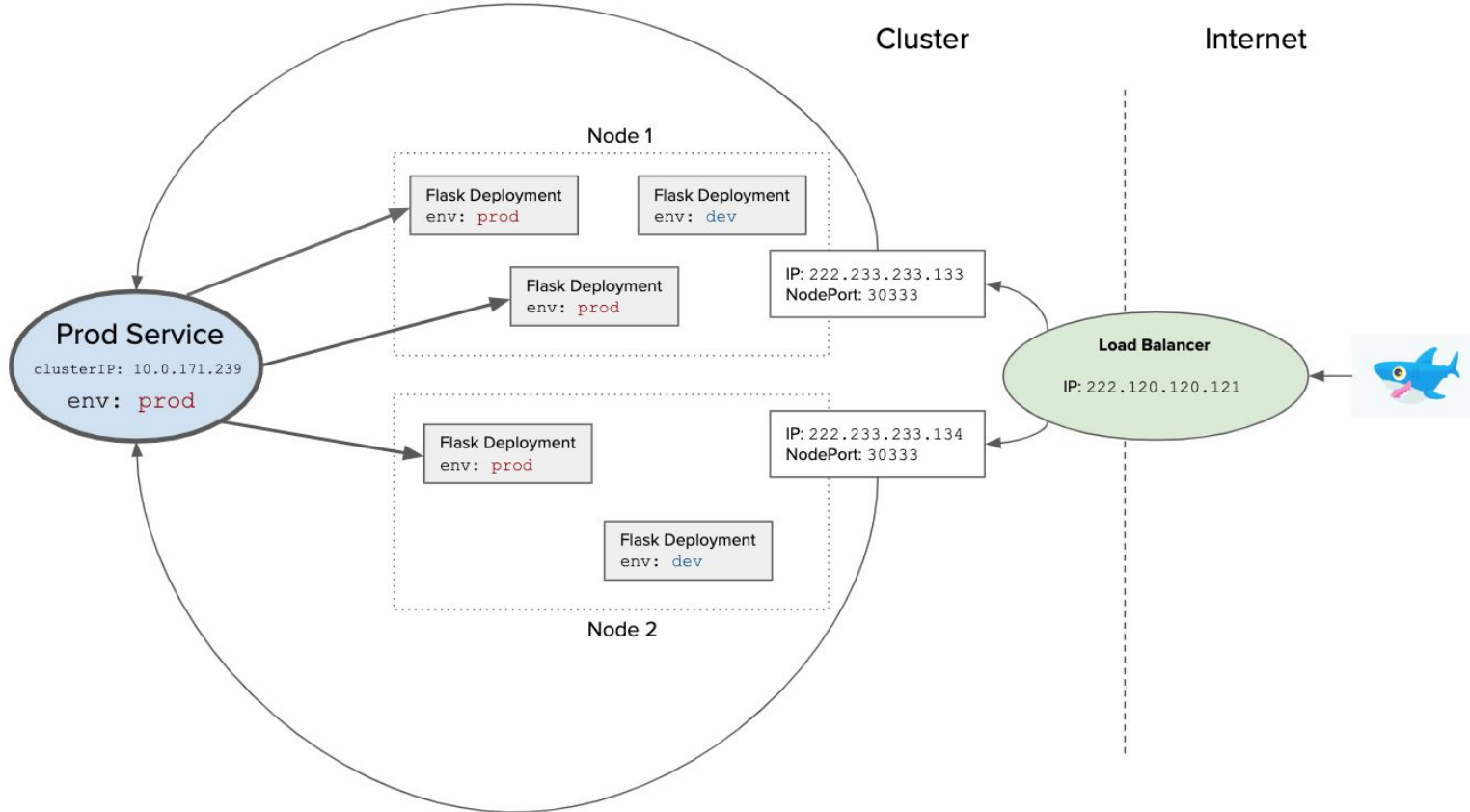


NodePort Service





LoadBalancer Service





Example: Flask App LoadBalancer Service

Service Manifest (cat k8s/flask-service.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
```

- [Deploy the Flask LoadBalancer Service](#)
 - `kubectl apply -f flask-service.yaml -n flask`
- Check that it's up (may have to wait for external IP)
 - `kubectl get svc -n flask`
 - `curl loadbalancer_external_ip`
- Get external IPs of Nodes (for NodePort services)
 - `kubectl get node -o wide`



Other Kubernetes Resources



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
 - Passwords, credentials, etc.
- Versatile, can be created and used in a number of ways
 - Env vars
 - Mounted as Volumes attached to Pods



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



Helm: a K8S “Package Manager”

- Tool for managing Kubernetes applications
 - Think “apt-get” for Ubuntu / package managers
- Architecture
 - Helm (client)
 - Tiller (server, runs in the cluster)
- How it works
 - Charts
 - `helm install stable/wordpress`
- Sample apps: Wordpress, Prometheus, MySQL, Drupal, ...



Where to go from here?

- [Kubernetes For Fullstack Developers Curriculum](#)
- [Kubernetes White Paper](#)
- [DigitalOcean Kubernetes Community Tutorials](#)
- [Kubernetes Official Documentation](#)
- [Kubernetes GitHub Project](#)
- [The History of Kubernetes and the Community Behind It](#)
- [K9s](#)



Any questions?

Thank you!

