Kubernetes (k8s)

Rajesh G

CTO, Managing Partner
https://unigps.in

Training Objectives

At the end of training,

participants should be able to

❑ Know Kubernetes and Be a Helmsman

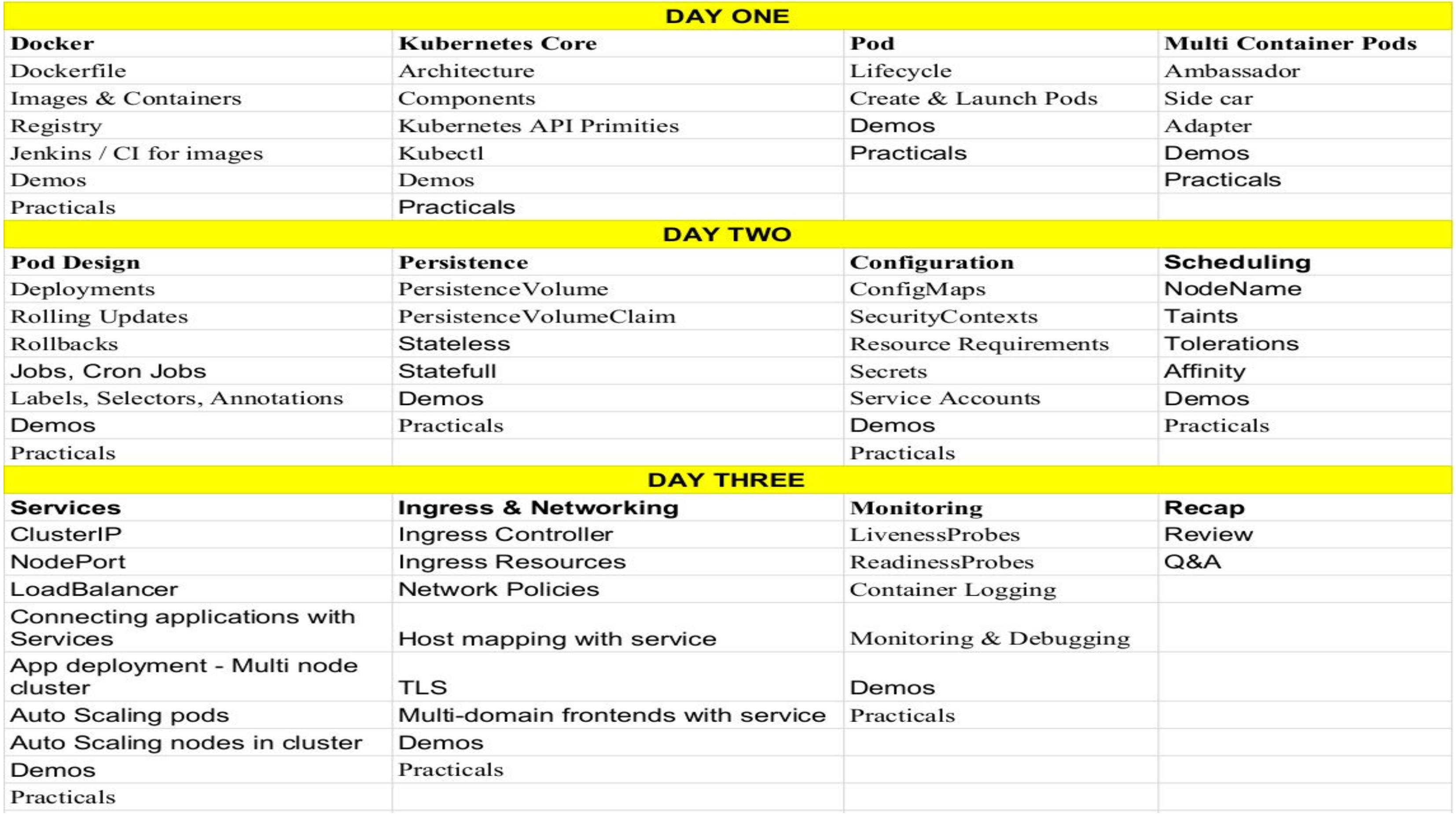
❑ Create and run PODs

❑ Bundle applications & Deploy

❑ Service apps using Load Balancers

❑ Troubleshoot

Table of Contents



DAY ONE			
Docker	**Kubernetes Core**	**Pod**	**Multi Container Pods**
Dockerfile	Architecture	Lifecycle	Ambassador
Images & Containers	Components	Create & Launch Pods	Side car
Registry	Kubernetes API Primities	Demos	Adapter
Jenkins / CI for images	Kubectl	Practicals	Demos
Demos	Demos		Practicals
Practicals	Practicals		
DAY TWO			
Pod Design	**Persistence**	**Configuration**	**Scheduling**
Deployments	PersistenceVolume	ConfigMaps	NodeName
Rolling Updates	PersistenceVolumeClaim	SecurityContexts	Taints
Rollbacks	Stateless	Resource Requirements	Tolerations
Jobs, Cron Jobs	Statefull	Secrets	Affinity
Labels, Selectors, Annotations	Demos	Service Accounts	Demos
Demos	Practicals	Demos	Practicals
Practicals		Practicals	
DAY THREE			
Services	**Ingress & Networking**	**Monitoring**	**Recap**
ClusterIP	Ingress Controller	LivenessProbes	Review
NodePort	Ingress Resources	ReadinessProbes	Q&A
LoadBalancer	Network Policies	Container Logging	
Connecting applications with Services	Host mapping with service	Monitoring & Debugging	
App deployment - Multi node cluster	TLS	Demos	
Auto Scaling pods	Multi-domain frontends with service	Practicals	
Auto Scaling nodes in cluster	Demos		
Demos	Practicals		
Practicals			

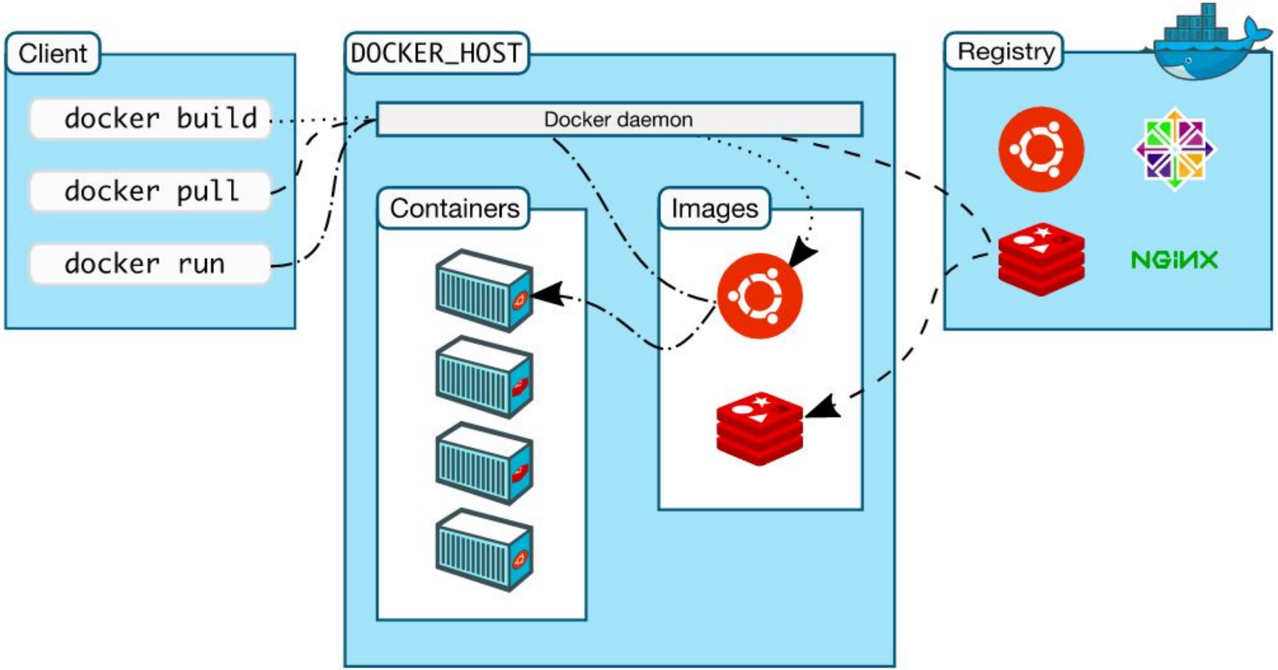
Docker

- Overview
- Dockerfile
- Images & Containers
- Registry
- Jenkins / CI
- Demo

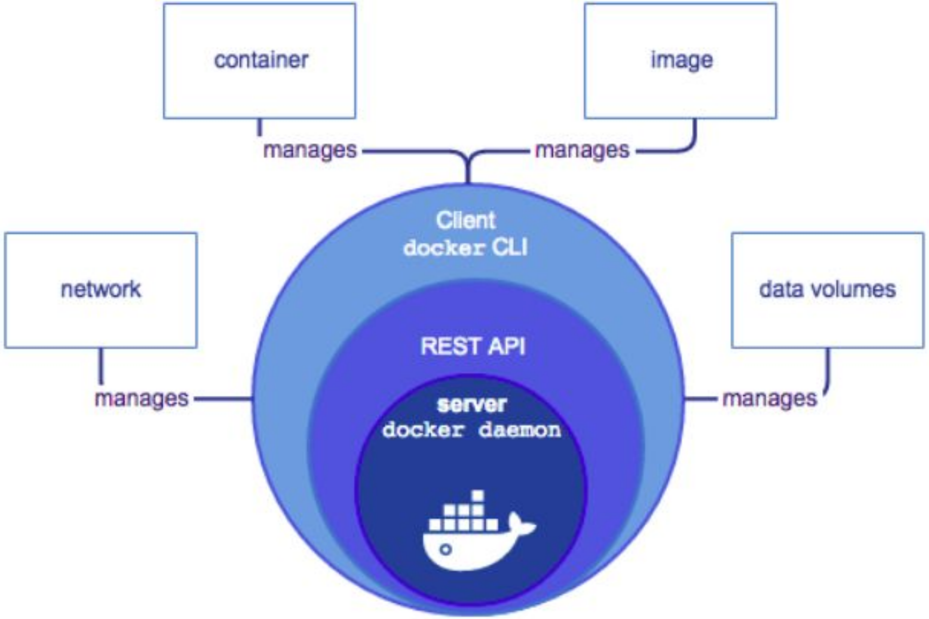
Containers - Benefits

- Flexible: Even the most complex applications can be containerized.
- Lightweight: Containers leverage and share the host kernel.
- Interchangeable: You can deploy updates and upgrades on-the-fly.
- Portable: You can build locally, deploy to the cloud, and run anywhere.
- Scalable: You can increase and automatically distribute container replicas.
- Stackable: You can stack services vertically and on-the-fly
- Running more workload on the same hardware

Docker Architecture



Docker Architecture



Introducing the Dockerfile

A Dockerfile is a text document that contains

- a set of instructions required to assemble the app (image) and/ run it

```
Usage:

docker build [OPTIONS] PATH | URL | -

Options:
      --add-host list           Add a custom host-to-IP mapping (host:ip)
      --compress                Compress the build context using gzip
      --cpu-quota int           Limit the CPU CFS (Completely Fair Scheduler) quota
  -f, --file string             Name of the Dockerfile (Default is 'PATH/Dockerfile')
      --force-rm                Always remove intermediate containers
      --label list              Set metadata for an image
  -m, --memory bytes            Memory limit
      --pull                    Always attempt to pull a newer version of the image
      --rm                      Remove intermediate containers after a successful build (default true)
  -t, --tag list                Name and optionally a tag in the 'name:tag' format
```

Introducing the Dockerfile

Example:

- docker build -f Dockerfile .

```
rajesh@rajesh-Gazelle:~/git/dockers/trainora-aug/images/simple$ cat Dockerfile
FROM alpine:latest

MAINTAINER rajesh@unigps.in
```

Introducing the Dockerfile

Few more variations:

- docker build -t myfirstimage -f Dockerfile .
- docker build -f /home/rajesh/git/dockers/trainora-aug/images/simple/Dockerfile-myfirstimage .
- docker build -t myfirstimage -f ./simple/Dockerfile ./simple/
- docker build -t myimage -t rajesh/myimage:1.0.0 -t localhost:5000/rajesh/myimage:1.0.0 .

Introducing the Dockerfile

- ENV - to set environment variables
- EXPOSE - to expose ports
- FROM - base image
- LABEL - to add metadata to image
- HEALTHCHECK - to check if container is running
- USER - to set user and group
- VOLUME - to specify mount point from external host
- WORKDIR - workdir to run any of the commands

Introducing the Dockerfile

- ARG - variable used during build time
- CMD - to provide defaults to executing container
- RUN - to execute commands in new layer
- COPY - Copy file,dir or remote url to image
- ADD - Copy file,dir or remote url to image
- ENTRYPOINT - to configure container as executable
- MAINTAINER - the image maintainer

RUN COPY ADD instructions create new layers in the image stack - refer layering section

Building Images (Python)

rajesh@rajesh-Gazelle:~/git/dockers/trainora-aug/images/python$ cat Dockerfile

```
FROM python:2.7-slim
WORKDIR /app
ADD app.py /app
ADD requirements.txt /app
RUN pip install --trusted-host pypi.python.org -r requirements.txt
EXPOSE 80
ENV name world
CMD ["python","app.py"]
```

Build

- `docker build -t mypython .`

Run

- `docker run -p 80:80 mypython`

Dockerfile - Example (Apache)

```
FROM bitnami/minideb-extras:jessie-r23
LABEL maintainer "Bitnami <containers@bitnami.com>"

# Install required system packages and dependencies
RUN install_packages libapr1 libaprutil1 libc6 libexpat1 libffi6 libgmp10 libgnutls-deb0-28 libhogweed2 libldap-2.4-2 libnettle4
libp11-kit0 libpcre3 libsasl2-2 libssl1.0.0 libtasn1-6 libuuid1 zlib1g
RUN bitnami-pkg unpack apache-2.4.29-1 --checksum
42114e87aafb1d519ab33451b6836873bca125d78ce7423c5f7f1de4a7198596
RUN ln -sf /opt/bitnami/apache/htdocs /app

COPY rootfs /

ENV APACHE_HTTPS_PORT_NUMBER="443" \
    APACHE_HTTP_PORT_NUMBER="80" \
    BITNAMI_APP_NAME="apache" \
    BITNAMI_IMAGE_VERSION="2.4.29-r1" \
    PATH="/opt/bitnami/apache/bin:$PATH"

EXPOSE 80 443

WORKDIR /app
ENTRYPOINT ["/app-entrypoint.sh"]
CMD ["nami","start","--foreground","apache"]
```

Dockerfile - Example (Jenkins CI)

```
FROM jenkinsci/jenkins:latest
LABEL maintainer "r1co@post-box.cc"

USER root

# install docker cli
RUN mkdir -p /tmp/_install && cd /tmp/_install && wget https://get.docker.com/builds/Linux/x86_64/docker-latest.tgz  && tar -xvzf docker-latest.tgz && cd docker && cp docker /usr/bin/docker && rm -rf  /tmp/_install
RUN chmod +x /usr/bin/docker
# add jenkins to docker group
RUN groupadd -g 999 docker
RUN usermod -a -G docker jenkins
# install docker-compose
RUN curl -L https://github.com/docker/compose/releases/download/1.7.1/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose
RUN chmod +x /usr/local/bin/docker-compose
USER jenkins
```

Dockerfile - Example (Multi stage)

```
FROM golang:1.7.3 AS builder
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go    .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=builder /go/src/github.com/alexellis/href-counter/app .
 CMD ["./app"]
```

Docker Hub - store & retrieve

https://hub.docker.com (register and create login)

- docker tag alpine rajeshgheware/alpine:rajesh
- docker push rajeshgheware/alpine:rajesh
- docker pull rajeshgheware/alpine:rajesh

Registry Server

- With no docker volume (uses default volume for container)

  -
    ```
    docker run -d -p 5000:5000 --name registry registry:2
    ```

  -
    ```
    docker push localhost:5000/rajesh/alpine:test
    ```

  -
    ```
    Docker pull localhost:5000/rajesh/alpine:test
    ```
- With docker volume

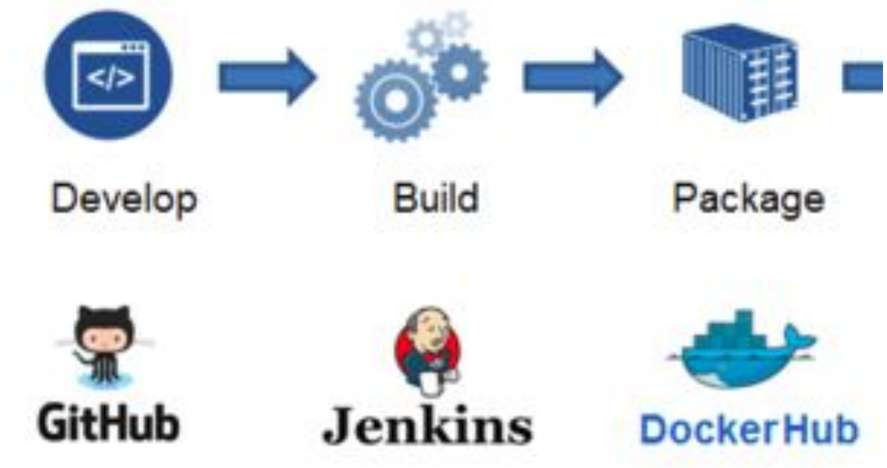
  -
    ```
    docker volume create docker_registry
    ```

  -
    ```
    docker run -d -p 5000:5000 -v docker_registry:/var/lib/registry --name registry registry:2
    ```

  -
    ```
    docker container stop registry && docker container rm -v registry
    ```
- With Volume Mount on Host

  -
    ```
    docker run -d -p 5000:5000 -v /media/deepti/Ubuntu/home/docker_registry:/var/lib/registry --name registry registry:2
    ```

Build Image using CI / Jenkins



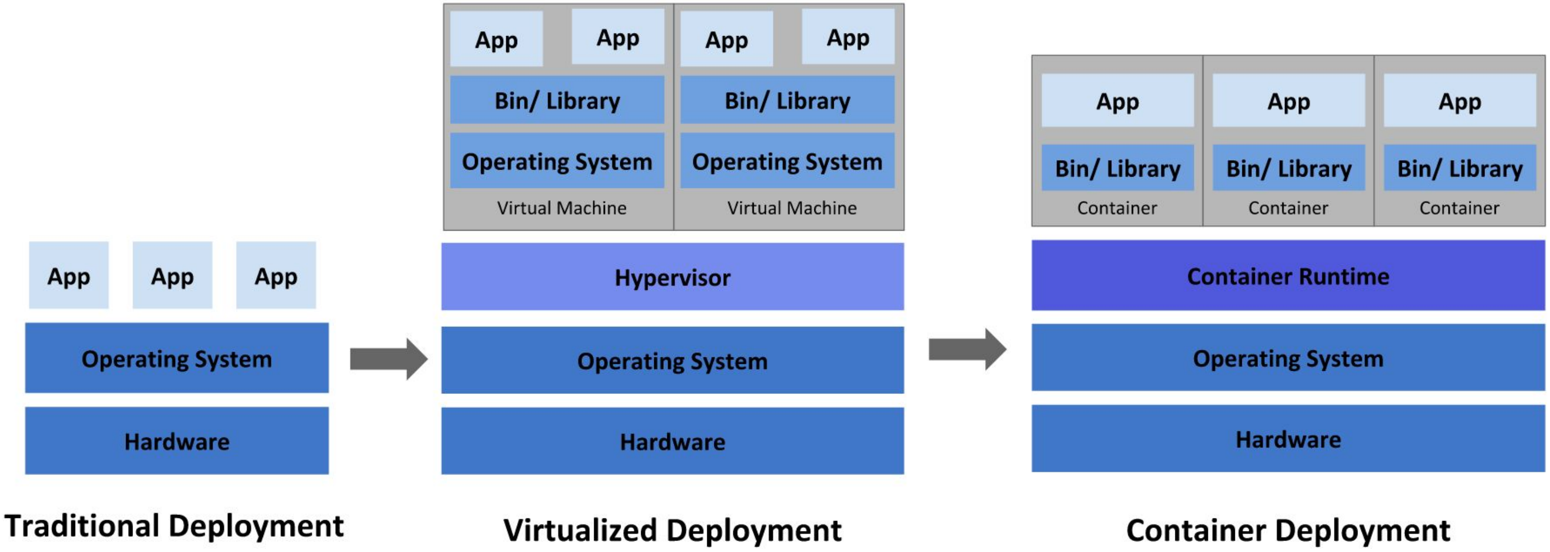
```
docker run -p 8080:8080 -p 50000:50000 -v /var/run/docker.sock:/var/run/docker.sock r1co/jenkins-docker
```

https://jenkins.io/doc/tutorials/build-a-node-js-and-react-app-with-npm/ `(ci for node reactjs build with pipeline)`

Kubernetes Core

- Architecture
- Components
    - Master Components
    - Node Components
    - Add ons
- API Primitives
- Kubectl
- Demo
- Practicals

Deployment - Journey



App
App
App
Operating System
Hardware
Traditional Deployment

App
App
App
App
Bin/ Library
Bin/ Library
Operating System
Operating System
Virtual Machine
Virtual Machine
Hypervisor
Operating System
Hardware
Virtualized Deployment

App
App
App
Bin/ Library
Bin/ Library
Bin/ Library
Container
Container
Container
Container Runtime
Operating System
Hardware
Container Deployment

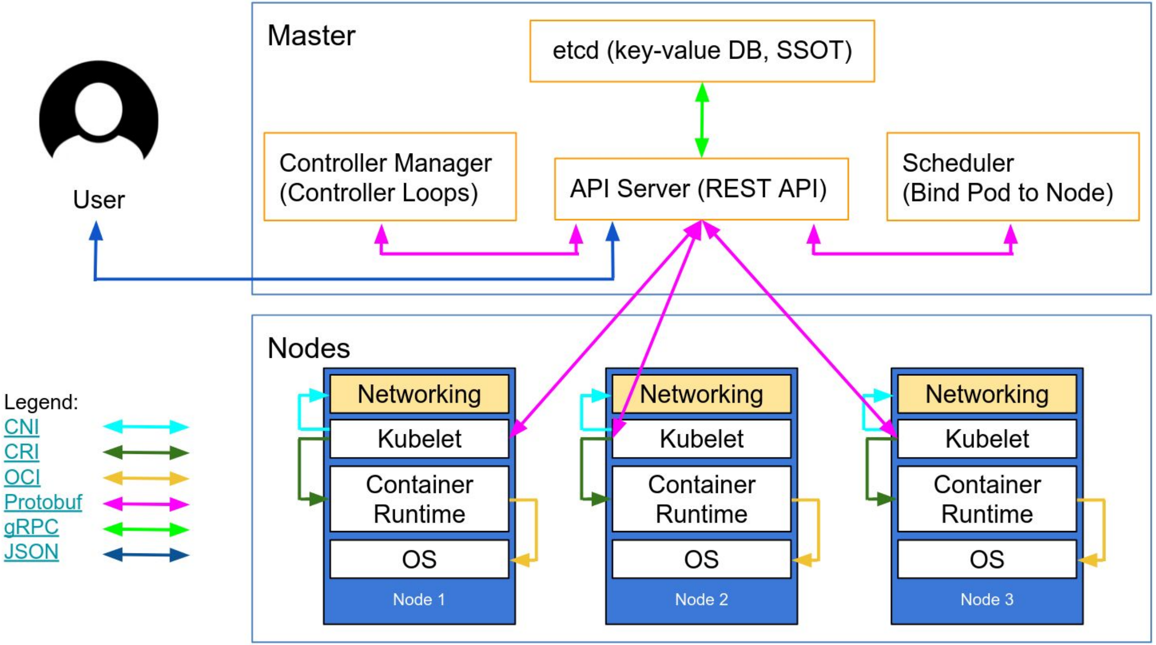
What is / why Kubernetes

Kubernetes - Helmsman (in ancient greek): Guy who steers ship / boat

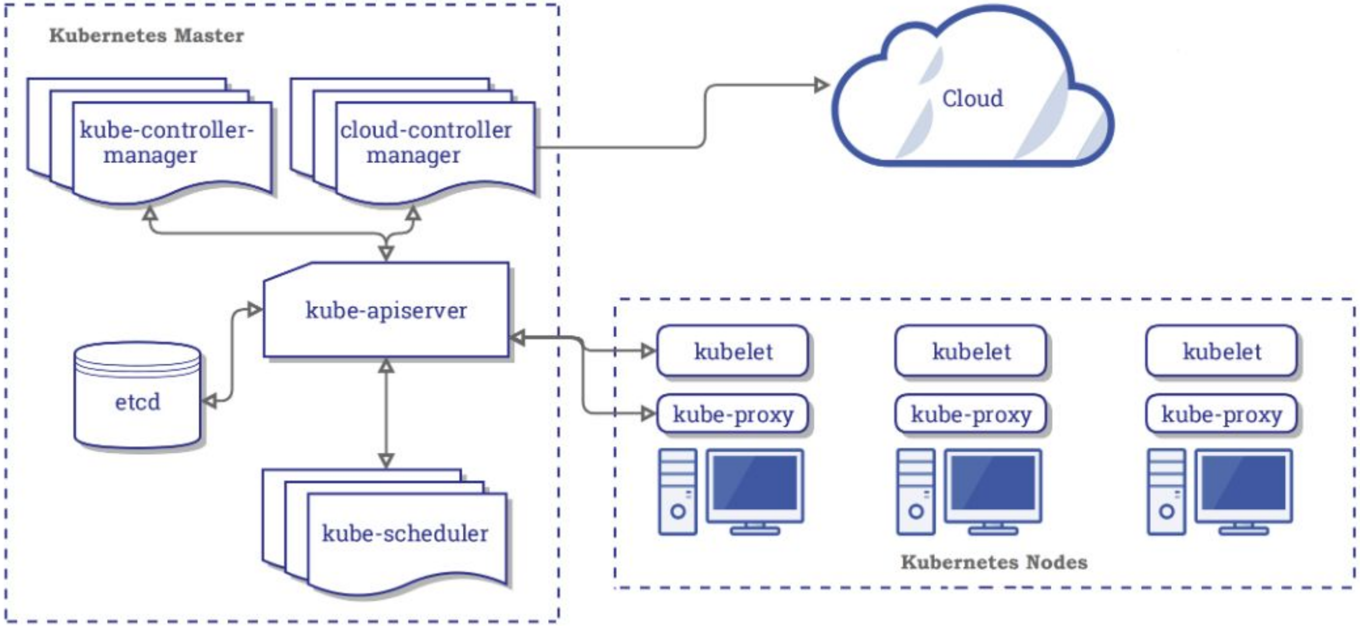
Why Kubernetes?

- Service Discovery & Load Balancing
- Storage Orchestration
- Automated rollouts & rollbacks
- Automatic bin packing
- Self-healing
- Secret and configuration Management

Architecture - Overview



Architecture (view 2)



Master Components - ETCD

- Distributed reliable key-value store that is simple, secure & fast
- Uses RAFT based consensus algorithm to work in distributed environment
- Key value store distributed database
- Runs on port 2379

Master Components - API Server

- The central management entity
- Only component that connects to ETCD
- Designed for horizontal scaling

Connectivity:

- External: kubectl
- Internal: kubelet
- Persistent Storage: ETCD

Master Components - Scheduler

Schedules pods on appropriate Node(s)

Watches for newly created PODs that have no nodes assigned

Decision Parameters:

- Resource requirements (memory, cpu, disk type say SSD)
- Hardware, Software, Policy requirements
- Affinity, Anti-affinity
- Data locality
- Inter workload interference
- Deadlines

Master Components - Kube Controller

- Node Controller
  - Responsible for noticing and responding when nodes go down
- Replication Controller
  - Responsible for maintaining the correct number of pods for every replication controller object in the system
- Endpoints Controller
  - Populates the Endpoints object (that is, joins Services & Pods)
- Service Account & token Controller
  - Create default accounts and API access tokens for new namespaces

Master Components - Cloud Controller

- Route Controller
  - For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
- Service Controller
  - For setting up routes in the underlying cloud infrastructure
- Service Controller
  - For creating, updating and deleting cloud provider load balancers
- Volume Controller
  - For creating, attaching, and mounting volumes, and interacting with the cloud provider to orchestrate volumes

Node Components - kube-proxy

- Network proxy that runs on every node in cluster
- Maintains network rules on nodes
- Uses OS packet filtering layer else forwards traffic itself

Node Components - Container RT

- Docker
- Containerd
- cri-o
- rktlet

Addon Components

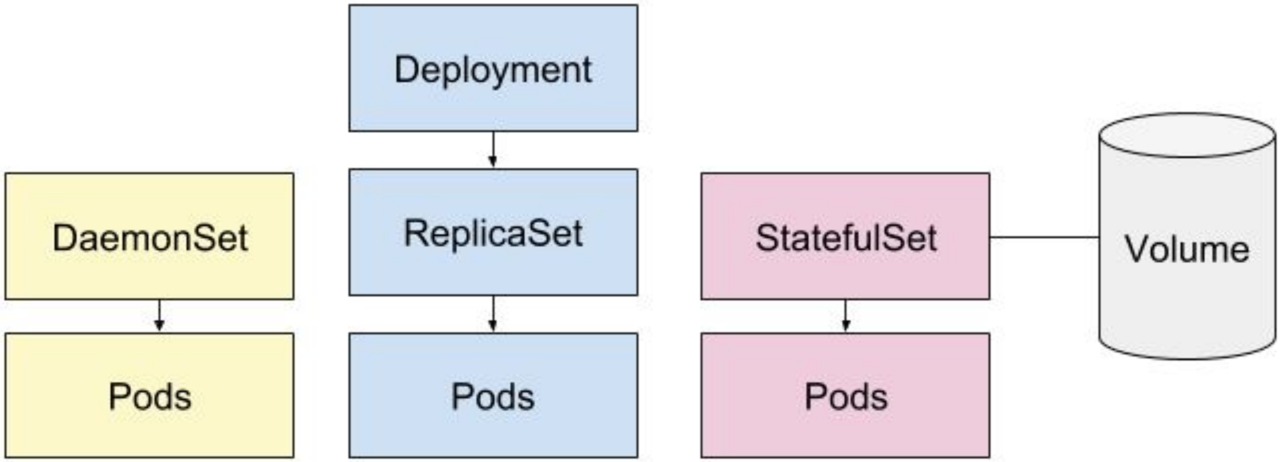
- Cluster DNS
  - Cluster DNS is a DNS server, in addition to the other DNS server(s) in your environment, which serves DNS records for Kubernetes services

- Web UI
  - General purpose, web-based UI for Kubernetes clusters to view and manager cluster

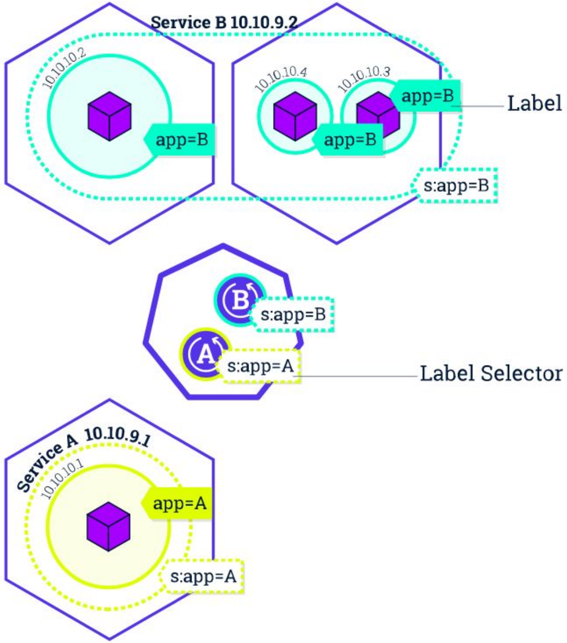
- Container Resource Monitoring
  - Generic time-series metrics about containers in a central database, and provides a UI for browsing that data

- Cluster level Logging
  - Mechanism responsible for saving container logs to a central log store with search/browsing interface

Objects


Deployment
DaemonSet
ReplicaSet
StatefulSet
Volume
Pods
Pods
Pods

Objects


Service B 10.10.9.2
10.10.10.2
app=B
10.10.10.4
10.10.10.3
app=B
app=B
Label
s:app=B
B
s:app=B
A
s:app=A
Label Selector
Service A 10.10.9.1
10.10.10.1
app=A
s:app=A

Node Components - kubelet

- Runs on every node
- Ensures containers are running & healthy in PODs
- Doesn’t manage container not created by K8S

Kubectl

Command line tool to control kubernetes cluster

- Imperative commands to manage objects (basic & intermediate)
- Deploy commands
- Cluster Management commands
- Troubleshooting and Debugging
- Advanced, Settings and Other

kubectl - commands

- kubectl get pods
- kubectl describe pod hello-world
- kubectl describe pod/nodejs
- kubectl delete pod webapp
- kubectl cluster-info
- kubectl get pods -o yaml
- kubectl get services -o json
- kubectl get pods --sort-by=.metadata.name
- kubectl get rs,deployments,service
- kubectl describe nods
- kubectl get pod/<pod-name> svc/<svc-name>
- kubectl get pod -l name=<label-name>
- kubectl delete pods --all
- kubectl get nodes -o json | jq ‘.items[] | {name:.metadata.name, cap:.status.capacity}’
- kubectl get nodes -o yaml | egrep ‘\sname:|cpu:|memory:’
- kubectl get all

- kubectl run hello --image=tutum/hello-world --port=80
- kubectl run -it busybox --image=busybox --restart=Never
- kubectl run nginx --image=nginx --replicas=1

Exercises (30 mins)

Run Hello World POD using tutum/hello-world image (kubectl run…) & then

- Get POD summary (kubectl get …)
- Get POD details (kubectl describe …)
- Get POD IP (kubectl describe pod… -o yaml | egrep….podIP:)
- Delete the POD created above (kubectl delete …)
- Verify using kubectl get all
- View cluster info

POD

- Overview
- Lifecycle
- Init Containers
- Preset
- Topology Spread
- Ephemeral Containers

POD - Overview

- Smallest deployable unit
- Supports multiple cooperating processes (containers) that form cohesive unit of service
- Ephemeral Entity

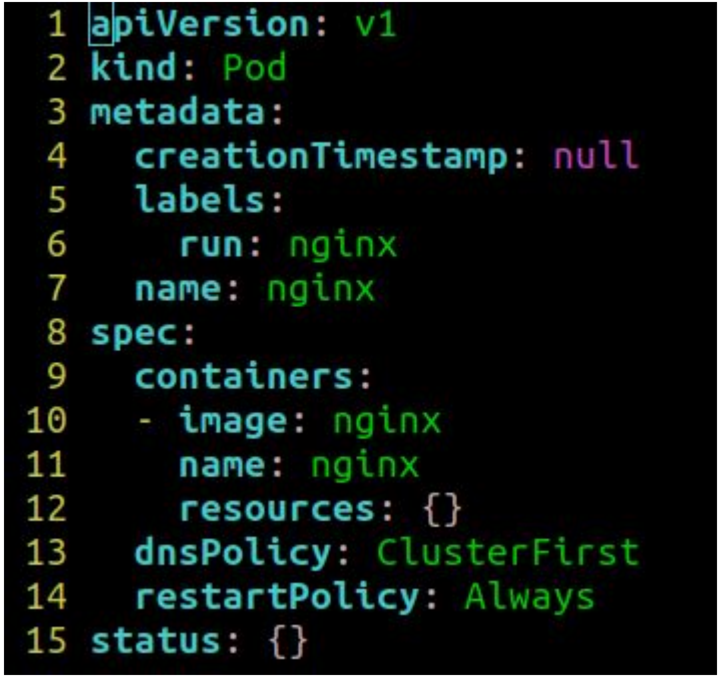
Encapsulates

- application container(s)
- Storage resources
- Unique network IP

Shared Resources:

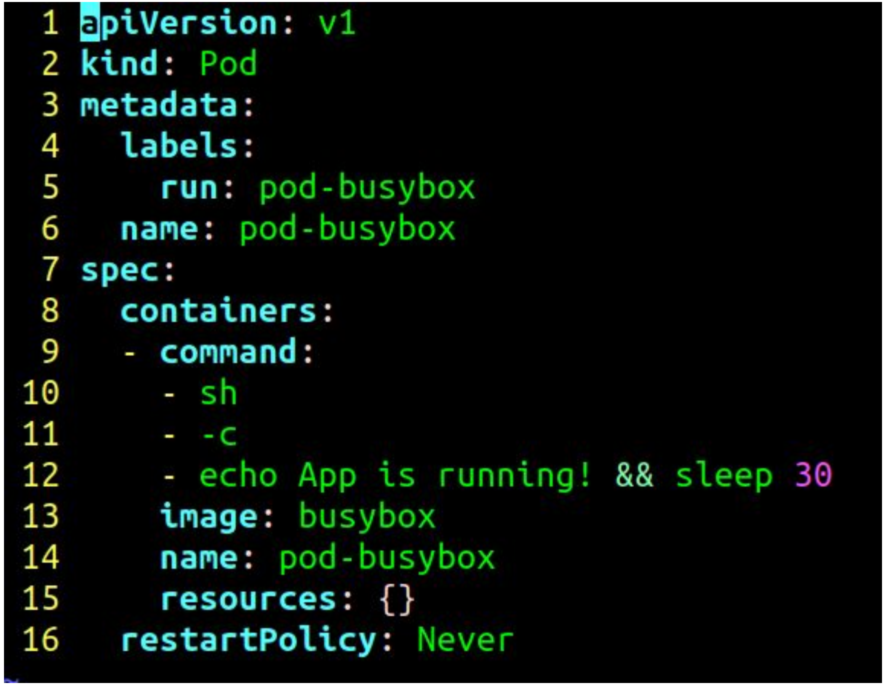
- Networking
- Storage

Example



```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

Example

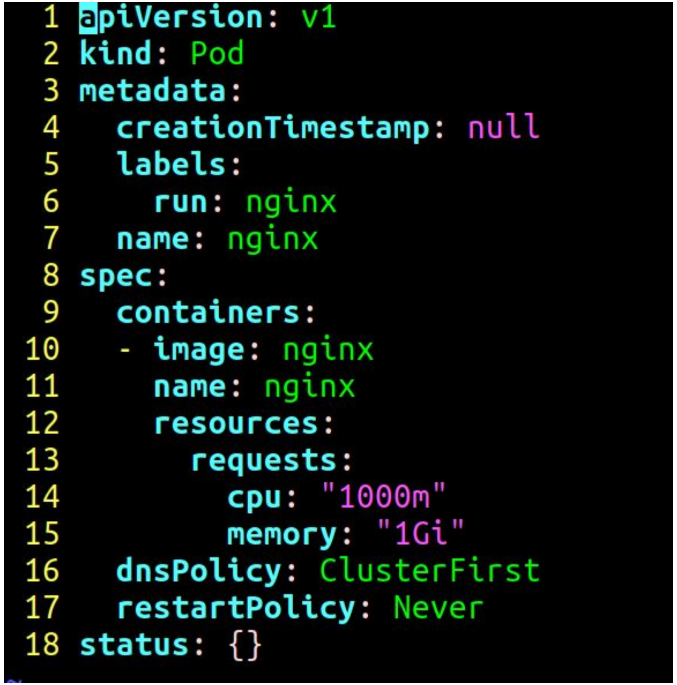


```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: pod-busybox
  name: pod-busybox
spec:
  containers:
  - command:
    - sh
    - -c
    - echo App is running! && sleep 30
    image: busybox
    name: pod-busybox
    resources: {}
  restartPolicy: Never
```

POD - Lifecycle

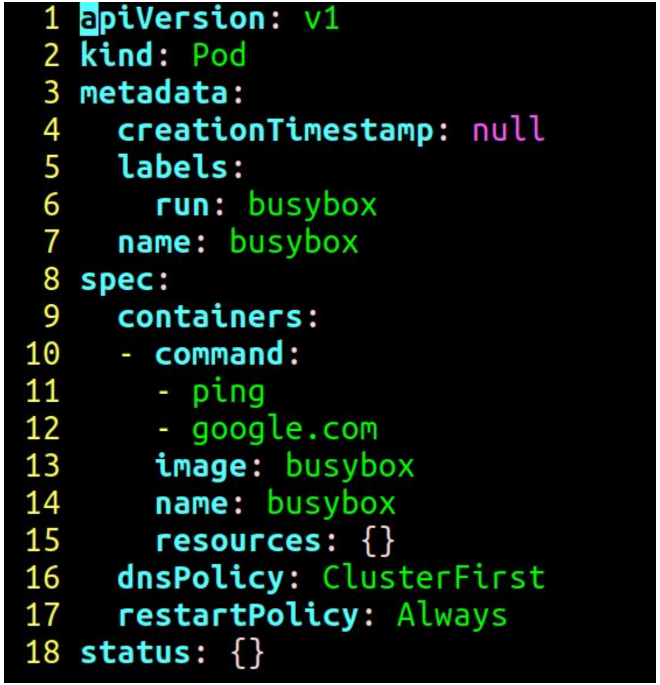
- Phase
  - Pending (waiting to be scheduled, image downloading)
  - Running (all containers started and ready to serve)
  - Succeeded (all containers exited with success)
  - Failed (all containers exited but at least one with failure)
  - Unknown (unable to fetch status as node is unreachable)
- Container States
  - Waiting, Running, Terminated
- Restart Policy (**Always**, Never, OnFailure)
- Conditions
  - Type: PodScheduled, ContainersReady, Initialized, Ready
    (lastProbeTime, lastTransitionTime, Message, reason, status)
- Probes
  - Startup, Readiness, Liveness
- Lifecycle hooks

Phase - Pending



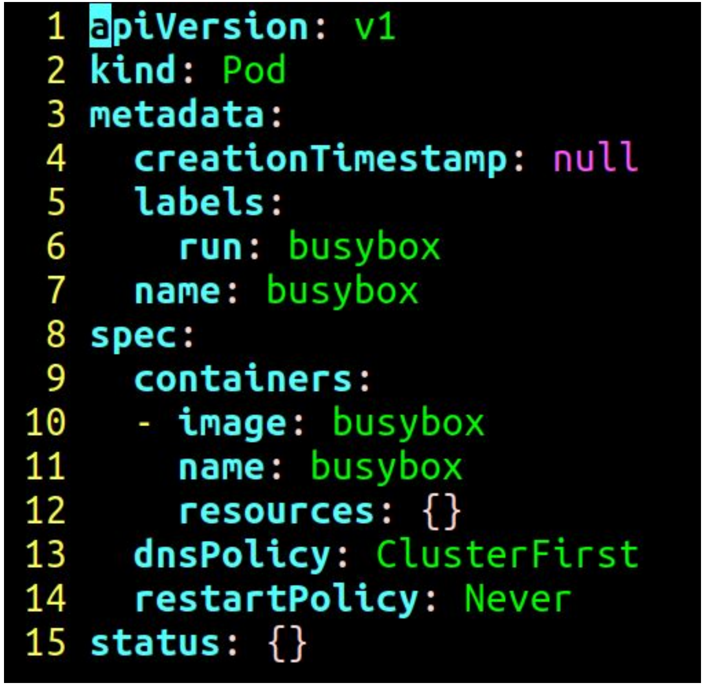
```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
    resources:
      requests:
        cpu: "1000m"
        memory: "1Gi"
  dnsPolicy: ClusterFirst
  restartPolicy: Never
status: {}
```

Phase - Running



```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: busybox
  name: busybox
spec:
  containers:
  - command:
    - ping
    - google.com
    image: busybox
    name: busybox
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

Phase - Succeeded



```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: busybox
  name: busybox
spec:
  containers:
  - image: busybox
    name: busybox
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Never
status: {}
```

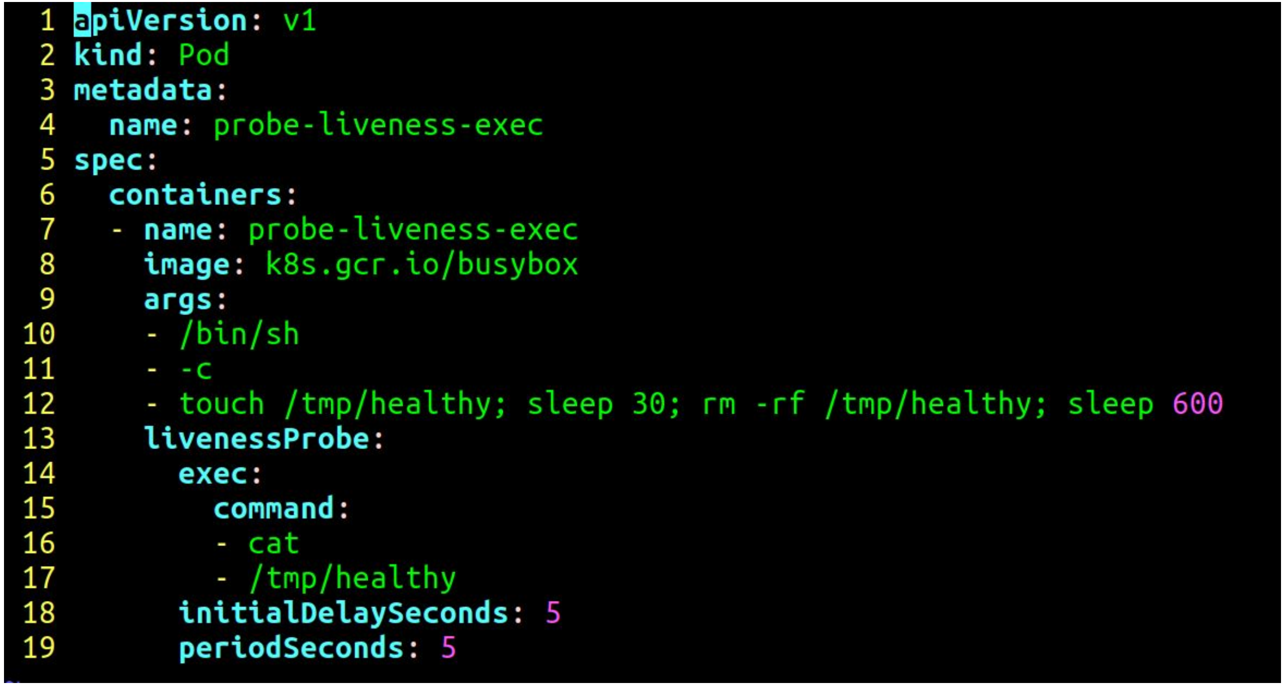
Probes

- Types
  - Startup
  - Readiness
  - Liveness
- Methods
  - Http
  - Tcp
  - Command

- Settings
  - initialDelaySeconds
  - periodSeconds
  - timeoutSeconds
  - successThreshold
  - failureThreshold

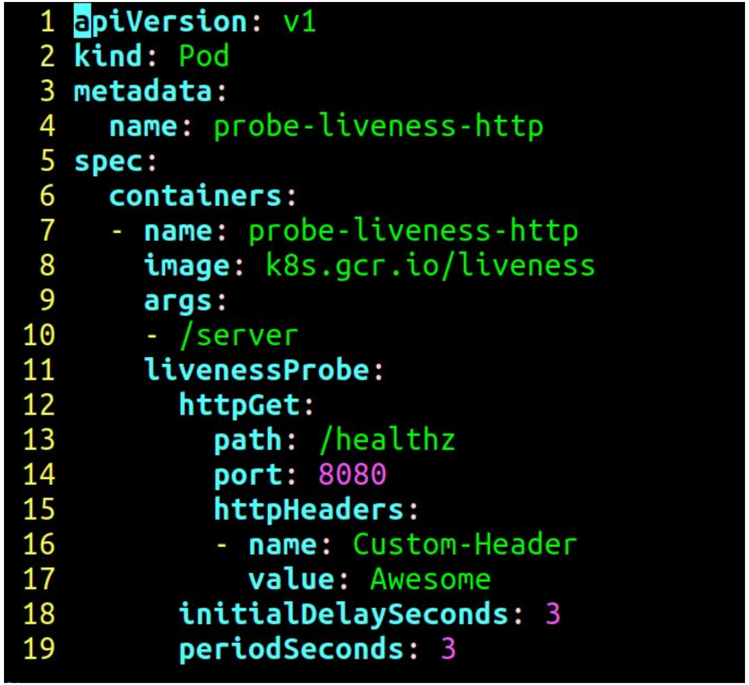
- Http
  - Host
  - Scheme
  - Path
  - Port
  - Headers

Probe - Liveness - Exec



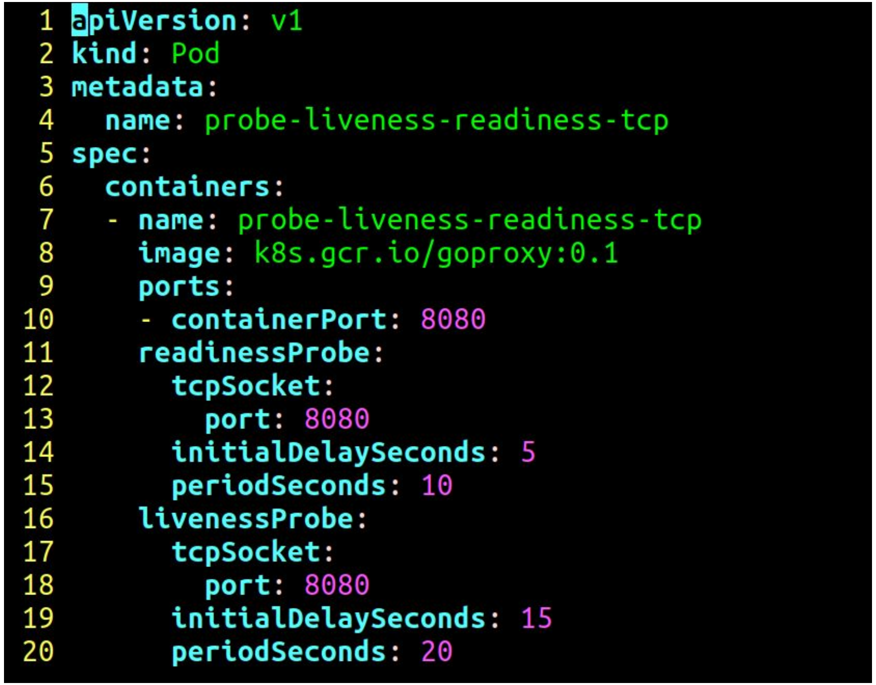
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: probe-liveness-exec
spec:
  containers:
  - name: probe-liveness-exec
    image: k8s.gcr.io/busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
```

Probe - Liveness - http



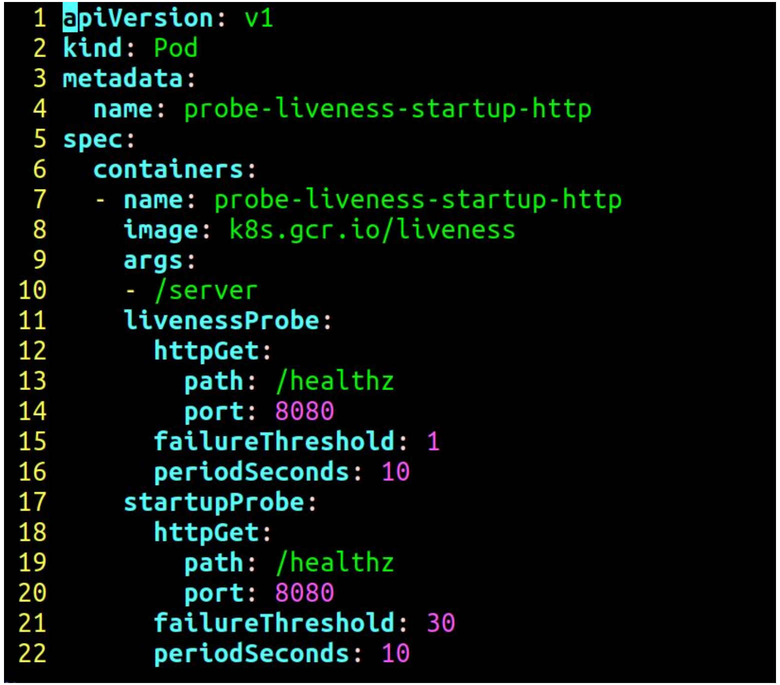
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: probe-liveness-http
spec:
  containers:
  - name: probe-liveness-http
    image: k8s.gcr.io/liveness
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
        - name: Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
```

Probe - Liveness - readiness - http



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: probe-liveness-readiness-tcp
spec:
  containers:
  - name: probe-liveness-readiness-tcp
    image: k8s.gcr.io/goproxy:0.1
    ports:
    - containerPort: 8080
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 10
    livenessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 15
      periodSeconds: 20
```

Probe - Liveness - startup - http



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: probe-liveness-startup-http
spec:
  containers:
  - name: probe-liveness-startup-http
    image: k8s.gcr.io/liveness
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
      failureThreshold: 1
      periodSeconds: 10
    startupProbe:
      httpGet:
        path: /healthz
        port: 8080
      failureThreshold: 30
      periodSeconds: 10
```

POD Init Containers

- Always run to completion
- Must complete successfully before next one
- Readiness probes not supported
- Run(s) before application containers

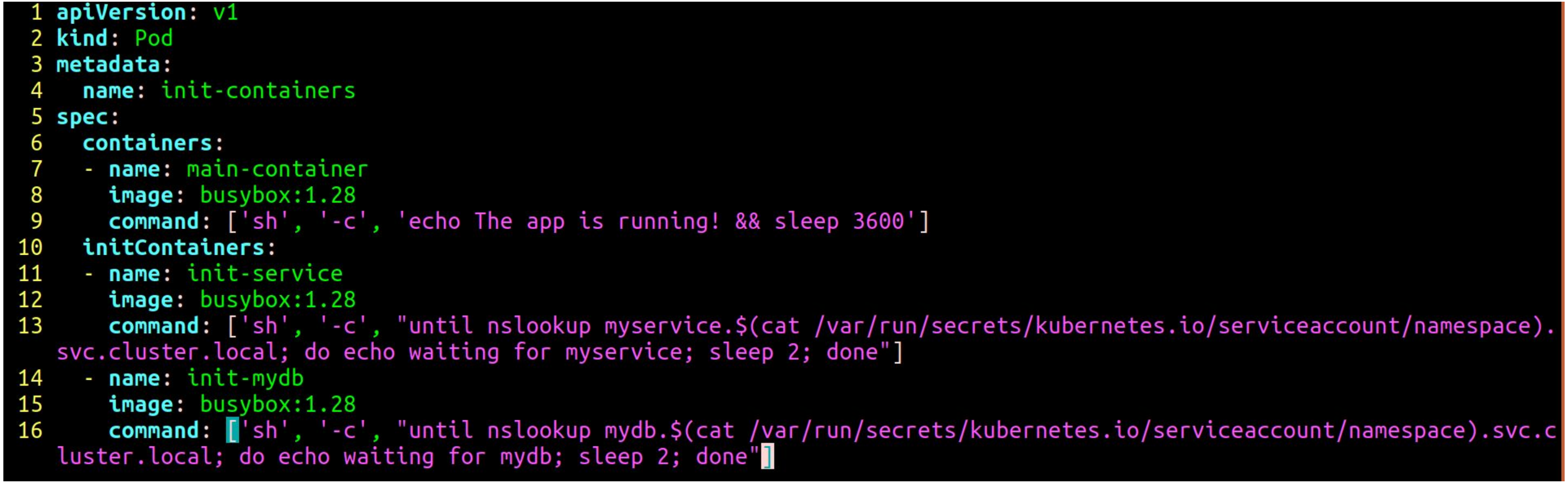
Examples:

- Custom code / utilities to run before app containers
- Block / delay app container startup
- App container image building can be separate

POD Init - Statuses

- Init:N/M
- Init:Error
- Init:CrashLoopBackOff
- Pending
- PodIniliazing
- Running

Example

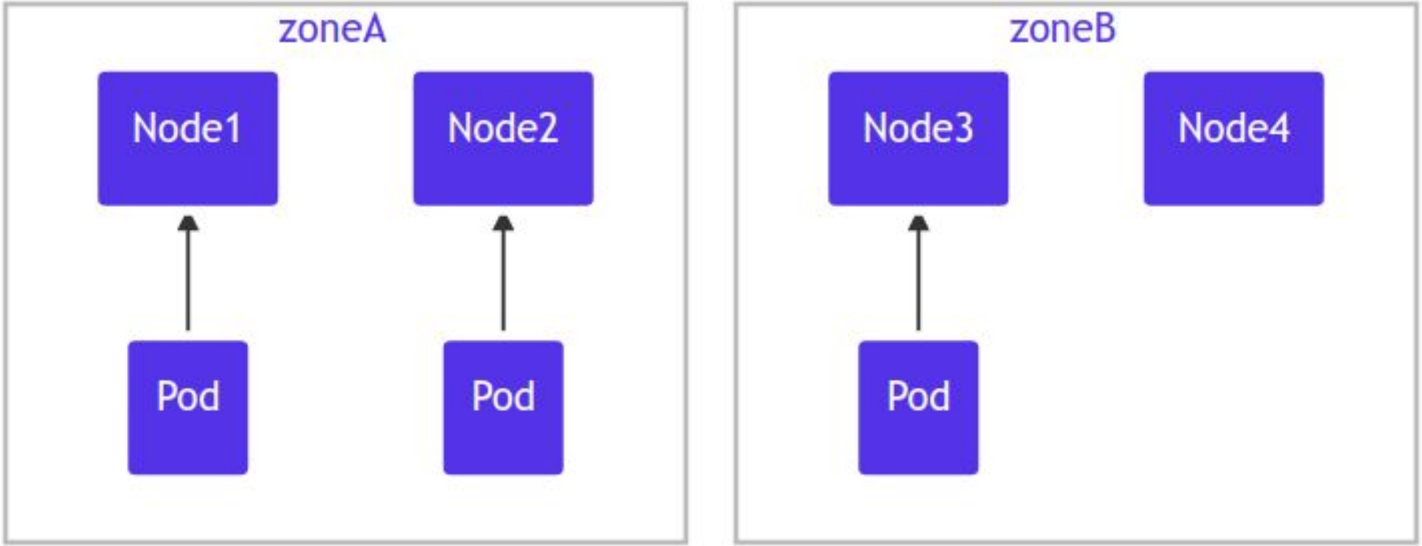


```
apiVersion: v1
kind: Pod
metadata:
  name: init-containers
spec:
  containers:
  - name: main-container
    image: busybox:1.28
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-service
    image: busybox:1.28
    command: ['sh', '-c', "until nslookup myservice.$(cat /var/run/secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.local; do echo waiting for myservice; sleep 2; done"]
  - name: init-mydb
    image: busybox:1.28
    command: ['sh', '-c', "until nslookup mydb.$(cat /var/run/secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.local; do echo waiting for mydb; sleep 2; done"]
```

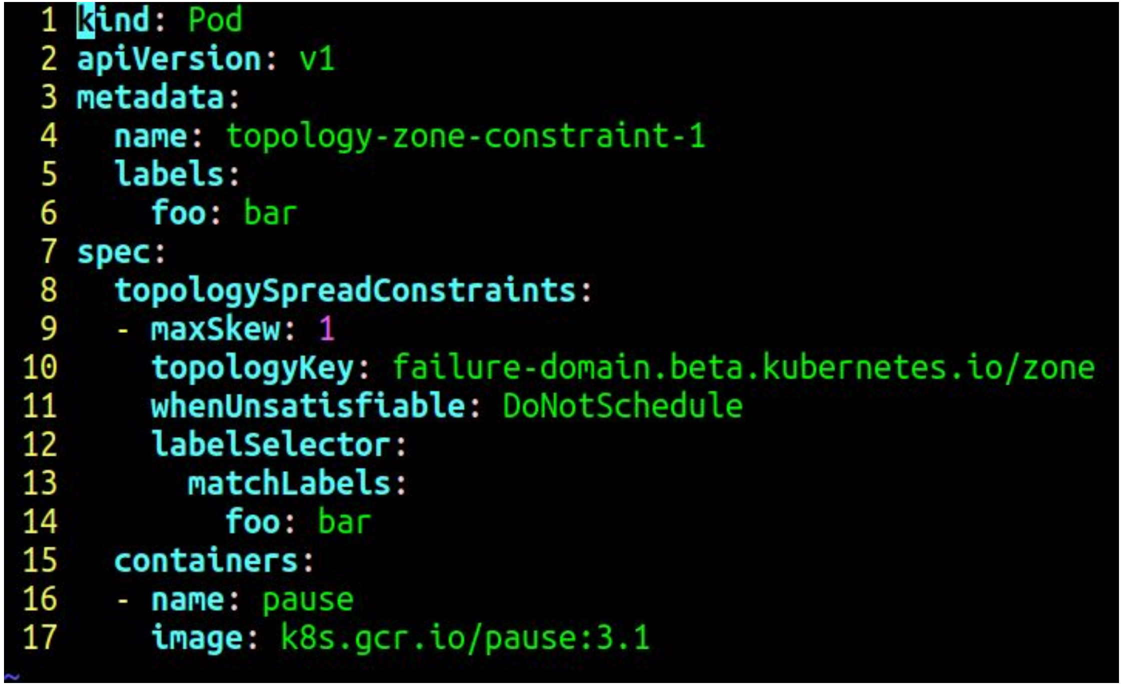
POD - Topology Spread

- Objectives
  - To control how Pods are spreads across regions, zones, nodes and other user defined topology domains
  - To achieve high availability
  - To achieve efficient resource utilization
- Spread Constraints
  - maxSkew
  - topologyKey
  - whenUnsatisfiable (DoNotSchedule / ScheduleAnyway)
  - labelSelector

POD - Topology Spread

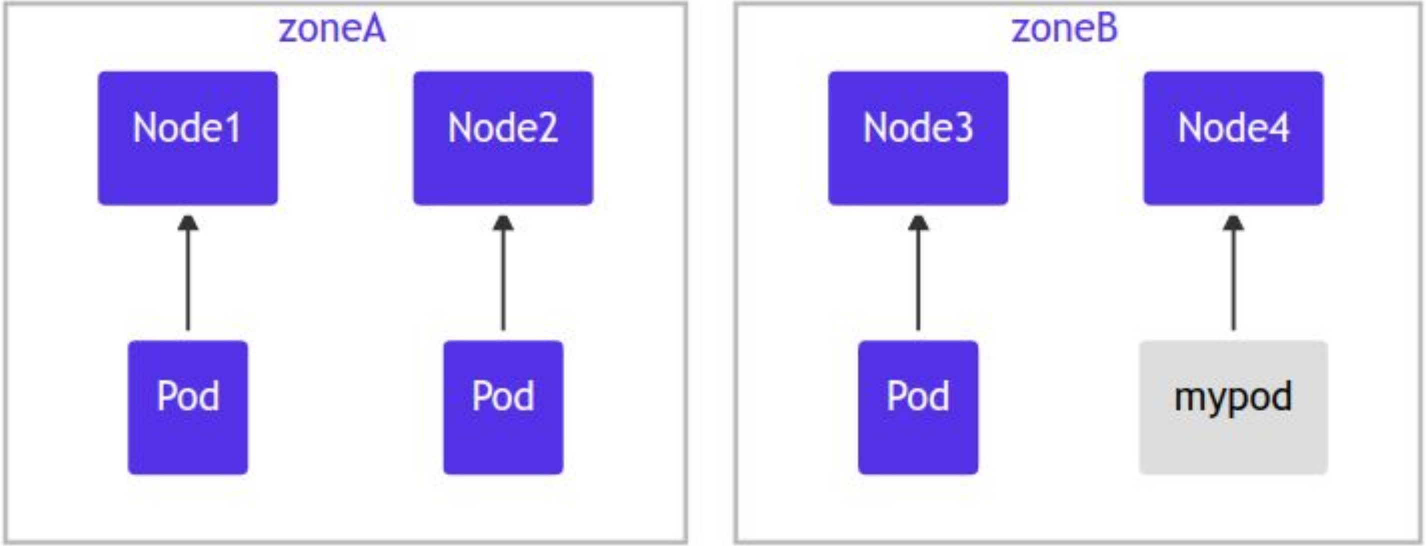


POD - Topology Spread

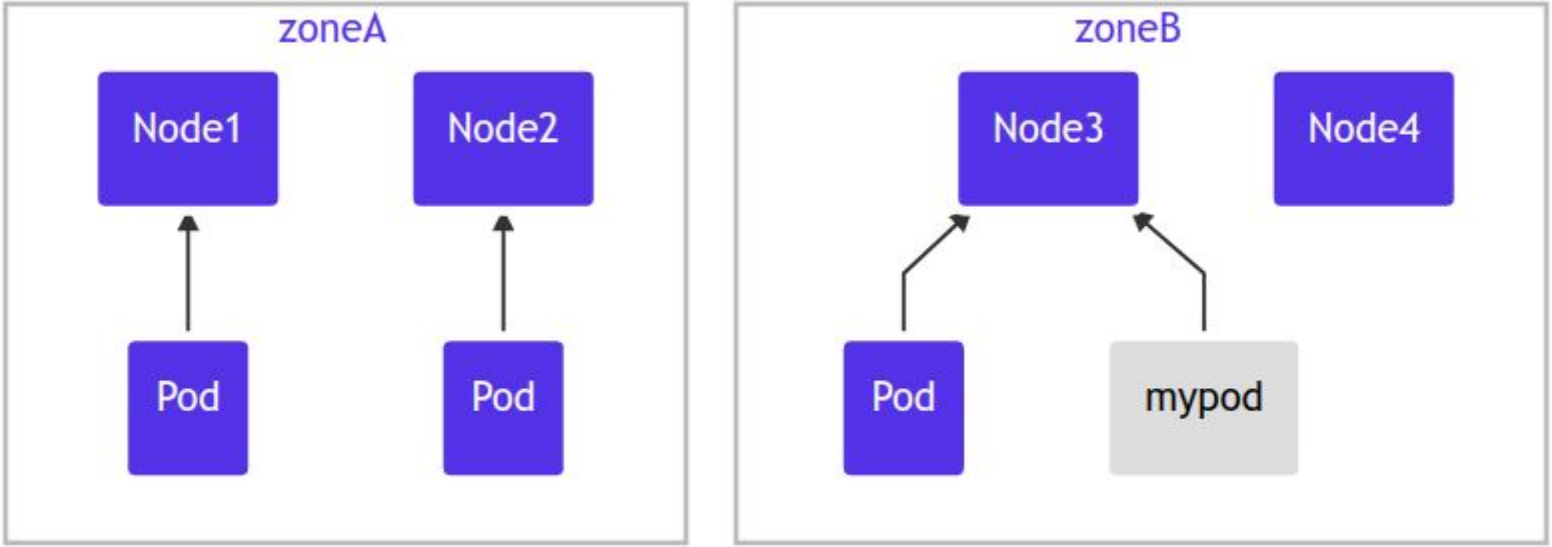


```yaml
kind: Pod
apiVersion: v1
metadata:
  name: topology-zone-constraint-1
  labels:
    foo: bar
spec:
  topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: failure-domain.beta.kubernetes.io/zone
    whenUnsatisfiable: DoNotSchedule
    labelSelector:
      matchLabels:
        foo: bar
  containers:
  - name: pause
    image: k8s.gcr.io/pause:3.1
```

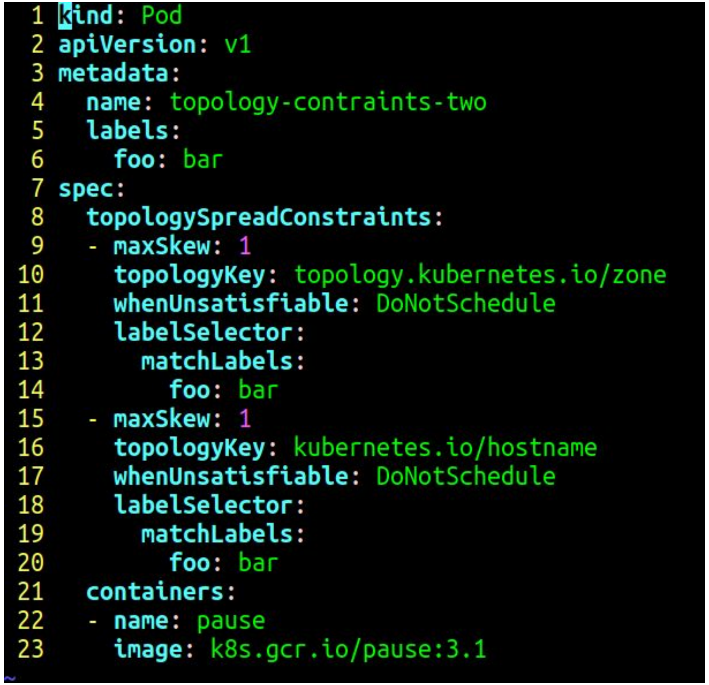
POD - Topology Spread



POD - Topology Spread

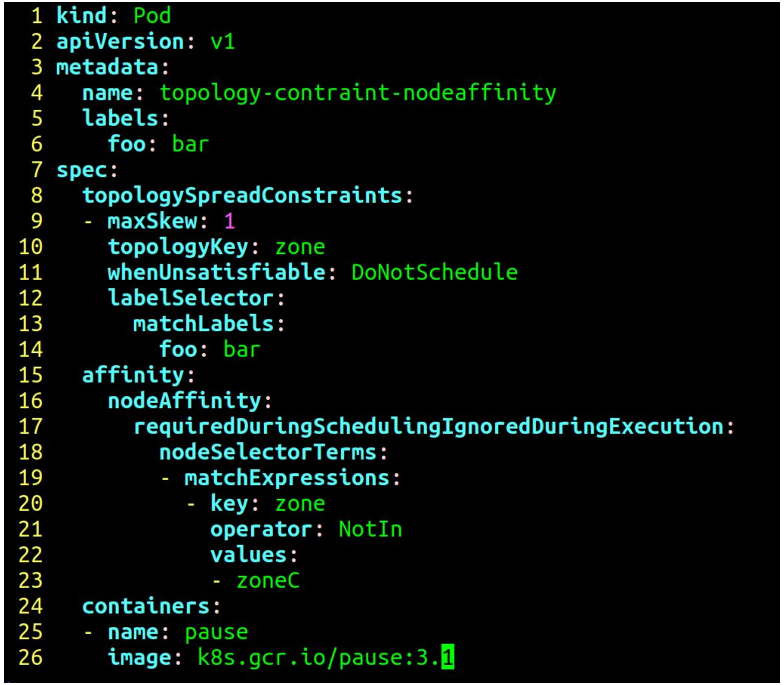


Topology - Multiple Constraints



```yaml
kind: Pod
apiVersion: v1
metadata:
  name: topology-contraints-two
  labels:
    foo: bar
spec:
  topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: topology.kubernetes.io/zone
    whenUnsatisfiable: DoNotSchedule
    labelSelector:
      matchLabels:
        foo: bar
  - maxSkew: 1
    topologyKey: kubernetes.io/hostname
    whenUnsatisfiable: DoNotSchedule
    labelSelector:
      matchLabels:
        foo: bar
  containers:
  - name: pause
    image: k8s.gcr.io/pause:3.1
```

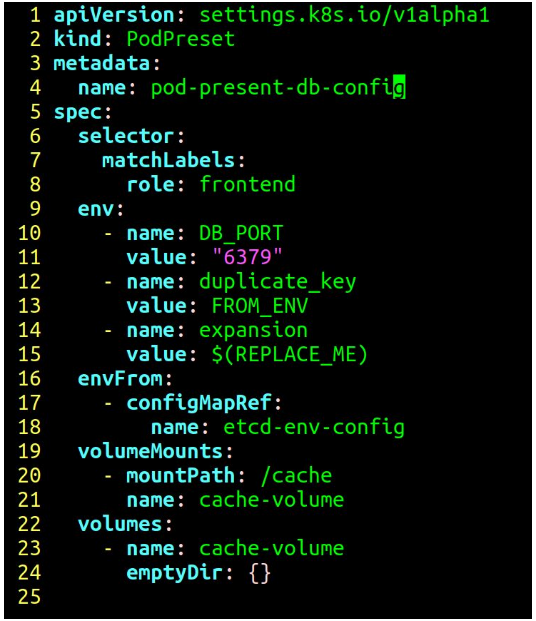
Topology - Constraint & Node Affinity



```yaml
kind: Pod
apiVersion: v1
metadata:
  name: topology-contraint-nodeaffinity
  labels:
    foo: bar
spec:
  topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: zone
    whenUnsatisfiable: DoNotSchedule
    labelSelector:
      matchLabels:
        foo: bar
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: zone
            operator: NotIn
            values:
            - zoneC
  containers:
  - name: pause
    image: k8s.gcr.io/pause:3.1
```

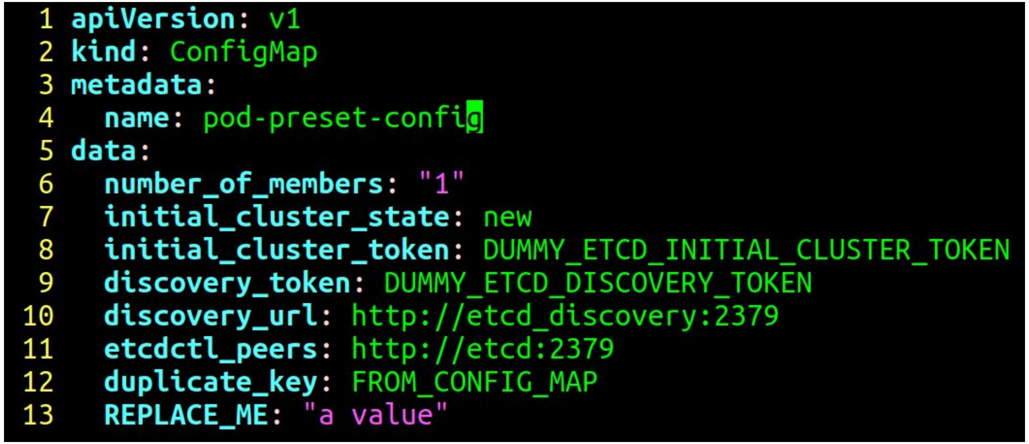
POD Preset

- To inject certain info at creation time
- Can include secrets, volume, volume mounts, environment variables



```yaml
apiVersion: settings.k8s.io/v1alpha1
kind: PodPreset
metadata:
  name: pod-present-db-config
spec:
  selector:
    matchLabels:
      role: frontend
  env:
    - name: DB_PORT
      value: "6379"
    - name: duplicate_key
      value: FROM_ENV
    - name: expansion
      value: $(REPLACE_ME)
  envFrom:
    - configMapRef:
        name: etcd-env-config
  volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
    - name: cache-volume
      emptyDir: {}

```

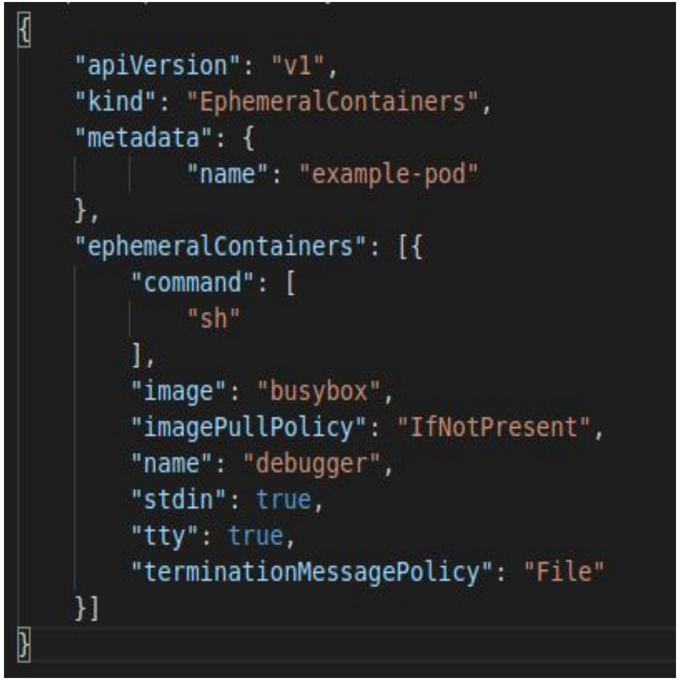


```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: pod-preset-config
data:
  number_of_members: "1"
  initial_cluster_state: new
  initial_cluster_token: DUMMY_ETCD_INITIAL_CLUSTER_TOKEN
  discovery_token: DUMMY_ETCD_DISCOVERY_TOKEN
  discovery_url: http://etcd_discovery:2379
  etcdctl_peers: http://etcd:2379
  duplicate_key: FROM_CONFIG_MAP
  REPLACE_ME: "a value"
```

POD - Ephemeral

- Meant for interactive troubleshooting inside POD
- No resource guarantees
- Never restart automatically
- Process Namespace sharing

```
kubectl replace --raw /api/v1/namespaces/default/pods/example-pod/ephemeralcontainers -f ec.json
```



```
{
    "apiVersion": "v1",
    "kind": "EphemeralContainers",
    "metadata": {
            "name": "example-pod"
    },
    "ephemeralContainers": [{
        "command": [
            "sh"
        ],
        "image": "busybox",
        "imagePullPolicy": "IfNotPresent",
        "name": "debugger",
        "stdin": true,
        "tty": true,
        "terminationMessagePolicy": "File"
    }]
}
```

Exercises (30 mins)

- Create POD with init containers
  - Main app container from tutum/hello-world
  - Init container using busybox to fetch google.com/index.html and save to /www/google.html
- Create POD to keep in different state (Pending, Running, Succeeded)
- Create POD with postStart and preStop hooks
- Create POD with readiness probe for container readiness requiring 60 seconds
- Create POD and restart if not live after 30 seconds having 10 s interval
- Create POD that requires 90 s to start but restart every 30 s if not live
- Create two PODs and make sure they are not deployed on the same node
- Create db config and inject it into two PODs

Multi Container Pods

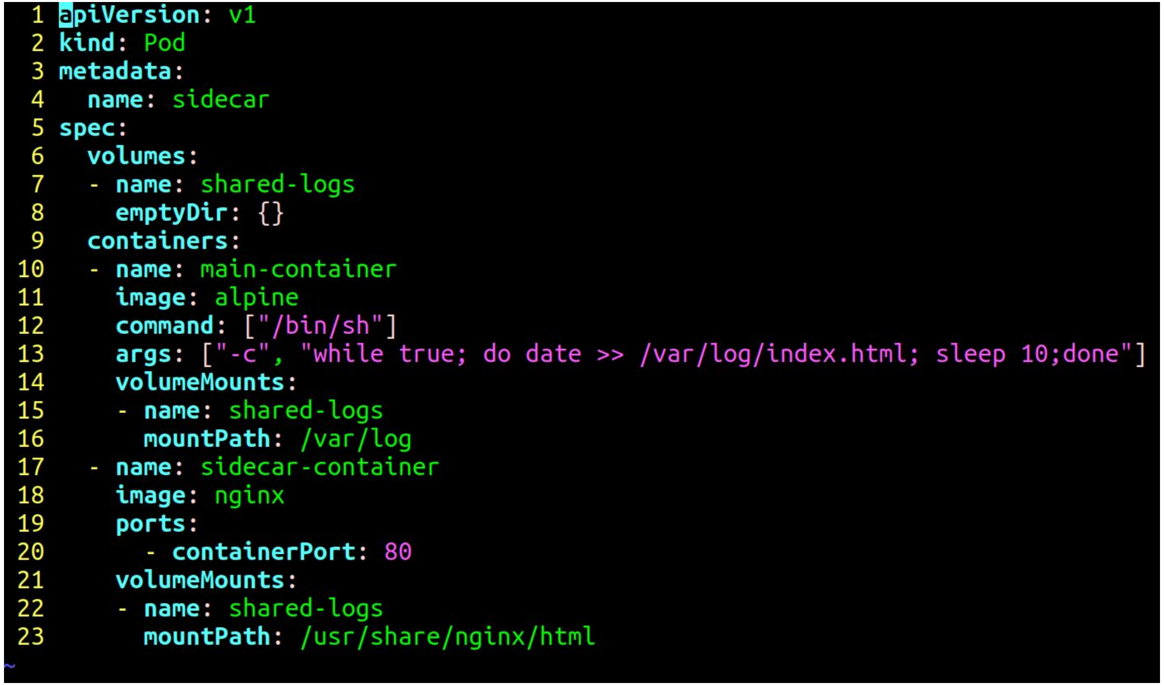
- Ambassador
- Sidecar
- Adapter
- Demo
- Practicals

Patterns - POD

- To extend the functionality of the existing container
- To have helper process enhancing work of the existing container
- To send logs to external server

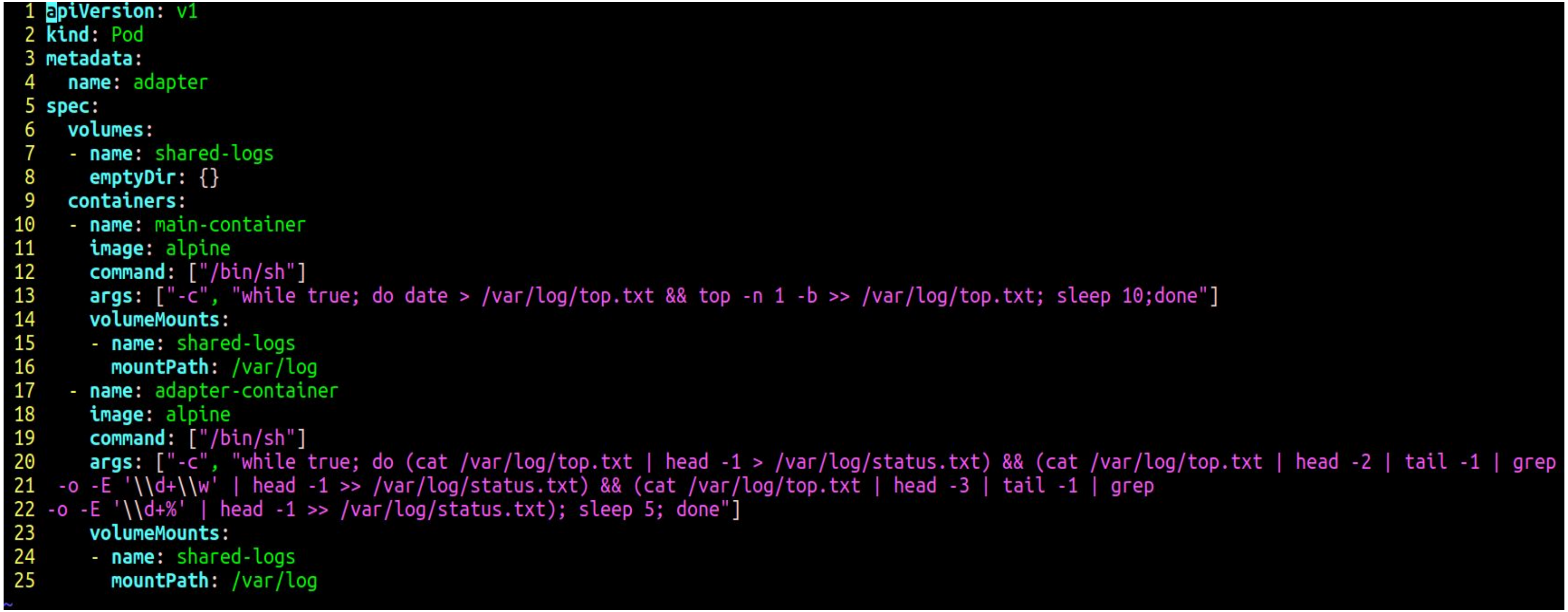
- Types
  - Sidecar - To export logs
  - Ambassador - To proxy connection
  - Adapter - To standardise and normalize output

Pattern - Sidecar



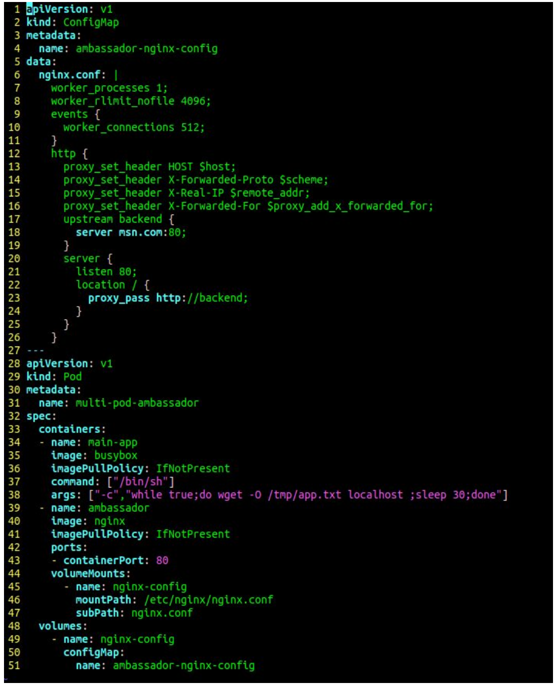
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: sidecar
spec:
  volumes:
  - name: shared-logs
    emptyDir: {}
  containers:
  - name: main-container
    image: alpine
    command: ["/bin/sh"]
    args: ["-c", "while true; do date >> /var/log/index.html; sleep 10;done"]
    volumeMounts:
    - name: shared-logs
      mountPath: /var/log
  - name: sidecar-container
    image: nginx
    ports:
      - containerPort: 80
    volumeMounts:
    - name: shared-logs
      mountPath: /usr/share/nginx/html
```

Pattern - Adapter



```
apiVersion: v1
kind: Pod
metadata:
  name: adapter
spec:
  volumes:
  - name: shared-logs
    emptyDir: {}
  containers:
  - name: main-container
    image: alpine
    command: ["/bin/sh"]
    args: ["-c", "while true; do date > /var/log/top.txt && top -n 1 -b >> /var/log/top.txt; sleep 10;done"]
    volumeMounts:
    - name: shared-logs
      mountPath: /var/log
  - name: adapter-container
    image: alpine
    command: ["/bin/sh"]
    args: ["-c", "while true; do (cat /var/log/top.txt | head -1 > /var/log/status.txt) && (cat /var/log/top.txt | head -2 | tail -1 | grep
 -o -E '\\d+\\w' | head -1 >> /var/log/status.txt) && (cat /var/log/top.txt | head -3 | tail -1 | grep
-o -E '\\d+%' | head -1 >> /var/log/status.txt); sleep 5; done"]
    volumeMounts:
    - name: shared-logs
      mountPath: /var/log
~
```

Pattern - Ambassador



```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ambassador-nginx-config
data:
  nginx.conf: |
    worker_processes 1;
    worker_rlimit_nofile 4096;
    events {
      worker_connections 512;
    }
    http {
      proxy_set_header HOST $host;
      proxy_set_header X-Forwarded-Proto $scheme;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      upstream backend {
        server msn.com:80;
      }
      server {
        listen 80;
        location / {
          proxy_pass http://backend;
        }
      }
    }
---
apiVersion: v1
kind: Pod
metadata:
  name: multi-pod-ambassador
spec:
  containers:
  - name: main-app
    image: busybox
    imagePullPolicy: IfNotPresent
    command: ["/bin/sh"]
    args: ["-c","while true;do wget -O /tmp/app.txt localhost ;sleep 30;done"]
  - name: ambassador
    image: nginx
    imagePullPolicy: IfNotPresent
    ports:
    - containerPort: 80
    volumeMounts:
      - name: nginx-config
        mountPath: /etc/nginx/nginx.conf
        subPath: nginx.conf
  volumes:
    - name: nginx-config
      configMap:
        name: ambassador-nginx-config
```

Exercises

- Create POD having below
    - Main container: busybox
        - Appending ping outputs to /tmp/index-input.html
    - Sidecar container: busybox
        - Copy the index.html to /tmp/index-output.html
- Create POD having below
    - Main container: busybox
        - Appending ping outputs to /tmp/index.html
    - Adapter container: busybox
        - Copy the index.html to /tmp/index-output.html
- Create POD having below
    - Main container: busybox
        - Storing localhost ping outputs to /tmp/index-input.html
    - Ambassador container: nginx
        - Proxying localhost:80 to https://www.brainupgrade.in

Pod Design

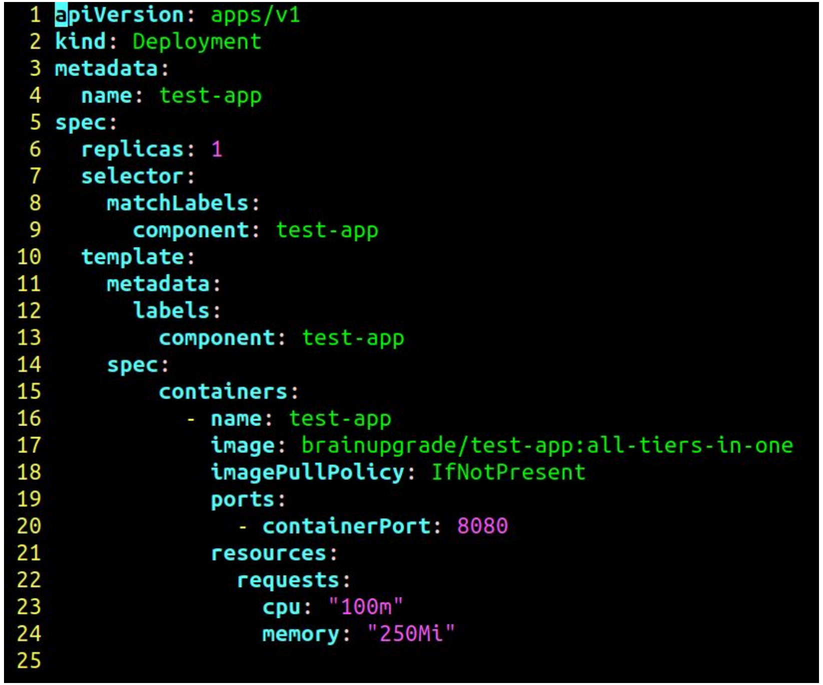
- Deployments
- Rolling Updates & Rollbacks
- App Deployment - multi node
- Auto scaling pods
- Auto scaling pod across availability zones
- Jobs, Cron Jobs
- Labels, Selectors, Annotations
- Demo
- Practicals

Deployments

Use Cases

- To rollout a set of PODs
- To declare a new set of PODs
- To rollback to an earlier version of deployment
- To scale up deployment to facilitate more load
- To pause the deployment / rollout
- To autoscale deployment when cpu usage threshold reached

Deployment - Example



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-app
spec:
  replicas: 1
  selector:
    matchLabels:
      component: test-app
  template:
    metadata:
      labels:
        component: test-app
    spec:
        containers:
          - name: test-app
            image: brainupgrade/test-app:all-tiers-in-one
            imagePullPolicy: IfNotPresent
            ports:
              - containerPort: 8080
            resources:
              requests:
                cpu: "100m"
                memory: "250Mi"
```

Deployment - Commands

- kubectl create deployment nginx --image=nginx:1.15
- kubectl get deployment/nginx
- kubectl describe deployment/nginx
- kubectl rollout history deployment/nginx
- kubectl set image deployment/nginx nginx=nginx:1.16
- kubectl rollout history deployment/ nginx
- kubectl rollout undo deployment/nginx
- kubectl rollout undo deployment/nginx --to-revision=2
- kubectl rollout pause deployment/nginx
- kubectl rollout resume deployment/nginx
- kubectl scale --replicas=3 deployment/nginx
- kubectl autoscale deployment/nginx --min=2 --max=10

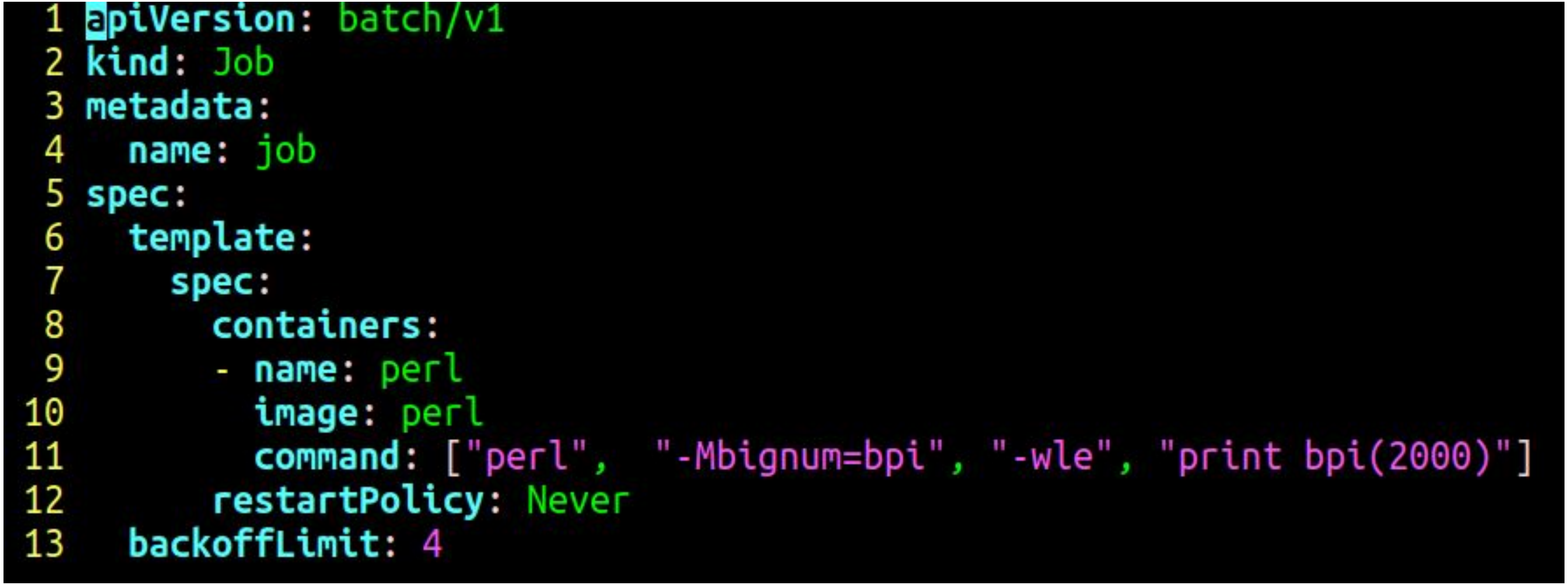
Job

- To provide reliable parallel execution of tasks

- Examples:
  - Send emails, transcode files, Scan database for a set of rows,

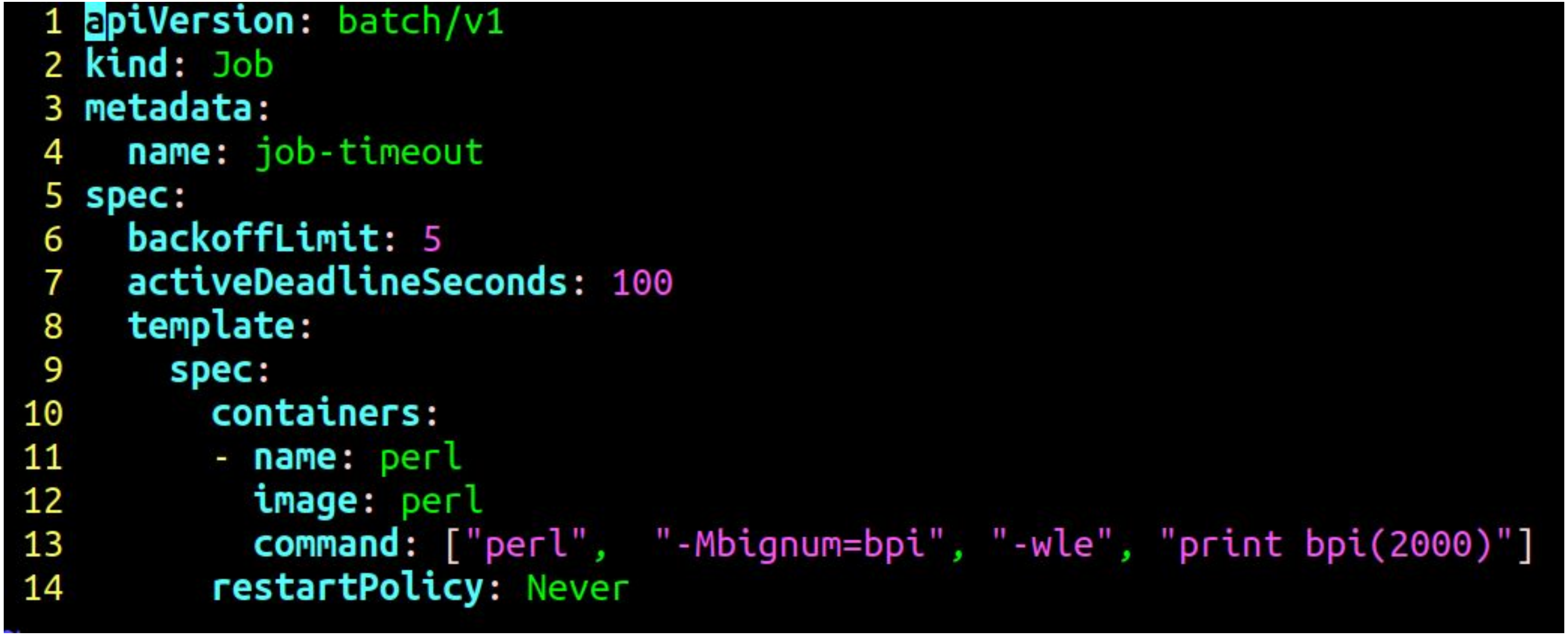
- Patterns
  - Non parallel job
  - Fixed completion count job
  - Work queue job

Job



```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: job
spec:
  template:
    spec:
      containers:
      - name: perl
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

Job - Timeout



```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: job-timeout
spec:
  backoffLimit: 5
  activeDeadlineSeconds: 100
  template:
    spec:
      containers:
      - name: perl
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
```

CronJob

- Creates jobs on a repeating schedule
- Schedule times are based on kube-controller-manager
- Useful for tasks like migrating data to reporting server, sending emails, creating backups etc
- Schedule tasks at specific time (like when cluster is idle)

Key Configurations:

- startingDeadlineSeconds - Missed occurrences in last X seconds will be counted
- concurrencyPolicy
  - If Allow, then job will run at least once
  - If Forbid, will be missed if previous instance is still running

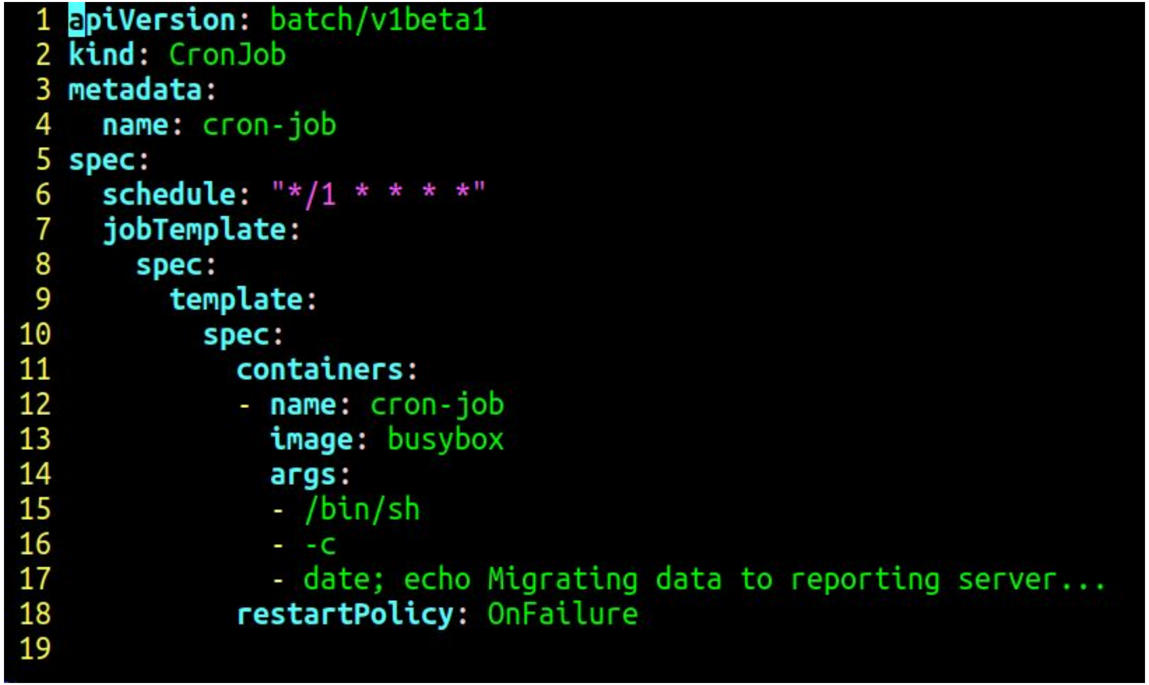
CronJob - Expression

```
# ┌───────────── minute (0 - 59)
# │ ┌───────────── hour (0 - 23)
# │ │ ┌───────────── day of the month (1 - 31)
# │ │ │ ┌───────────── month (1 - 12)
# │ │ │ │ ┌───────────── day of the week (0 - 6) (Sunday to Saturday;
# │ │ │ │ │                                   7 is also Sunday on some systems)
# │ │ │ │ │
# │ │ │ │ │
  # * * * * * <command to execute>
```

Examples:

- `*/15 0,8,16 * * * echo running backup    (every 15 minutes of 0,8 & 16th hour)`
- `30 0 * * 6 /home/oracle/scripts/export_dump.sh   (last day of week at 00:30)`
- `1 0 * * * printf "" > /var/log/apache/error_log   (everyday at 00:01 )`

CronJob - Expression



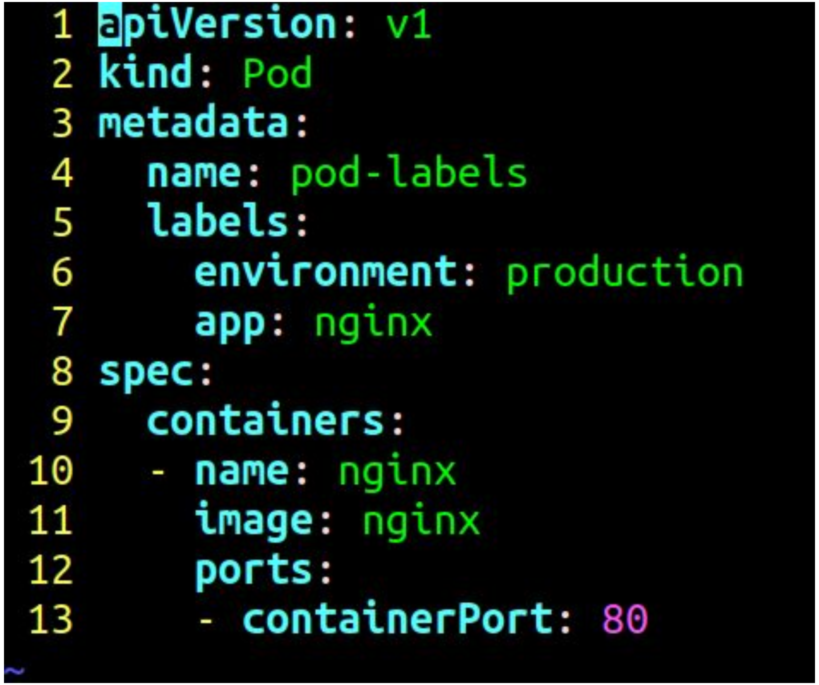
```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: cron-job
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: cron-job
            image: busybox
            args:
            - /bin/sh
            - -c
            - date; echo Migrating data to reporting server...
          restartPolicy: OnFailure

```

Labels

- Labels
  - Key value pairs attached to objects
  - To specify identifying attributes of objects
  - To organize and select subset of objects
  - To query objects efficiently (cli as well gui monitoring tools)
  - Attached at creation time and can be added / modified at any time
  - Label key must be unique per object
- Example labels:
  - `"release" : "stable", "release" : "canary"`
  - `"environment" : "dev", "environment" : "qa", "environment" : "production"`
  - `"tier" : "frontend", "tier" : "backend", "tier" : "cache"`
  - `"partition" : "customerA", "partition" : "customerB"`
  - `"track" : "daily", "track" : "weekly"`

Labels



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-labels
  labels:
    environment: production
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

Selectors

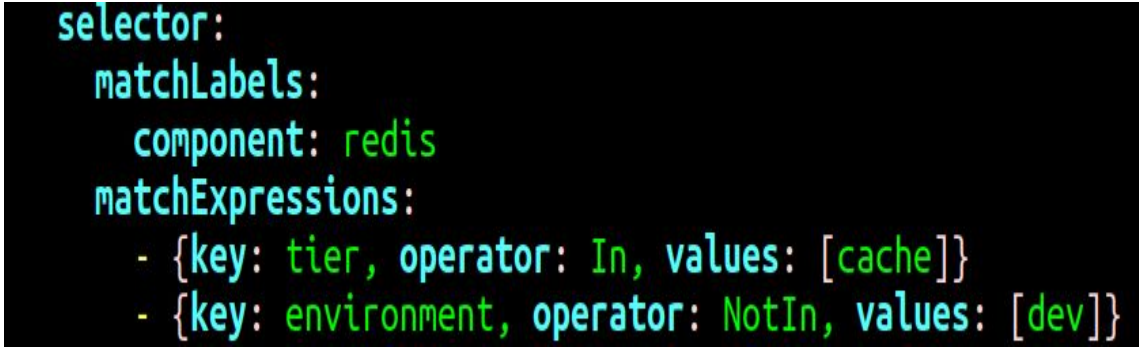
- Equality Based
  - `environment = production`
  - `tier != frontend`

- Set Based
  - `environment in (production, qa)`
  - `tier notin (frontend, backend)`
  - `partition`
  - `!partition`

Selectors - Examples

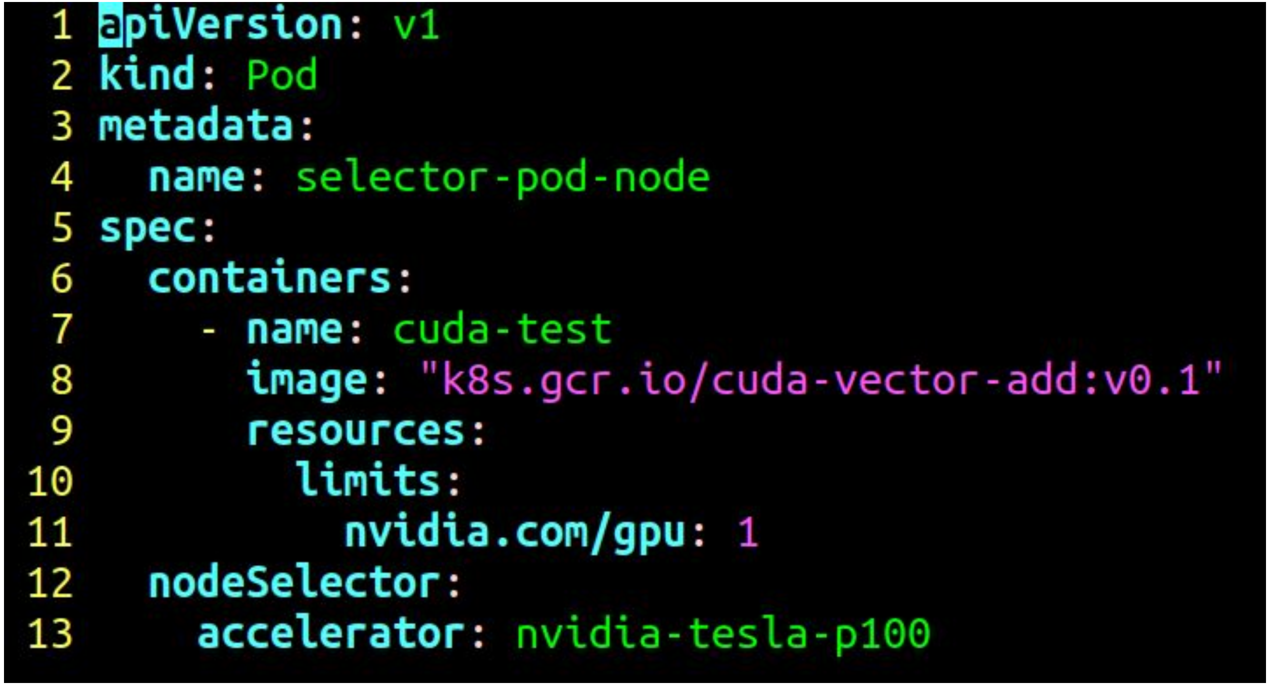
- `kubectl get pods -l environment=production,tier=frontend`
- `kubectl get pods -l 'environment in (production),tier in (frontend)'`
- `kubectl get pods -l 'environment in (production, qa)'`
- `kubectl get pods -l 'environment,environment notin (frontend)'`

Jobs, Deployments, ReplicaSet, Daemonset



```yaml
selector:
  matchLabels:
    component: redis
  matchExpressions:
    - {key: tier, operator: In, values: [cache]}
    - {key: environment, operator: NotIn, values: [dev]}
```

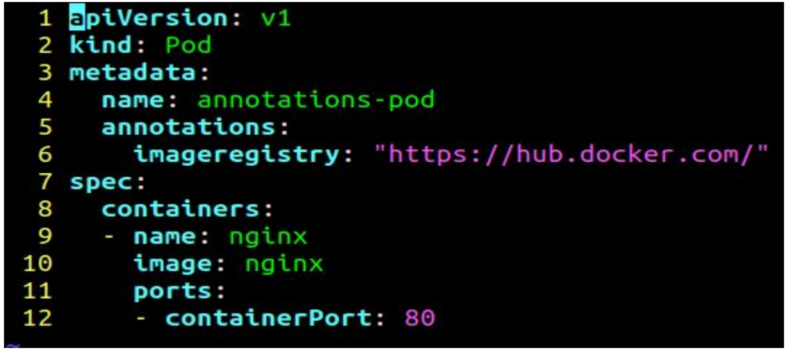
Selectors - Examples



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: selector-pod-node
spec:
  containers:
    - name: cuda-test
      image: "k8s.gcr.io/cuda-vector-add:v0.1"
      resources:
        limits:
          nvidia.com/gpu: 1
  nodeSelector:
    accelerator: nvidia-tesla-p100
```

Annotations

- To attach non-identifying arbitrary metadata to objects
- Usage
  - Pointers for debugging purposes
  - Build, release, image hashes etc
  - Author info, contact details
  - Metadata to help tools for deployment, management, introspection



```
apiVersion: v1
kind: Pod
metadata:
  name: annotations-pod
  annotations:
    imageregistry: "https://hub.docker.com/"
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

Exercises (30 mins)

- Create POD with init containers
  - Main app container from tutum/hello-world
  - Init container using busybox to fetch google.com/index.html and save to /www/google.html
- Create a deployment to run 3 replicas of nginx container
- Scale down the replicas to 1
- Scale up replicas to 8
- View the roll out history
- Switch to rollout version 2
- View deployments, rc, pod using kubectl
- Scale down replicas to 5
- Update image to nginx:1.9.1 and immediately try another rollout with nginx:1.7.1
- Observe if rollout with nginx:1.9.1 was completed or not

Persistence

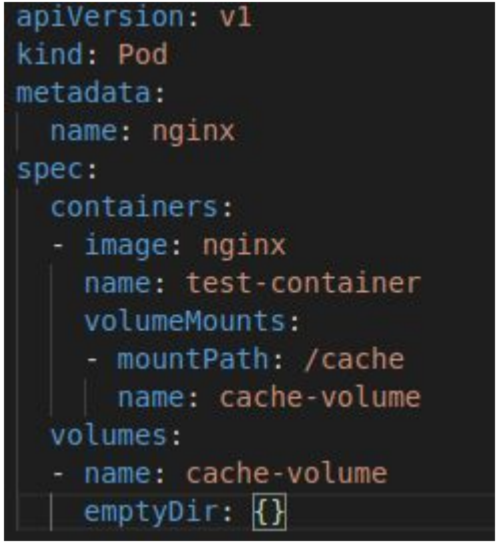
- Persistence Volume
- Persistence Volume Claim
- Statefulset
- Daemonset

Overview

- Ephemeral
  - Tightly coupled with POD lifetime
  - Deleted when POD is removed
  - Example: emptydir

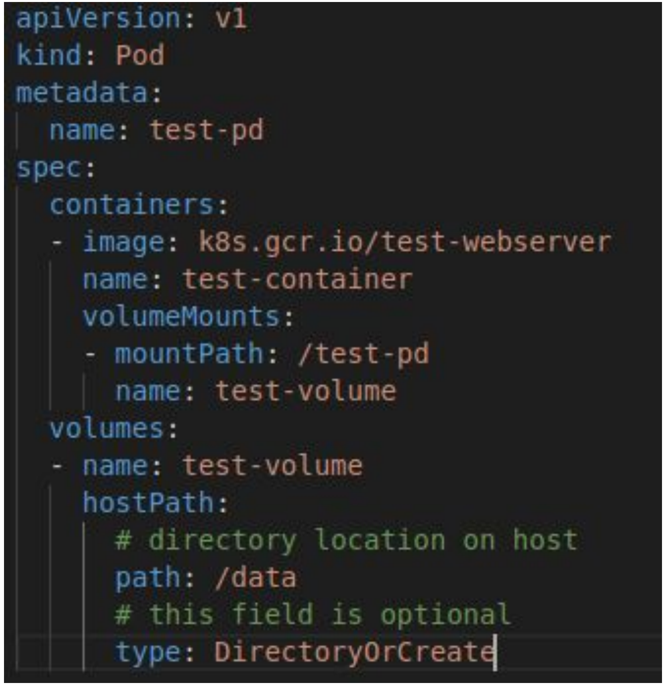
- Persistent
  - Survives POD reboots
  - Meant for long term and independent of POD / Node lifecycle
  - Examples: hostpath, local, NFS, Cloud storage (EBS etc)

Examples - emptyDir



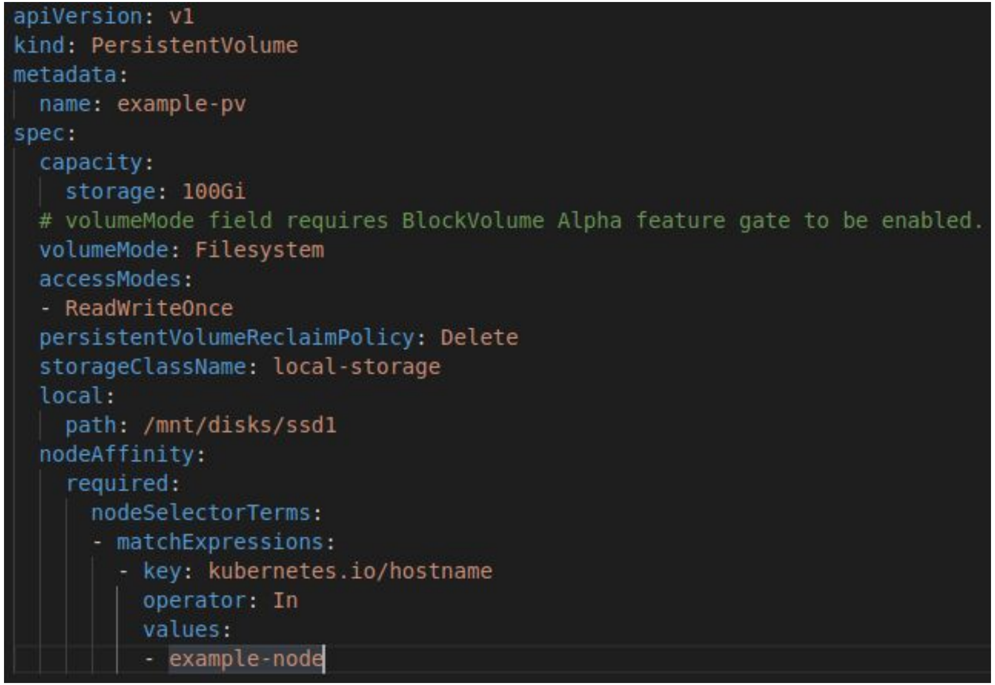
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
    name: test-container
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
```

Examples - hostpath (file/dir)



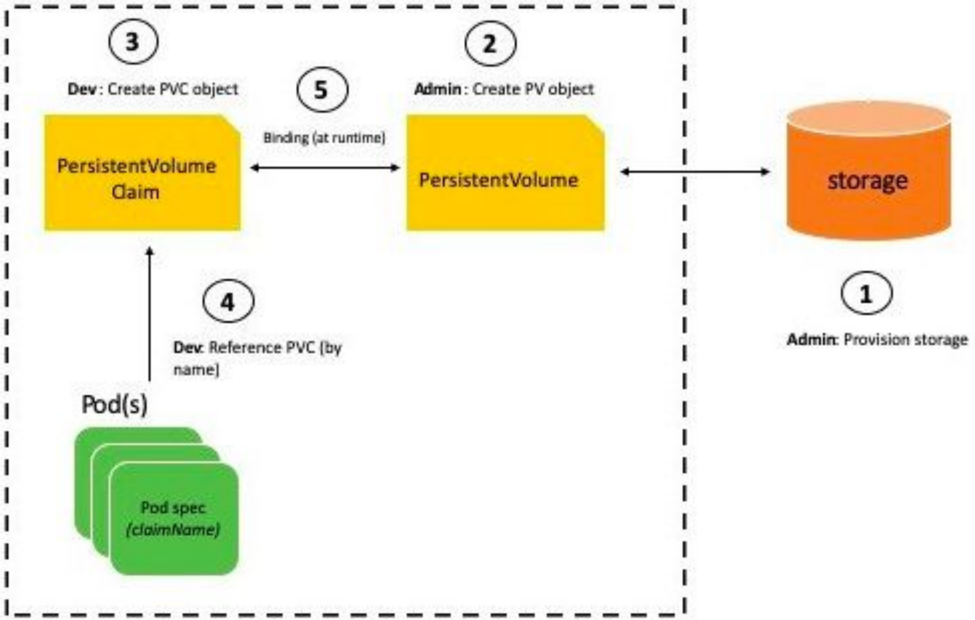
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /test-pd
      name: test-volume
  volumes:
  - name: test-volume
    hostPath:
      # directory location on host
      path: /data
      # this field is optional
      type: DirectoryOrCreate
```

Persistent Volume - static & local

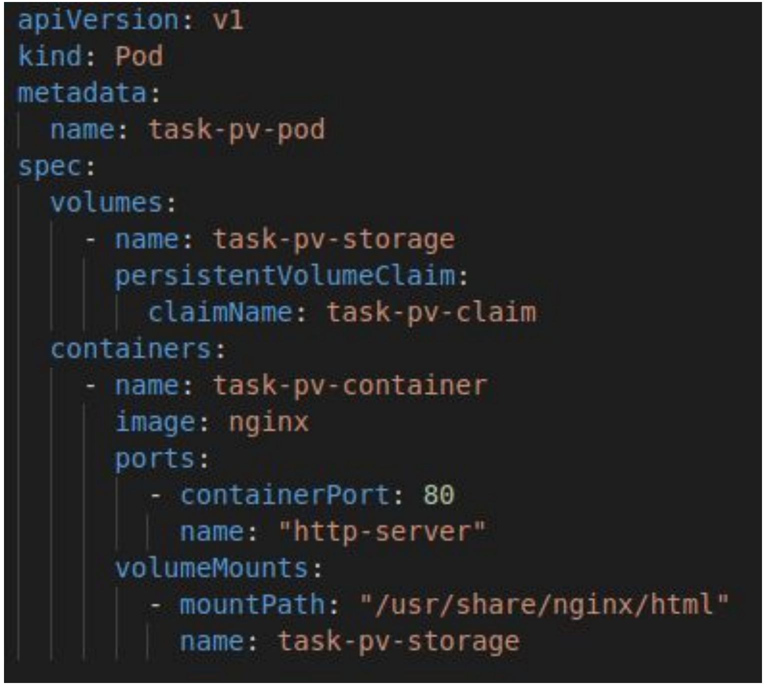


```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv
spec:
  capacity:
    storage: 100Gi
  # volumeMode field requires BlockVolume Alpha feature gate to be enabled.
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: /mnt/disks/ssd1
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - example-node
```

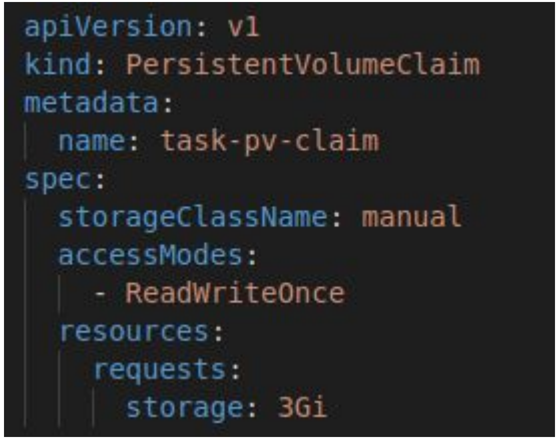
Persistent Volume - static



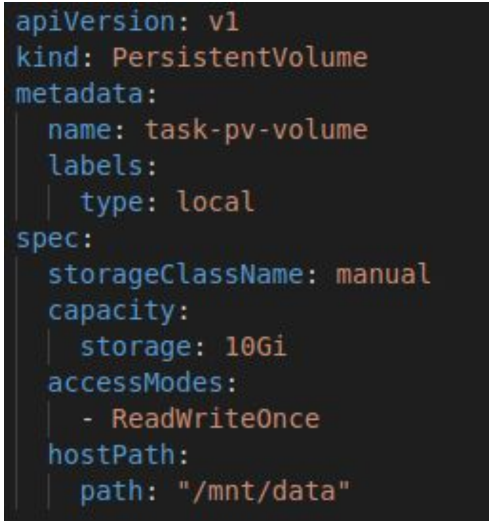
Example



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
        claimName: task-pv-claim
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
```

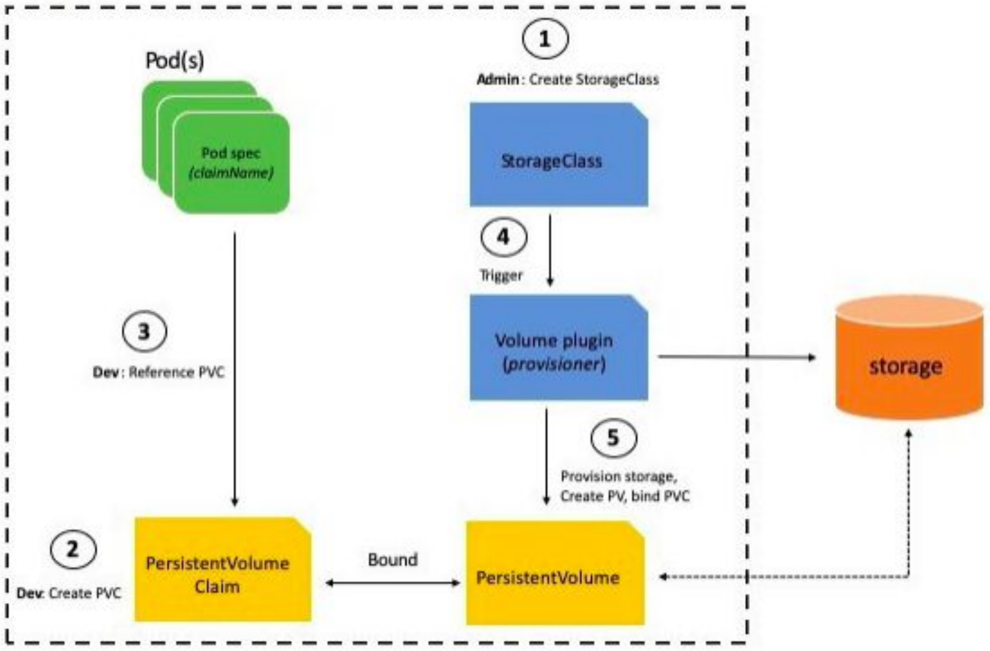


```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```



```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

Persistent Volume - Dynamic



Persistent Volumes

- GCEPersistentDisk
- AWSElasticBlockStore
- AzureFile
- AzureDisk
- CSI
- FC (Fibre Channel)
- FlexVolume
- Flocker
- NFS
- iSCSI
- RBD (Ceph Block Device)
- CephFS
- Cinder (OpenStack block storage)
- Glusterfs
- VsphereVolume
- Quobyte Volumes
- HostPath (Single node testing only – local storage is not supported in any way and WILL NOT WORK in a multi-node cluster)
- Portworx Volumes
- ScaleIO Volumes
- StorageOS

Controllers - StatefulSet

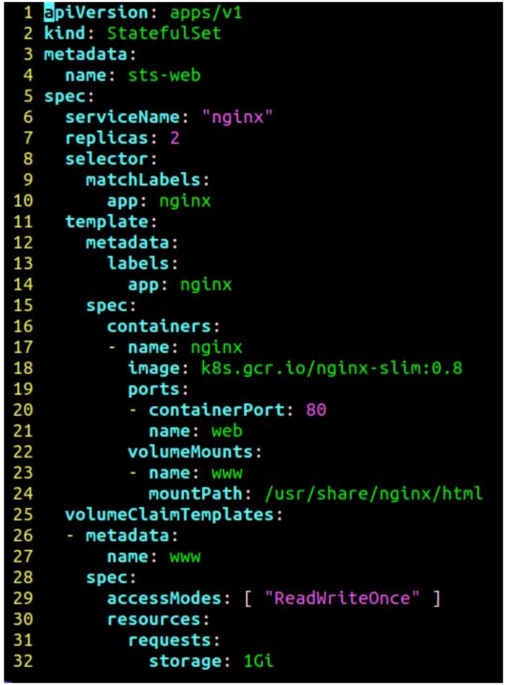
Use Cases

- Stable, unique network identifiers
- Stable, persistent storage
- Ordered, graceful deployment and scaling
- Ordered, automated rolling updates

Limitations

- Requires headless service (manual way)
- No automatic deletion of referenced volumes
- No PODs deletion guarantee when StatefulSet is deleted
- Rolling Updates not consistent always

StatefulSet - Example



```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: sts-web
spec:
  serviceName: "nginx"
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi
```

- Scale Up
- Scale Up
- Update (image)

StatefulSet - Example

- Scale Up
- Scale down
- Rolling Update (image)
- kubectl set image sts/sts-web nginx=nginx:1.18
- Staged Update

  -
    ```
    kubectl patch statefulset web -p
    '{"spec":{"updateStrategy":{"type":"RollingUpdate","rollingUpdate":{
    "partition":3}}}}'
    ```

Controllers - DaemonSet

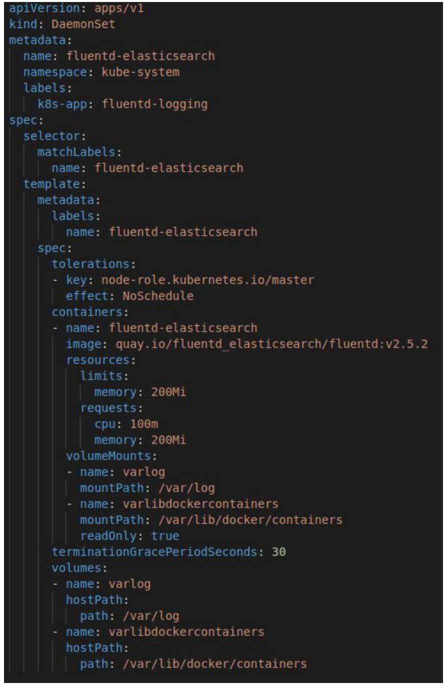
Purpose

- To run a copy of a POD on all / some node(s)

Use Cases

- Storage cluster daemon (gluster, ceph)
- Log Collectors (fluentd, logstash)
- Node Monitoring daemons (Prometheus, Dynatrace, collectd)

DaemonSet - Example



```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: fluentd-elasticsearch
        image: quay.io/fluentd_elasticsearch/fluentd:v2.5.2
        resources:
          limits:
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
        volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
```

Exercises (15 mins)

- Create POD (nginx / redis) to use volume emptyDir
- Launch POD and login into POD
- Create test file
- Kill the container process (nginx / redis)
- Observe POD status and login into POD again
- Verify if test file exists

NOTE:

POD has restartPolicy as Always

Ephemeral storage is associated till POD is deleted

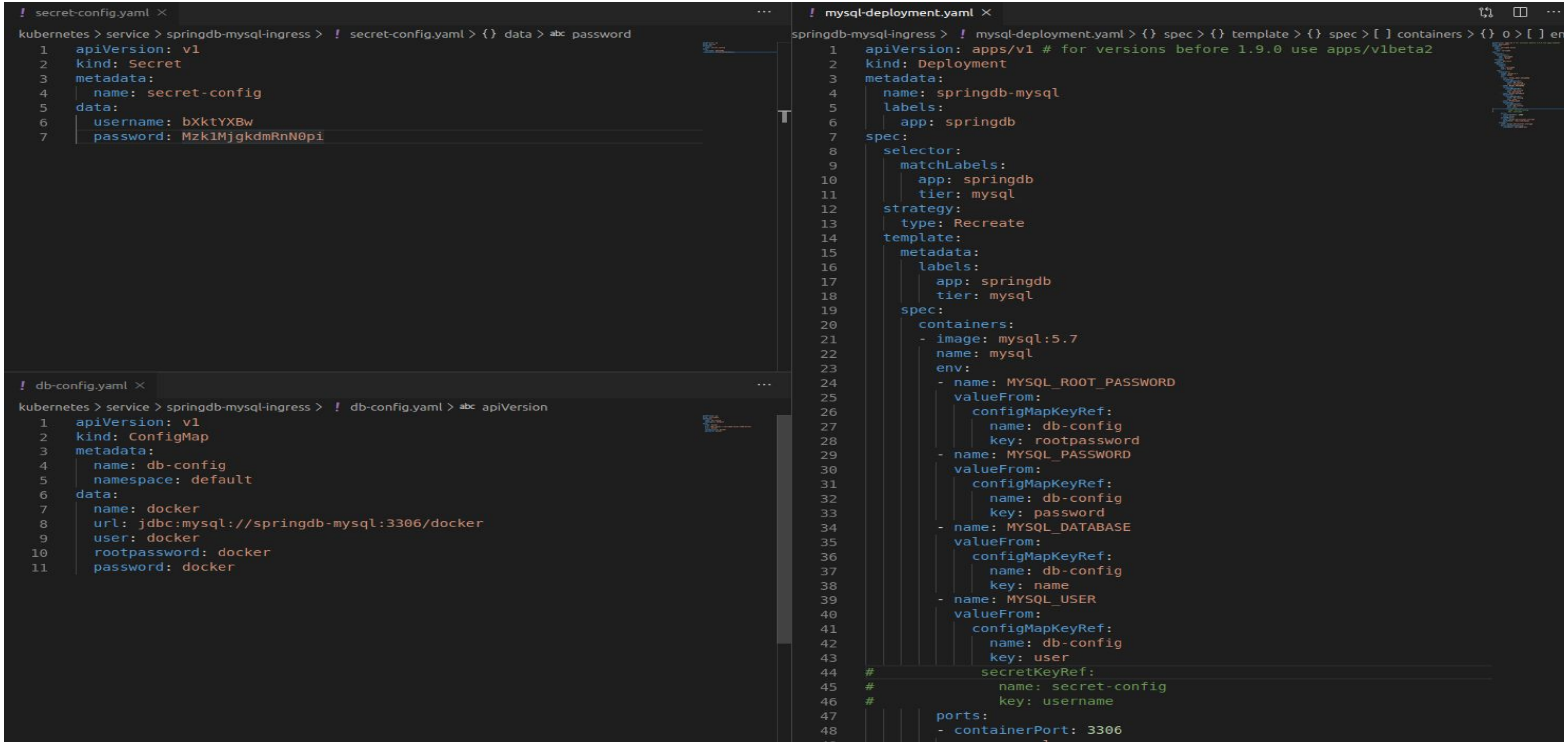
Exercises (15 mins)

- Create nginx POD that uses pvc for serving web files
- Define pvc that uses pv
- Define pv that refers to host path /mnt/data
- Create index.html echoing ‘hello k8s’ under host path
- Verify that nginx serves the index.html contents that you saved

Configuration

- Config Maps, Environment
- Security Contexts
- Resource Requirements
- Secrets
- Service Accounts
- Demo
- Practicals

Example


secret-config.yaml
kubernetes > service > springdb-mysql-ingress > secret-config.yaml > {} data > abc password

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: secret-config
data:
  username: bXktYXBw
  password: Mzk1MjgkdmRnN0pi
```

db-config.yaml
kubernetes > service > springdb-mysql-ingress > db-config.yaml > abc apiVersion

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: db-config
  namespace: default
data:
  name: docker
  url: jdbc:mysql://springdb-mysql:3306/docker
  user: docker
  rootpassword: docker
  password: docker
```

mysql-deployment.yaml
springdb-mysql-ingress > mysql-deployment.yaml > {} spec > {} template > {} spec > [ ] containers > {} 0 > [ ] en

```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: springdb-mysql
  labels:
    app: springdb
spec:
  selector:
    matchLabels:
      app: springdb
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: springdb
        tier: mysql
    spec:
      containers:
      - image: mysql:5.7
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            configMapKeyRef:
              name: db-config
              key: rootpassword
        - name: MYSQL_PASSWORD
          valueFrom:
            configMapKeyRef:
              name: db-config
              key: password
        - name: MYSQL_DATABASE
          valueFrom:
            configMapKeyRef:
              name: db-config
              key: name
        - name: MYSQL_USER
          valueFrom:
            configMapKeyRef:
              name: db-config
              key: user
#            secretKeyRef:
#              name: secret-config
#              key: username
        ports:
        - containerPort: 3306
```

Scheduling

- Node Name
- Taints
- Tolerations
- Affinity
- Demo
- Practicals

Topology

Purpose

- To route the traffic in the same node / cluster / zone wherever possible

Advantages

- Lower latency
- Cost Optimization as inter zonal cloud requests costs

How to achieve

- Enable Service Topology feature (API Server and Kube proxy)
- Label endpoints, node, cluster, zone appropriately

Scheduling - Priorities

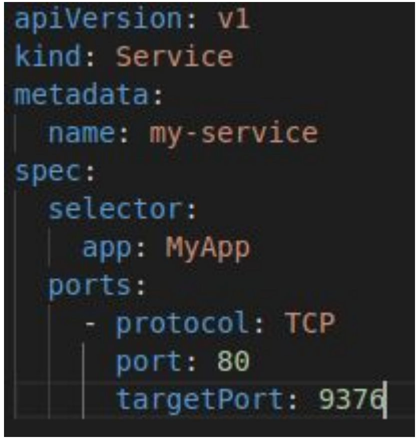
- Node capacity based on priority weightage on its resources (cpu, memory, disk etc)
- Node score - Input to Kube scheduler
- Resource specification on PODs helps better computing capacity management

Services

- Cluster IP
- Node Port
- Load Balancer
- Connecting using services
- Demo
- Practicals

Service

- An abstract way to expose an application running on pod as network service.
- Frontends and backends of application can connect without worrying about POD IPs

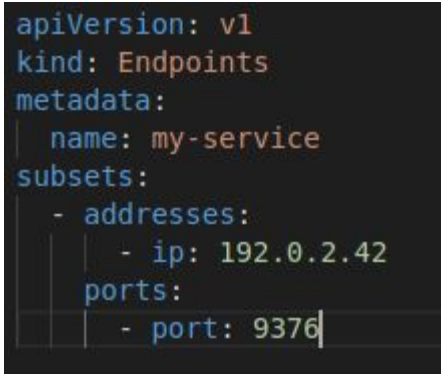


```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Service without selector

Use Cases:

- External DB Cluster in production
- To point a service in another namespace / cluster
- Uses session affinity while connecting to backend PODs



```yaml
apiVersion: v1
kind: Endpoints
metadata:
  name: my-service
subsets:
  - addresses:
      - ip: 192.0.2.42
    ports:
      - port: 9376
```

Service Types

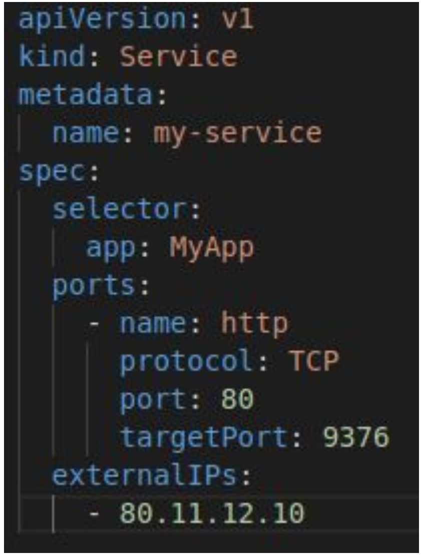
- Cluster IP
  - Service exposed on cluster internal IP
  - Reachable only within cluster

- Node Port
  - Exposed on each Node IP at static port

- Load Balancer
  - Exposed through external cloud load balancer

- External Name
  - Exposed through the contents of external field via CNAME record

External IP



```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 9376
  externalIPs:
    - 80.11.12.10
```

Ingress & Networking

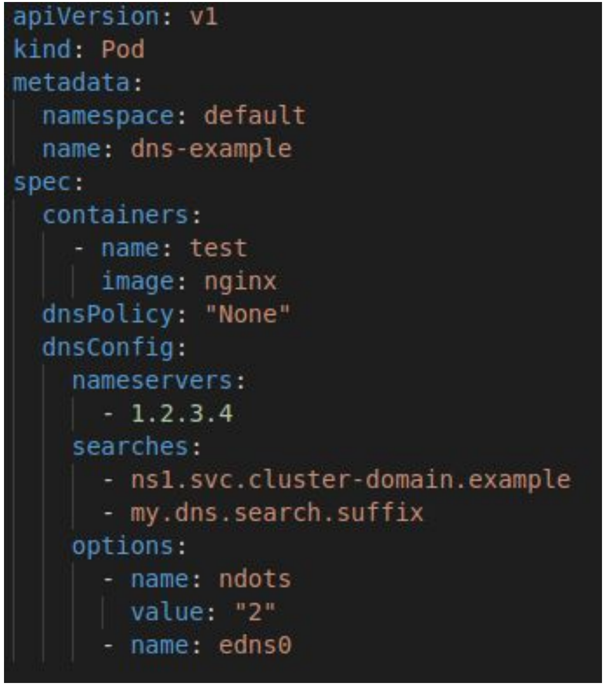
- Ingress Controller
- Ingress Resources
- Network Policies
- Host mapping with service
- TLS
- Multi domain frontends with service
- Demo
- Practicals

DNS

DNS Policy

- Default
  - Inherit name resolution from Node
- ClusterFirst
  - Forwards to upstream nameserver for unresolved name queries
- ClusterFirstWithHostNet
  - Only for PODs running with hostNetwork
- None
  - POD explicitly defines it using dnsConfig

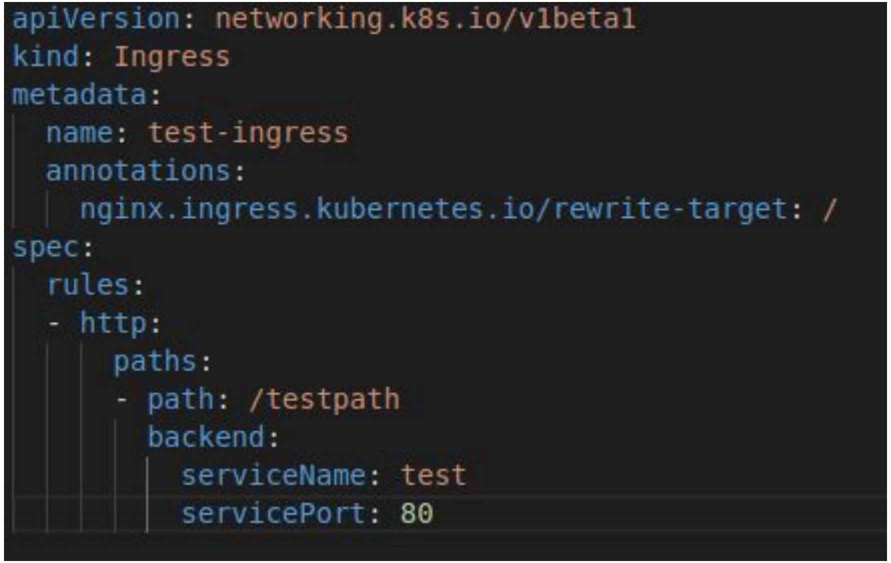
Custom DNS - Example



```yaml
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: dns-example
spec:
  containers:
    - name: test
      image: nginx
  dnsPolicy: "None"
  dnsConfig:
    nameservers:
      - 1.2.3.4
    searches:
      - ns1.svc.cluster-domain.example
      - my.dns.search.suffix
    options:
      - name: ndots
        value: "2"
      - name: edns0
```

Ingress

- Provides load balancing, SSL Termination and Name based virtual hosting
- Provides externally reachable URLs to Services
- Used for HTTP / HTTPS protocols



```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: test-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        backend:
          serviceName: test
          servicePort: 80
```

Examples

- Web server (tutum)

- API Services offering from UniGPS

Exercises (15 mins)

- Create deployment based on nginx image with 3 replicas
- Create NodePort service to map to the PODs created by above deployment
- View service URL and access it using browser
- Create another service of type Ingress
- View service URL and Access the service outside cluster
- Create one more service of type ClusterIP
- View service URL and find a way to access it

Exercise - Scenario

Assume that based on your recently acquired K8S expertise, you are tasked by your firm to develop real time video based fleet monitoring service with below high level Objectives:

- New video service should be independent of any other earlier services (/API) developed
- Deployment should be as easy as possible
- New service should be provided to end customers via video.unigps.in
- You are expected to use current k8s setup and extend on it

Outcome expected:

- Yaml based definitions of deployment, service and domain based routing and load balancing.
- Service deployment should have at least 5 instances of replica

Monitoring

- Liveness Probes
- Readiness Probes
- Container Logging
- Monitoring & Debugging
    - Debug - live debugging with Telepresence
    - Shell to running container
- Demo
- Practicals

Monitoring Dashboard

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-beta8/aio/deploy/recommended.yaml

kubectl proxy
```

http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/

Cluster Access

To view cluster configuration

```
kubectl config view
```

Reverse proxy to API server

```
kubectl proxy --port=8080
```

Port Forwarding

```
kubectl port-forward <pod> 7000:6379
```

```
kubectl port-forward <deployment> 7000:6379
```

```
kubectl port-forward <svc> 7000:6379
```

To access LoadBalancer service on localhost

```
minikube tunnel
```

Introspection & debugging

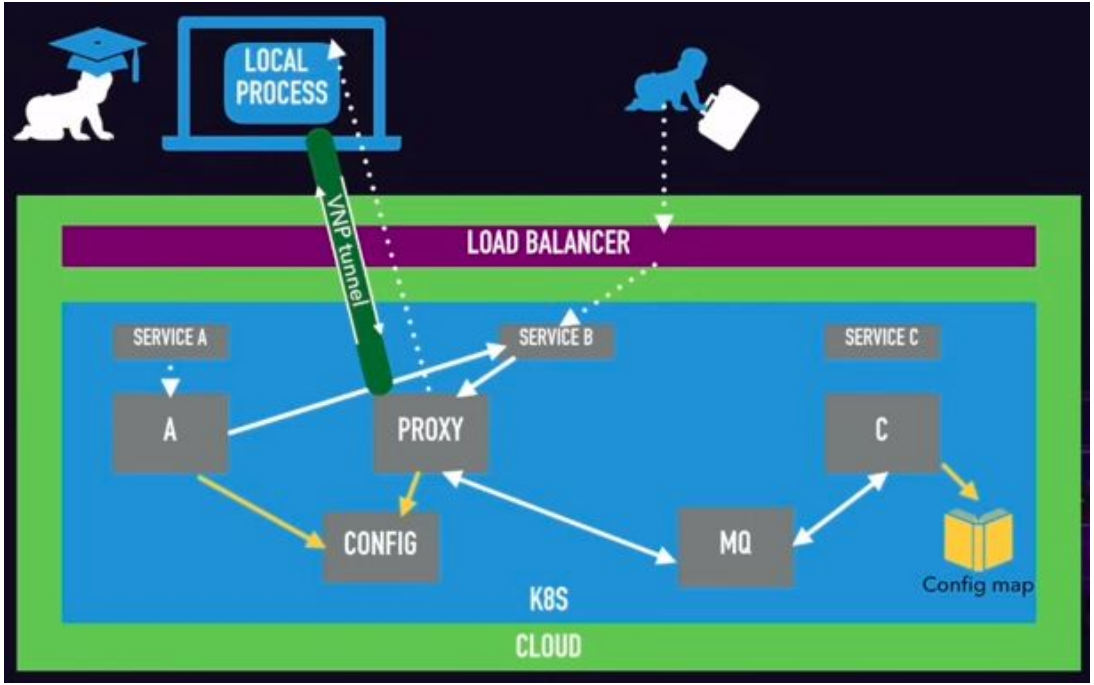
```
kubectl get pods
kubectl get pod <pod-name> -o yaml
kubectl describe <pod-name>
kubectl describe <pod-name> -o yaml
kubectl get events
kubectl get events --namespace=my-namespace   (--all-namespaces)
kubectl get nodes
kubectl get node <node-name>
kubectl get node <node-name> -o yaml
kubectl describe node <node-name>
kubectl describe node <node-name> -o yaml
```

Live debugging using IDE

```
telepresence --swap-deployment hostnames --namespace default --run mvn spring-boot:run
-Dspring-boot.run.jvmArguments="-Xdebug
-Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5005"
```

Project: git/rest-service

Telepresence - Live debugging



Shell to running container

```
rajesh@rajesh-Gazelle:~/git/kubernetes/debugging/shell$ kubectl apply -f shell-demo.yaml

kubectl get pod shell-demo
kubectl exec -it shell-demo -- /bin/bash

root@shell-demo:/# ls /

root@shell-demo:/# echo Hello shell demo > /usr/share/nginx/html/index.html
root@shell-demo:/# apt-get update
root@shell-demo:/# apt-get install curl
root@shell-demo:/# curl localhost

kubectl exec shell-demo env

kubectl exec -it my-pod --container main-app -- /bin/bash
```

Best Practices

- Configuration - specify latest stable API version
- Keep config files in version control before pushing to cluster
- Prefer YAML over JSON
- Group related objects into one file whenever it makes more sense
- Don’t specify default values unnecessarily
- Put Object descriptions as part of annotations
- Don’t use naked PODs
- Create service before deployments
- Avoid using hostPort for POD
- Use labels effectively
- Use image tag instead of using latest as the default
- Use kubectl run and expose to launch single container deployments & services

Best Practices- Security

4 Cs of Cloud Native Security

- Code
- Container
- Cluster
- Cloud

Areas of caution / concern:

- API Server: Avoid exposing Master Nodes / API server publicly
- Nodes Access: Should allow only master nodes to communicate on specified ports
- Access to Cloud API: Based on principle of least privilege, Cloud API access to K8S should be provided
- ETCD: Master only should have access and data should be encrypted

Recap

- Review
- Q & A

Thank You for your active participation!

rajesh@unigps.in

9880195215

https://www.linkedin.com/in/rajesh-g-b48495/