TEKsystems Global Services | WORKFORCE DEVELOPMENT

Configuration Management and DevOps: Automate Infrastructure Deployments

PARTICIPANT GUIDE

Content Usage Parameters

Content refers to material including instructor guides, student guides, lab guides, lab or hands-on activities, computer programs, etc. designed for use in a training program

1

Content is subject to copyright protection

2

Content may only be leveraged by students enrolled in the training program

3

Students agree not to reproduce, make derivative works of, distribute, publicly perform and publicly display content in any form or medium outside of the training program

4

Content is intended as reference material only to supplement the instructor-led training

Core Kubernetes

Kubernetes Services Overview

Kubernetes Service Objects:Microservices

Services provide abstraction between layers of an application

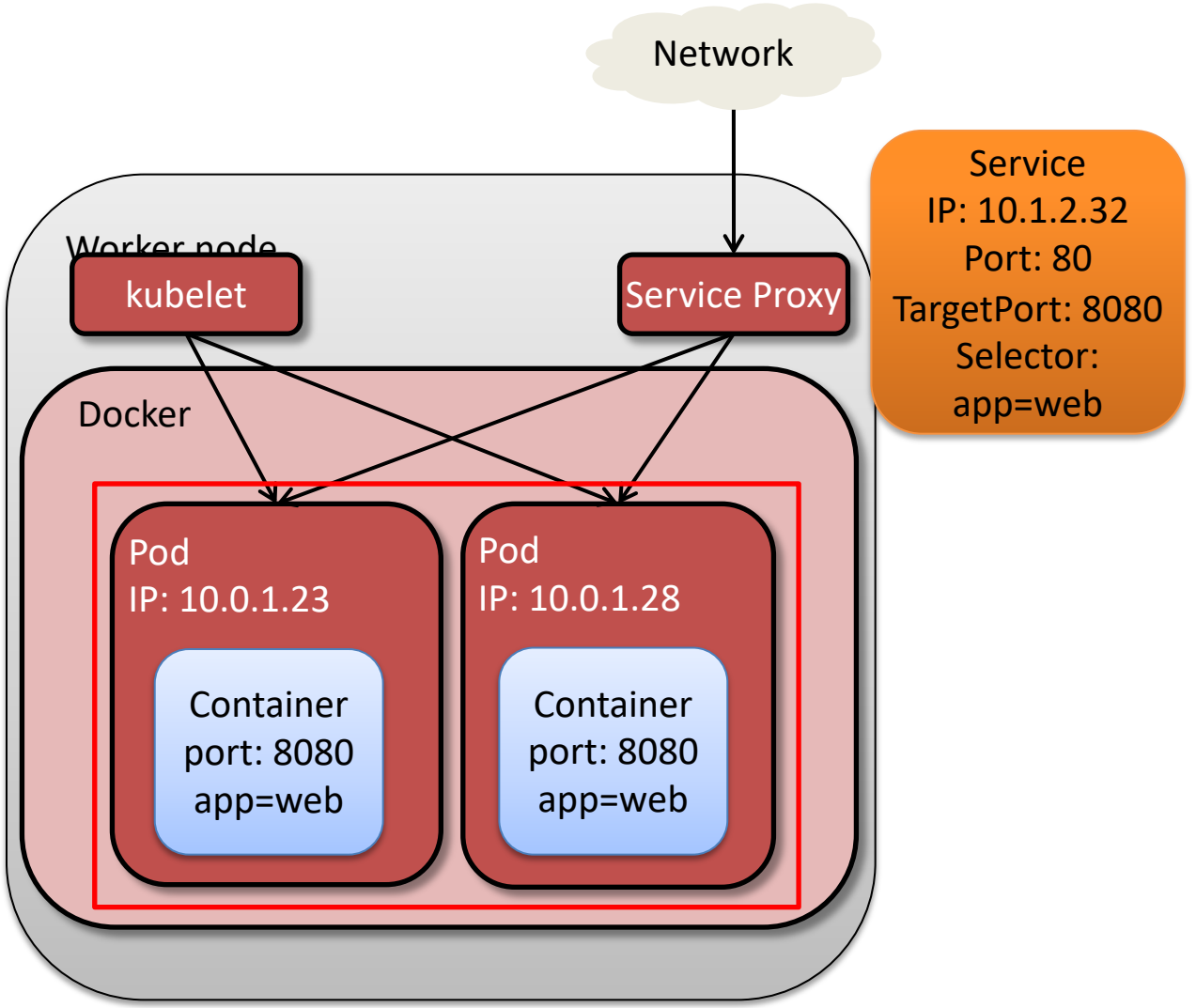
- **Service** object provides a stable IP for a collection of Pods
- Services use a label **selector** to target a specific set of pods as endpoints to receive proxied traffic
- Clients can reliably connect via the service IP and port(s), even as individual endpoint pods are dynamically created & destroyed
- Can model other types of backends using services without selectors

```yaml
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  ports:
    - port: 80
      targetPort: 8080
  selector:
    app: wordpress
    tier: frontend
  type: LoadBalancer
```

sampleservice.yaml

Services Provide Abstraction Layer for Applications

- Services can be used for communications between application tiers
- Services can also be used to expose applications outside the K8s cluster
- Services distribute requests over the set of Pods matching the service's selector
  - Service functions as TCP and UDP proxy for traffic to Pods
  - Service maps its defined ports to listening ports on Pod endpoints



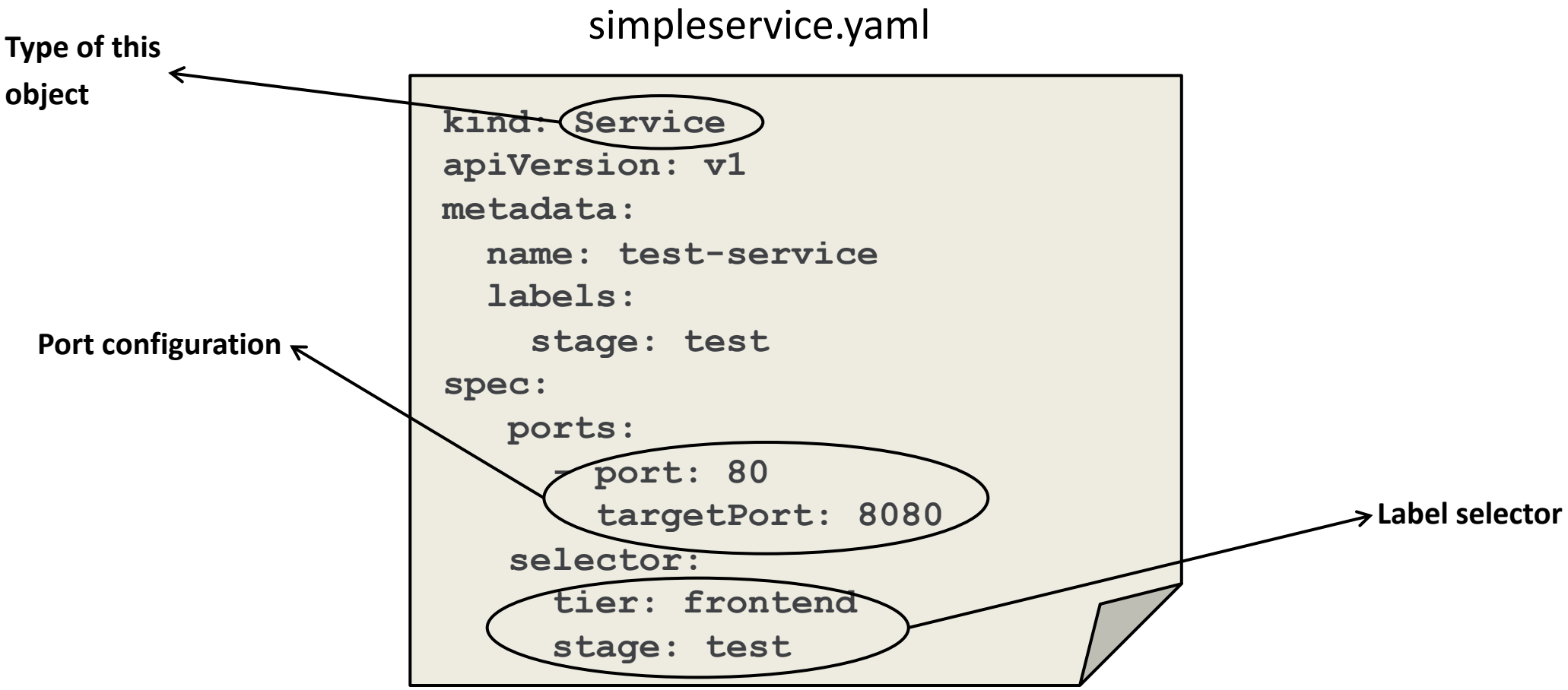
Defining a Service Using a Manifest File

Services can defined in YAML or JSON, like other K8s resources

- **kind** field value is 'Service'
- **metadata** includes
  - **name** to assign to Service
- **spec** includes the ports associated with the Service
  - **port** is the Service's port value
  - t**argetPort** is connection port on selected pods (default: **port** value)
- **selector** specifies a set of label KV pairs to identify the endpoint pods for the Service

```
kind: Service
apiVersion: v1
metadata:
  name: test-service
  labels:
    stage: test
spec:
   ports:
     - port: 80
       targetPort: 8080
   selector:
     tier: frontend
     stage: test
```

Reviewing a Service Manifest File



ServiceTypes and Exposing Applications

- By default, a Service is assigned a cluster-internal IP – good for back-ends
  - **ServiceType** ClusterIP
- To make a front-end Service accessible outside the cluster, there are other ServiceTypes available
  - **ServiceType** NodePort exposes Service on each Node's IP on a static port
  - **ServiceType** LoadBalancer exposes the Service externally using cloud provider's load balancer
- Can also use a Service to expose an external resource via **ServiceType** ExternalName

```
kind: Service
apiVersion: v1
metadata:
  name: test-service
  labels:
    stage: test
spec:
   ports:
     - port: 80
       targetPort: 8080
       nodePort: 30080
   selector:
     tier: frontend
   type: NodePort
```

Exposing Services at L7 through Ingresses

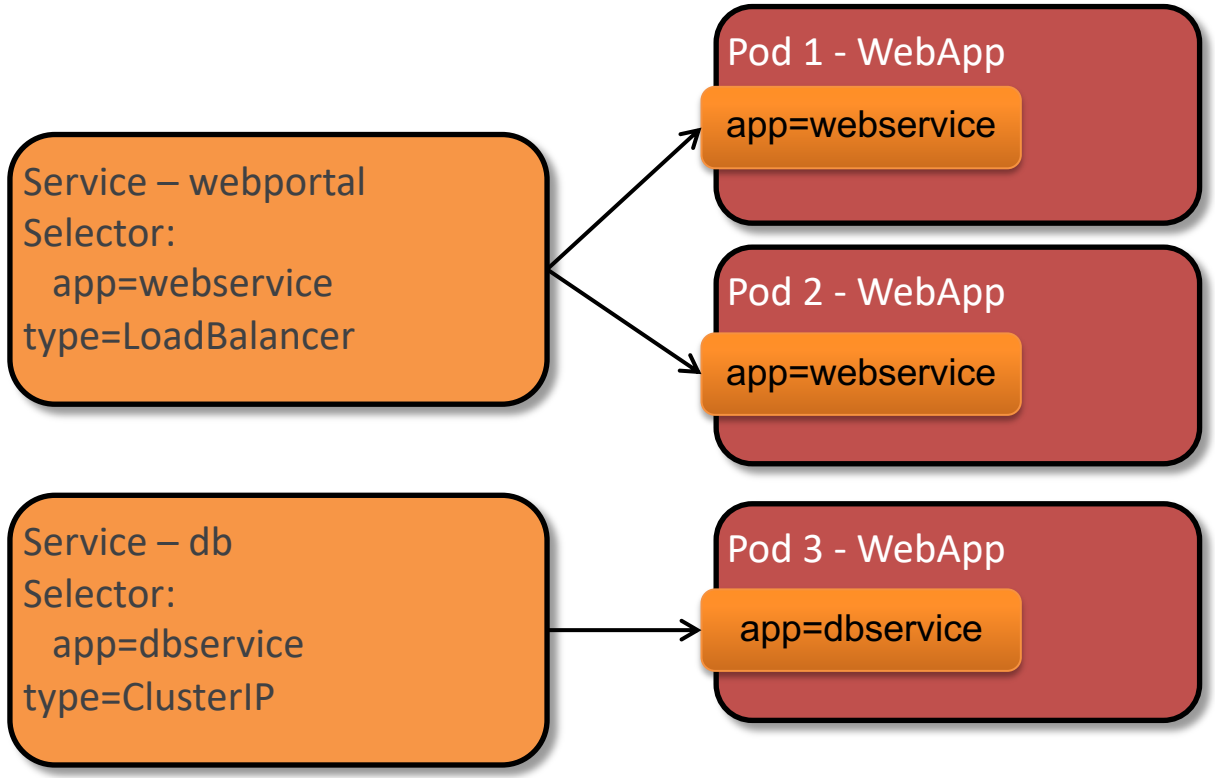
- Kubernetes also provides the facility to define **Ingress** resource to configure an external loadbalancer at L7
- Spec of Ingress resource is a set of rules matching HTTP host/url paths to specific Service backends
- Ingresses require the cluster to be running an appropriately configured Ingress controller to function (e.g. nginx)
- Useful for implementing fanout, Service backends for virtual hosts, etc.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: test-ingress
spec:
  rules:
  -host: bar.foo.com
   http:
      paths:
      - path: /first
        backend:
          serviceName:
firstservice
          servicePort: 80
      - path: /second
        backend:
          serviceName:
secondservice
          servicePort: 80
```

Selecting Pods as Service Endpoints

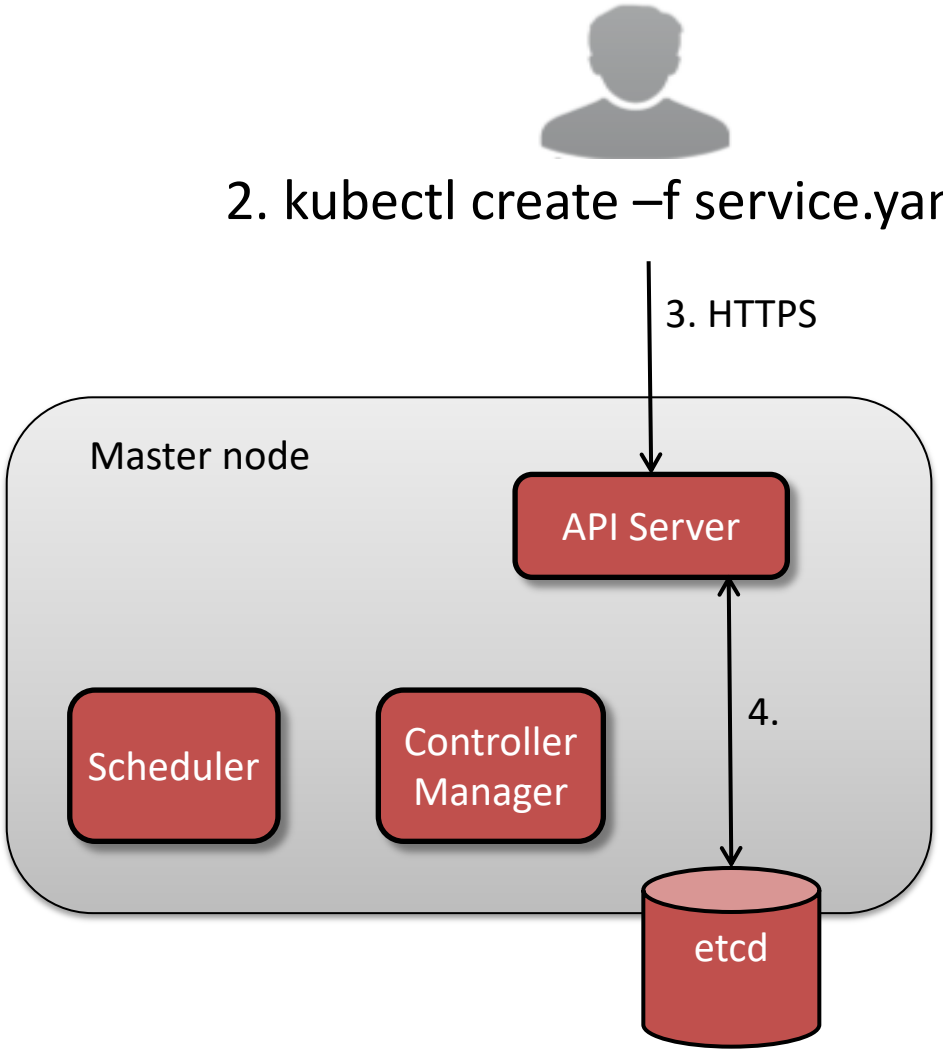
Service's pod selector based on labels

- Multiple pods can have the same label, unlike pod names which are unique in the namespace
- K8s system re-evaluates Service's selector continuously
- K8s maintains **endpoints** object of same name with list of pod IP:port's matching Service's selector



Services Management

Service Creation Process

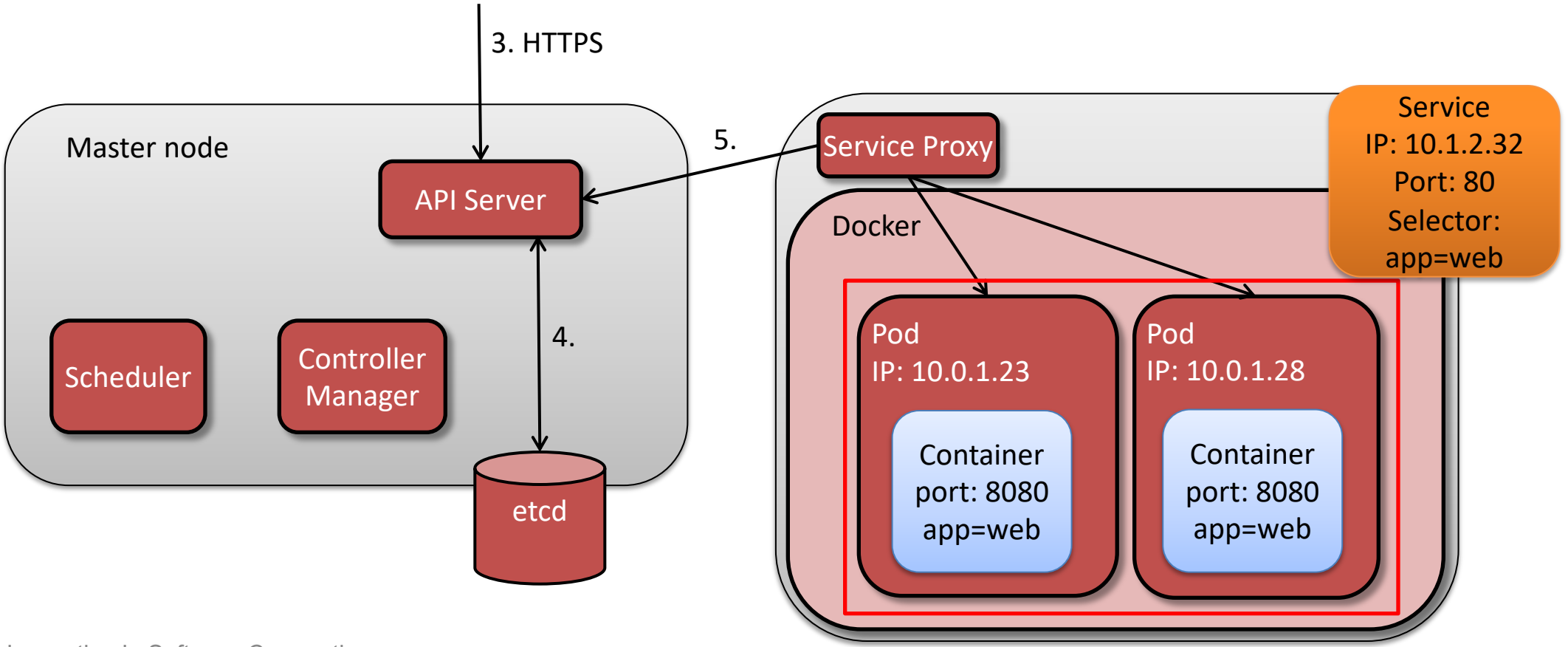


1. User writes a Service manifest file
2. User requests creation of Service from manifest via CLI
3. CLI tool marshals parameters into K8s RESTful API request (HTTP POST)
4. kube-apiserver creates new Service object record in etcd

Service Creation Process

2. kubectl create –f service.yaml

5. kube-proxy on node sees new Service
6. kube-proxy configures local iptables rules to forward traffic to endpoints



Accessing Services Externally

Checking the service external IP

- Configured as type LoadBalancer, a configured cluster will provide an externally accessible IP for your Service

```
$ kubectl get services test-service
NAME         CLUSTER-IP   EXTERNAL-IP     PORT(S)    AGE
test-service 10.3.247.123 104.154.105.198 8080/TCP   4h
```

Deleting Services

Services can be deleted anytime

- Service deletion does not affect pods targeted by the Service's selector
- Can be done referencing the Service name or a manifest for the resource

```
$ kubectl delete -f simpleservice.yaml
```

```
$ kubectl delete service test-service
```

Service Discovery via DNS

Kubernetes advertises services via cluster DNS

- Kubernetes uses a cluster-internal DNS add-on to create and manage records for all Services in the cluster
- Pod's DNS search list includes its own namespace and cluster default domain by default

```
$ kubectl get svc
NAME           CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
kubernetes     10.0.0.1      <none>        443/TCP          11d
test-service   10.0.0.102    <nodes>       8080:30464/TCP   2d

kubectl exec -ti busybox1 -- nslookup test-service.default
Server:    10.0.0.10
Address 1: 10.0.0.10 coredns.kube-system.svc.cluster.local

Name:      test-service.default
Address 1: 10.0.0.102 test-service.default.svc.cluster.local
```

Deployment Strategies Overview

What is a Deployment Strategy?

Approaches to manage risks on updating Deployments

- On each Deployment update/change, all pods in the deployment will be deleted and recreated
- Recreation process can have service impacts, especially for large Deployments
- A Deployment strategy defines how this rebuild process is done, to minimize downtime due to application failures or malfunctions

Types of Deployment Strategies

Kubernetes supports two basic strategies, but users can also leverage multiple Deployments when applying changes

- Strategies for single Deployments
  - Recreate
  - RollingUpdate
- Strategic approaches using two Deployments with a Service
  - Canary deployments
  - Blue/Green deployments

Each approach has a specific behavior and advantages/disadvantages.

Deployment Strategy: RollingUpdate

Deployment Strategy: RollingUpdate

RollingUpdate is DEFAULT strategy for Deployments

- When a change is made to the Deployment, the old Replica Set pods are scaled down as new pods are created by the new Replica Set
- A minimum number of running Pods is specified, so the Deployment will never be totally out of Pods to respond to service requests
- During the update process, the requested replica count may be temporarily exceeded

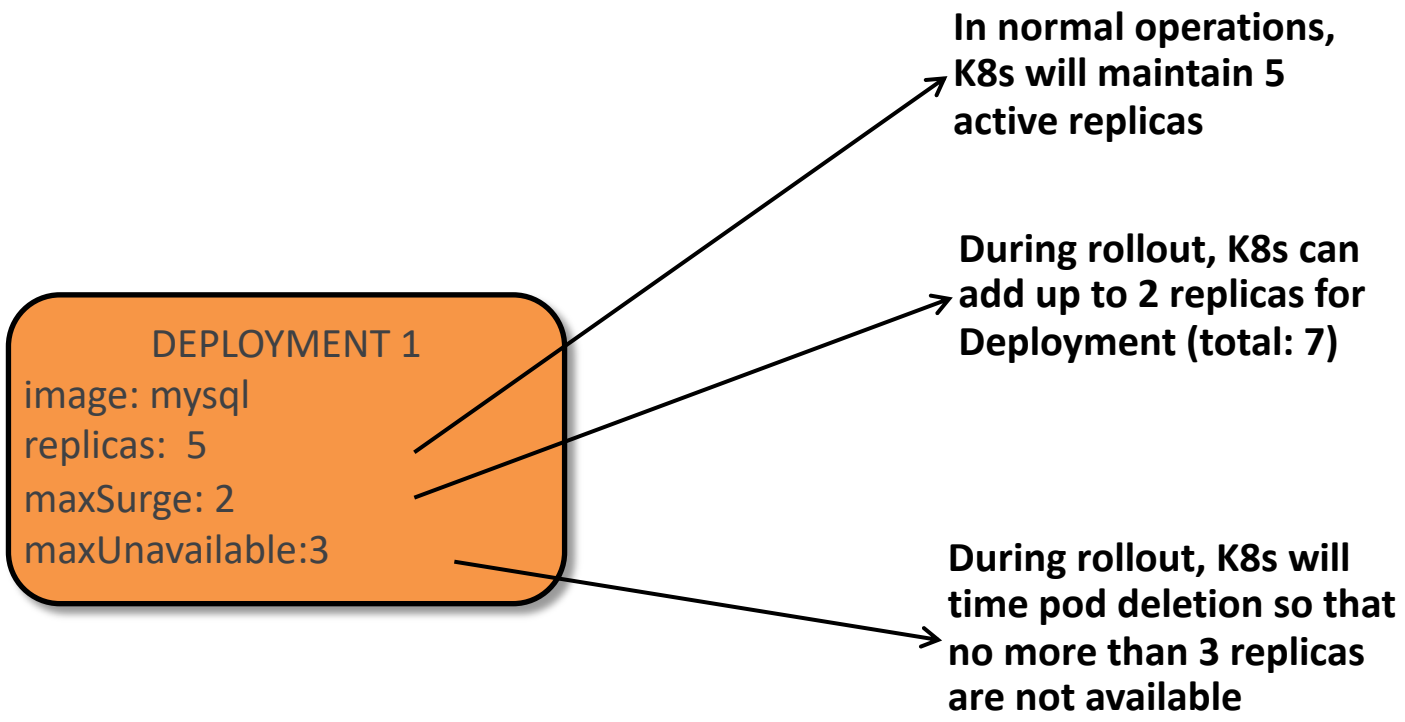
Deployment Strategy: RollingUpdate

Configure parameters to control the update process

```
...
metadata:
  name: tomcat-deployment
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable:10%
template:
    metadata:
      labels:
        type: webserver
...
```

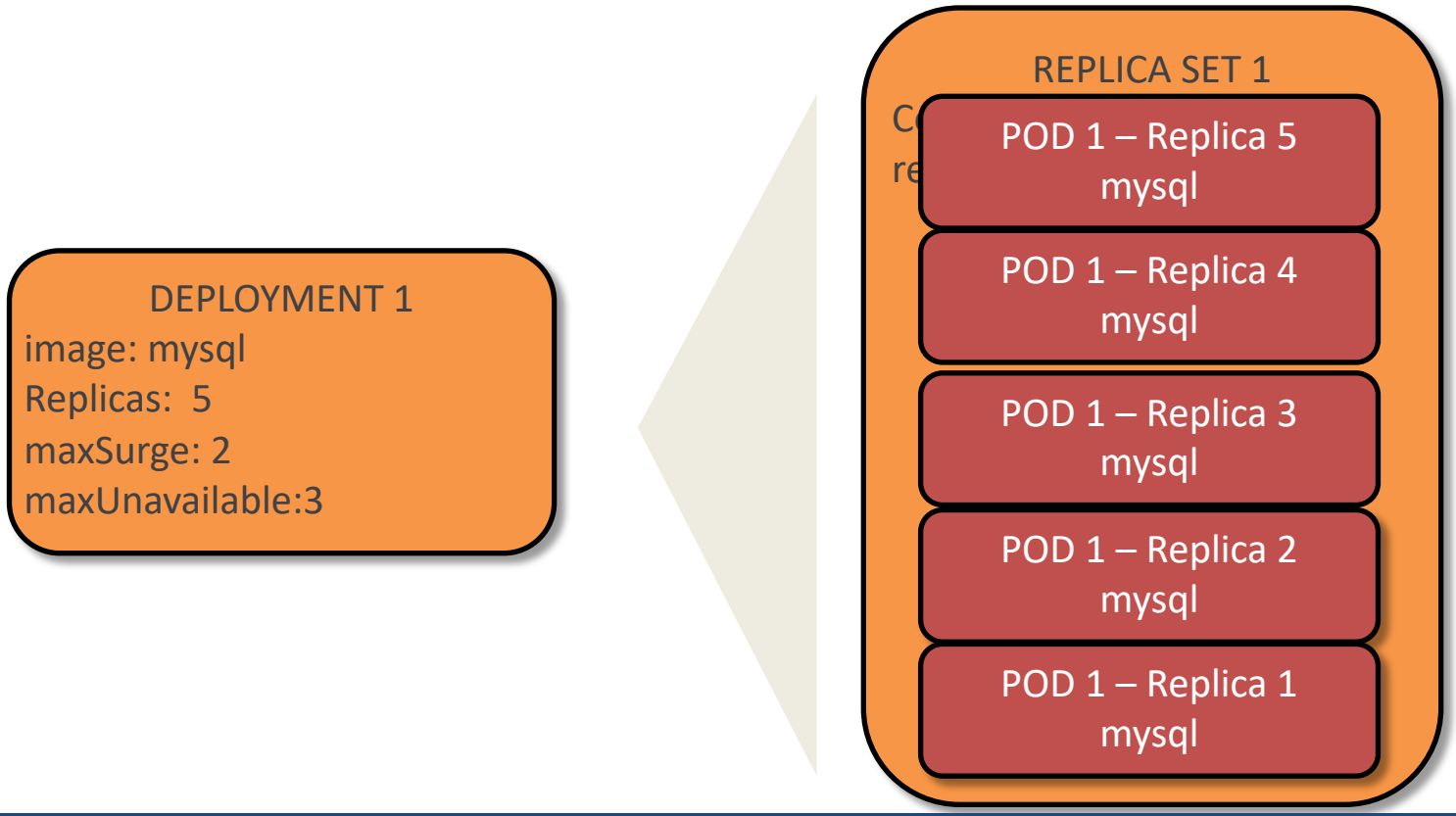
- **maxSurge**: number or percentage of additional Pods that can be created exceeding the replica count during update
  - Default value of 25%
- **maxUnavailable**: number of Pods that can be unavailable during the update
  - Default value of 25%

Deployment Strategy: RollingUpdate



Deployment Strategy: RollingUpdate

Initial State

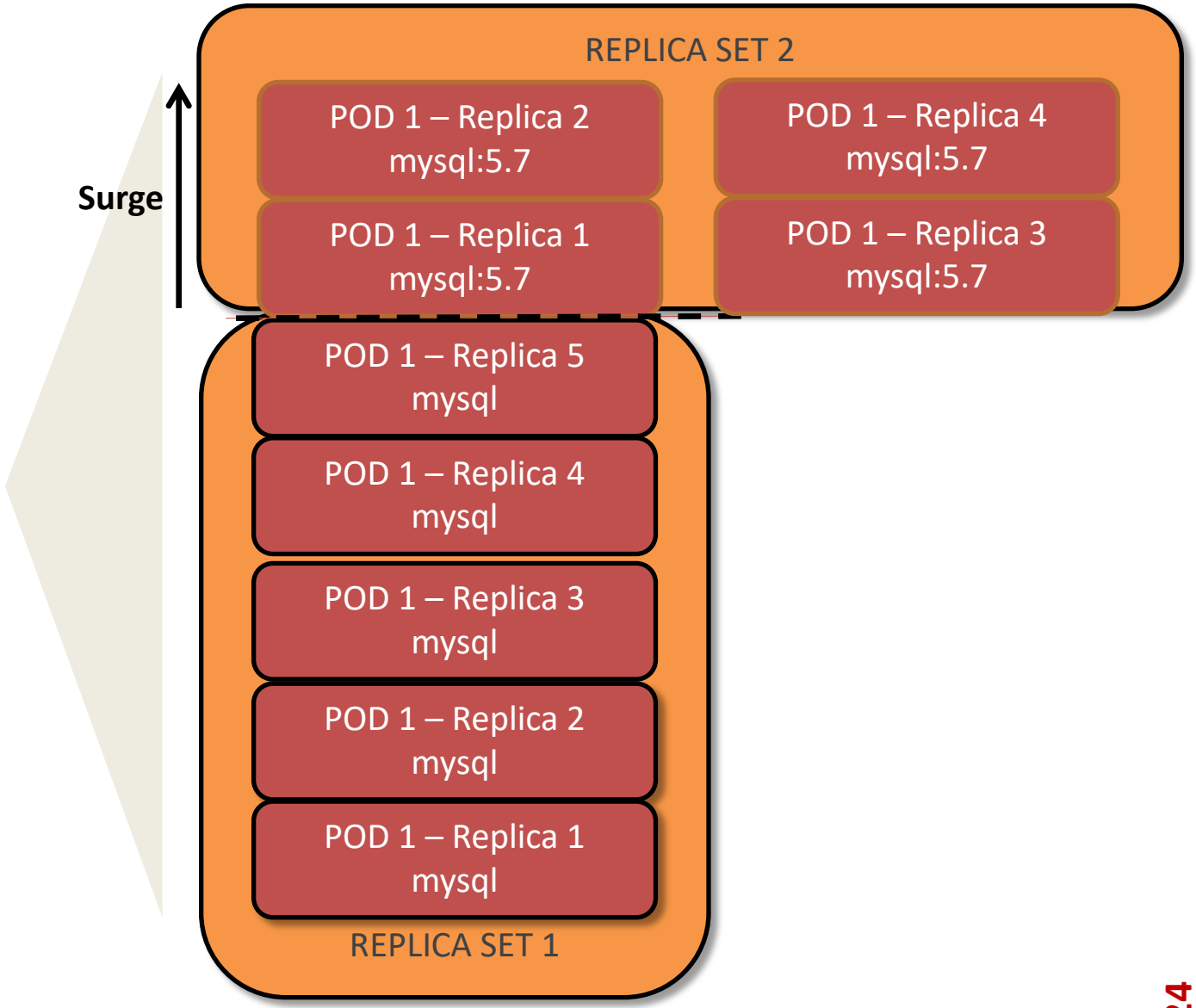



```
$ vi simpledeployment.yaml
...
    image: mysql:5.7
...
$ kubectl apply –f simpledeployment.yaml
```

Deployment Strategy: RollingUpdate

Rollout in progress

DEPLOYMENT 1
image: mysql:5.7
Replicas: 5
maxSurge: 2
maxUnavailable:3



- Initial surge of new pods on new Replica Set
- Original Replica Set scaled back as new RS scaled out

Deployment Strategy: RollingUpdate

Rollout complete

DEPLOYMENT 1
image: mysql:5.7
replicas: 5
maxSurge: 2
maxUnavailable:3

REPLICA SET 2
image: mysql:5.7
replicas: 5

- POD 1 – Replica 5 mysql:5.7
- POD 1 – Replica 4 mysql:5.7
- POD 1 – Replica 3 mysql:5.7
- POD 1 – Replica 2 mysql:5.7
- POD 1 – Replica 1 mysql:5.7

REPLICA SET 1
image: mysql
replicas: 0

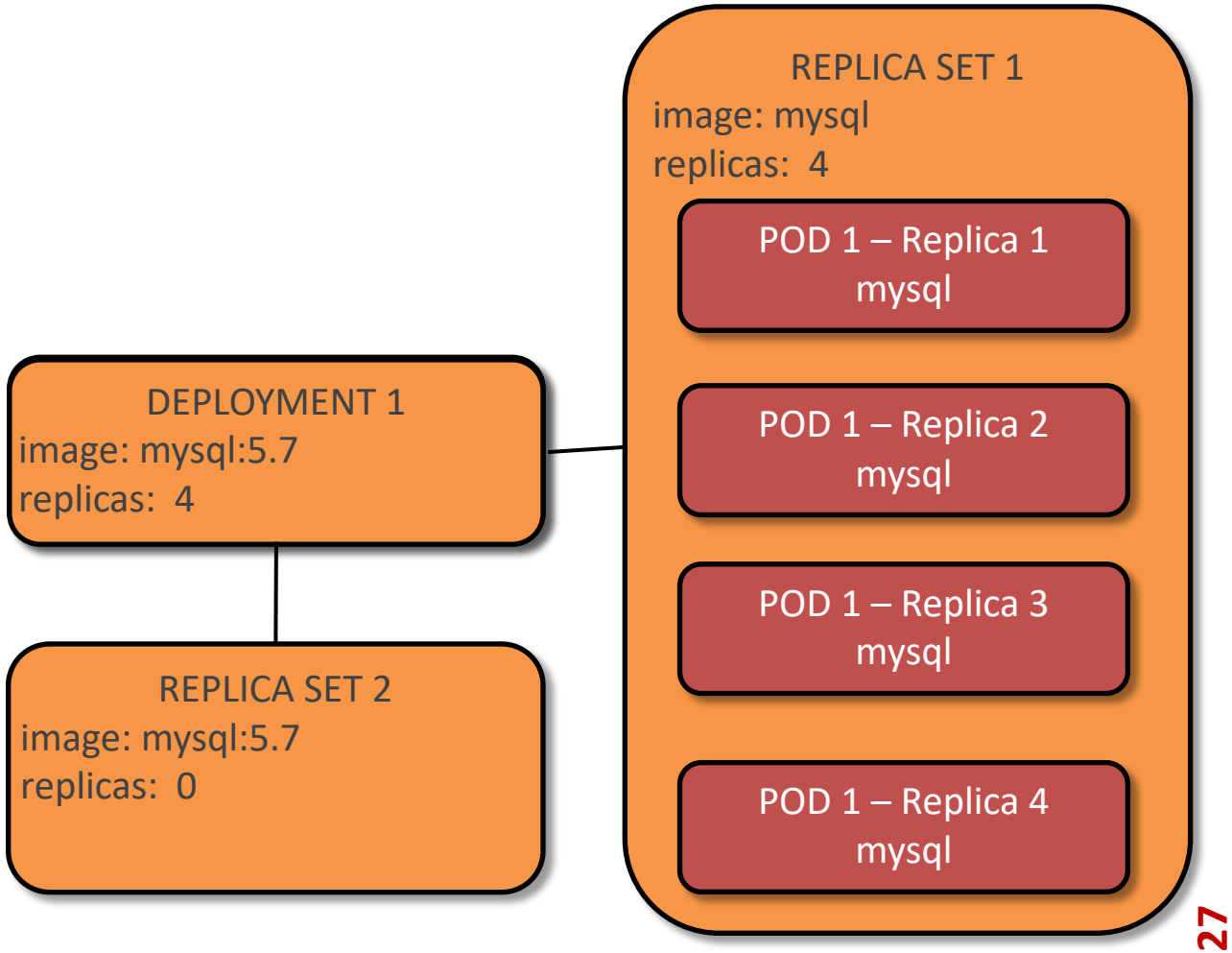
- By default, old, inactive Replica Set saved – previous version of the Deployment

Deployment Strategy: Recreate

Deployment Strategy: Recreate

Simplest strategy for deployments

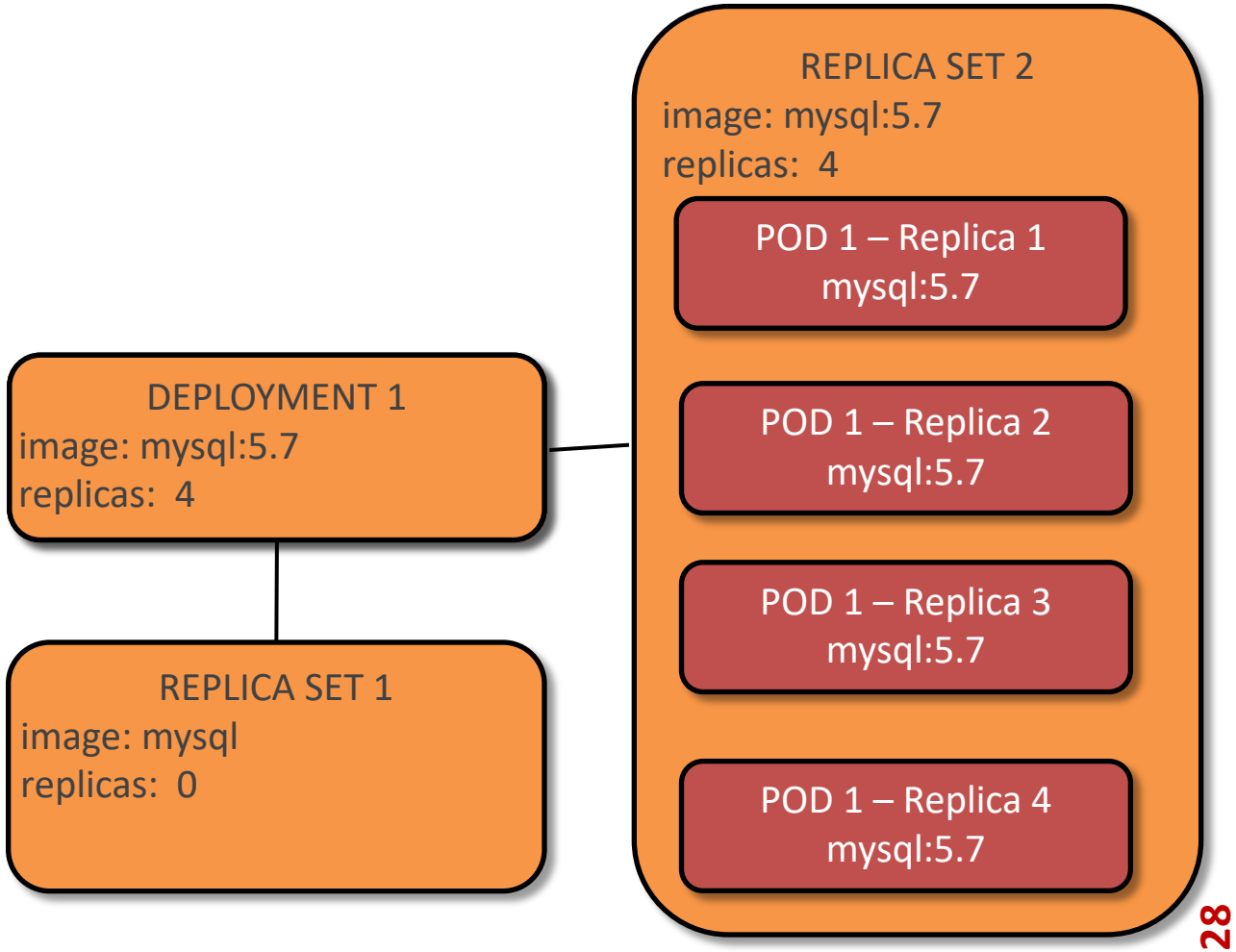
- When a change is made to a Deployment's spec, all Pods are removed and then recreated
  - Old Replica Set pods are killed
  - Then, new Replica Set starts pods
- May lead to downtime during the process while new pods are started



Deployment Strategy: Recreate

Simplest strategy for deployment updates

- When a change is made to a Deployment's template, all Pods are removed and then recreated
  - Old Replica Set pods are killed
  - New Replica Set starts pods
- May cause downtime due to delay between old pods terminating and new pods becoming available



Deployment Strategy: Recreate

Strategies are defined in the spec of a Deployment

- **strategy** parameter in Deployment spec sets the strategy to be used for updates
- If no parameter value is set, the default is **RollingUpdate**

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: tomcat-deployment
spec:
  replicas: 3
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        type: webserver
    spec:
      containers:
      - name: tomcat-container
        image: tomcat
        ports:
        - containerPort: 8080
```

Updating Using Multiple Deployments

RollingUpdate using Multiple Deployments

Controlled testing of new versions in production

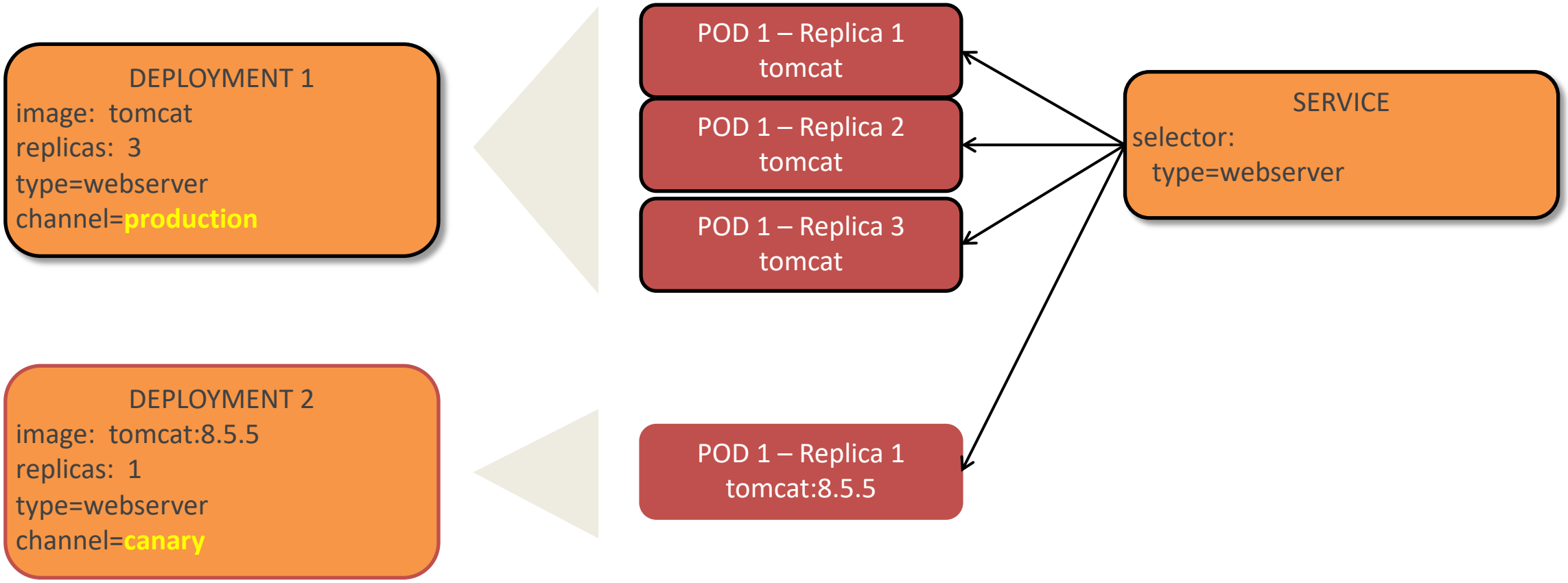
- Assume an application running as a Deployment, exposed as a Service
- To apply a new application version in production, a second Deployment can be used using labels in common with the first Deployment
  - Canary deployment allows for limited testing of new version in production
  - Blue/green deployment

Strategic Approach: Canary Deployment

Controlled testing of the update on production

- Consider a Service selecting pods from a Deployment of application pods
- In a canary deployment, a second Deployment (Canary Deployment) is created with pods for the new version, with labels matching the Service's selector
- Service directs some requests to pods on the Canary, allowing testing of changes in production
- If a malfunction is detected, it will only impact a small portion of the Pods and can be undone.

Strategic Approach: Canary Deployment



Strategic Approach: Canary Deployment

Decisions after running the canary Deployment in production

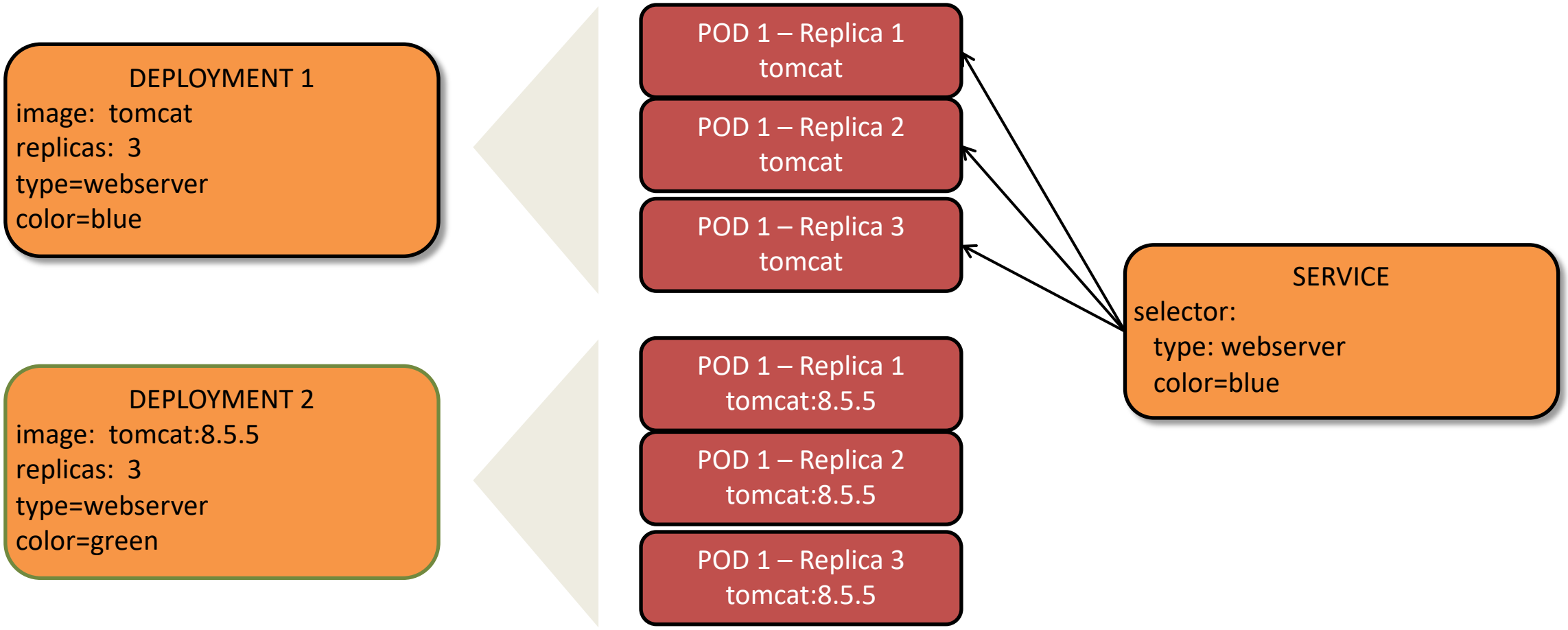
- If the application error rate is not increasing and Canary Deployment is stable:
  - The main Deployment can be updated to the newer version (using Rolling Update for example) and then the Canary can be discarded; OR
  - The Canary can be scaled up and reconfigured and the old Deployment can be discarded.
- If the test results in failure, the Canary deployment can be deleted

Strategic Approach: Blue/Green Deployment

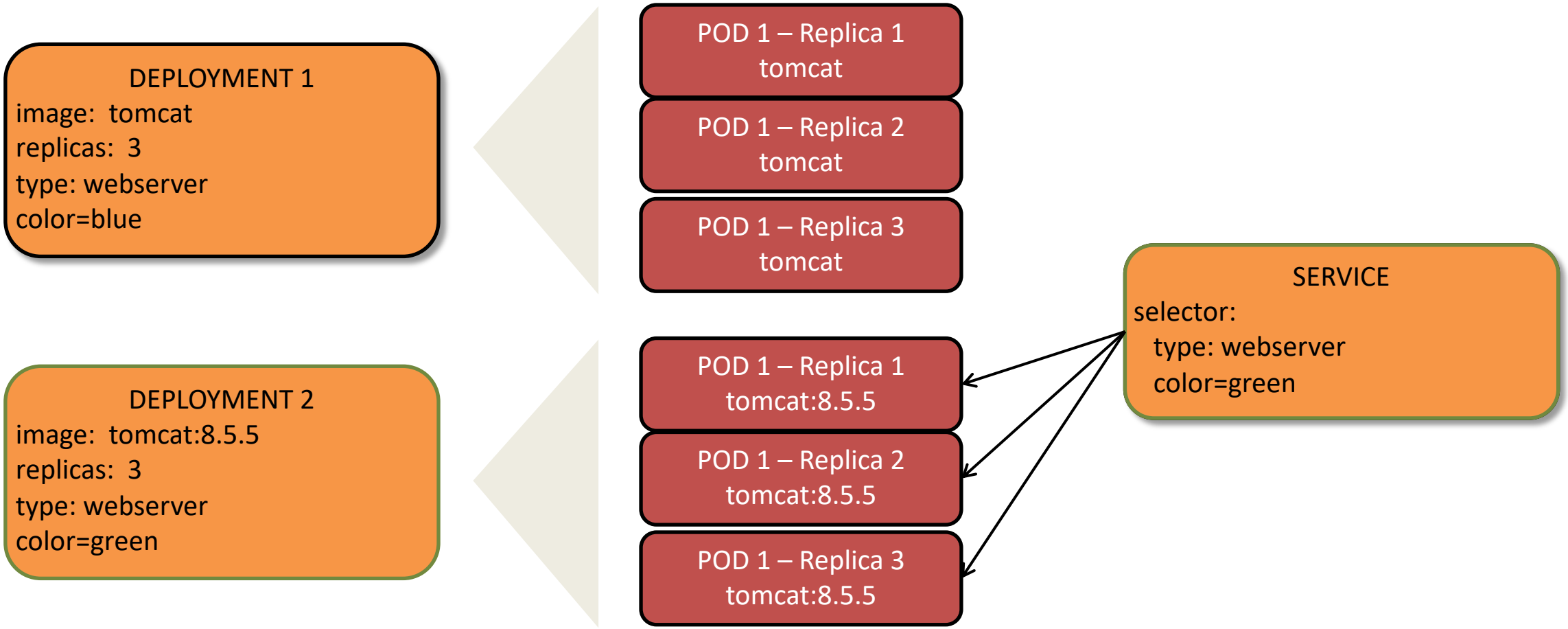
Complete environment switch from one version to another

- With a Blue/Green deployment, you create a new full-scale Deployment in addition to the current production Deployment
- Reconfiguring the pod label selector on the application's Service allows choice of directing requests to old Deployment or new Deployment
- Similar to effect of Replace strategy without application downtime

Strategic Approach: Blue/Green Deployment



Strategic Approach: Blue/Green Deployment



ConfigMap

ConfigMap

ConfigMap

- Many applications require configuration via:
  - Config Files
  - Command-Line Arguments
  - Environment Variables
- These need to be decoupled from images to keep portable
- ConfigMap API provides mechanisms to inject containers with configuration data
- Store individual properties or entire config files/JSON blobs
- Key-Value Pairs

ConfigMap

- Not meant for sensitive information
- PODs or controllers can use ConfigMaps

1. Populate the value of environment variables
2. Set command-line arguments in a container
3. Populate config files in a volume

```
kind: ConfigMap
apiVersion: v1
metadata:
   creationTimestamp: 2016-02-18T19:14:38Z
   name: example-config
   namespace: default
data:
   example.property.1: hello
   example.property.2: world
   example.property.file: |-
      property.1=value-1
      property.2=value-2
      property.3=value-3
```

ConfigMap from directory

- 2 files in docs/user-guide/configmap/kubectl
  - game.properties
  - ui.properties

game.properties

```
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
secret.code.passphrase=UUDDLRLRBABAS
secret.code.allowed=true
secret.code.lives=30
```

ui.properties

```
color.good=purple
color.bad=yellow
allow.textmode=true
how.nice.to.look=fairlyNice
```

ConfigMap from directory

```
kubectl create configmap game-config --from-file=docs/user-guide/configmap/kubectl
```

```
kubectl describe configmaps game-config
Name: game-config
Namespace: default
Labels: <none>
Annotations: <none>
Data
====
game.properties: 121 bytes
ui.properties: 83 bytes
```

ConfigMap from directory

```
kubectl get configmaps game-config -o yaml
```

```
apiVersion: v1
data:
    game.properties: |-
        enemies=aliens
        lives=3
        …
ui.properties: |
       color.good=purple
       …
kind: ConfigMap
metadata:
    creationTimestamp: 2016-02-18T18:34:05Z
    name: game-config
    namespace: default
    resourceVersion: "407"-
    selfLink: /api/v1/namespaces/default/configmaps/game-config
    uid: 30944725-d66e-11e5-8cd0-68f728db1985
```

ConfigMap from files

```
kubectl create configmap \
game-config-2 \
--from-file=file1 \
--from-file=file2
```

```
kubectl get configmaps \
game-config-2 \
-o yaml
```

```
apiVersion: v1
data:
    game.properties: |-
        enemies=aliens
        lives=3
        …
ui.properties: |
       color.good=purple
       …
kind: ConfigMap
metadata:
    creationTimestamp: 2016-02-18T18:52:05Z
    name: game-config-2
    namespace: default
    resourceVersion: ”516"-
    selfLink: /api/v1/namespaces/default/configmaps/game-config-2
    uid: b4952dc3-d670-11e5-8cd0-68f728db1985
```

ConfigMap options

--from-file=/path/to/directory
--from-file=/path/to/file1 (/path/to/file2)
Literal key=value: --from-literal=special.how=very

ConfigMap in PODs

- Populate Environment Variables

```
apiVersion: v1
kind: ConfigMap
metadata:
    name: special-config
    namespace: default
data:
    special.how: very
    special.type: charm
```

```
apiVersion: v1
kind: Pod
metadata:
    name: dapi-test-pod
 spec:
   containers:
       - name: test-container
         image: busybox
         command: [“/bin/sh”, “-c, “env” ]
         env:
             - name: SPECIAL_LEVEL_KEY
               valueFrom:
                   configMapKeyRef:
                       name: special-config
                       key: special.how
   restartPolicy: Never
```

OUTPUT

```
SPECIAL_LEVEL_KEY=very
SPECIAL_TYPE_KEY=charm
```

ConfigMap Restrictions

- ConfigMaps must be created before they are consumed
- ConfigMaps can only be referenced by objects in the same namespace
- Quota for ConfigMap size not implemented yet
- Can only use ConfigMap for PODs created through API server.

Secrets

Application Secrets

What secrets do applications have?

- Database credentials
- API credentials & endpoints (Twitter, Facebook etc.)
- Infrastructure API credentials (Google, Azure, AWS)
- Private keys (TLS, SSH)
- Many more!

Application Secrets

It is a bad idea to include these secrets in your code.

- Accidentally push up to GitHub with your code
- Push into your file storage and forget about
- Etc.

Application Secrets

There are bots crawling GitHub searching for secrets

Real life example:
Dev put keys out on GitHub, woke up next morning with a ton of emails and missed calls from Amazon

- 140 instances running under Dev's account.
- $2,375 worth of Bitcoin mining

Create Secret

- Designed to hold all kinds of sensitive information
- Can be used by Pods (filesystem & environment variables) and the underlying kubelet when pulling images

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
Type: Opaque
data:
   password: mmyWfoidfIuL==
   username: NyhdOKwB
```

Pod Secret

```
kind: Pod
metadata:
  name: secret-env-pod
spec:
 containers:
 - name: mycontainer
   image: redis
   env:
   - name: SECRET_USERNAME
       valueFrom:
         secretKeyRef:
           name: mysecret
           key: username
```

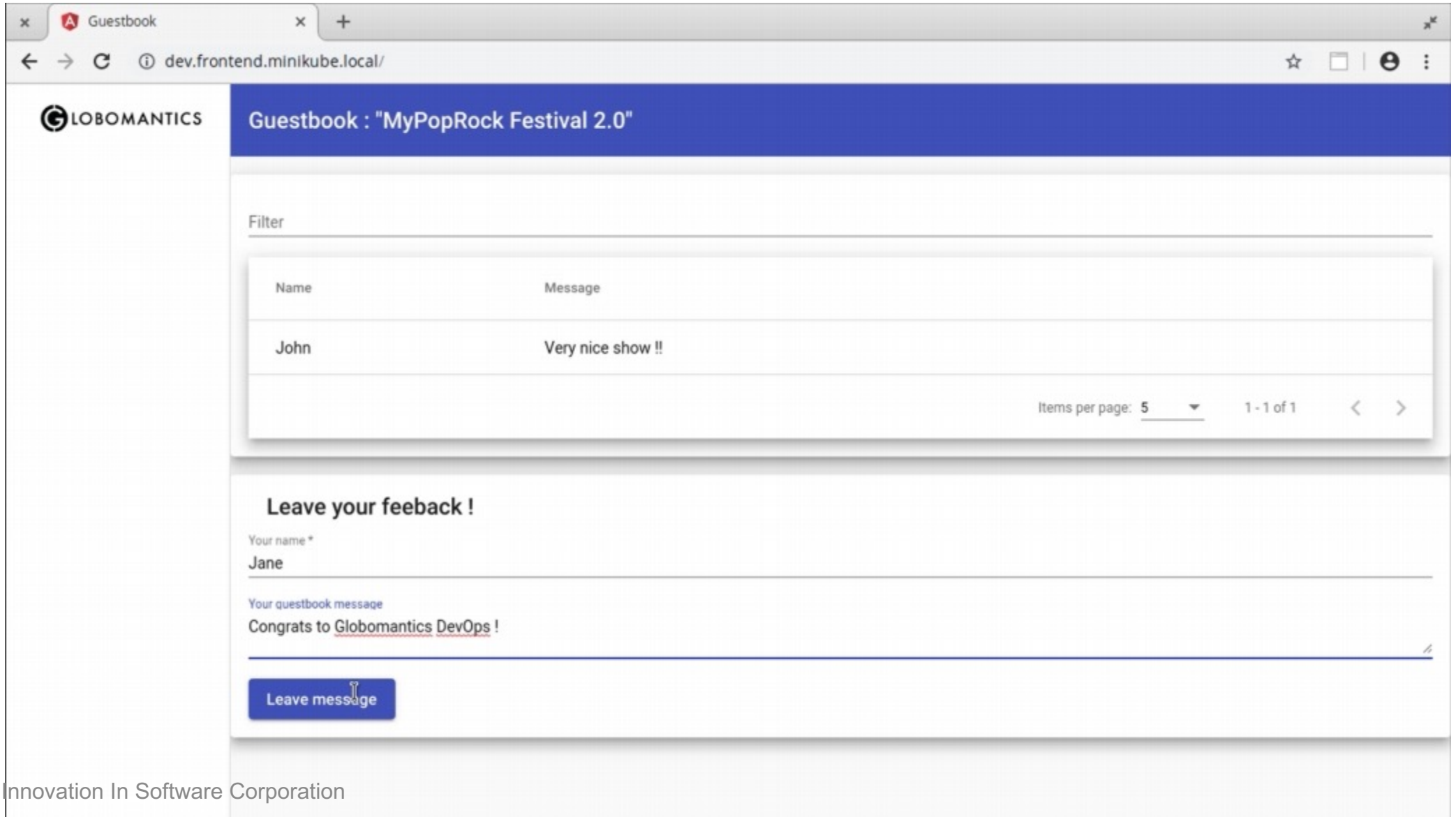
Volume Secret

```
spec:
  containers
  - name: mycontainer
    image: redis
    volumeMounts:
    - name: “secrets”
      mountPath: “/etc/my-secrets”
      readOnly: true
  volumes:
   - name: “secrets”
      secret:
        secretName: “mysecret”
```

Labs: ConfigMap & Secrets

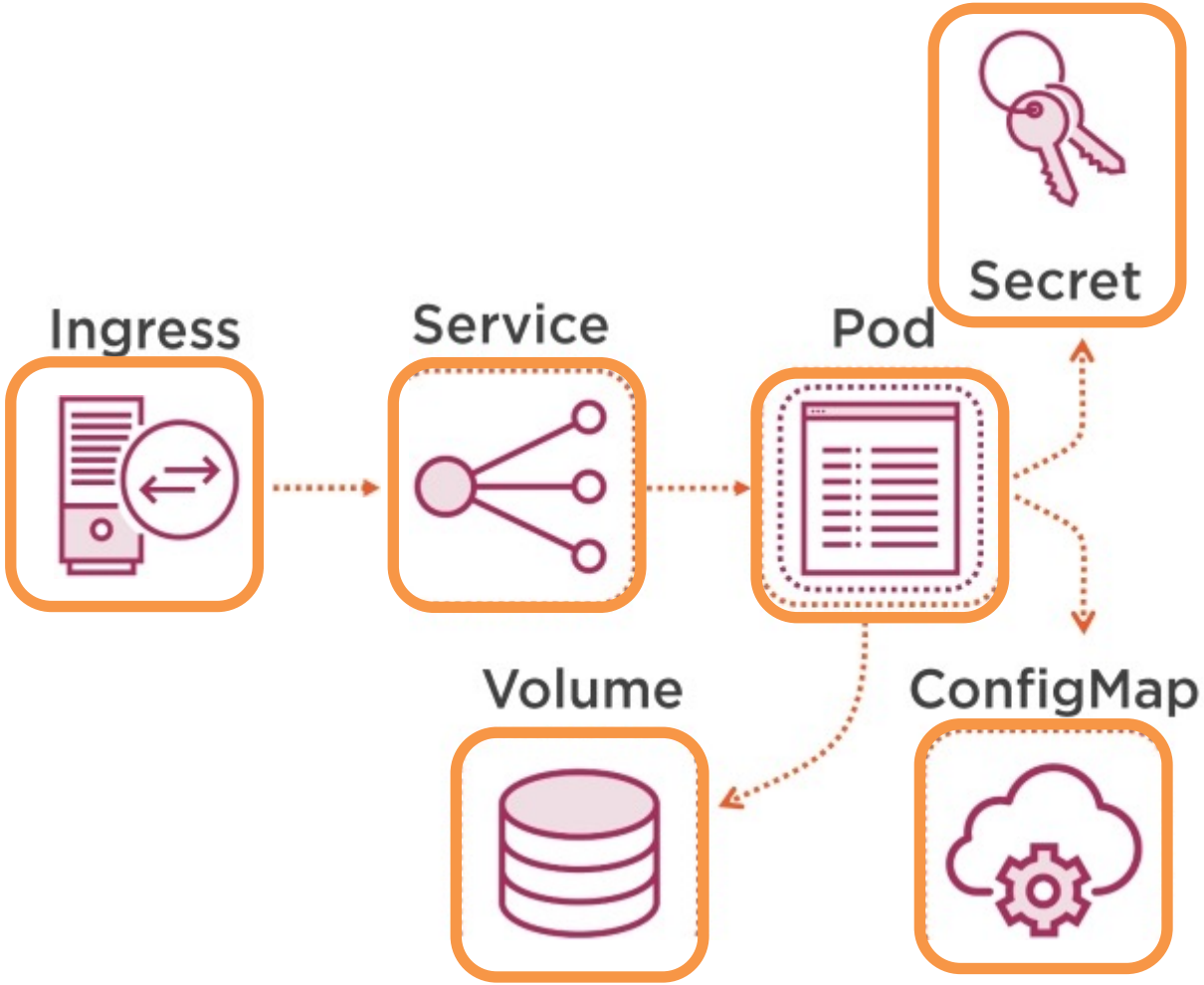
Helm: Package management

Helm sample application



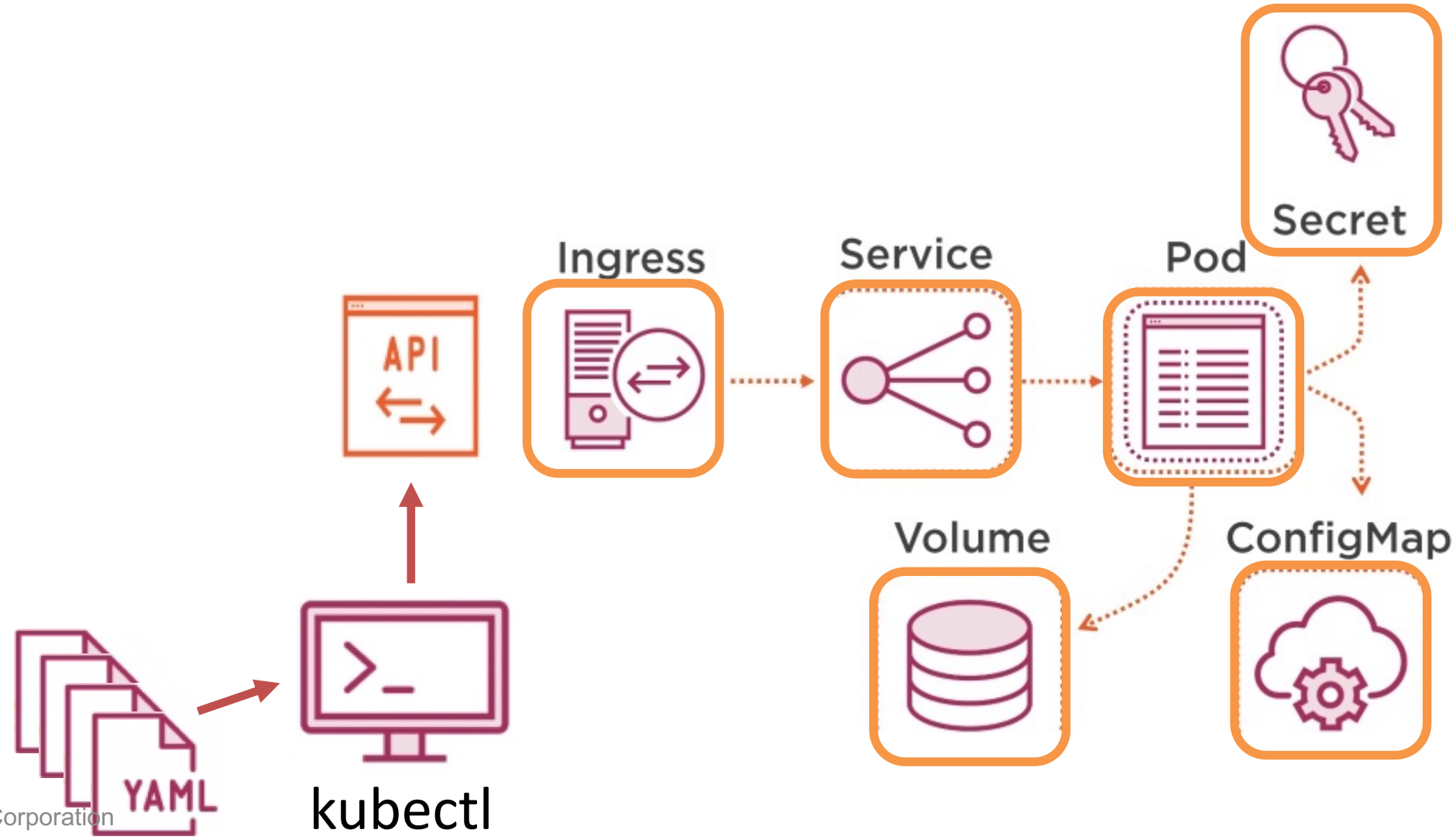
Native Kubernetes way

- Application
- Container
- Pod
- Service
- Ingress
- ConfigMap
- Secrets
- Volumes: PV, PVC, Storage



Native Kubernetes way

- Limitations
  - Packaging
  - Versioning



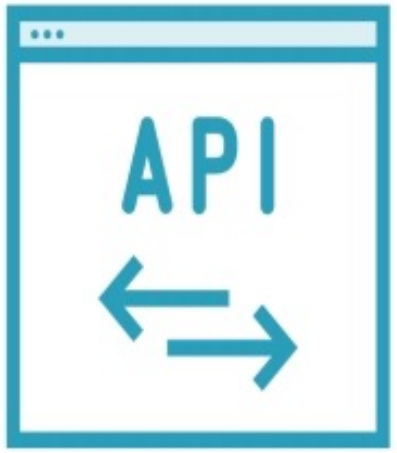
Guestbook: version 1

- ConfigMap
- Pod
- Service
- Ingress

Frontend

Guestbook: version 2

- ConfigMap
- Pod
- Service
- Ingress



- Secret
- Pod
- Service

- Secret
- PV
- PVC
- Pod
- Service

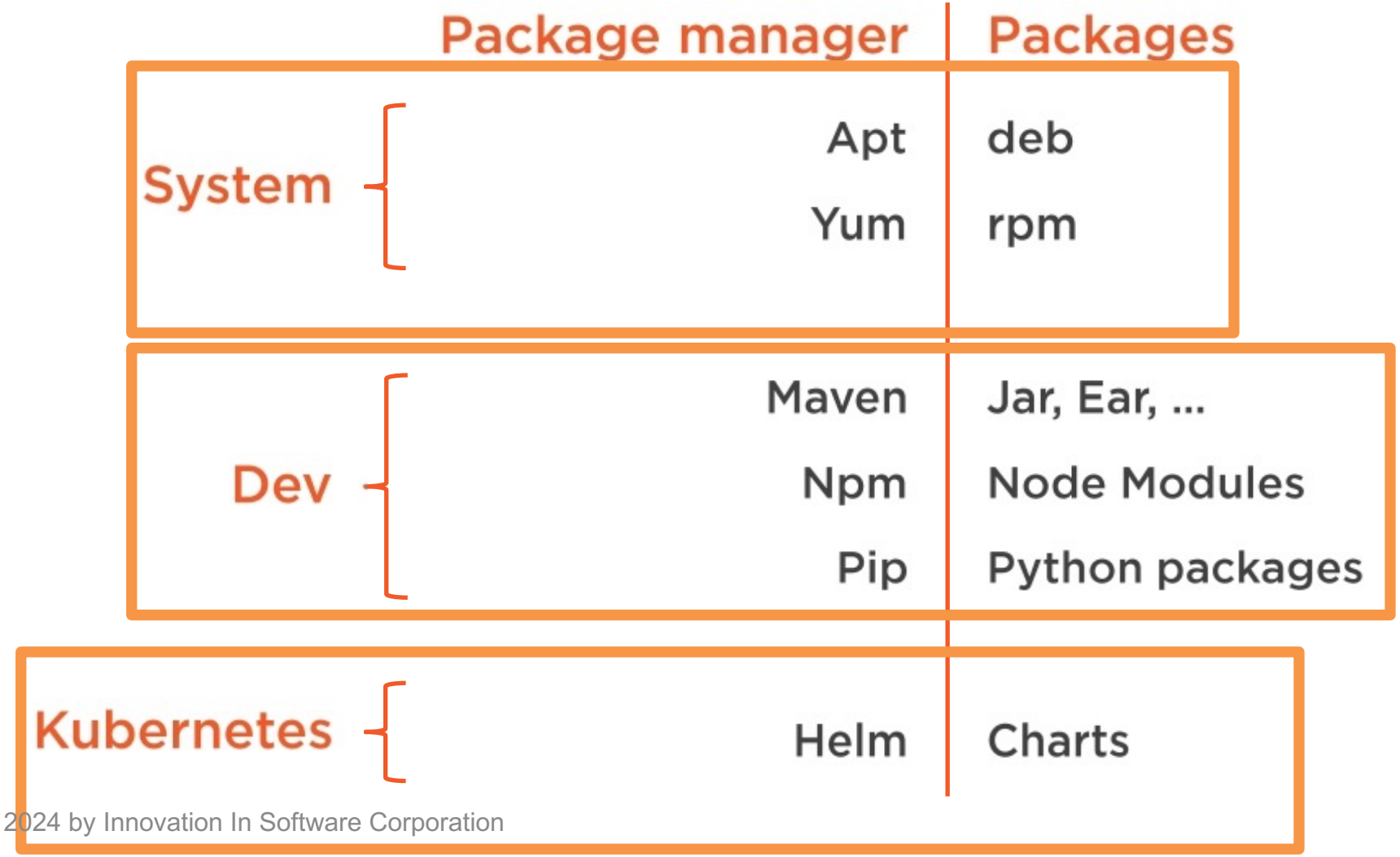
Frontend

Backend

Database

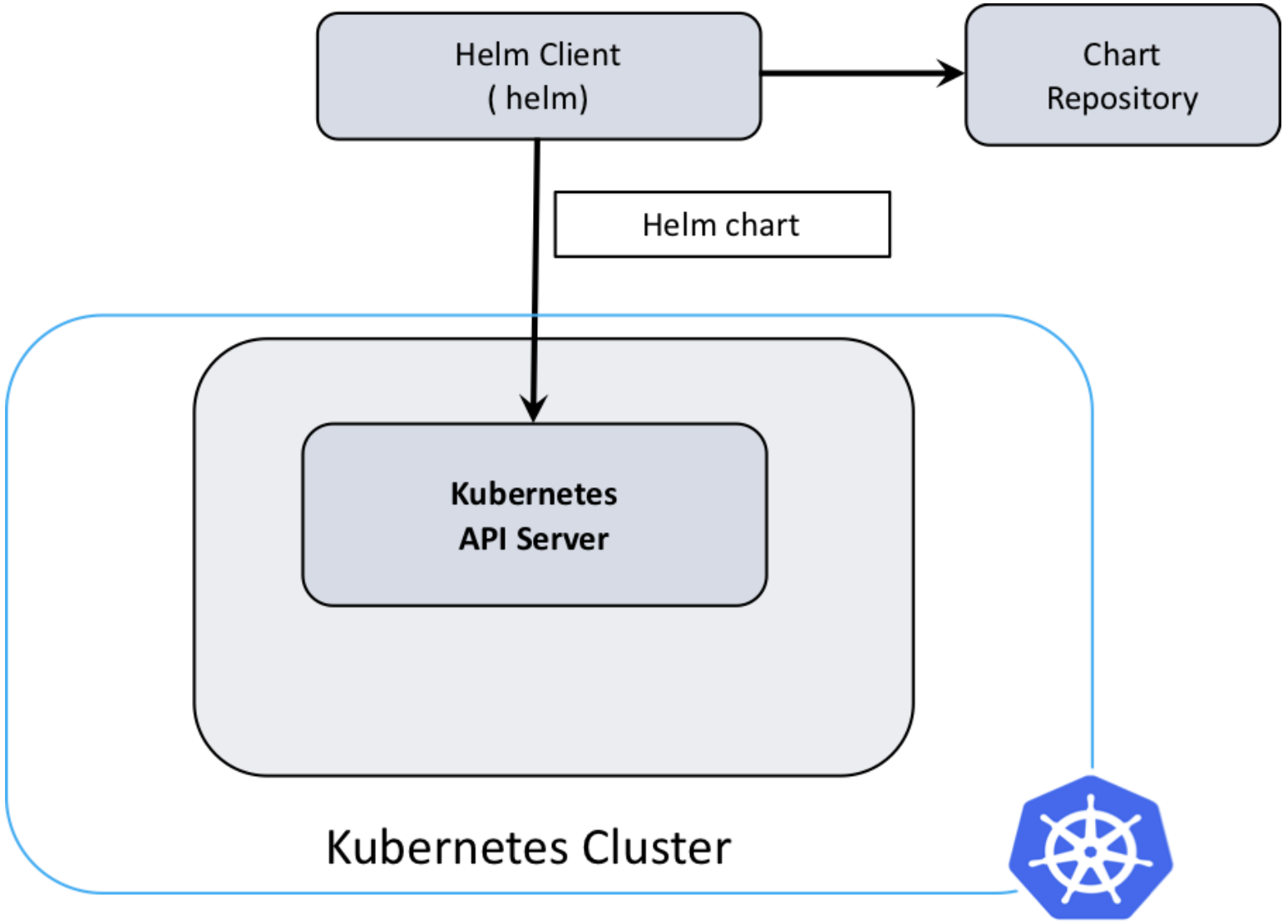
Painful to manage

Helm features



Helm architecture

- Helm:
  - develop charts locally
  - command-line



Helm features

Charts

Templates

Dependencies

Repositories

Helm Chart Structure

Helm Chart structure

- nginx-demo
  - Chart.yaml

- Chart properties
  - name
  - version
  - more..

Helm Chart structure

- nginx-demo
  - Chart.yaml
  - README.md

- Document chart
  - Overrides
  - Maintainer
  - Instructions

Helm Chart structure

- nginx-demo
  - Chart.yaml
  - README.md
- templates
  - deployment.yaml
  - ingress.yaml
  - service.yaml

- Kubernetes object definitions
  - customizable YAML templates

Helm Chart structure

- nginx-demo
  - Chart.yaml
  - README.md
- templates
  - deployment.yaml
  - ingress.yaml
  - service.yaml
- values.yaml

- Kubernetes object definitions
  - customizable YAML templates
    - Provide default values.

Helm Chart structure

- nginx-demo
  - Chart.yaml
  - README.md
- templates
  - deployment.yaml
  - ingress.yaml
  - service.yaml
  - NOTES.txt
- values.yaml

- Provide helpful output after installation
  - How to access application
  - Create user/pass

Helm Chart structure

- nginx-demo
  - Chart.yaml
  - README.md
- templates
  - deployment.yaml
  - ingress.yaml
  - service.yaml
  - NOTES.txt
  - tests
    - test-connection.yaml
- values.yaml

- You can create tests to confirm Chart works as expected.

Helm Chart structure

- nginx-demo
  - Chart.yaml
  - README.md
  - requirements.yaml
- templates
  - deployment.yaml
  - ingress.yaml
  - service.yaml
  - NOTES.txt
  - tests
    - test-connection.yaml
- values.yaml

- Define sub-charts and dependencies

Helm Chart.yaml

Chart.yaml

```
apiVersion: v1
appVersion: "1.0"
description: A Helm chart for Kubernetes
name: nginx-demo
version: 0.1.0
```

Major,Minor,Patch (SemVer 2.0)

- App you are installing

Helm Chart.yaml

Chart.yaml

```yaml
apiVersion: v1
appVersion: "1.0"
description: A Helm chart for Kubernetes
name: nginx-demo
version: 0.1.0
```

Major,Minor,Patch (SemVer 2.0)

- Version of Helm chart

Common Helm commands

Action	Command
Install a Release	`helm install [chart]`
Upgrade a Release revision	`helm upgrade [release] [chart]`
Rollback to a Release revision	`helm rollback [release] [revision]`
Print Release history	`helm history [release]`
Display Release status	`helm status [release]`
Show details of a release	`helm get [release]`
Uninstall a Release	`helm delete [release]`
List Releases	`helm list`

Lab: Helm

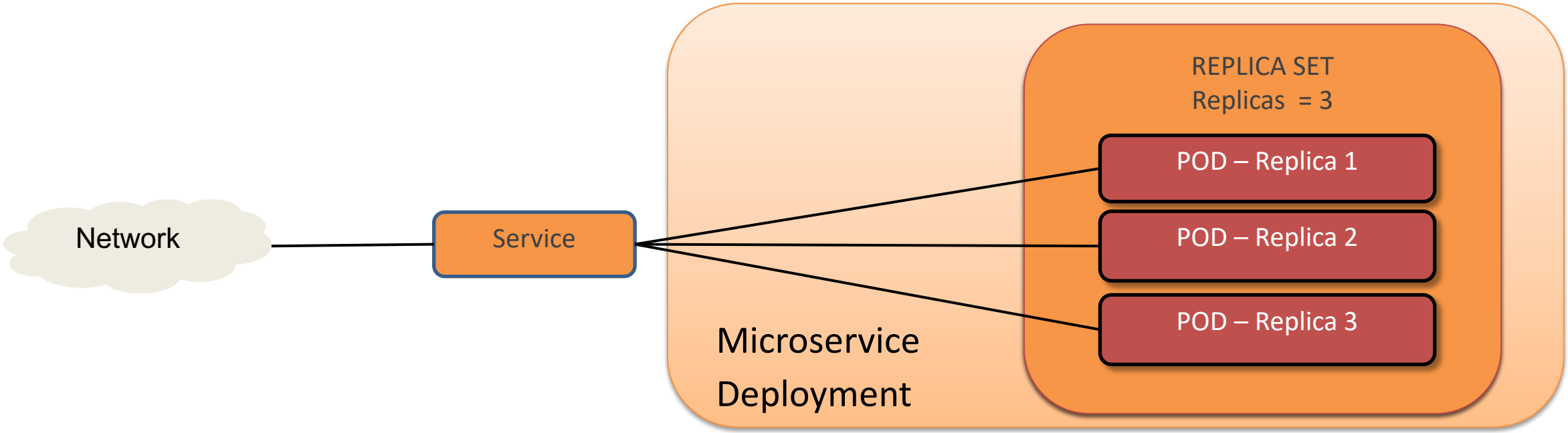
Scaling Deployments

Horizontal Pod Autoscaler (HPA)

Kubernetes Deployments for Microservices

Kubernetes is perfect for Microservices management

- Applications built as services or collections of microservices easily modeled using Kubernetes Deployment and Service resources
- Key property is that application or component can be scaled horizontally



Basic Microservices Principles

- Application is composed of in(ter)dependent, independently deployable components
  - E.g. containerized components packaged as Docker images
- Design of application is service-oriented, with information exchange between components via standard interfaces, e.g.
  - RESTful APIs
  - AMQP messaging
- Maximize use of stateless components that can be scaled horizontally via replication
  - Horizontal scaling provides elastic capacity for handing demand

Application Scaling Strategies

Several ways to achieve scalability

- **Pre-defined application scale**: application deployed with a predefined, static number of resources => not the Kubernetes way
- **Manual scaling**: deployment scale reconfiguration driven by operator
- **Auto-scaling**: automatic scaling of resources based on a defined trigger (number of hits, CPU usage, etc)

Application Scaling Strategies

Manual scaling

- K8s supports manual scaling of Deployment to adjust replica count
- Operator just needs to adjust declaration of how many replicas are desired, and K8s system will manage to that number
  - Pod creation and placement completely automatic
  - Pods automatically re-created if nodes fail

```
$ kubectl scale --replicas=2 deployment/tomcat

$ kubectl scale --replicas=8 deployment/tomcat

$ kubectl scale --replicas=4 deployment/tomcat
```

```
$ kubectl edit deployment/tomcat
```

```
$ kubectl apply –f simpledeployment.yaml
```

Application Scaling Strategies

Auto-scaling in Kubernetes

- Kubernetes has a controller object for automatic scaling of Deployments (or Replica Sets or ReplicationControllers)
- HorizontalPodAutoscaler is control loop to adjust scale of pod set depending on one or more metrics, evaluated at configurable interval (default 30s)
  - Requires metrics-server to be deployed on the cluster to provide metric data
  - Metrics include CPU and RAM utilization for pods
- Typical scenario: increase Deployment replica count when average Pod CPU utilization is above threshold value for specified period of time
- Note: application must support horizontal scaling!

Application Scaling Strategies

Auto-scaling in Kubernetes

- HPA will either reduce or increase the number of pods in the set to a level that will enable each pod in the set to match the desired baseline usage as closely as possible. The baseline is calculated based on the CPU limit in the pod specifications, for example:

```
resources:
  requests:
    cpu: 25m
  limits:
    cpu: 100m
```

Automatic Scaling of a Deployment

Creating a HorizontalPodAutoscaler resource

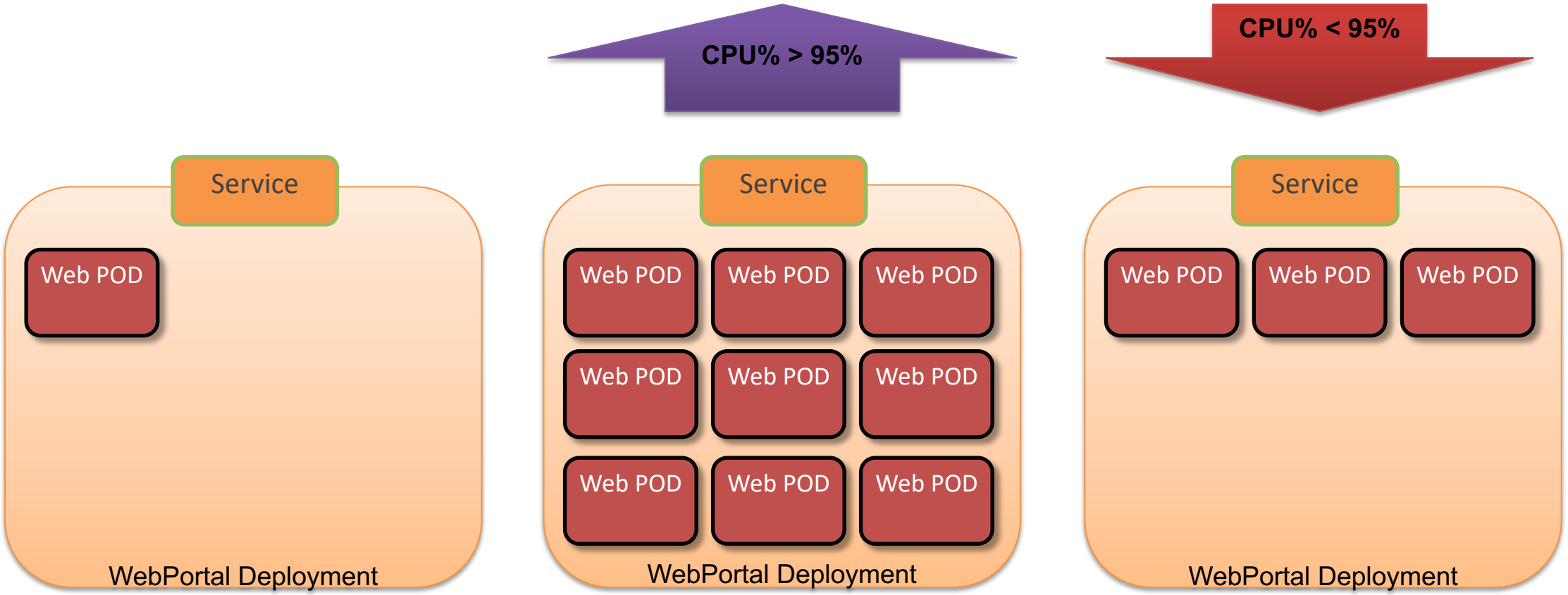
- The hpa is created similarly to any other Kubernetes resource
  - Use a manifest file for source tracking
  - Can create directly with *kubectl*
- 30 seconds is the default for considering the threshold
- Scaling is metrics-driven, either resource metrics from metrics-server, custom metrics API, or external metrics API

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: tomcat-autoscaler
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: tomcat-deployment
  minReplicas: 2
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

simplescaler.yaml

Auto-Scaling of a Deployment

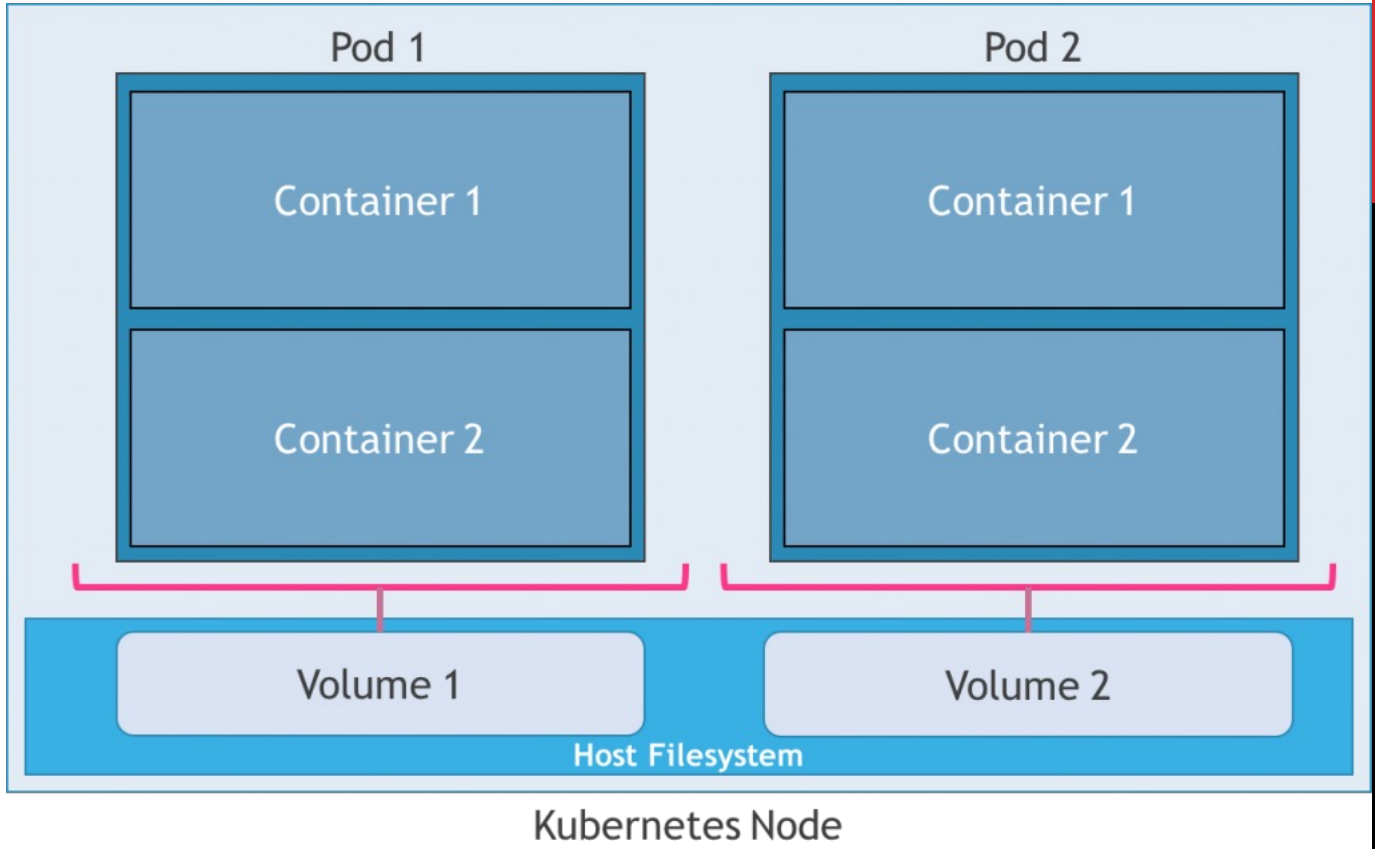
HorizontalPodAutoscaler with a 95% CPU usage threshold



Persistent Volumes

Host Based Storage

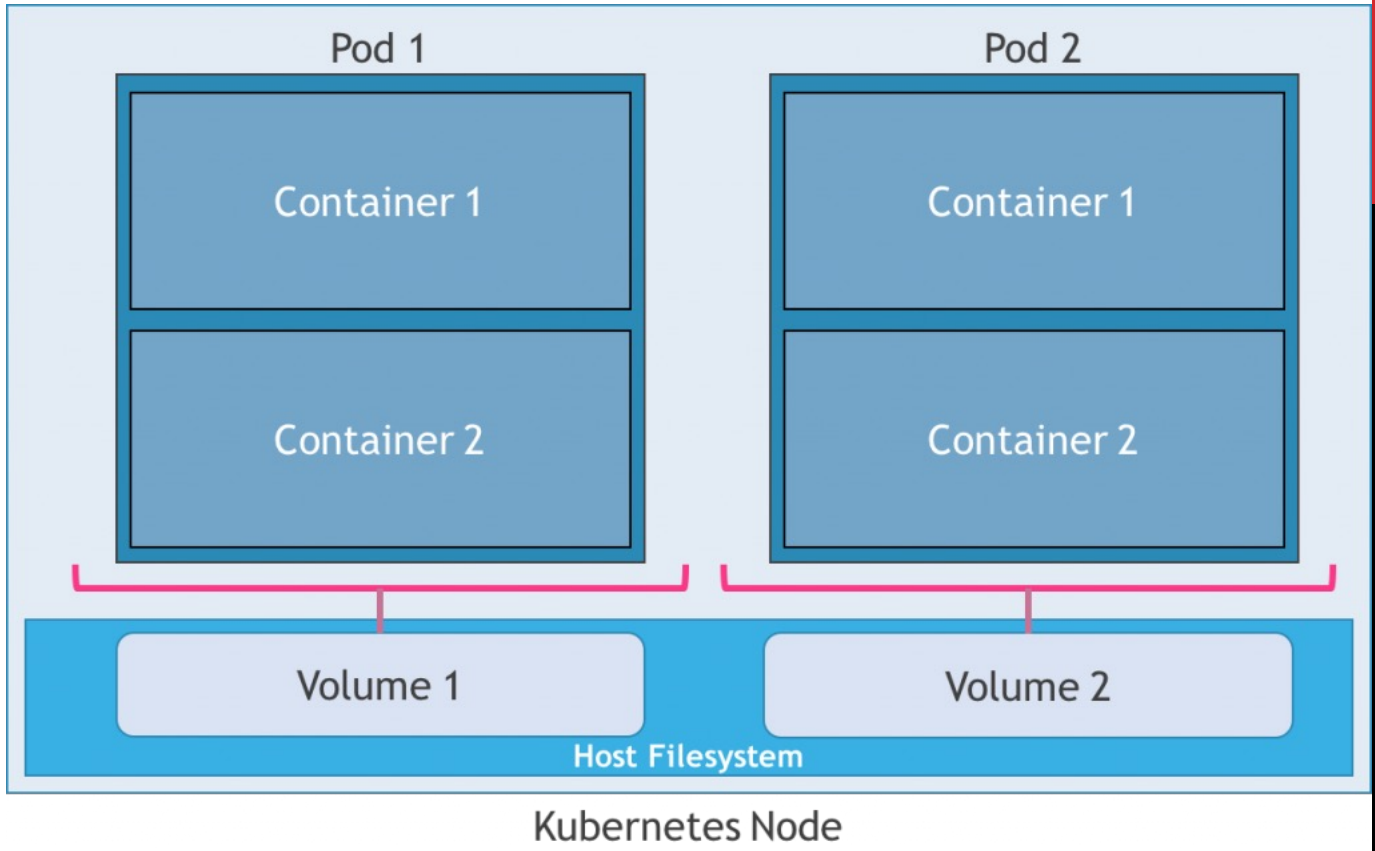
- Containers in a Pod share the same volume
- Host based storage
  - Local filesystem
- Removed when Pod is deleted
- Types of local storage
  - emptyDir
  - hostPath



Host Based Storage

- Common use cases
  - Scratch disk
  - Store temp config data

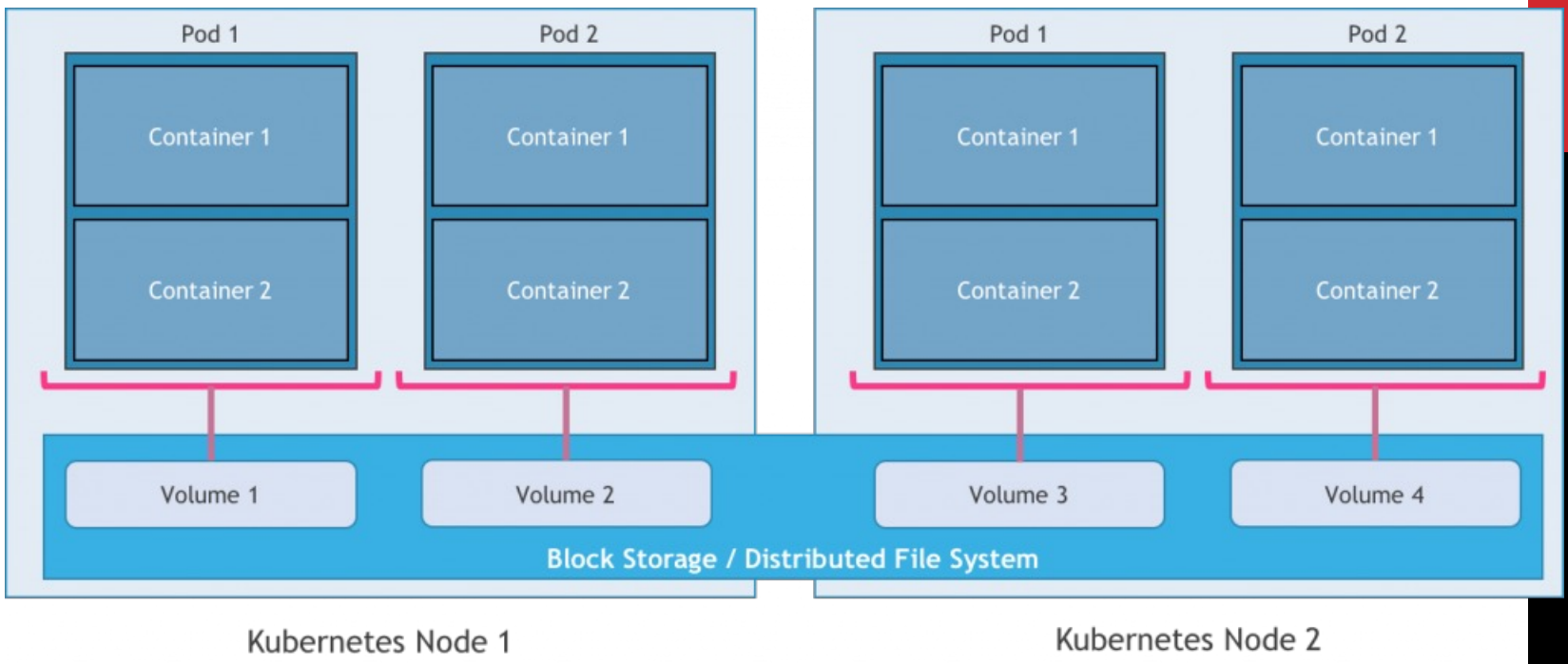
- hostPath
  - Directories on host
  - Owned by root



Non-Host Based Storage

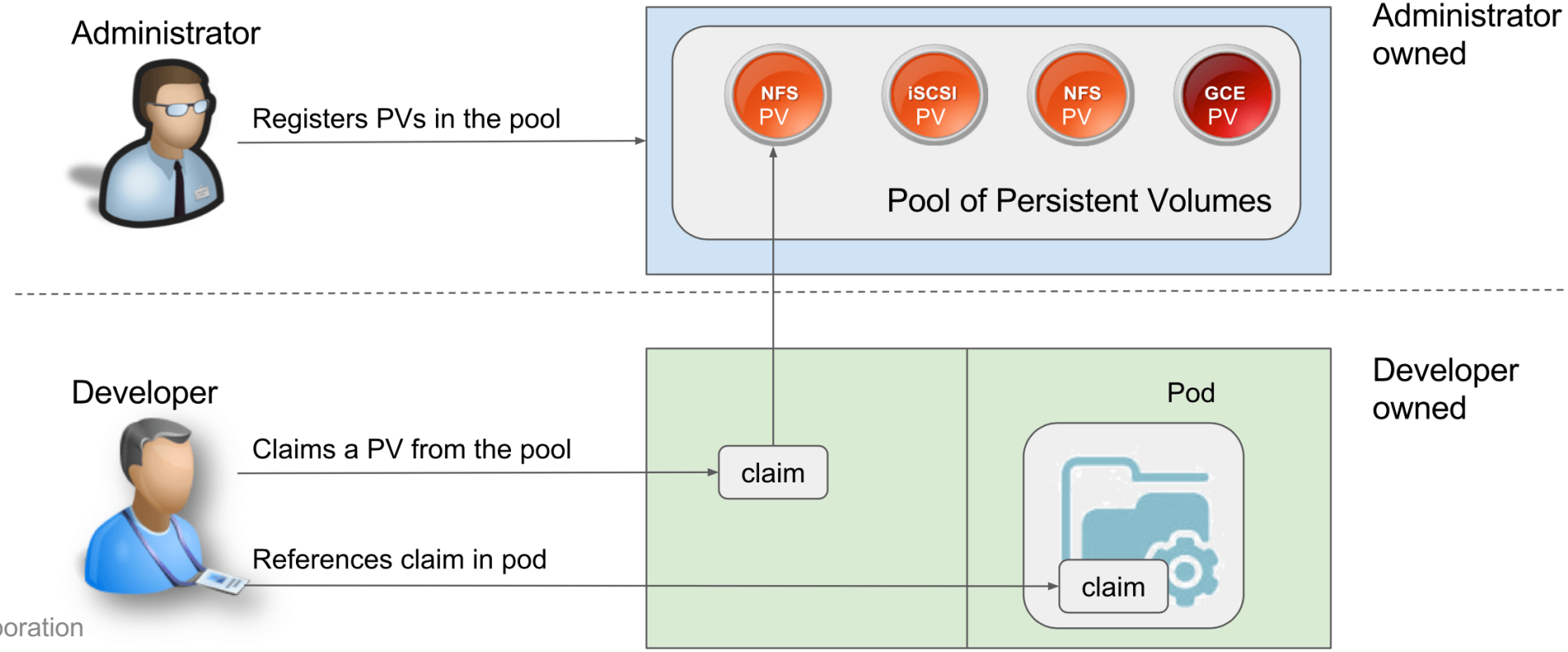
- Common use cases
  - Persistent data
  - DB, Stateful apps

- EBS
- GCE Persistent Disks
- NFS, NetApp, GlusterFS



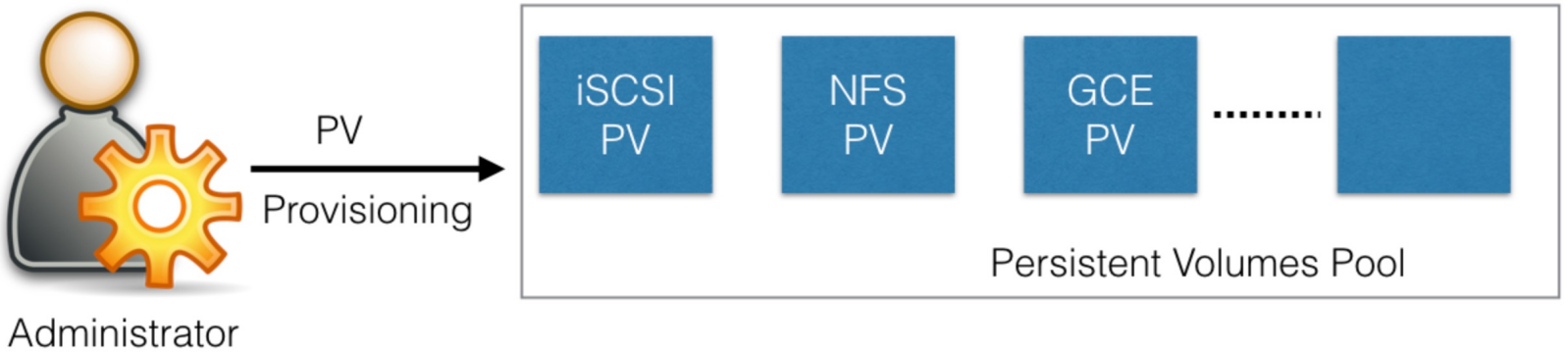
Persistent Volumes & Claims

- Operations creates PVs
  - Real storage details
  - Storage “LUN”
- Devs claim a PV
  - Self-service
  - Flexibility
  - Speed



Persistent Volumes

- Admin managed
- Quotas
- Access control
- Storage pools

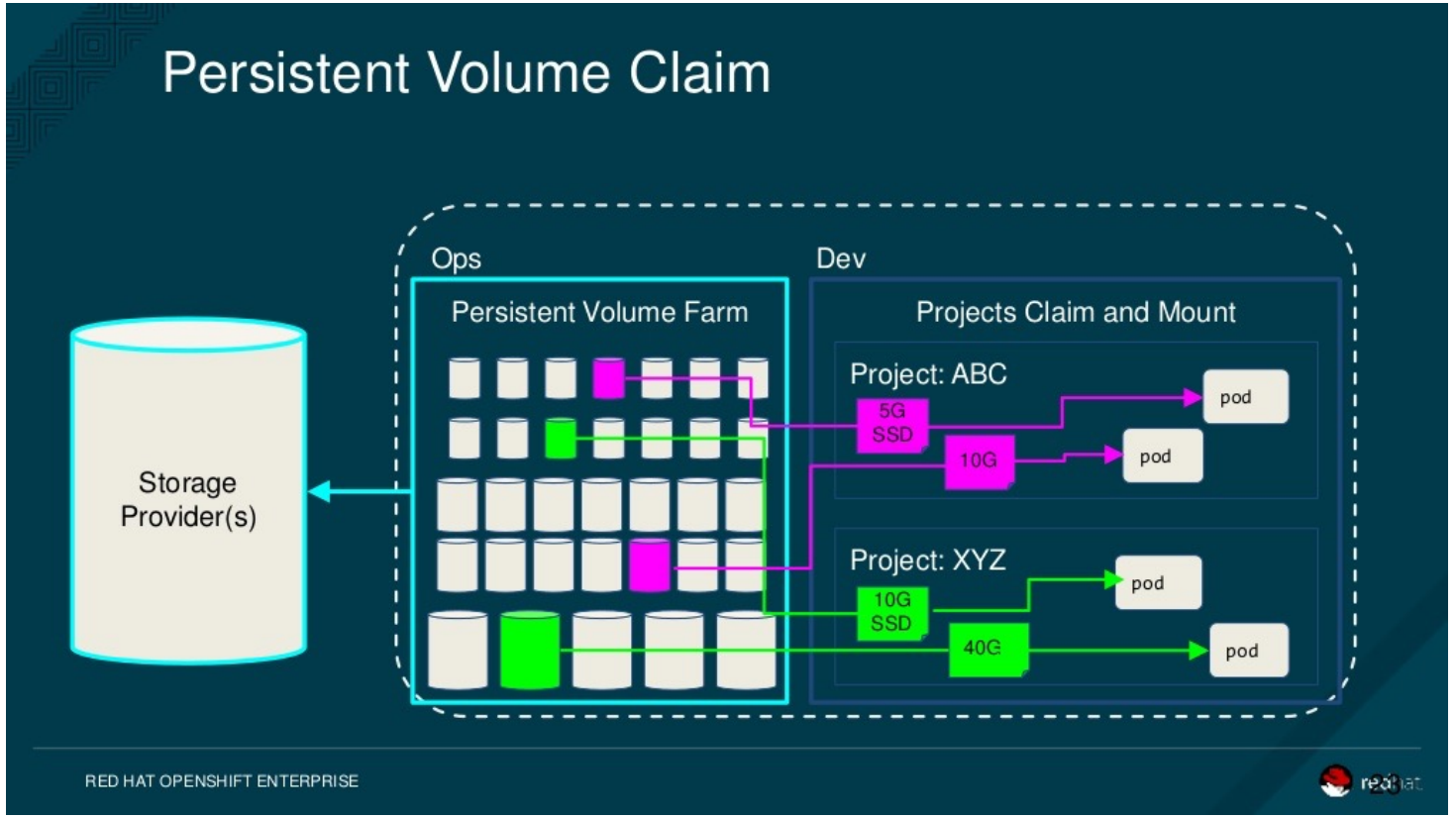


Persistent Volumes

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-aws
spec:
  capacity:
    storage: 12Gi
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  awsElasticBlockStore:
    fsType: "ext4"
    volumeID: "vol-f37a03aa"
```

Persistent Volume Claims

- Allows Devs to claim storage without worrying about management.
- Devs only see storage assigned to projects they are in.



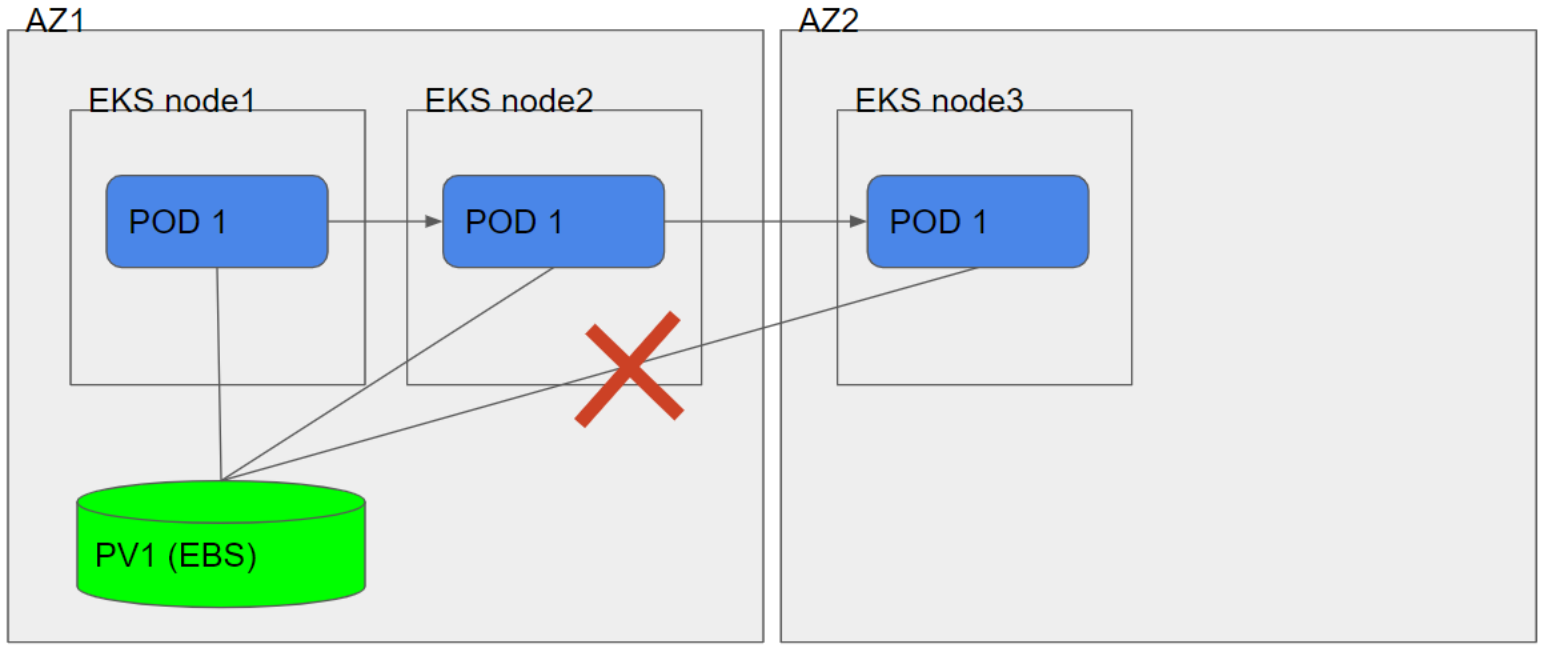
Persistent Volume Claims

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pv-claim
  labels:
    app: mongodb
spec:
  accessModes:
   - ReadWriteOnce
  resources:
     requests:
       storage: 10Gi
```

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: rsvp-db
spec:
  replicas: 1
  template:
    metadata:
      labels:
        appdb: rsvpdb
    spec:
      containers:
      - name: rsvpd-db
        image: mongo:3.3
        ports:
        - containerPort: 27017
        volumeMounts:
        - name : mongodb-persistent-storage
          mountPath : /data/db
      volumes:
      - name: mongodb-persistent-storage
        persistentVolumeClaim:
          claimName: mongodb-pv-claim
```

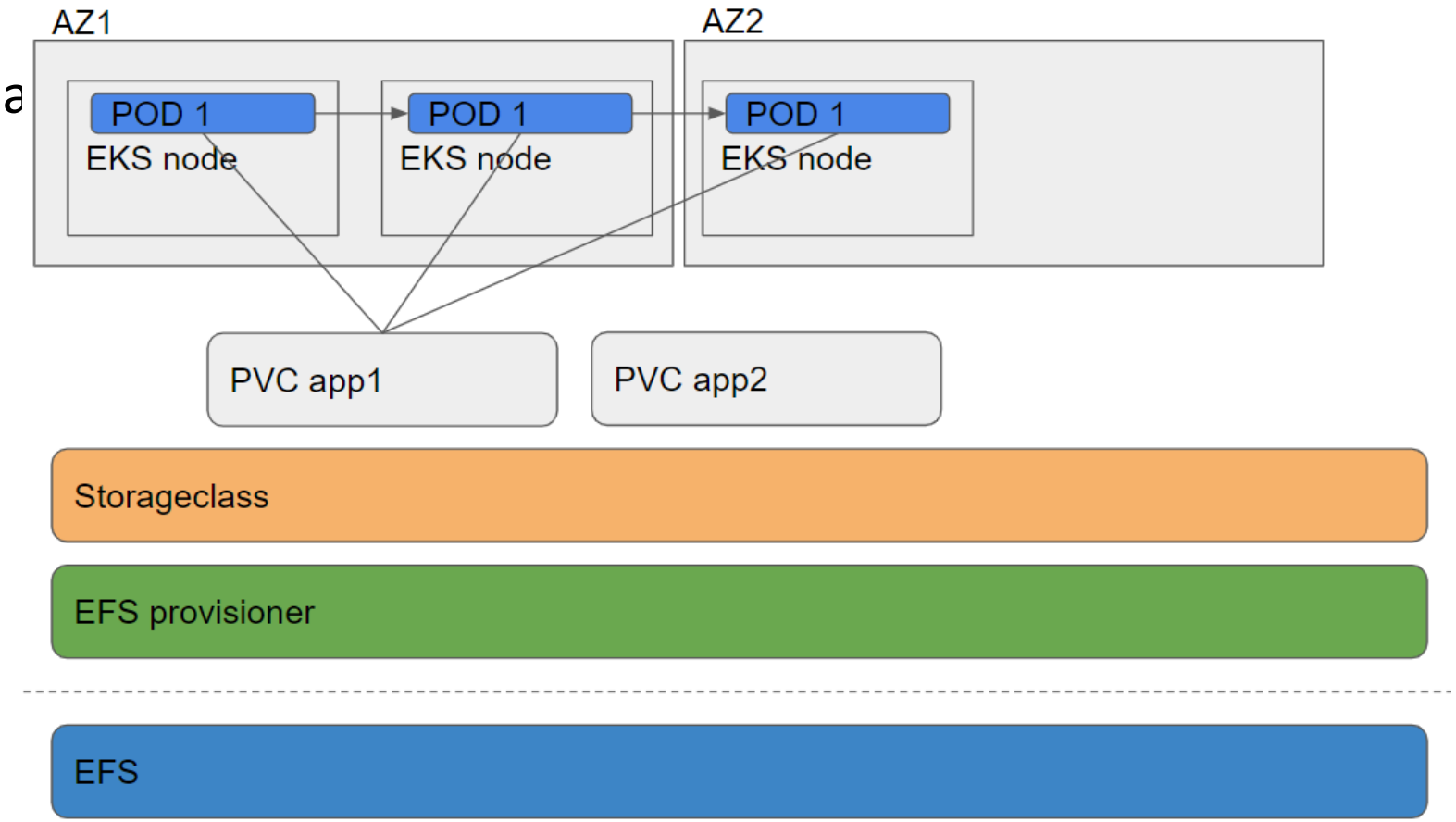
Elastic Block Store

- EBS volumes are only available in a single AZ.
- If you have nodes in multiple AZs the volume can only connect to one.



Elastic File Store

- EBS volumes are available in many AZs.
- Can connect to multiple pods a once.
- File locking



StatefulSets

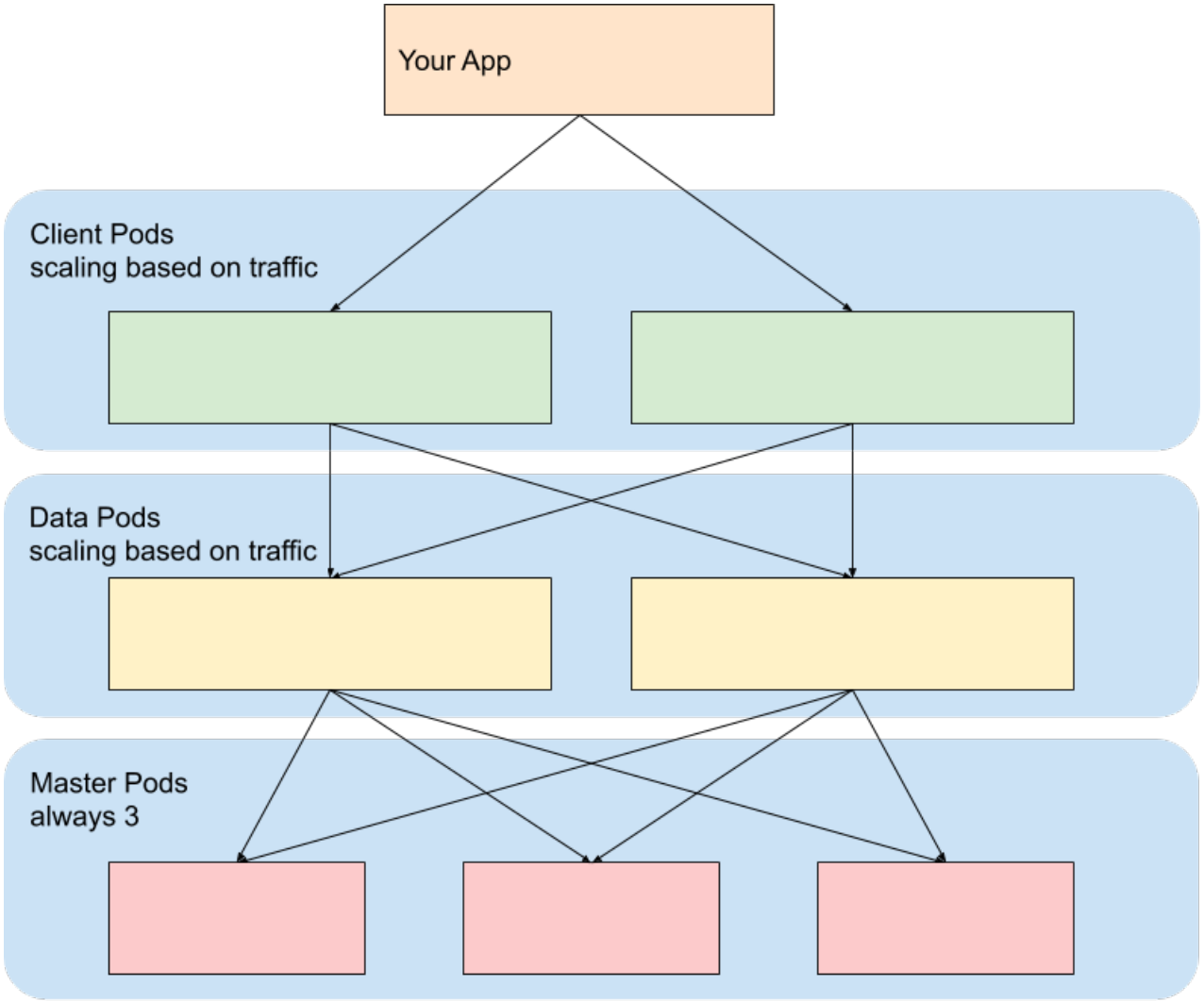
StatefulSets

- StatefulSets require a persistent volume for storing state.
  - Ensure the same PersistentVolumeClaim stays bound to the same Pod throughout its lifetime.
- Deployments:
  - Ensures the group of Pods within the Deployment stay bound to a PersistentVolumeClaim.
- Headless Service:
  - No load balancing, single dedicated IP

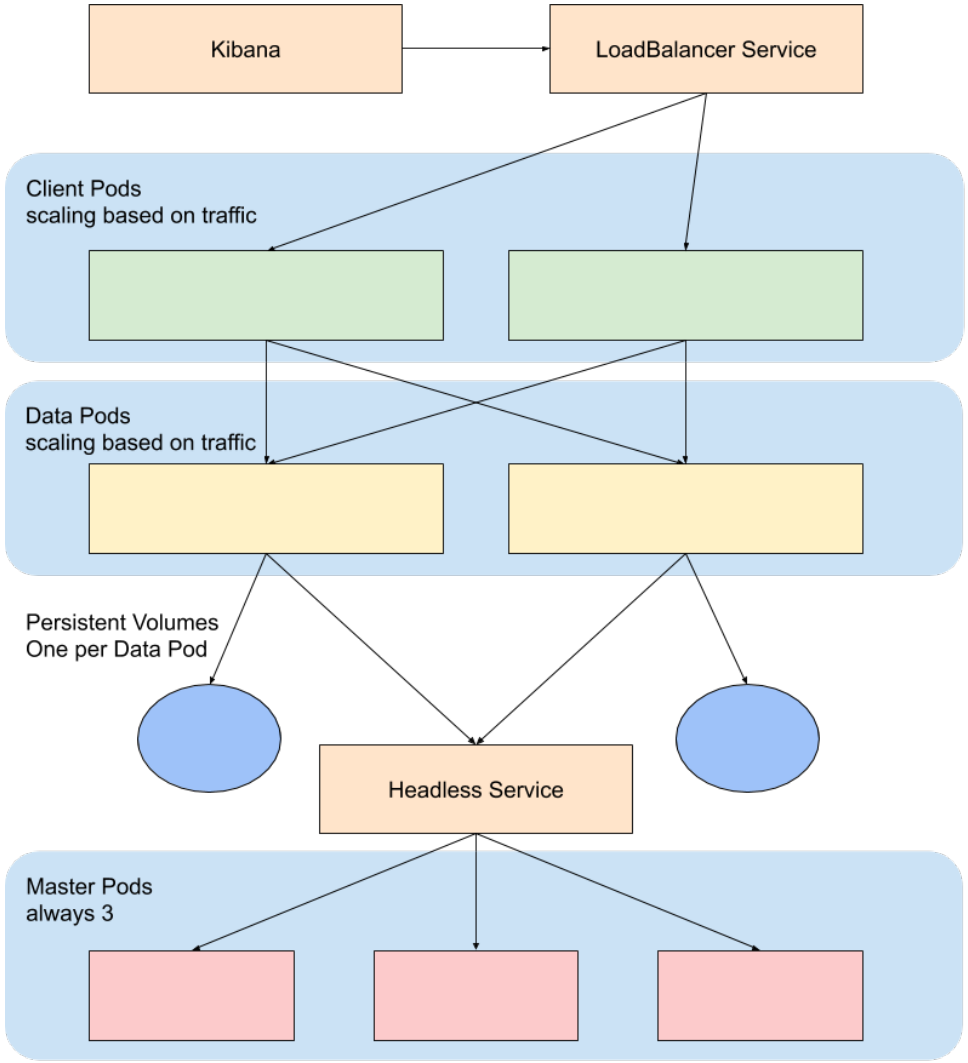
StatefulSets

- Ordinal scaling
  - Pod with a unique naming convention. e.g. If you create a StatefulSet with name Elastic, it will create a pod with name elastic-0, and for multiple replicas of a StatefulSet, their names will increment like Elastic-0, Elastic-1, Elasic-2, etc

StatefulSets: Elasticseach



StatefulSets: Elasticseach



Lab: StatefulSets