

Introduction to Kubernetes



Fall 2020, CSCI-GA 2820, Graduate Division, Computer Science

Instructor:
John J Rofrano

Senior Technical Staff Member | DevOps Champion
IBM T.J. Watson Research Center
rofrano@cs.nyu.edu (@JohnRofrano) 

What will you learn?

- What is Kubernetes
- Why would you use a Kubernetes for deployment
- Kubernetes Architecture Overview
- How to deploy your own containers in Kubernetes





Source for This Lab

- The source for this lab can be found on GitHub at:

```
$ git clone https://github.com/nyu-devops/lab-kubernetes.git  
$ cd lab-kubernetes  
$ vagrant up
```

Kubernetes



Kubernetes is the de facto Orchestrator for Containers

The screenshot shows the Amazon EKS homepage. At the top, there's a navigation bar with links for Menu, AWS logo, Contact Sales, Products, Solutions, Pricing, More, English, My Account, and Sign In to the Console. Below the navigation is a banner with the text "Amazon EKS" and "Highly available and scalable Kubernetes service". A large blue hexagonal background image is visible. A yellow button labeled "Sign up for the Preview" is prominent. The main content area contains a detailed description of Amazon EKS, mentioning its managed nature, Kubernetes integration, and support for containerized applications across multiple Availability Zones.

Amazon Elastic Container Service for Kubernetes (Amazon EKS) is a managed service that makes it easy for you to run Kubernetes on AWS without needing to install and operate your own Kubernetes clusters. Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications. Operating Kubernetes for production applications presents a number of challenges. You need to manage the scaling and availability of your Kubernetes masters and persistence layer by ensuring that you have chosen appropriate instance types, running them across multiple Availability Zones, monitoring their health, and replacing unhealthy nodes. You need to patch and upgrade your masters and worker nodes to ensure that you are running the latest version of Kubernetes. This all requires expertise and a lot of manual work. With Amazon EKS, upgrades and high availability are managed for you by AWS. Amazon EKS runs three Kubernetes masters across three Availability Zones in order to ensure high availability. Amazon EKS automatically detects and replaces unhealthy masters, and it provides automated version upgrades and patching for the masters.

The screenshot shows the IBM Cloud Kubernetes Service homepage. The top navigation bar includes links for IBM, Cloud, Why IBM, Products, Solutions, Garage, Pricing, Blog, Docs, Support, and a search bar. Below the navigation is a section for the "Kubernetes Service" with links for Details, FAQ, and Resources. The main content area features a heading "IBM Cloud Kubernetes Service" and a sub-headline "Explore now with a starter account on IBM Cloud to start creating your clusters". It includes two buttons: "Sign up" and "Get started with clusters". To the right is a video player showing a man speaking, with a play button icon. Below the video is a section titled "What is IBM Cloud Kubernetes Service?" which describes it as a managed Kubernetes offering with various features like intelligent scheduling, security, and automation. It also includes a "Get the FAQs" button and a "Let's talk" button.

IBM Cloud Kubernetes Service

Explore now with a starter account on IBM Cloud to start creating your clusters

Sign up Get started with clusters

What is IBM Cloud Kubernetes Service?

A managed Kubernetes offering to deliver powerful tools, an intuitive user experience and built-in security for rapid delivery of applications that you can bind to cloud services related to IBM Watson®, IoT, DevOps and data analytics. As a certified K8s provider, IBM Cloud Kubernetes Service provides intelligent scheduling, self-healing, horizontal scaling, service discovery and load balancing, automated rollouts and rollbacks, and secret and configuration management. The Kubernetes service also has advanced capabilities around simplified cluster management, container security and isolation policies, the ability to design your own cluster, and integrated operational tools for consistency in deployment.

Get the FAQs Let's talk

Microsoft Azure

SALES 1 800 867 1389 | MY ACCOUNT | PORTAL | Search | FREE ACCOUNT >

Why Azure Solutions Products Documentation Pricing Training Partners Blog Resources Support

Azure Container Service (AKS)

Simplify the deployment, management, and operations of Kubernetes

Use a fully managed Kubernetes container orchestration service or choose other orchestrators.

Start free > Already using Azure? Try Container Service now >

Explore Container Service: Pricing details Documentation Roadmap

Announcing the public preview of Managed Kubernetes for Azure Container Service (AKS) >

Why choose Kubernetes?

- An open-source system for automating deployment, scaling, and management of containerized applications.
- No Vendor Lock-In
- Large Community of support
- Robust platform for container orchestration
- Based on 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the community



Kubernetes is an Orchestration Platform

- **Scheduling**
decide where containers run
 - **Lifecycle and Health**
keep containers running and restart them if they fail
 - **Scaling**
grow and shrink deployments as needed
 - **Naming and Discovery**
help containers find each other
 - **Load Balancing**
distribute traffic across containers
- ...and a whole lot more



Why Do You Need Container Orchestration?

- Deploy applications to servers without worrying about specific servers
- Scale the application horizontally up and down
- Restore the application if the server on which it worked fails
 - This is called container auto-healing or rescheduling



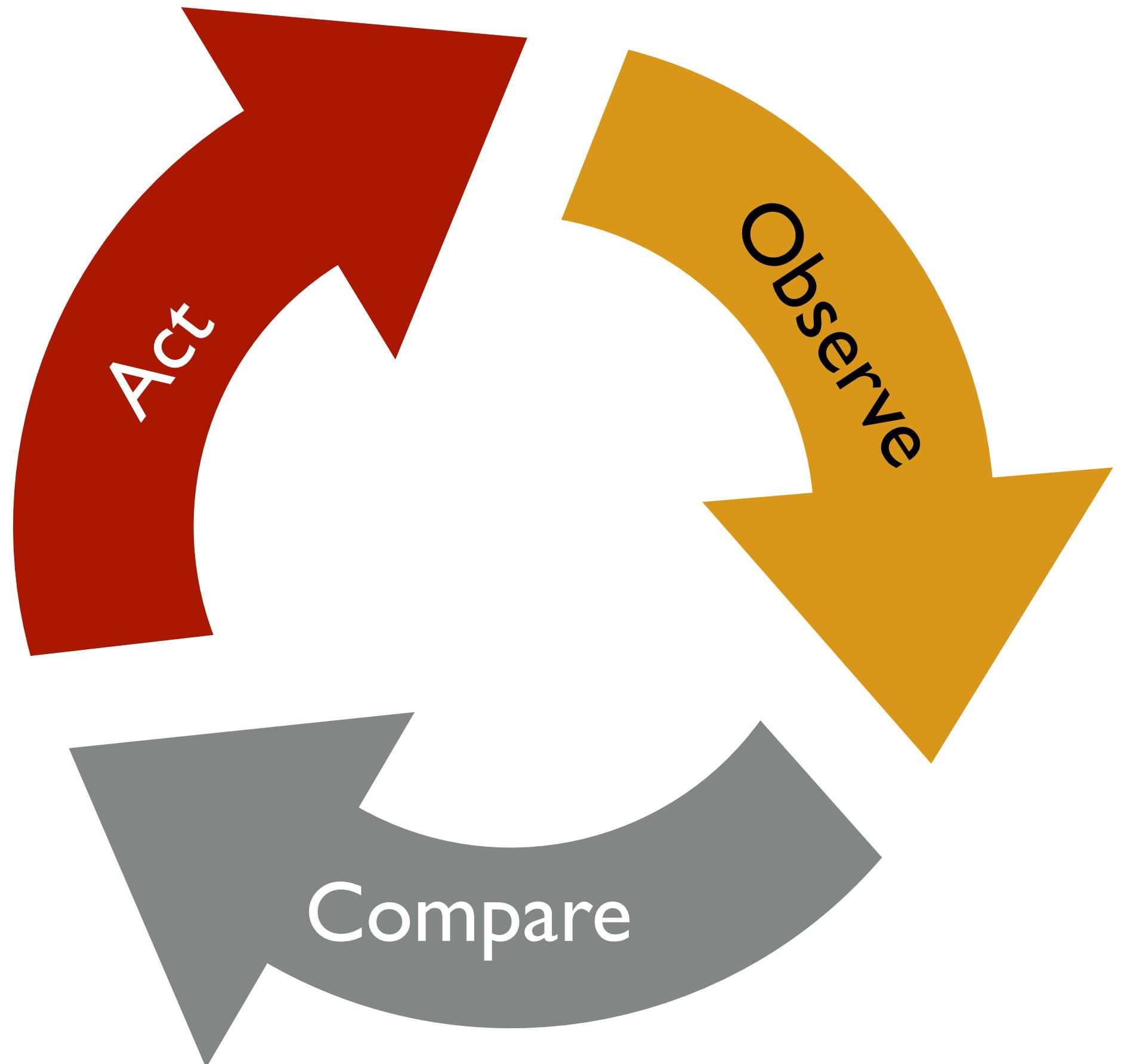
Kubernetes has a Declarative API

Kubernetes is a Declarative Model

You express the desired state

Kubernetes maintains it

What could be simpler?

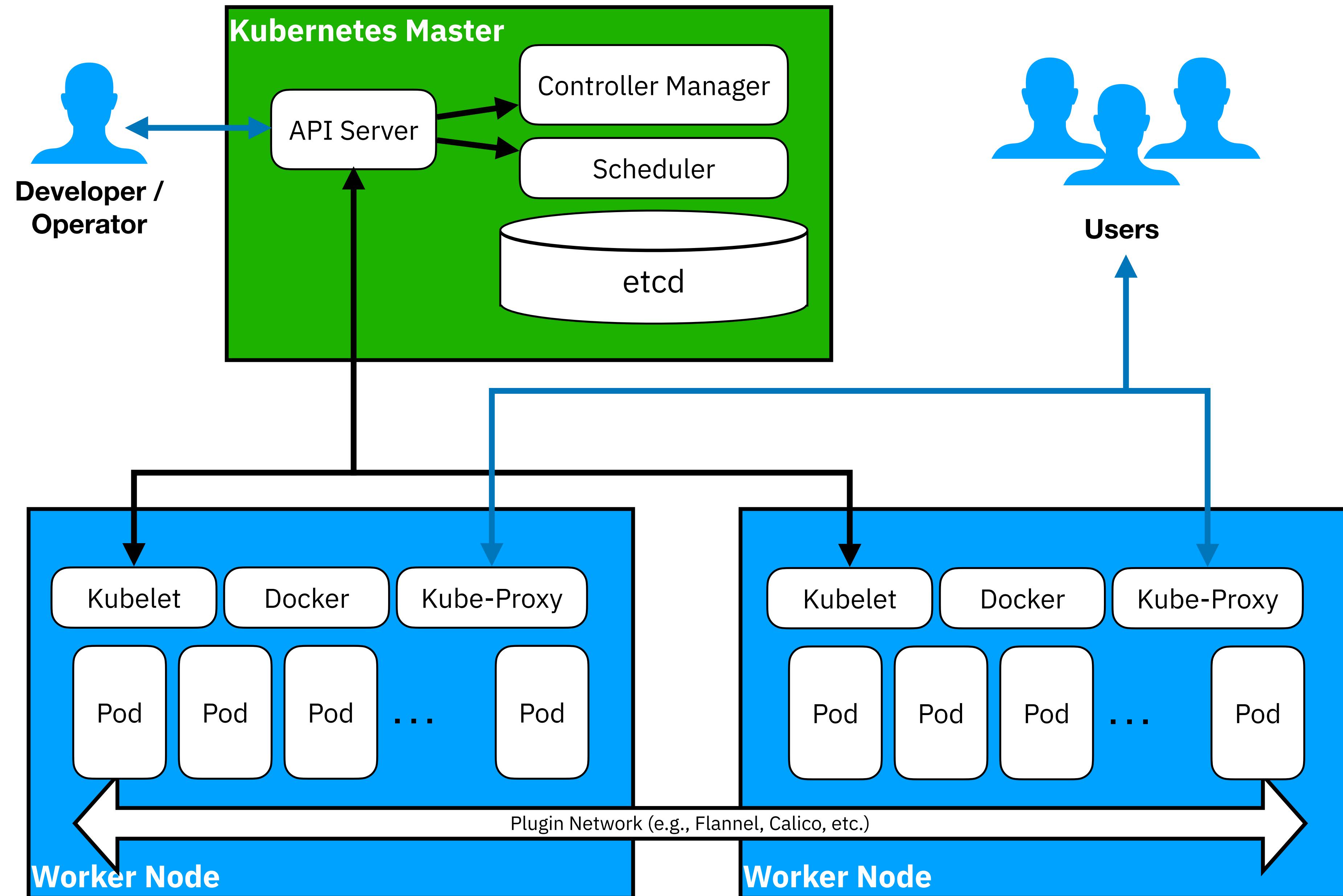


Kubernetes Architecture

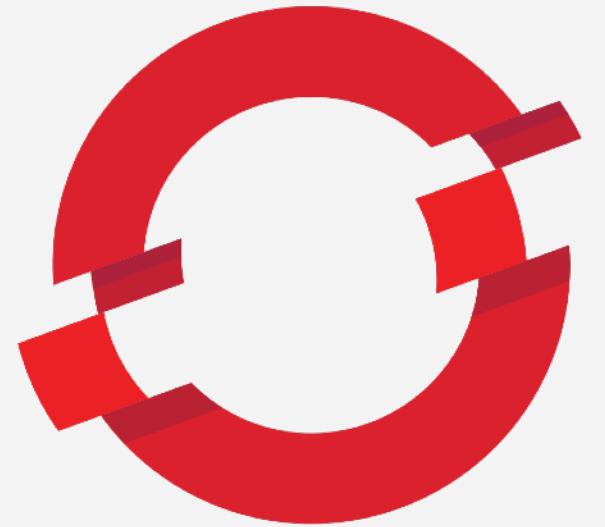


Kubernetes Architecture

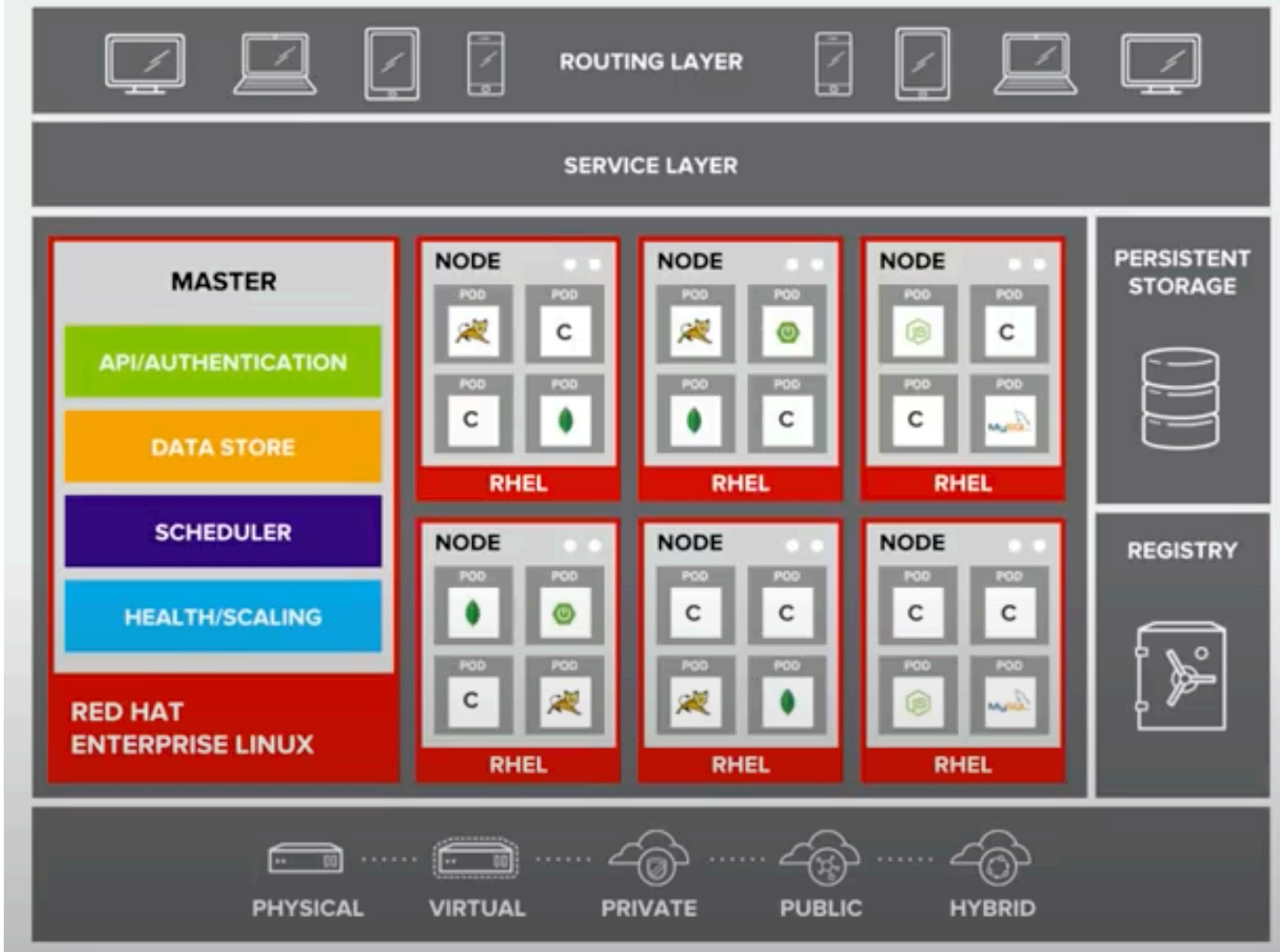
- There can be one or more Master Nodes (HA)
- There can be zero or more Worker Nodes
- Everyone communicates via the API Server
- Kubelet on each Worker Node acting as an agent
- Everything can be on one node for development use



OpenShift Architecture

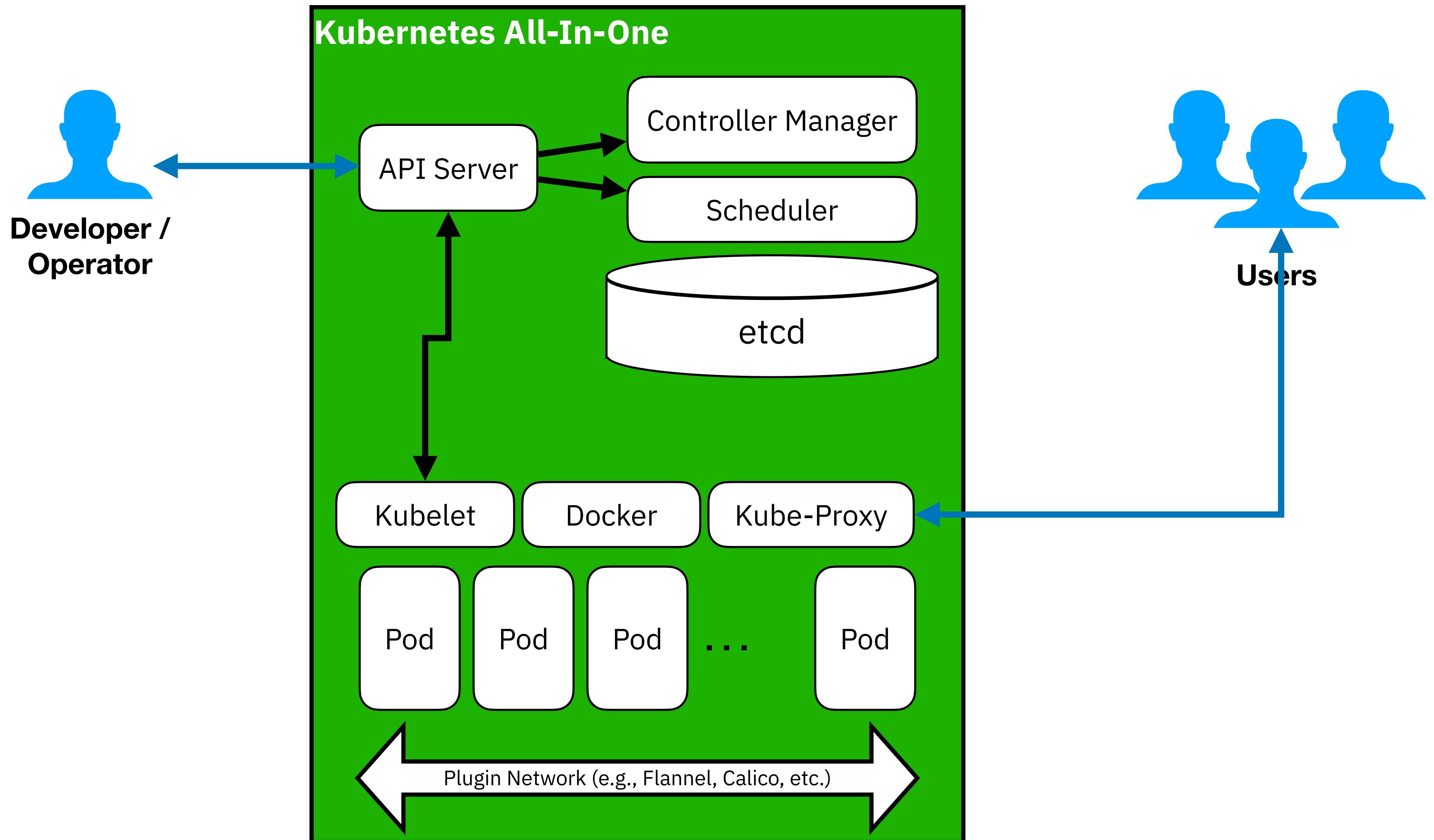


OPENSIFT



Kubernetes All-In-One

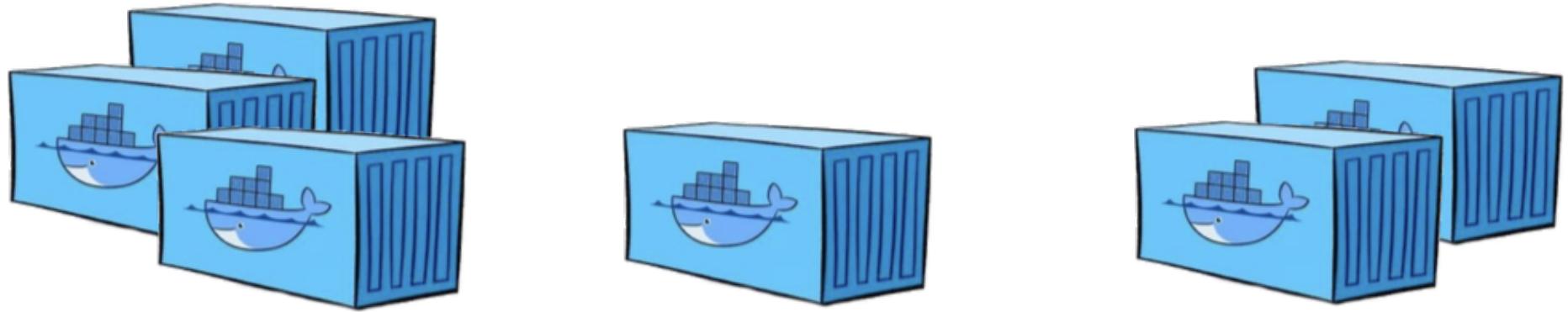
- You can run Kubernetes all in one VM for development work (not for production!)
- **MiniKube** is great for setting this up
- **Minishift** will deploy a development version of RedHat OpenShift 3.x which uses Kubernetes
- **CRC** (Code Ready Container) will deploy OpenShift 4.x



Kubernetes is the new "Cloud OS"

Managing Containers with VMs

- SysAdmins must decide where to place containers
- Workload balancing is manual



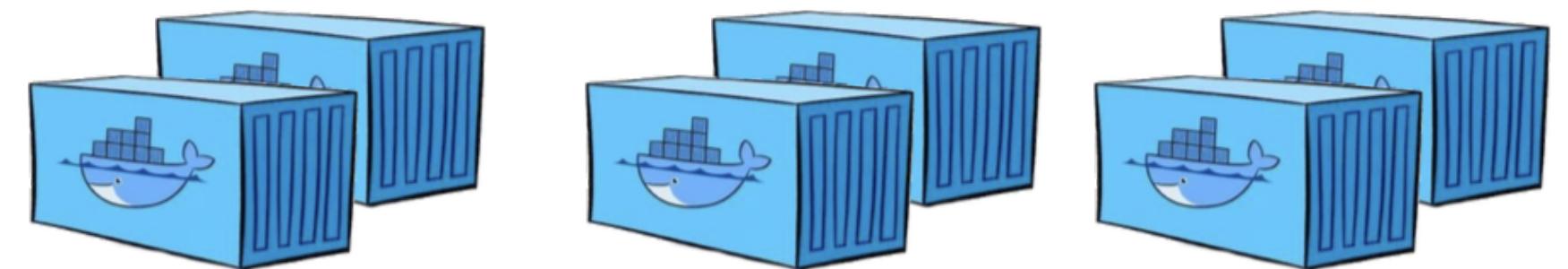
Virtual
Machine

Virtual
Machine

Virtual
Machine

Managing Containers with Kubernetes

- Kubernetes schedules and optimizes workloads automatically



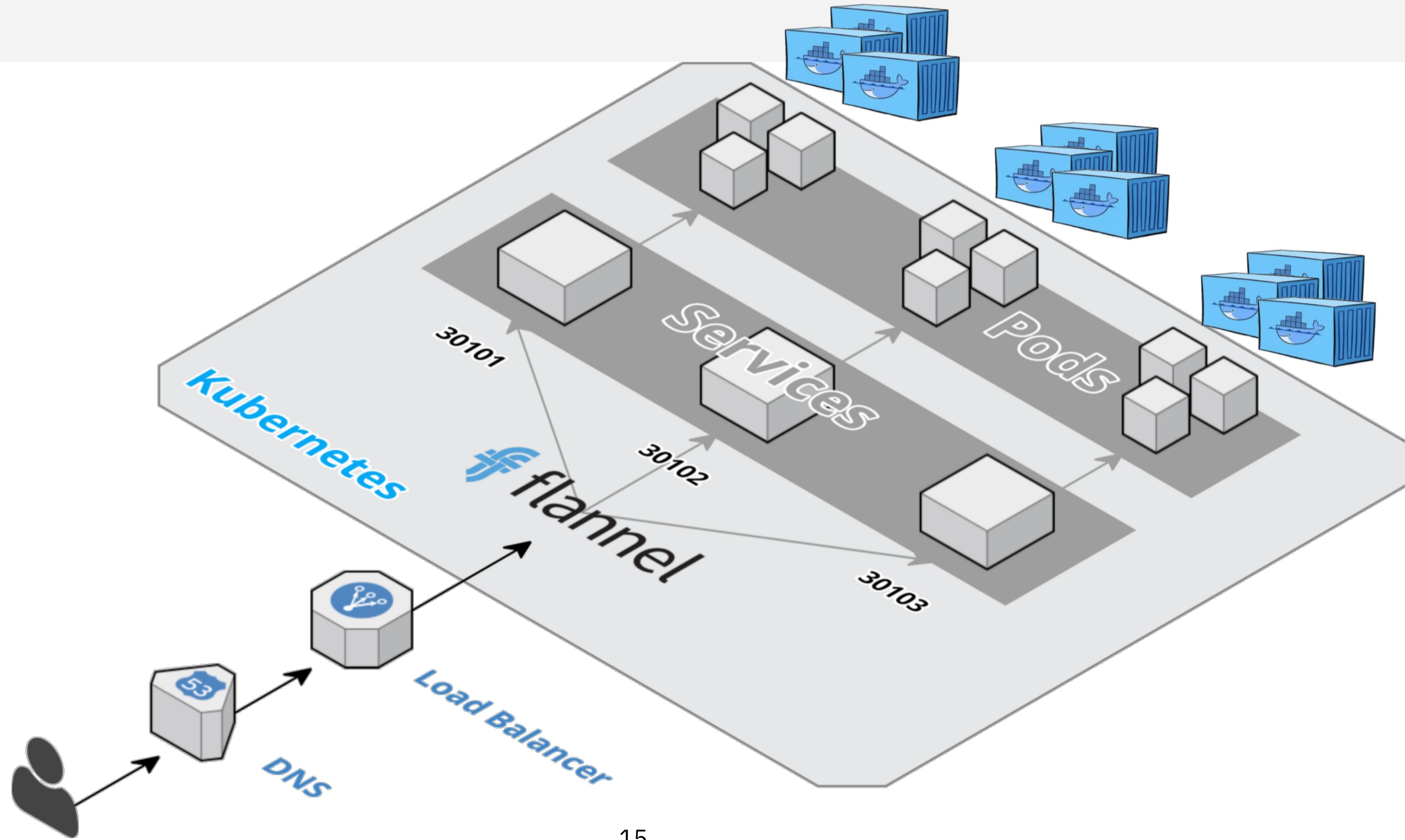
Kubernetes

Master VM

Node VM

Node VM

Kubernetes @ 20,000 ft.



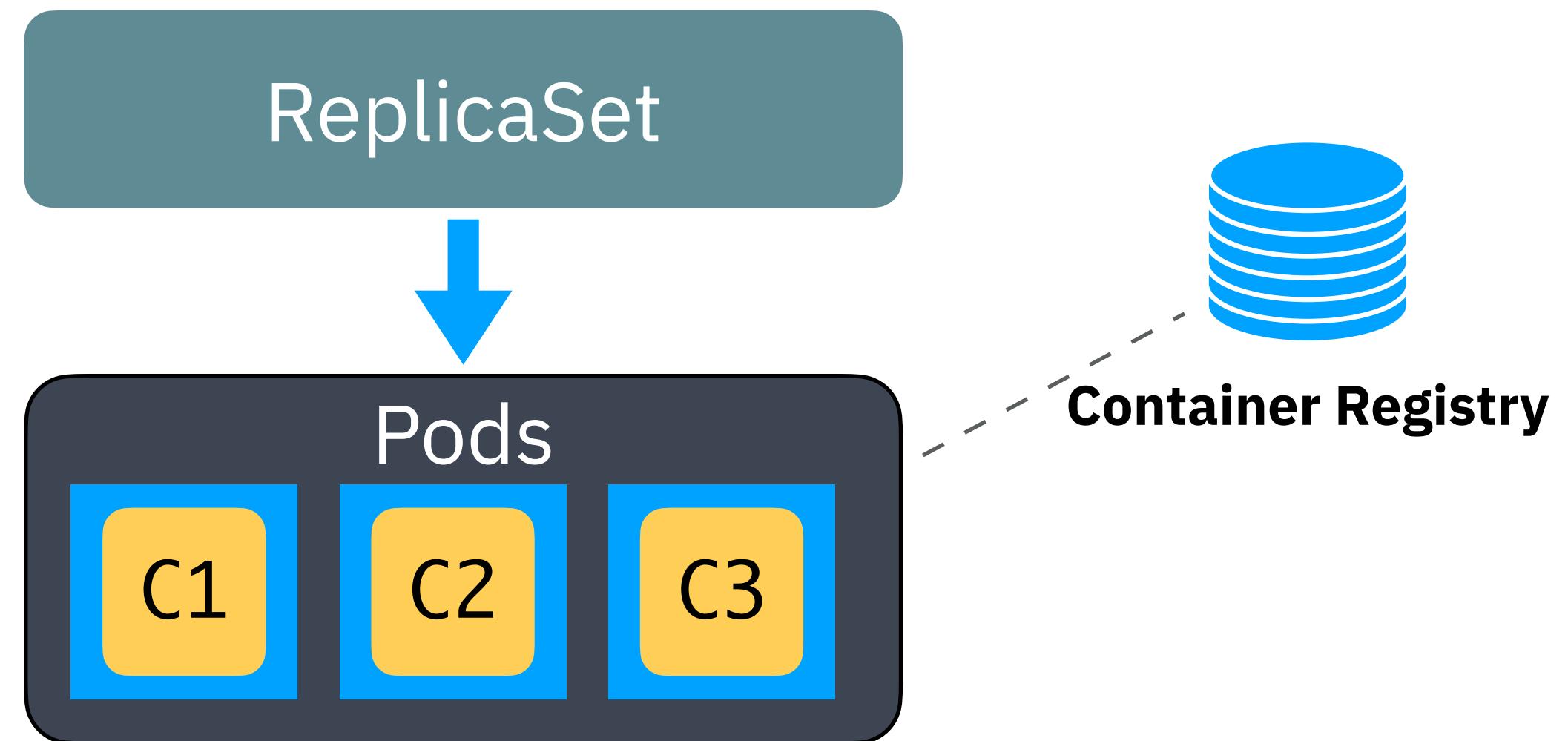
Kubernetes Pods

- Pods
 - Containers run in Pods
 - The Pod is the smallest deployment possible
 - Containers are deployed from a container registry (public or private)



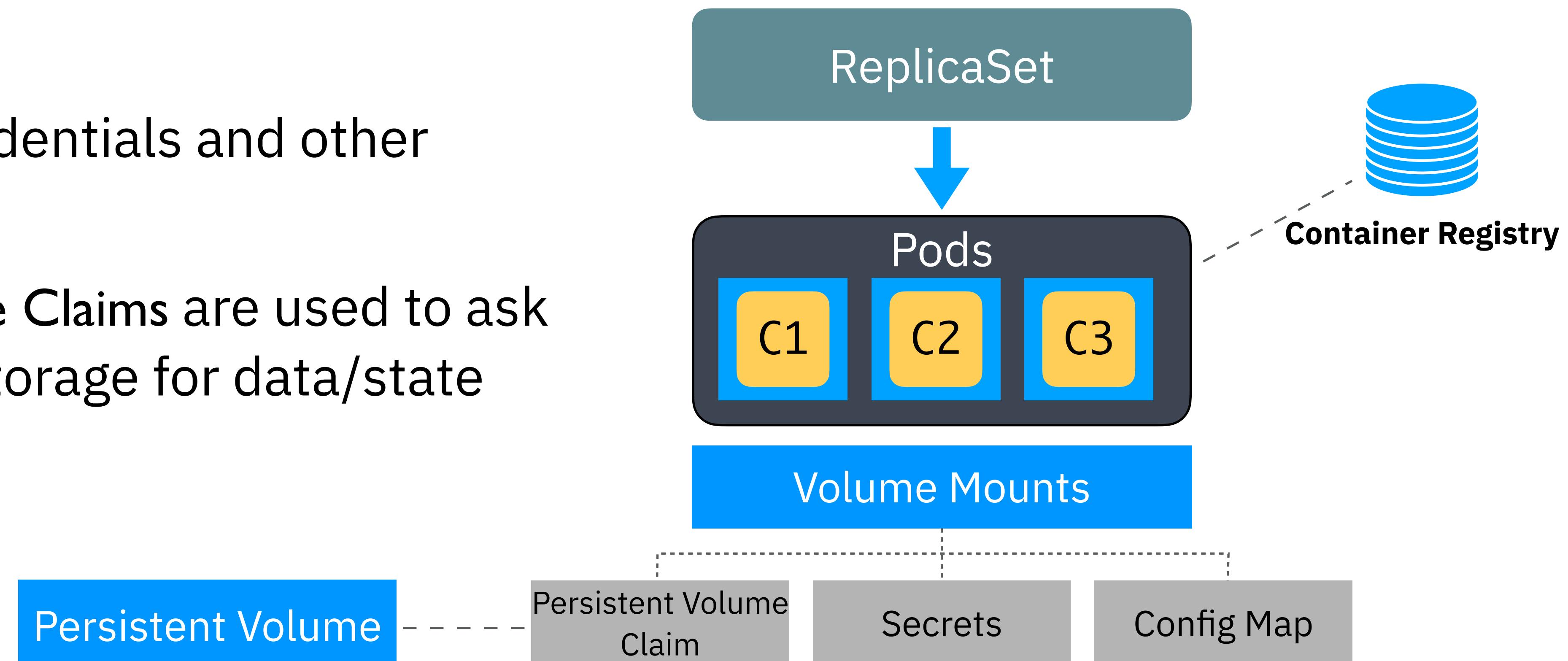
Kubernetes ReplicaSets

- ReplicaSet
 - ReplicaSets allow multiple copies of containers to be deployed
 - Each one in its own Pod
 - If a container dies, the ReplicaSet will spawn a new one



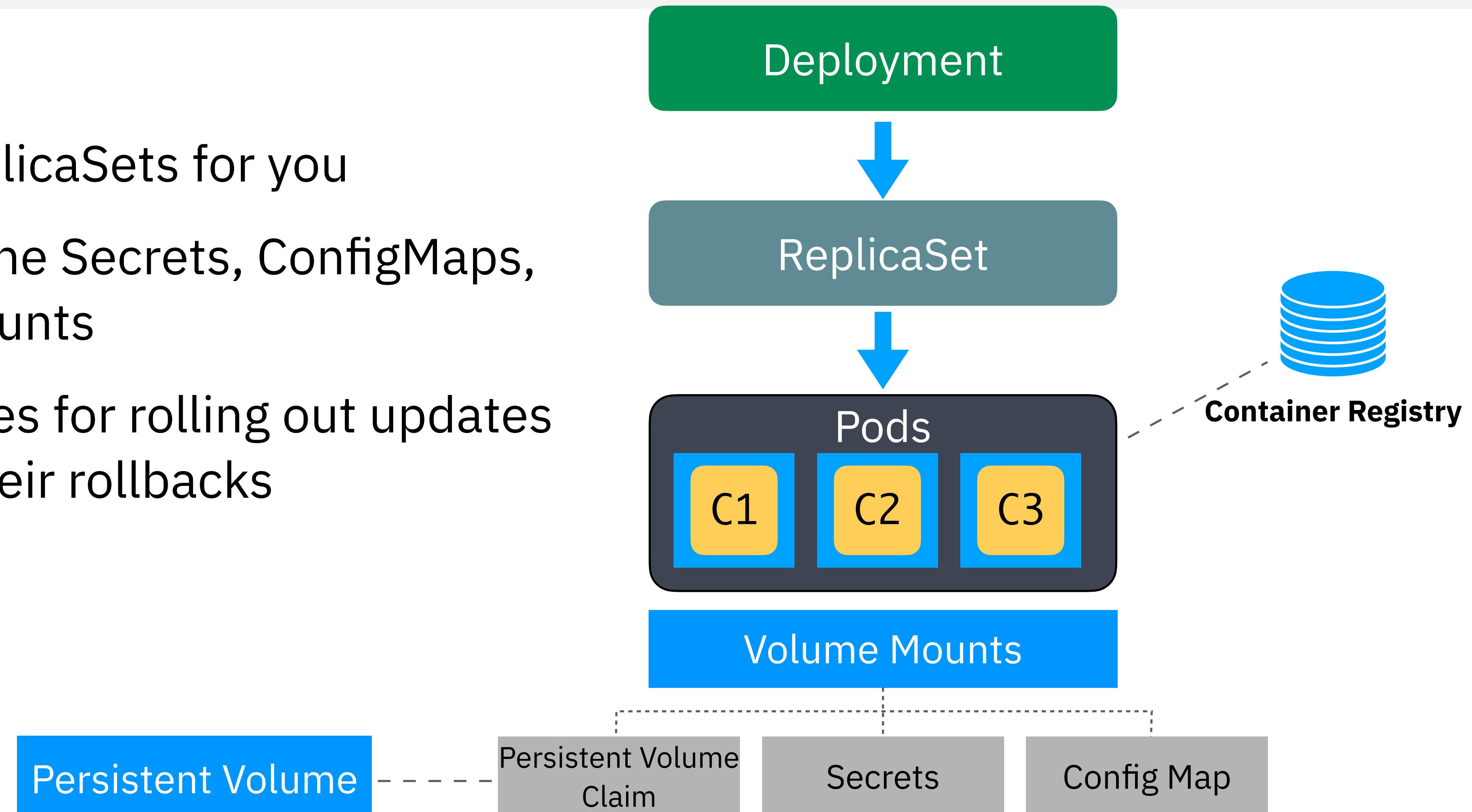
Kubernetes Volume Mounts

- Volumes
 - ConfigMaps hold configuration parameters
 - Secrets hold credentials and other secrets
 - Persistent Volume Claims are used to ask for persistent storage for data/state



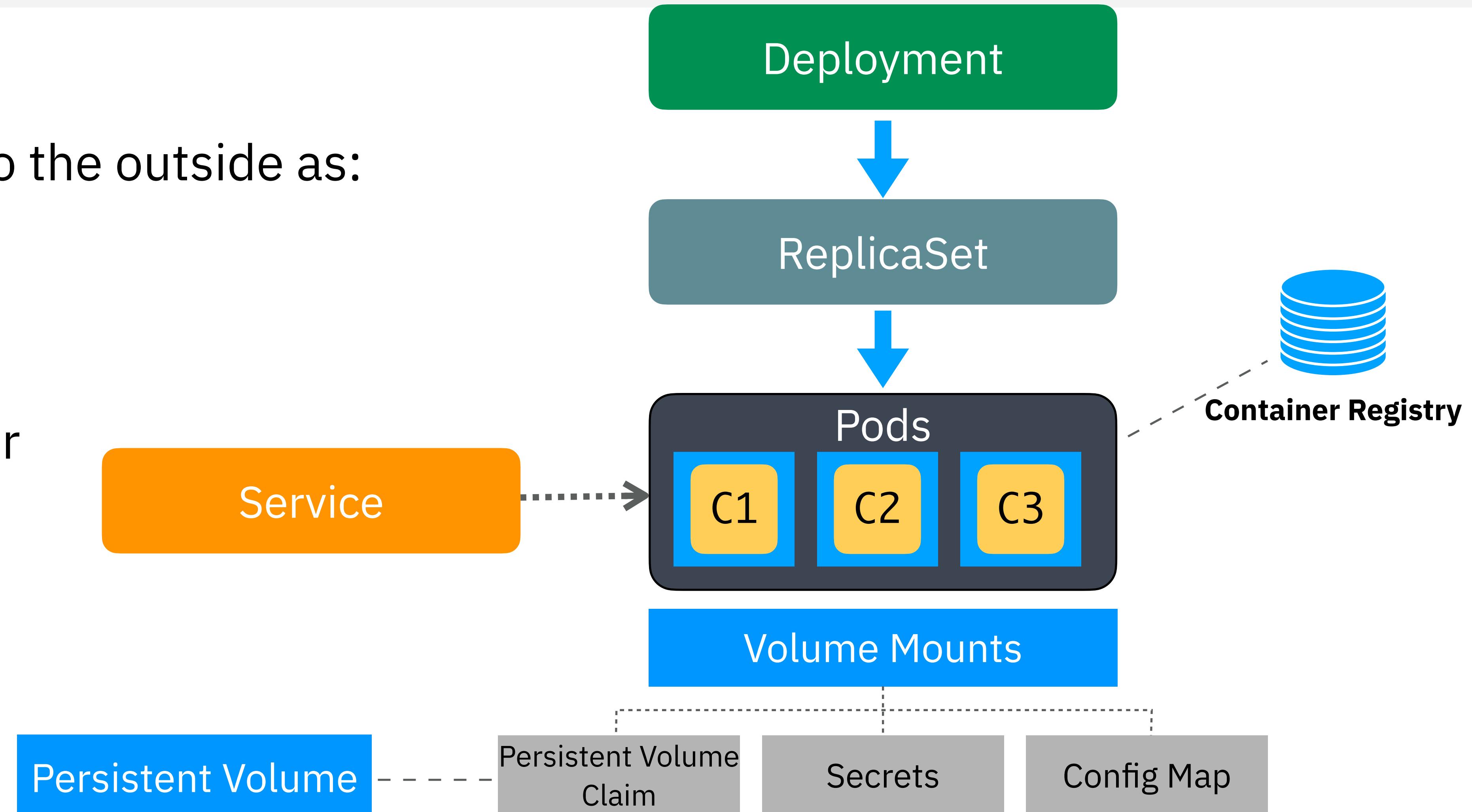
Kubernetes Deployments

- Deployment
 - Sets up the ReplicaSets for you
 - Also specified the Secrets, ConfigMaps, and Volume Mounts
 - Provides features for rolling out updates and handling their rollbacks



Kubernetes Service

- Service
 - Exposes Pods to the outside as:
 - ClusterIP
 - NodePort
 - Load Balancer

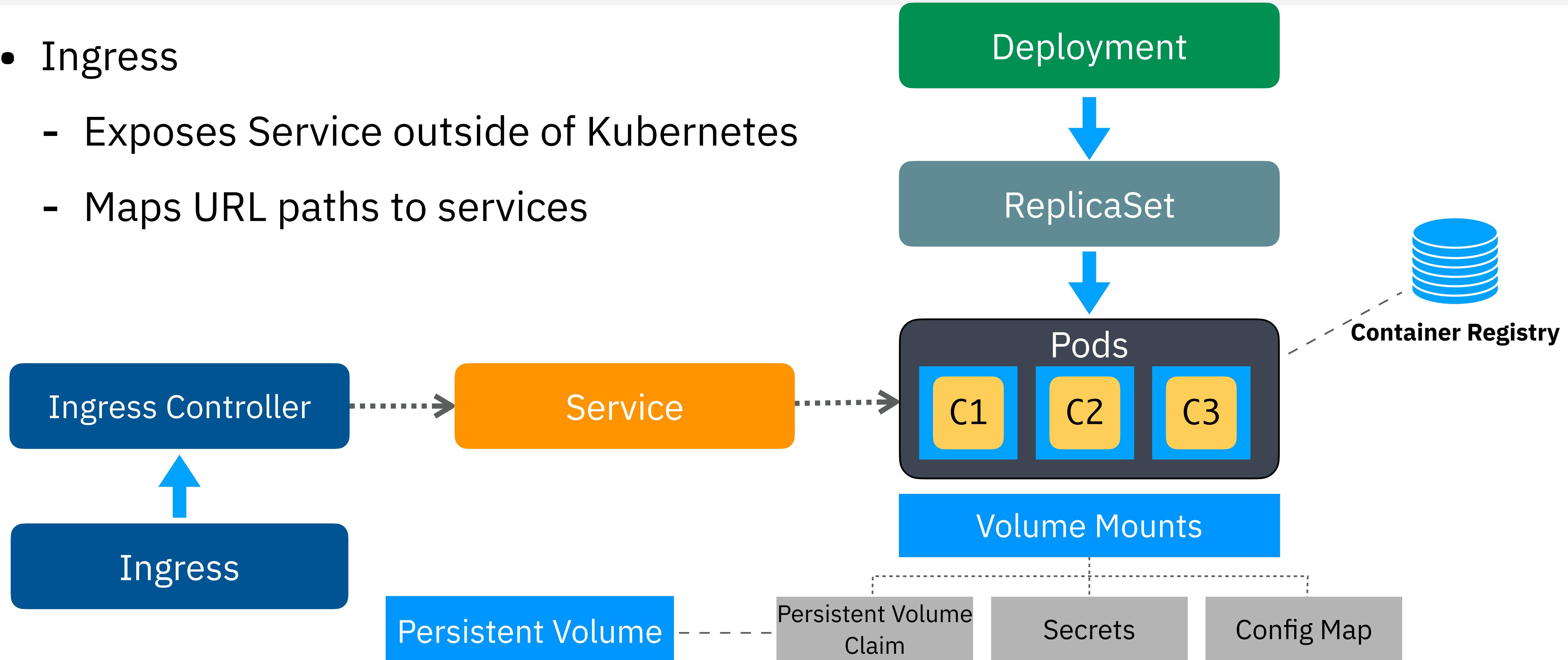


Types of Services

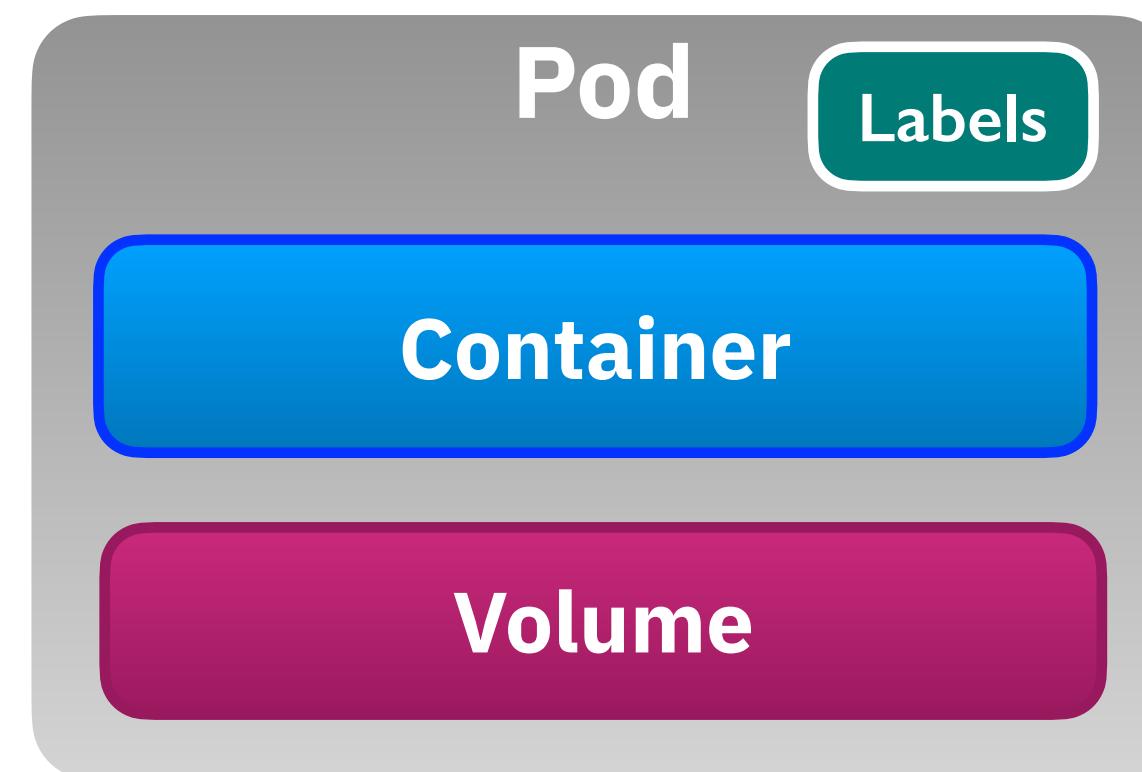
- **LoadBalancer** – Only available via cloud providers. Front end for service that balances the load across multiple backends from a single IP address
- **NodePort** – Exposes the service as an arbitrary port on every worker node in the cluster
- **ClusterIP** – Service is only accessible from other services within the cluster (no external exposure)

Kubernetes Ingress Controller

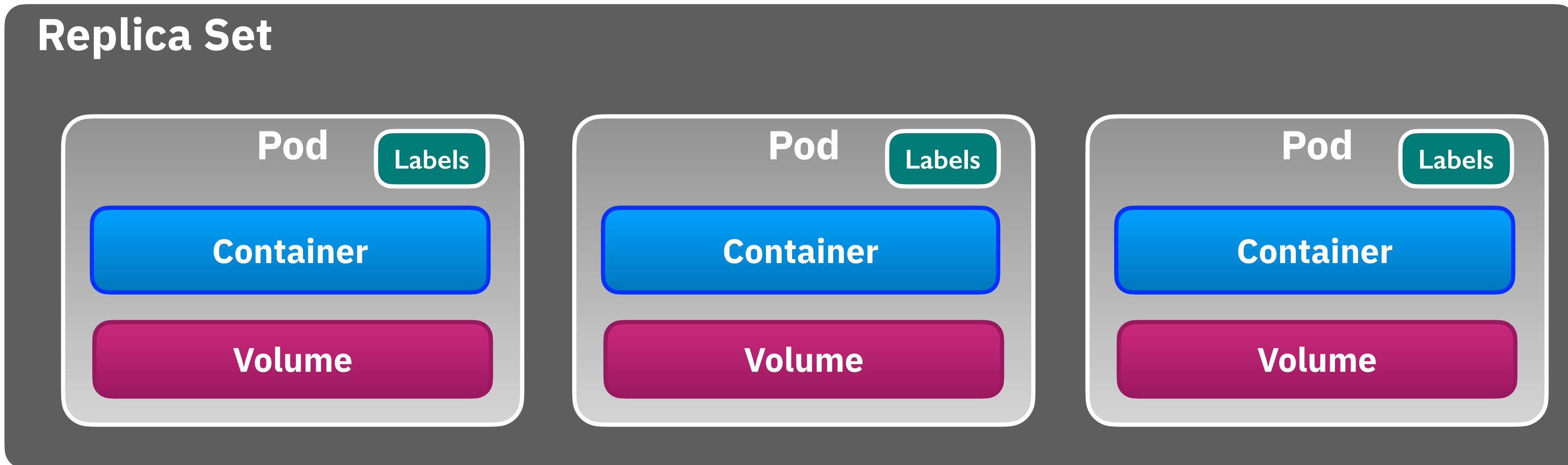
- Ingress
 - Exposes Service outside of Kubernetes
 - Maps URL paths to services



Services Map to Pods via Labels



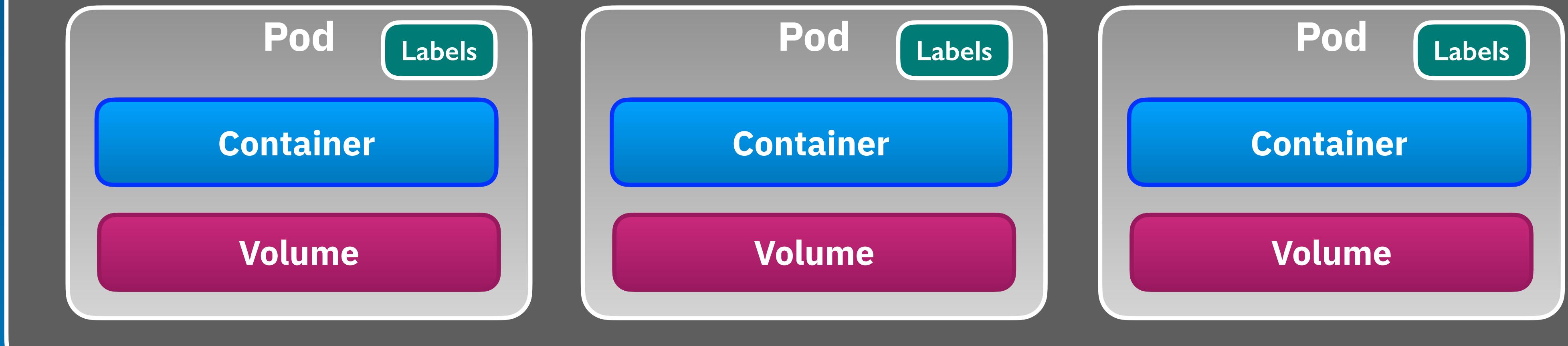
Services Map to Pods via Labels



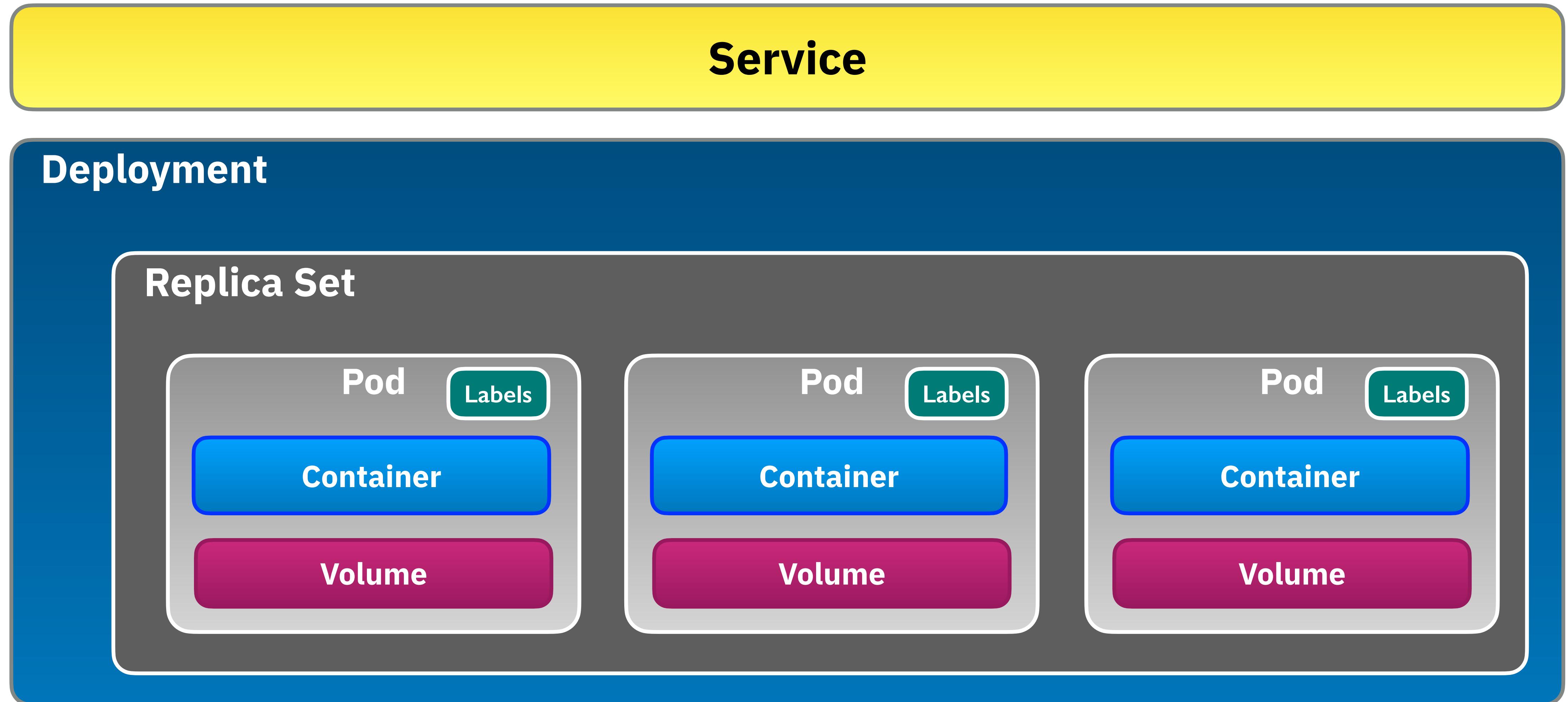
Services Map to Pods via Labels

Deployment

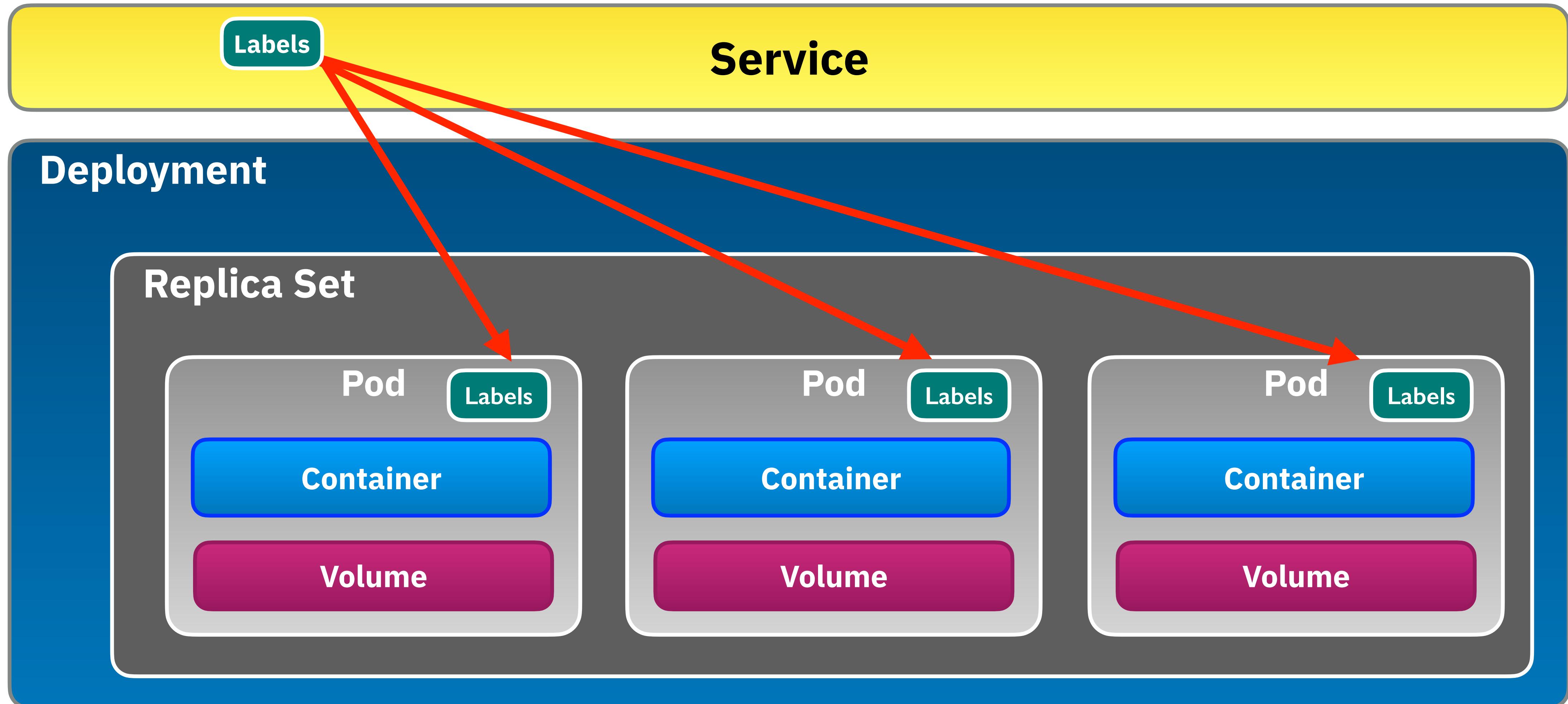
Replica Set



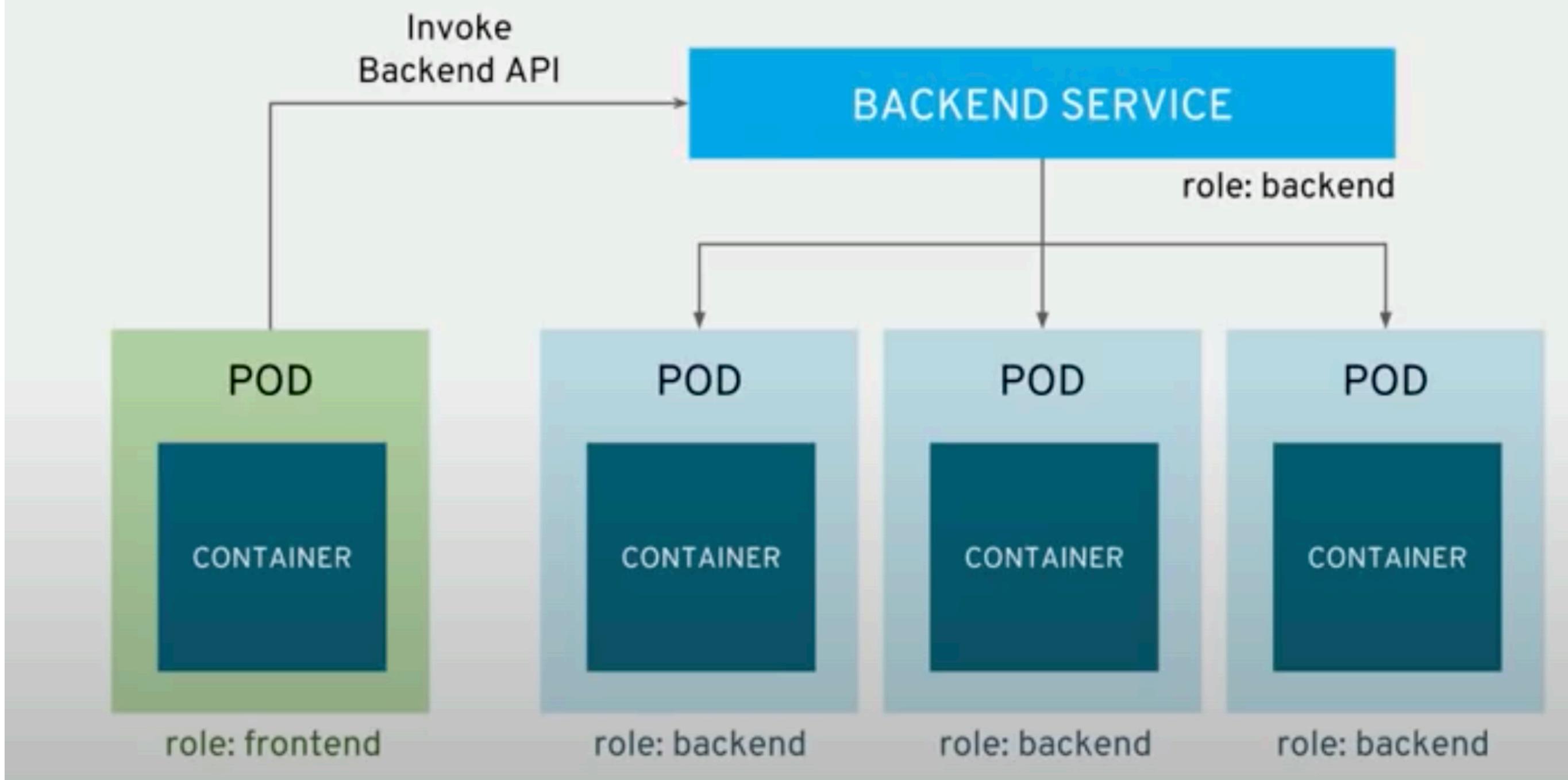
Services Map to Pods via Labels



Services Map to Pods via Labels



apps can talk to each other via services



Use Case for Multiple Containers in a Pod

- Pods can be used to host vertically integrated application stacks (e.g. LAMP), but their primary motivation is to support co-located, co-managed helper programs, such as:
 - Content management systems, file and data loaders, local cache managers, etc.
 - Log and checkpoint backup, compression, rotation, snapshotting, etc.
 - Data change watchers, log tailers, logging and monitoring adapters, event publishers, etc.
 - Proxies, bridges, and adapters
 - Controllers, managers, configurators, and updaters

Deployment using kubectl

- A containerized application can be deployed on Kubernetes using a deployment definition by executing a simple CLI command as follows:

```
$ kubectl create deployment <application-name> --image=<container-image>
```

Create a Service

- You can expose a deployment as a service using `kubectl expose`:

```
$ kubectl expose deployment <application-name> --type=<service-type> \
--port=<port_number>
```

Valid <service-type>s are:

- LoadBalancer
- NodePort
- ClusterIP

Deploy using YAML

- You can create a deployment and a service using yaml files

```
$ kubectl create -f deployment.yaml  
$ kubectl create -f service.yaml
```

- Update them

```
$ kubectl apply -f deployment.yaml  
$ kubectl apply -f service.yaml
```

- And just as easily delete them

```
$ kubectl delete -f deployment.yaml  
$ kubectl delete -f service.yaml
```

Deployment YAML

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: hitcounter
5 spec:
6   replicas: 3
7   selector:
8     matchLabels:
9       app: hitcounter
10  template:
11    metadata:
12      labels:
13        app: hitcounter
14  spec:
15    containers:
16      - image: hitcounter:1.0
17        imagePullPolicy: IfNotPresent
18        name: hitcounter
19        ports:
20          - containerPort: 8080
21            protocol: TCP
22        restartPolicy: Always
```

Service YAML

```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: hitcounter-service
5 spec:
6   type: NodePort
7   selector:
8     app: hitcounter
9   ports:
10    - name: primary
11      protocol: TCP
12      port: 8080
```

~

Linking Service to Pods

```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: hitcounter-service
5 spec:
6   type: NodePort
7   selector:
8     app: hitcounter
9   ports:
10    - name: primary
11      protocol: TCP
12      port: 8080
```

~

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: hitcounter
5 spec:
6   replicas: 3
7   selector:
8     matchLabels:
9       app: hitcounter
10  template:
11    metadata:
12      labels:
13        app: hitcounter
14  spec:
15    containers:
16      - image: hitcounter:1.0
17        imagePullPolicy: IfNotPresent
18        name: hitcounter
19        ports:
20          - containerPort: 8080
21            protocol: TCP
22        restartPolicy: Always
```

Linking Service to Pods

```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: hitcounter-service
5 spec:
6   type: NodePort
7   selector:
8     app: hitcounter
9   ports:
10    - name: primary
11      protocol: TCP
12      port: 8080
~
```

Look for Pods with this Label

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: hitcounter
5 spec:
6   replicas: 3
7   selector:
8     matchLabels:
9       app: hitcounter
10  template:
11    metadata:
12      labels:
13        app: hitcounter
14  spec:
15    containers:
16      - image: hitcounter:1.0
17        imagePullPolicy: IfNotPresent
18        name: hitcounter
19        ports:
20          - containerPort: 8080
21            protocol: TCP
22        restartPolicy: Always
```

Linking Service to Pods

Will match this Pod

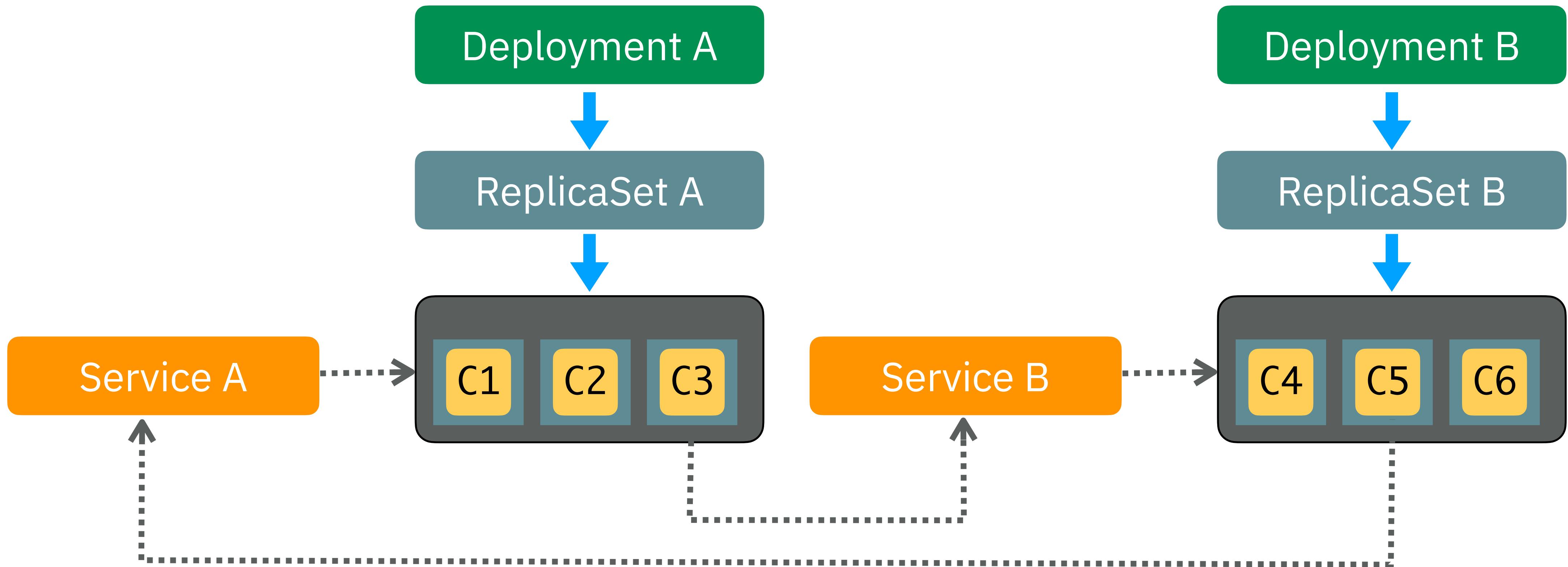
```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: hitcounter-service
5 spec:
6   type: NodePort
7   selector:
8     app: hitcounter
9   ports:
10    - name: primary
11      protocol: TCP
12      port: 8080
~
```

Look for Pods with this Label

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: hitcounter
5 spec:
6   replicas: 3
7   selector:
8     matchLabels:
9       app: hitcounter
10  template:
11    metadata:
12      labels:
13        app: hitcounter
14  spec:
15    containers:
16      - image: hitcounter:1.0
17        imagePullPolicy: IfNotPresent
18        name: hitcounter
19        ports:
20          - containerPort: 8080
21            protocol: TCP
22        restartPolicy: Always
```

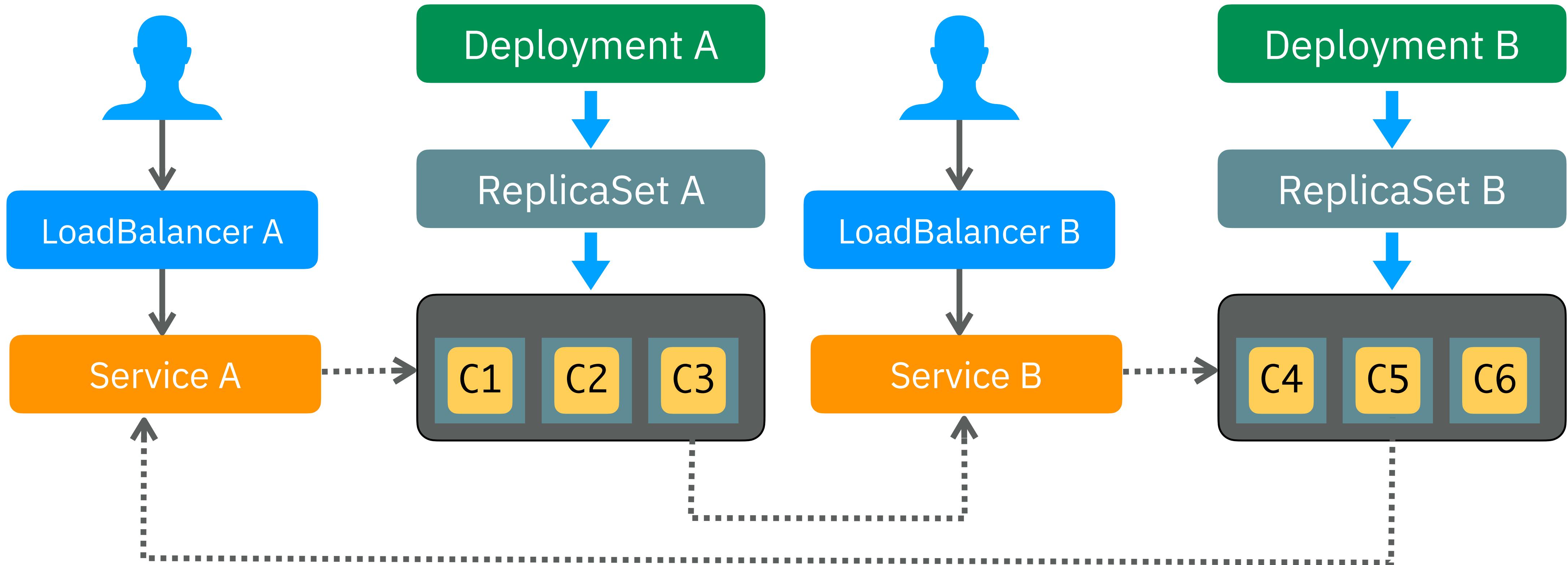
Service Discovery

- Kubernetes provides internal routing so services can find each other



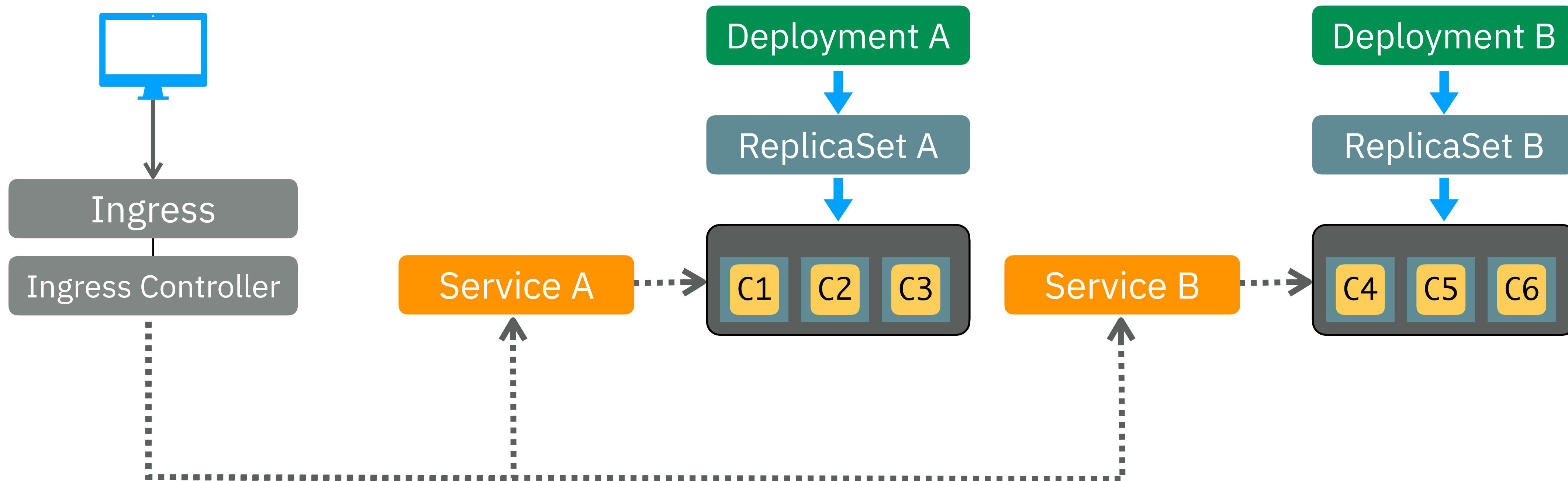
External Service Access

- LoadBalancers provide external access for a single service



External Routing

- Kubernetes provides an Ingress Controller to allow external network access or you can use NodePorts



Ingress Example

```
1 apiVersion: extensions/v1beta1
2 kind: Ingress
3 metadata:
4   name: ecommerce
5 spec:
6   # tls:
7   # - secretName: tls
8   rules:
9     - host: ecommerce.containers.mybluemix.net
10    http:
11      paths:
12        - path: /shopcarts
13          backend:
14            serviceName: shopcart-service
15            servicePort: 5000
16        - path: /catalog
17          backend:
18            serviceName: catalog-service
19            servicePort: 5000
20        - path: /orders
21          backend:
22            serviceName: order-service
23            servicePort: 5000
24        - path: /recommendations
25          backend:
26            serviceName: recommendation-service
27            servicePort: 5000
```

Ingress Example

```
1 apiVersion: extensions/v1beta1
2 kind: Ingress
3 metadata:
4   name: ecommerce
5 spec:
6   # tls:
7   # - secretName: tls
8   rules:
9     - host: ecommerce.containers.mybluemix.net
10    http:
11      paths:
12        - path: /shopcarts
13          backend:
14            serviceName: shopcart-service
15            servicePort: 5000
16        - path: /catalog
17          backend:
18            serviceName: catalog-service
19            servicePort: 5000
20        - path: /orders
21          backend:
22            serviceName: order-service
23            servicePort: 5000
24        - path: /recommendations
25          backend:
26            serviceName: recommendation-service
27            servicePort: 5000
```

Requests coming in on this URL

Ingress Example

```
1 apiVersion: extensions/v1beta1
2 kind: Ingress
3 metadata:
4   name: ecommerce
5 spec:
6   # tls:
7   # - secretName: tls
8   rules:
9     - host: ecommerce.containers.mybluemix.net
10    http:
11      paths:
12        - path: /shopcarts
13          backend:
14            serviceName: shopcart-service
15            servicePort: 5000
16        - path: /catalog
17          backend:
18            serviceName: catalog-service
19            servicePort: 5000
20        - path: /orders
21          backend:
22            serviceName: order-service
23            servicePort: 5000
24        - path: /recommendations
25          backend:
26            serviceName: recommendation-service
27            servicePort: 5000
```

Based on these PATHs

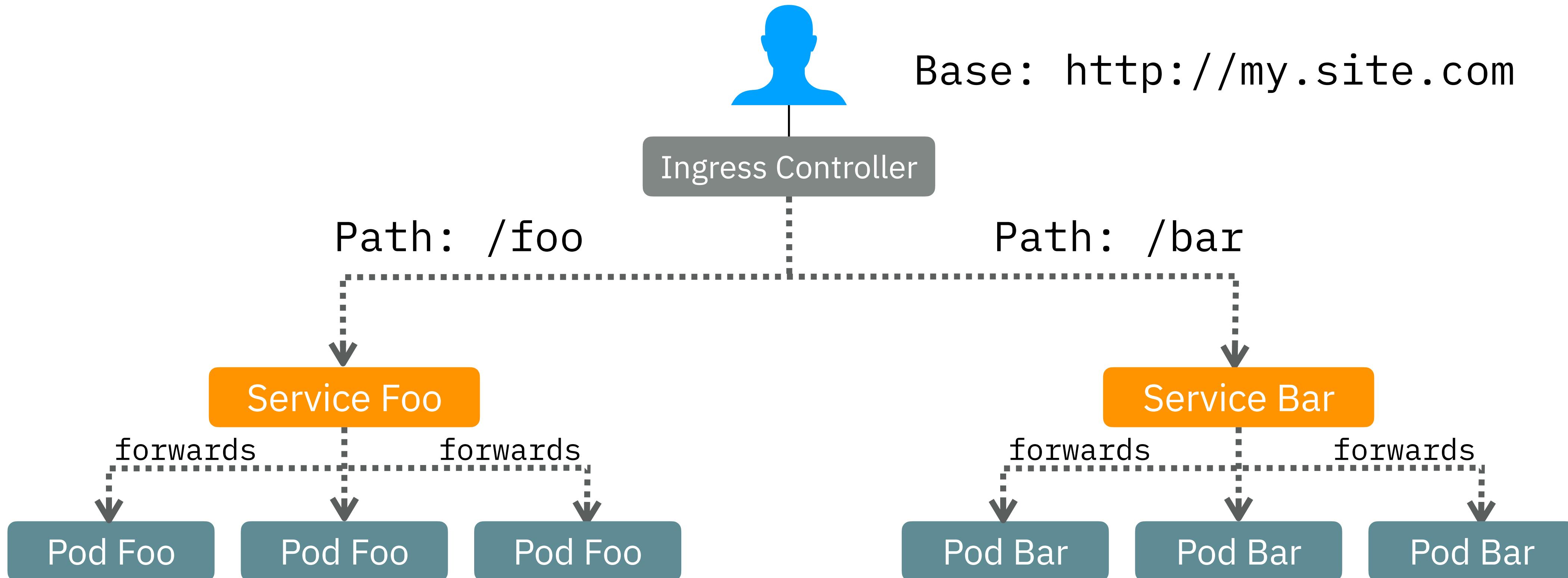
Ingress Example

```
1 apiVersion: extensions/v1beta1
2 kind: Ingress
3 metadata:
4   name: ecommerce
5 spec:
6   # tls:
7   # - secretName: tls
8   rules:
9     - host: ecommerce.containers.mybluemix.net
10    http:
11      paths:
12        - path: /shopcarts
13          backend:
14            serviceName: shopcart-service
15            servicePort: 5000
16        - path: /catalog
17          backend:
18            serviceName: catalog-service
19            servicePort: 5000
20        - path: /orders
21          backend:
22            serviceName: order-service
23            servicePort: 5000
24        - path: /recommendations
25          backend:
26            serviceName: recommendation-service
27            servicePort: 5000
```

Will be routed to these microservices

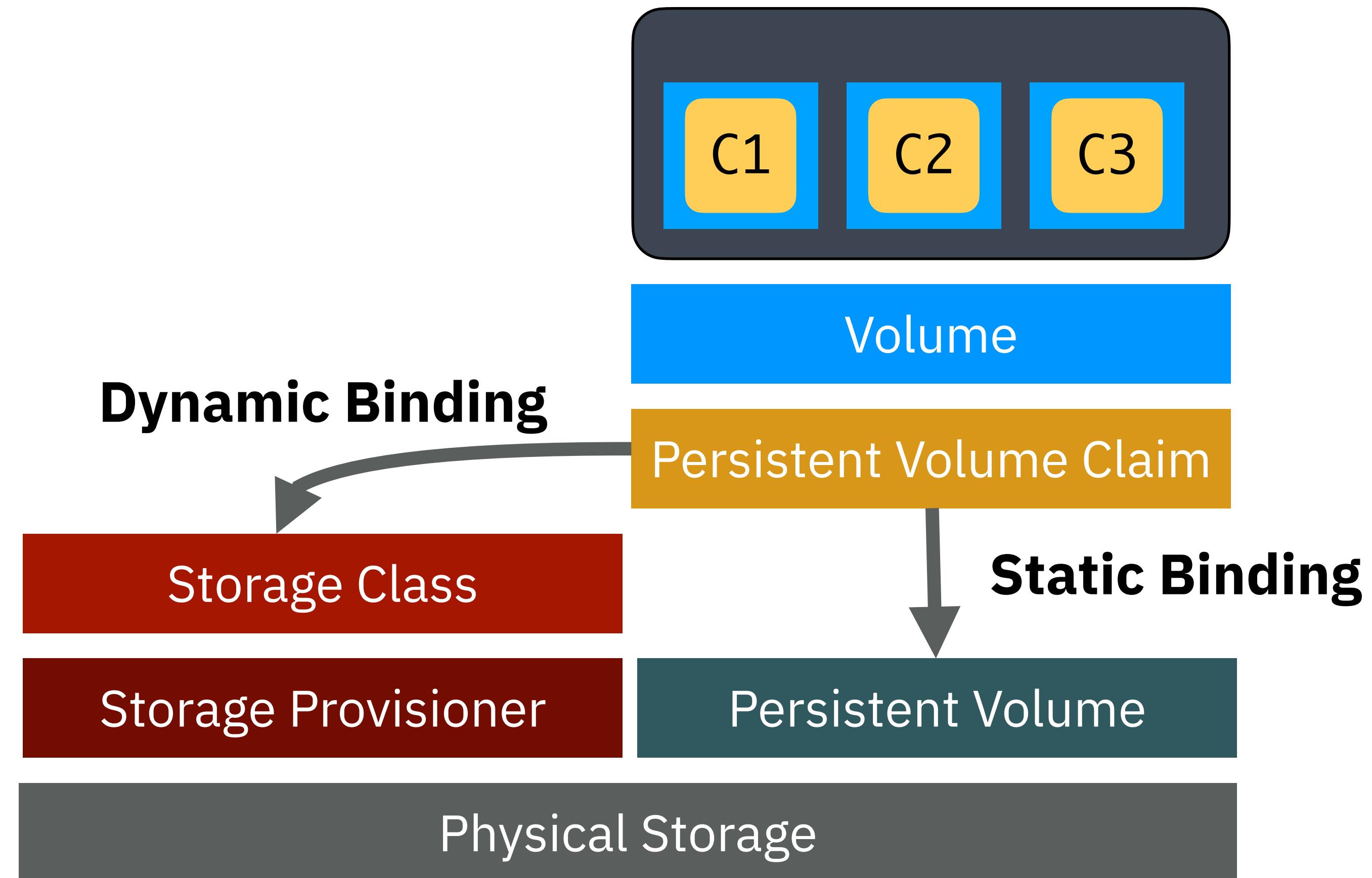
Ingress Controller

- Single entry point into multiple kubernetes services



Persistent Volumes

- Kubernetes loosely couples physical storage devices with containers by introducing an intermediate resource called persistent volume claims (PVCs).
- A PVC defines the disk size, disk type (ReadWriteOnce, ReadOnlyMany, ReadWriteMany) and dynamically links a storage device to a volume defined against a pod
- The binding process can either be done in a static way using PVs or dynamically be using a persistent storage provider



Example Volume Mount

- There is a volume named `redis-storage` and that is connected to the `redis` container via the `VolumeMount: /data/redis`

```
apiVersion: v1
kind: Pod
metadata:
  name: redis
spec:
  containers:
    - name: redis
      image: redis
      volumeMounts:
        - name: redis-storage
          mountPath: /data/redis
  volumes:
    - name: redis-storage
      emptyDir: {}
```

~

Example Volume Mount

- There is a volume named `redis-storage` and that is connected to the `redis` container via the `VolumeMount: /data/redis`

```
apiVersion: v1
kind: Pod
metadata:
  name: redis
spec:
  containers:
    - name: redis
      image: redis
      volumeMounts:
        - name: redis-storage
          mountPath: /data/redis
  volumes:
    - name: redis-storage
      emptyDir: {}
```

Volume named `redis-storage`

Example Volume Mount

- There is a volume named `redis-storage` and that is connected to the `redis` container via the `VolumeMount: /data/redis`

```
apiVersion: v1
kind: Pod
metadata:
  name: redis
spec:
  containers:
    - name: redis
      image: redis
      volumeMounts:
        - name: redis-storage
          mountPath: /data/redis
  volumes:
    - name: redis-storage
      emptyDir: {}
```

Mounted at: /data/redis

Volume named redis-storage

Configurations Management

(ConfigMap)

- Containers generally use environment variables for parameterizing their runtime configurations
- Kubernetes provides a way of managing more complex configuration files using a simple resource called ConfigMaps
- ConfigMaps can be created using directories, files or literal values using following CLI command:

```
$ kubectl create configmap <map-name> <data-source>  
  
# map-name: name of the config map  
# data-source: directory, file or literal value
```

Credentials Management

(Secrets)

- Similar to ConfigMaps, Kubernetes provides another valuable resource called Secrets for managing sensitive information such as passwords, OAuth tokens, and ssh keys.
- A secret can be created for managing basic auth credentials using the following way:

```
# write credentials to two files
$ echo -n 'admin' > ./username.txt
$ echo -n '1f2d1e2e67df' > ./password.txt

# create a secret
$ kubectl create secret generic app-credentials --from-file=./username.txt --from-file=./password.txt
```

Credentials From Environment Variables

(Secrets)

```
# export DATABASE_URI='postgres://admin:s3cr3t@postgres:5432/postgres'  
  
# kubectl create secret generic db-creds --from-literal=database-  
uri=$DATABASE_URI
```

- Creates the following

```
apiVersion: v1  
kind: Secret  
data:  
  database-uri: cG9zdGdyZXN6Ly9hZG1pbjpzM2NyM3RAcG9zdGdyZXN6NTQzMj9wb3N0Z3Jlcw==
```

Create Secrets from Literals

```
kubectl create secret generic dev-db-secret \
--from-literal=username=devuser \
--from-literal=password='s3cr3t'
```

```
apiVersion: v1
kind: Secret
metadata:
  name: dev-db-secret
type: Opaque
data:
  username: ZGV2dXNlcgo=
  password: czNjcjN0Cg==
```

Secret Yaml Example

```
1 apiVersion: v1
2 kind: Secret
3 metadata:
4   name: ecommerce-apikey
5   namespace: default
6 data:
7   secret: <place base64 encoded secret here>
```

```
$ echo -n "this is my secret" | base64
dGhpcyBpcyBteSBzZWNyZXQ=
```

Secret Yaml Example

```
1 apiVersion: v1
2 kind: Secret
3 metadata:
4   name: ecommerce-apikey
5   namespace: default
6 data:
7 secret: <place base64 encoded secret here>
```

Place the base64 encode string in secret

```
$ echo -n "this is my secret" | base64
dGhpcyBpcyBteSBzZWNyZXQ=
```

Using Secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: pet-creds
  namespace: default
data:
  binding: dGhpcyBpcyBteSBzZWNyZXQ=
```

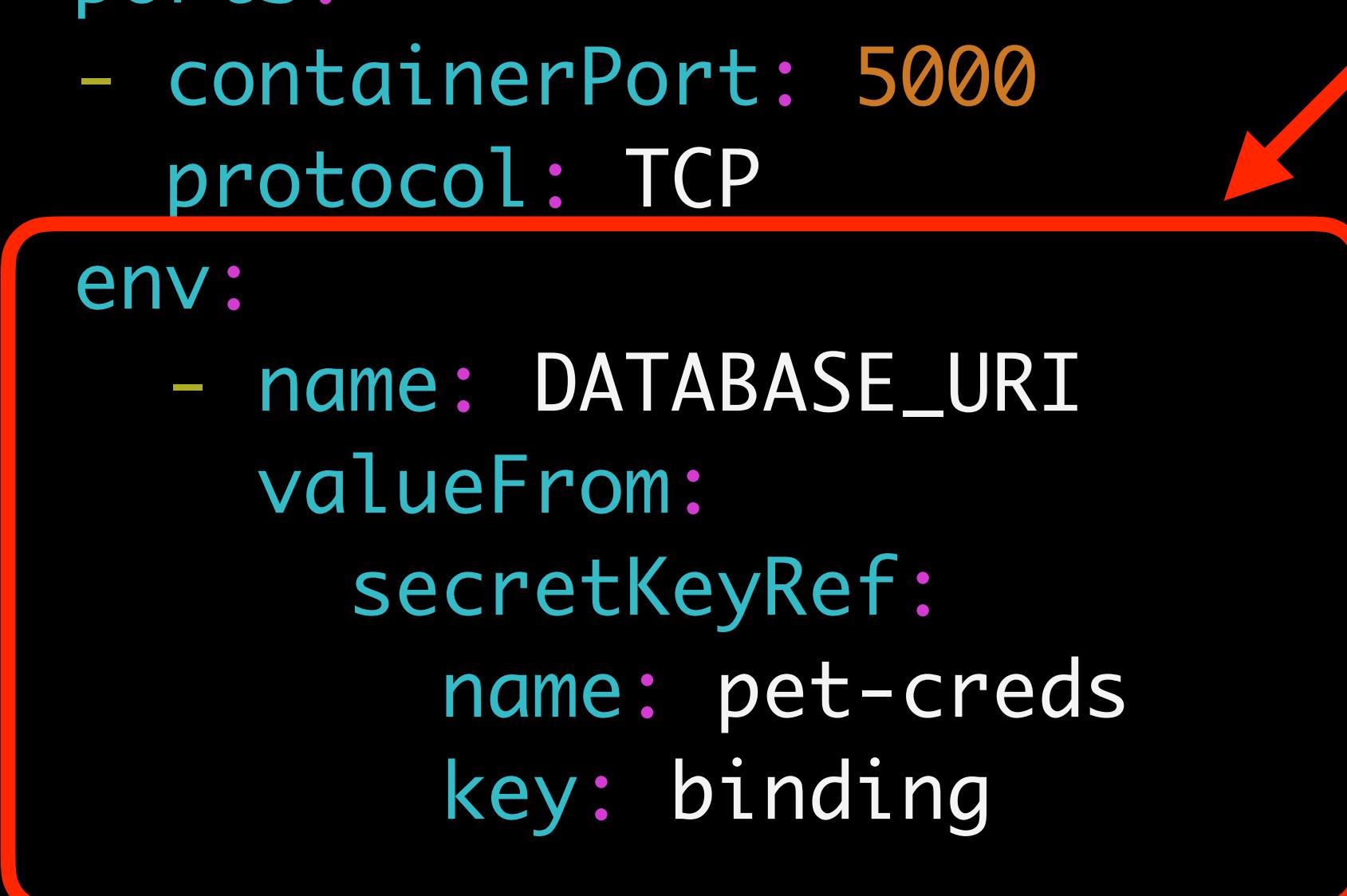
```
apiVersion: apps/v1
kind: Deployment
...
spec:
  containers:
    - name: pet-demo
      image: pet-demo:v1
      imagePullPolicy: IfNotPresent
      ports:
        - containerPort: 5000
          protocol: TCP
      env:
        - name: DATABASE_URI
          valueFrom:
            secretKeyRef:
              name: pet-creds
              key: binding
```

Using Secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: pet-creds
  namespace: default
data:
  binding: dGhpcyBpcyBteSBzZWNyZXQ=
```

```
apiVersion: apps/v1
kind: Deployment
...
spec:
  containers:
    - name: pet-demo
      image: pet-demo:v1
      imagePullPolicy: IfNotPresent
      ports:
        - containerPort: 5000
          protocol: TCP
      env:
        - name: DATABASE_URI
          valueFrom:
            secretKeyRef:
              name: pet-creds
              key: binding
```

Secrets exposed as environment variables



Kubernetes Rolling Updates (Zero Downtime Deployments)

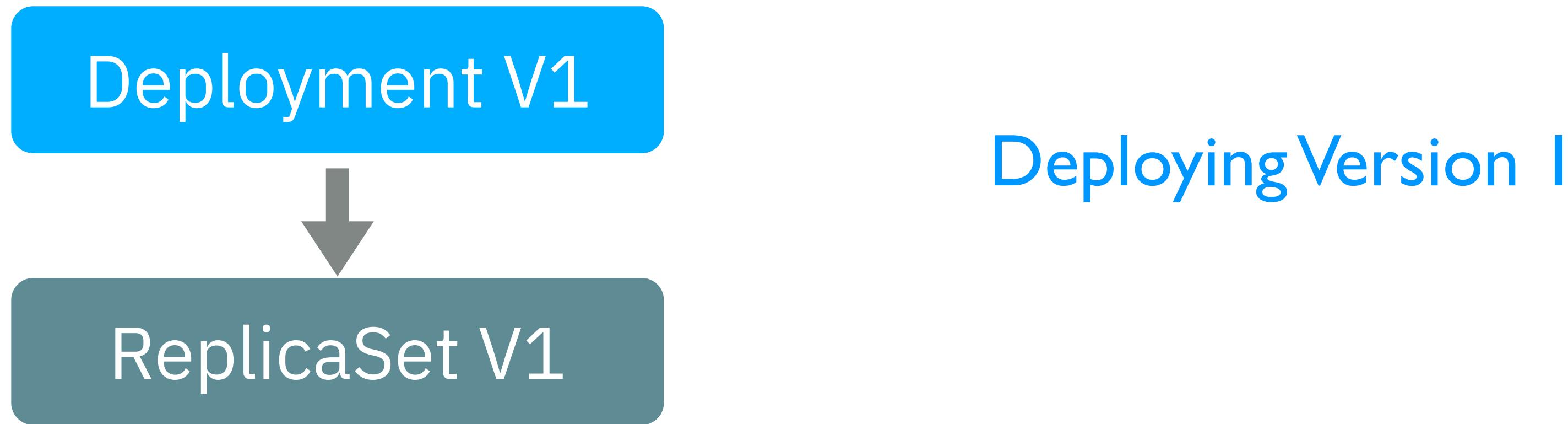
```
$ kubectl set image deployment/<application-name> <container-name>=<container-image-name>:<new-version>
```

Deployment V1

Deploying Version I

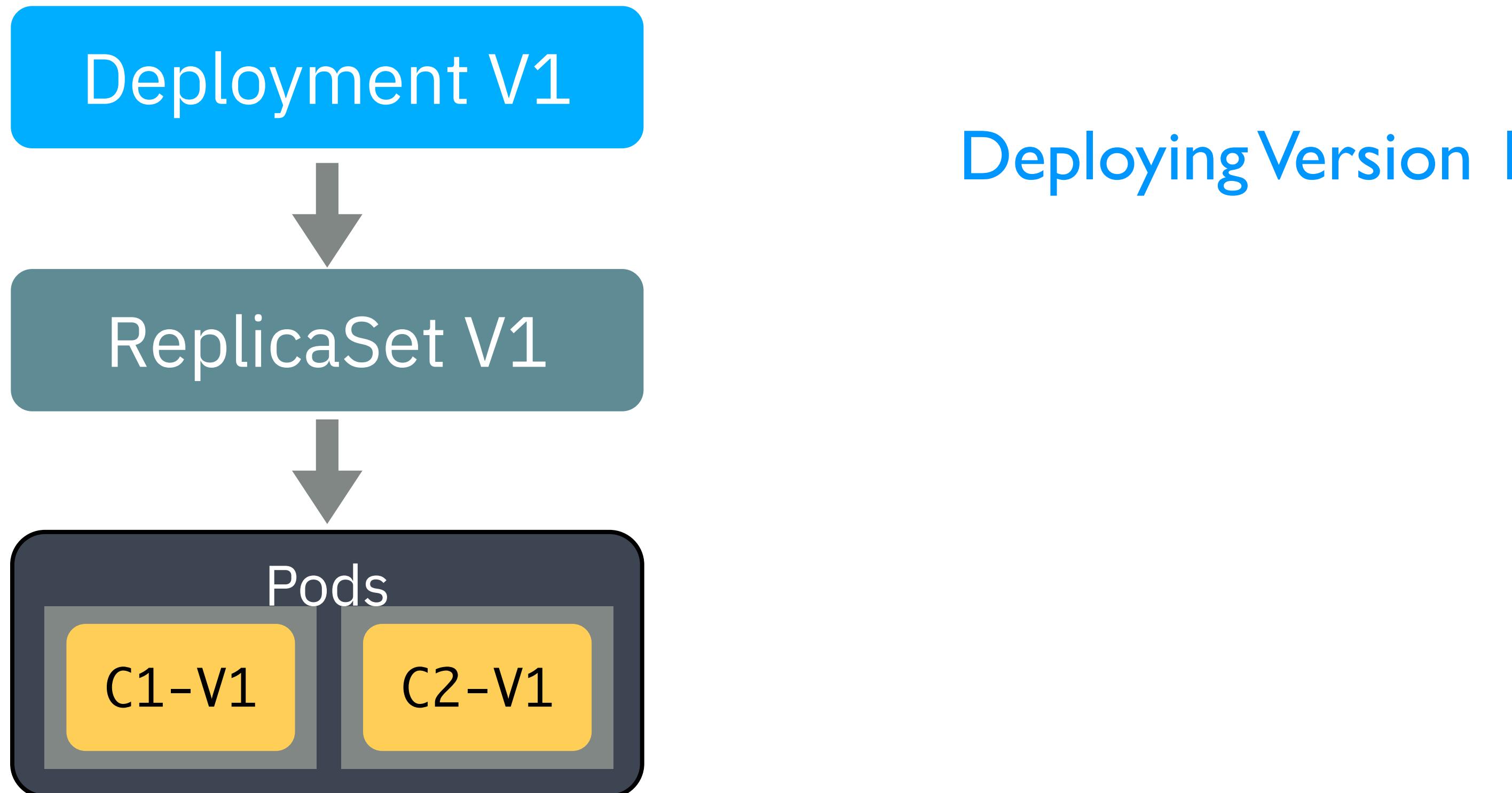
Kubernetes Rolling Updates (Zero Downtime Deployments)

```
$ kubectl set image deployment/<application-name> <container-name>=<container-image-name>:<new-version>
```



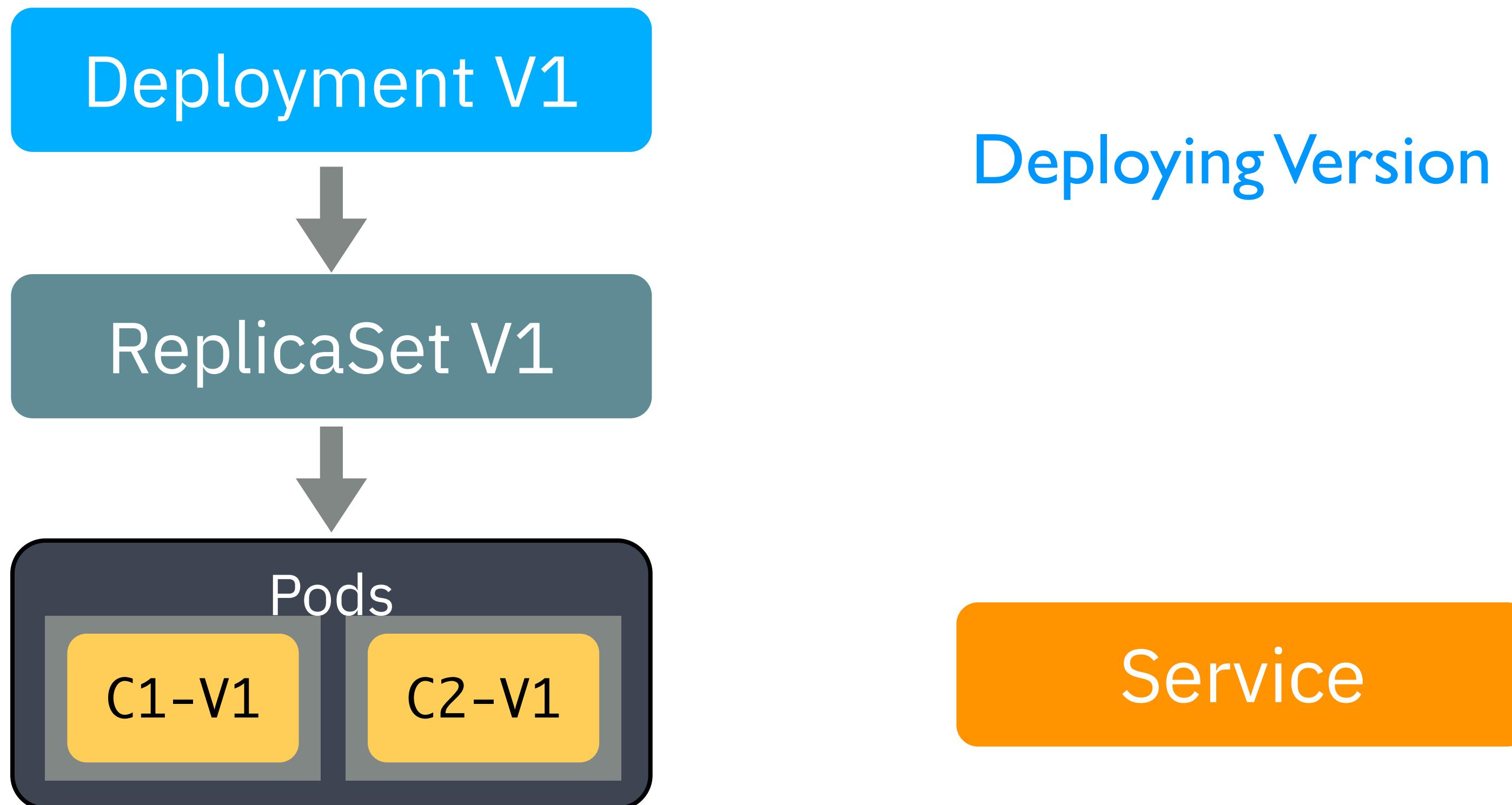
Kubernetes Rolling Updates (Zero Downtime Deployments)

```
$ kubectl set image deployment/<application-name> <container-name>=<container-image-name>:<new-version>
```



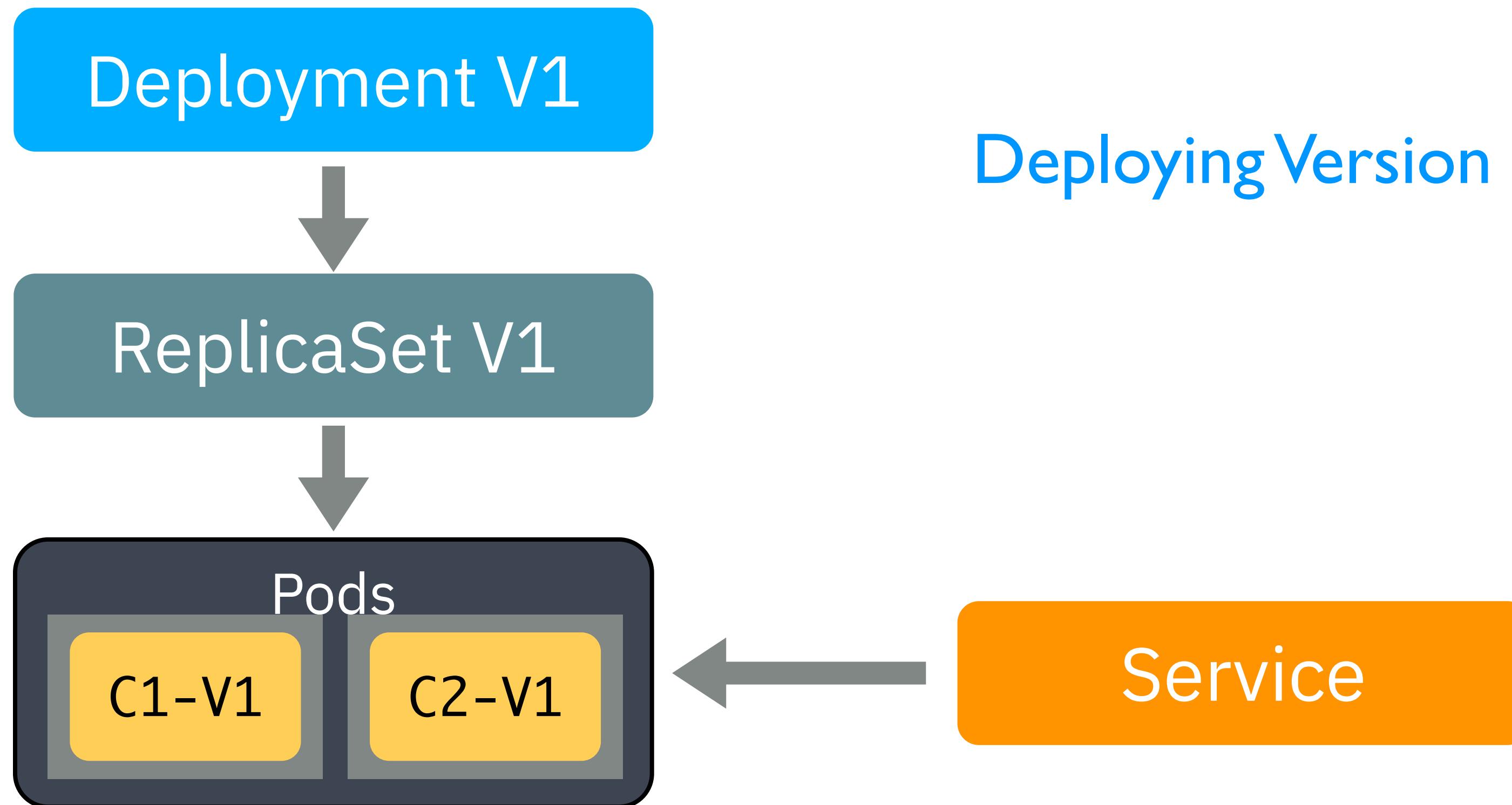
Kubernetes Rolling Updates (Zero Downtime Deployments)

```
$ kubectl set image deployment/<application-name> <container-name>=<container-image-name>:<new-version>
```



Kubernetes Rolling Updates (Zero Downtime Deployments)

```
$ kubectl set image deployment/<application-name> <container-name>=<container-image-name>:<new-version>
```



Kubernetes Rolling Updates (Zero Downtime Deployments)

```
$ kubectl set image deployment/<application-name> <container-name>=<container-image-name>:<new-version>
```



Kubernetes Rolling Updates (Zero Downtime Deployments)

```
$ kubectl set image deployment/<application-name> <container-name>=<container-image-name>:<new-version>
```



Kubernetes Rolling Updates (Zero Downtime Deployments)

```
$ kubectl set image deployment/<application-name> <container-name>=<container-image-name>:<new-version>
```



Kubernetes Rolling Updates (Zero Downtime Deployments)

```
$ kubectl set image deployment/<application-name> <container-name>=<container-image-name>:<new-version>
```



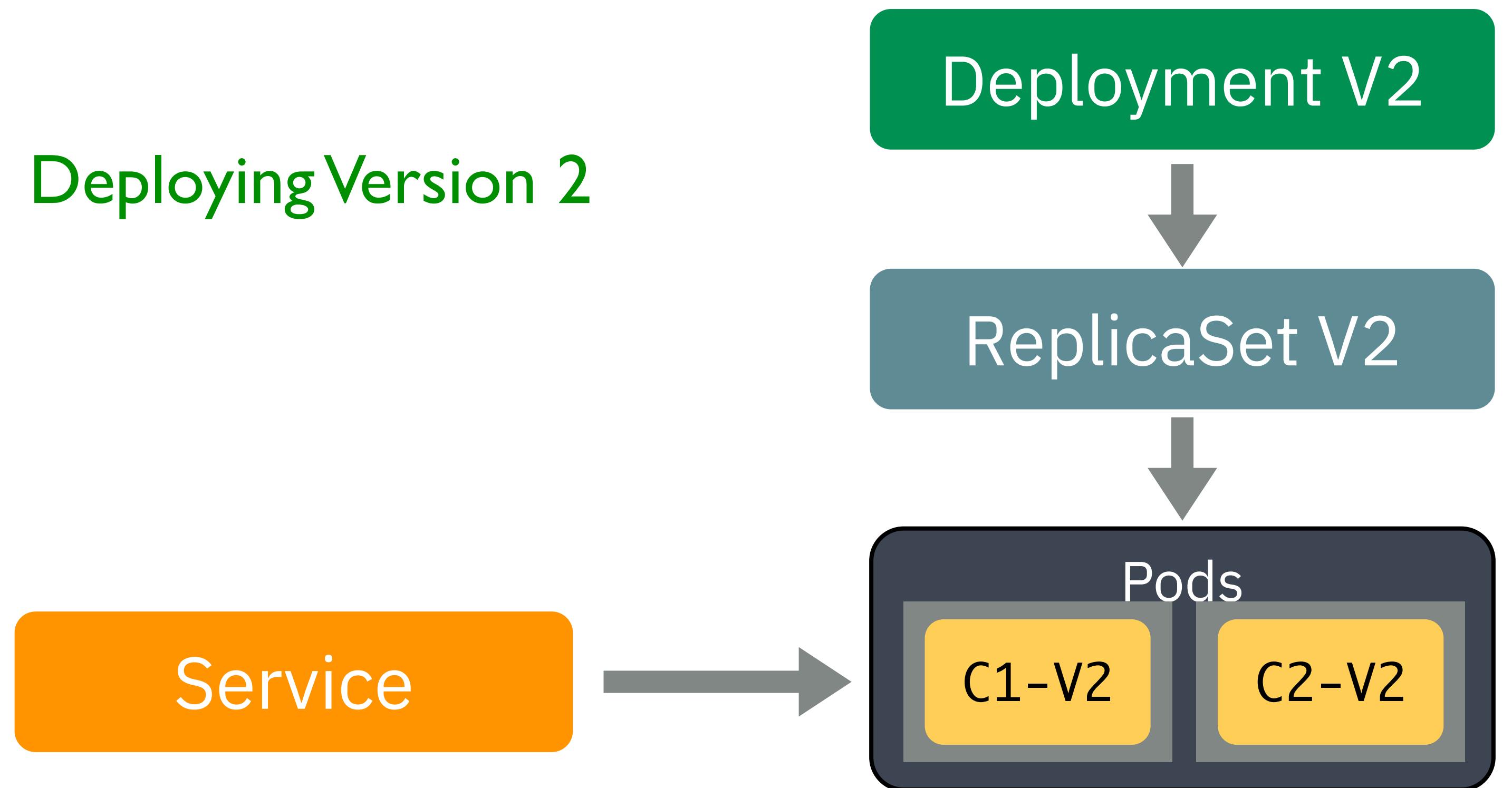
Kubernetes Rolling Updates (Zero Downtime Deployments)

```
$ kubectl set image deployment/<application-name> <container-name>=<container-image-name>:<new-version>
```



Kubernetes Rolling Updates (Zero Downtime Deployments)

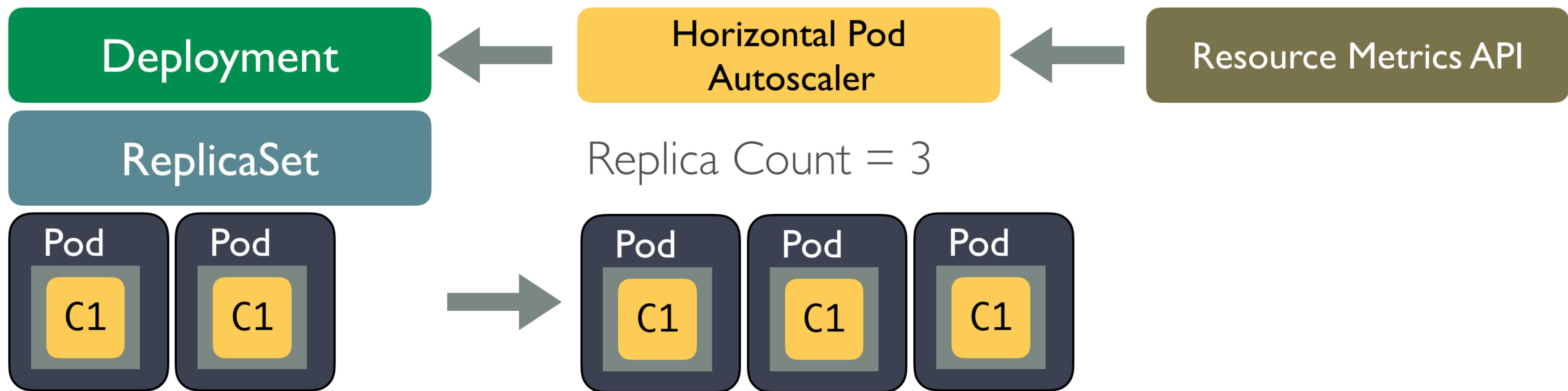
```
$ kubectl set image deployment/<application-name> <container-name>=<container-image-name>:<new-version>
```



Kubernetes Autoscaling

Kubernetes allows pods to be manually scaled either using ReplicaSets or Deployments. This can be achieved using the following CLI command:

```
$ kubectl scale --replicas=<desired-instance-count> deployment/<application-name>
```



Kubernetes Distributions

kubernetes, minikube, openshift, minishift, crc, microk8s, k3s, k3d

The screenshot shows the official Kubernetes website's 'Getting started' section. The 'TUTORIALS' tab is selected. A prominent 'Installing kubeadm' link is highlighted with a blue border. To its right is a large icon of a gear with a blue 'k' symbol.

The screenshot shows the 'TUTORIALS' section of the Kubernetes website. The 'Hello Minikube' tutorial is selected. It features a large 'Hello Minikube' heading and a brief description of what it covers.

The screenshot shows the Canonical MicroK8s page. It highlights the 'Autonomous low-ops Kubernetes for clusters, workstations, edge and IoT' feature. A 'certified' badge with a ship wheel icon is visible on the right.

The screenshot shows the Red Hat OpenShift homepage. It features a large 'OPENSHIFT 4.3 IS HERE' banner and a 'The Kubernetes platform for big ideas' section. A 'Get started' button is at the bottom.

The screenshot shows the OKD Minishift landing page. It features a large 'okd' logo and the text 'Minishift: Develop Applications Locally in a Containerized OKD Cluster'. A 'GET STARTED' button is at the bottom.

The screenshot shows the Rancher K3s page. It highlights 'Lightweight Kubernetes' and 'The certified Kubernetes distribution built for IoT & Edge computing'. A code snippet for curl command is shown, and a 'Great For' section lists 'Edge', 'IoT', 'CI', and 'ARM'.

Hands-On

Some Assembly Required

- Tools you will need to complete this lab:
 - Computer running macOS, Linux, or Windows*
 - Internet Access to download Docker Images
 - IBM Cloud Account for Kubernetes
 - Text Editor (i will use VSCode)
 - GitHub Account



* Windows users may need an ssh client

* PC users must have "VT-x/AMD-V hardware acceleration" turned on in your BIOS for VirtualBox to work

Source for This Lab

- The source for this lab can be found on GitHub at:

```
$ git clone https://github.com/nyu-devops/lab-kubernetes.git  
$ cd lab-kubernetes  
$ vagrant up  
$ cd /vagrant
```



Confirm that your cluster Works

- Make sure that your cluster is configured correctly

```
$ kubectl get all
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	3m44s

Deploy Nginx

- Create a Deployment

```
$ kubectl create deployment my-nginx --image=nginx:alpine
deployment.apps/my-nginx created
```

- Check that the deployment and pods were created

```
$ kubectl get deployments
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
my-nginx   1/1     1           1           3h42m

$ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
my-nginx-b49cf867-q7mjn      1/1     Running   0          3h42m
```

Deploy Nginx

- Create a Service

```
$ kubectl expose deploy my-nginx --type=NodePort --port=80
service/my-nginx exposed
```

- Check that the service was created

\$ kubectl get service						
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	
kubernetes	ClusterIP	172.21.0.1	<none>	443/TCP	317d	
my-nginx	NodePort	172.21.12.226	<none>	80:30371/TCP	3h42m	

Kubectl get all

```
vagrant@kubernetes:~$ kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/my-nginx-64f47865f-2z5w8	1/1	Running	0	2m47s
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP
service/my-nginx	NodePort	10.104.182.40	<none>	80:30965/TCP
NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/my-nginx	1/1	1	1	2m47s
NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/my-nginx-64f47865f	1	1	1	2m47s

Naming Convention

- We made a deployment called my-nginx and it created:

Deployment: my-nginx

ReplicaSet: my-nginx-b49cf867

Pod: my-nginx-b49cf867-q7mjn

Naming Convention

- We made a deployment called my-nginx and it created:



Naming Convention

- We made a deployment called my-nginx and it created:

Deployment: my-nginx

ReplicaSet: my-nginx-b49cf867

Pod name starts with ReplicaSet Name

Pod: my-nginx-b49cf867-q7mjn

Get the NodePort

\$ kubectl get service my-nginx					
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
my-nginx	NodePort	172.21.9.54	<none>	80:30187/TCP	37d

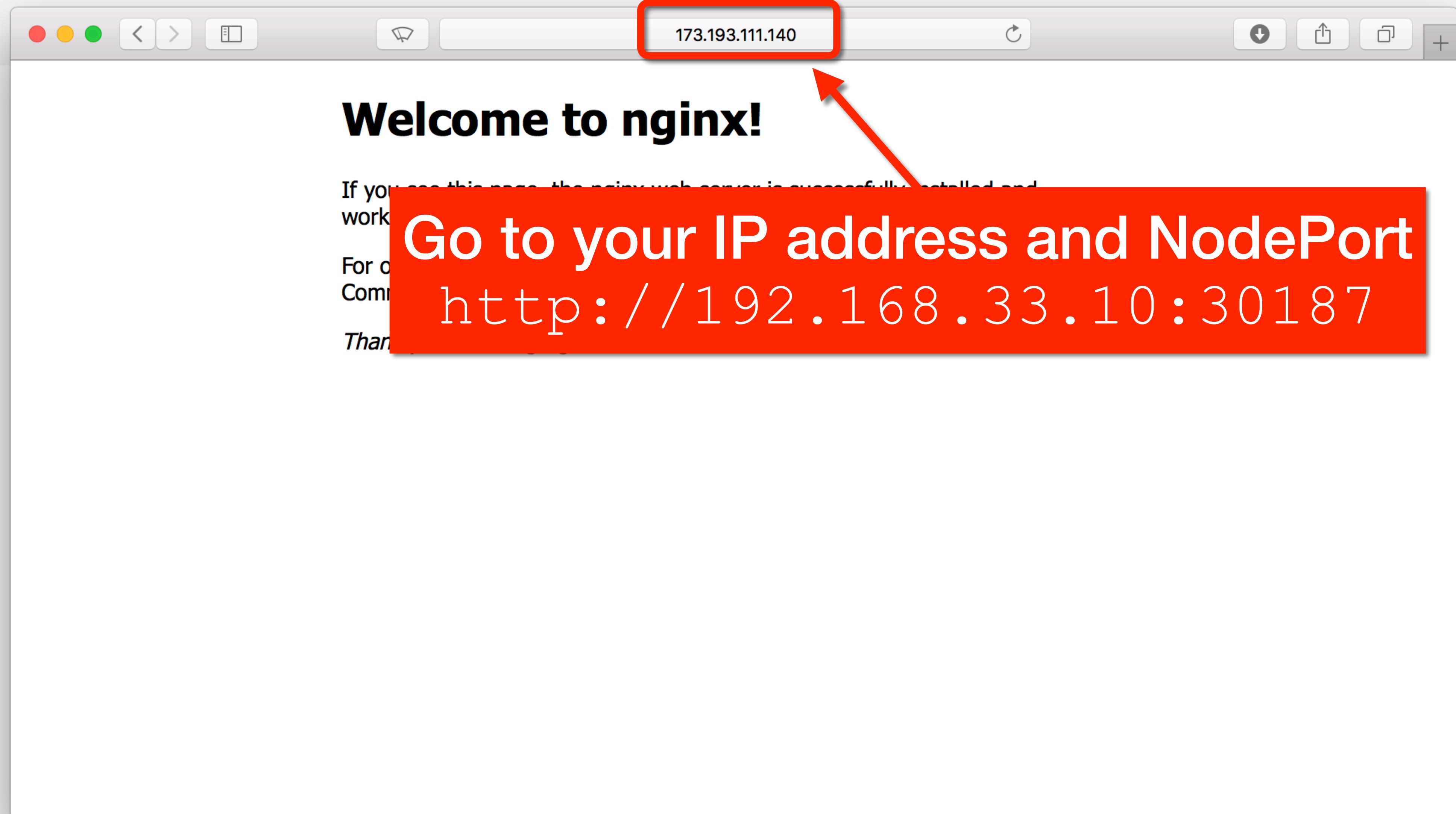
This is the NodePort that was assigned

Get the Public IP address

The IP address of the Vagrant Virtual Machine is in the Vagrantfile

192.168.33.10

Access NodePort



Destroy Nginx

- Delete a Service

```
$ kubectl delete svc my-nginx  
service "my-nginx" deleted
```

- Delete the deployment

```
$ kubectl delete deploy my-nginx  
deployment.apps "my-nginx" deleted
```

Introduction to Kubernetes



Let's Deploy our Project to Kubernetes



Build our Project as a Docker Image

- First let's build our project as a Docker image

```
$ cd /vagrant  
$ docker build -t hitcounter:1.0 .
```

- Or use docker-compose

```
$ cd /vagrant  
$ docker-compose build app
```

Push our Image to the Microk8s registry

- First let's tag the Docker image we just built

```
$ docker tag hitcounter:1.0 localhost:32000(hitcounter:1.0
```

- Then we push it to the remote registry

```
$ docker push localhost:32000(hitcounter:1.0
```

Create A Redis Service*

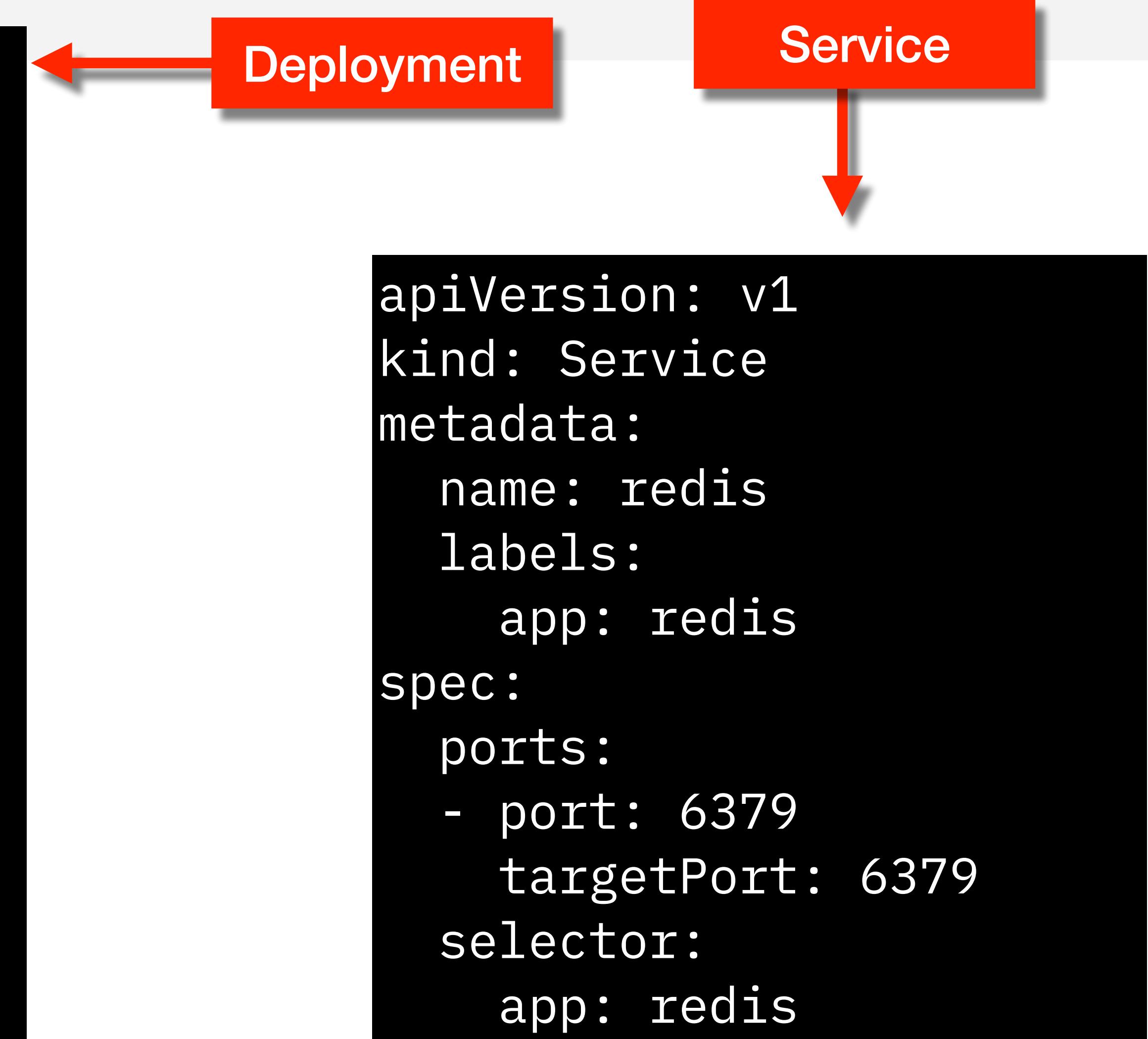
- The demo application requires Redis to store it's state
- We can deploy a Redis container from the `redis.yaml` file in the `./deploy` folder of the current repository

```
$ kubectl apply -f deploy/redis.yaml
deployment.apps/redis created
service/redis created
```

* Note: in a "real" deployment we would attach a volume to the Redis container to persist it's data beyond the life of the container

Redis.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis
  labels:
    app: redis
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
        - name: redis
          image: "redis:alpine"
          ports:
            - containerPort: 6379
              protocol: TCP
```



View the Logs from Redis

- You can view the logs of the pod to see how the service is doing

```
$ kubectl logs redis-784d69fbfd-qp5p5

1:C 01 Dec 2019 20:54:39.303 # o000o000o000o Redis is starting o000o000o000o
1:C 01 Dec 2019 20:54:39.303 # Redis version=5.0.6, bits=64, commit=00000000, modified=0, pid=1, just started
1:C 01 Dec 2019 20:54:39.303 # Warning: no config file specified, using the default config. In order to specify a
config file use redis-server /path/to/redis.conf
1:M 01 Dec 2019 20:54:39.305 * Running mode=standalone, port=6379.
1:M 01 Dec 2019 20:54:39.305 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/
somaxconn is set to the lower value of 128.
1:M 01 Dec 2019 20:54:39.305 # Server initialized
1:M 01 Dec 2019 20:54:39.305 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will
create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/
transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot.
Redis must be restarted after THP is disabled.
1:M 01 Dec 2019 20:54:39.305 * Ready to accept connections
```

Create your Credentials Secret

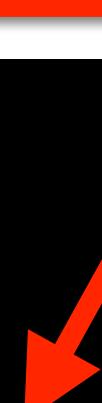
- Let's use the command line method of creating a secret for DATABASE_URI

```
$ export DATABASE_URI="redis://:@redis:6379/0"  
  
$ kubectl create secret generic redis-creds \  
--from-literal=uri=$DATABASE_URI  
  
secret/redis-creds created
```

Check your Credentials Secret

- Let's see what was created for **There is your base64 encoded secret**

```
$ kubectl get secret redis-creds -o yaml
apiVersion: v1
data:
  uri: cmVkaXM6Ly86QHJlZGlz0jYzNzkvMA==
kind: Secret
metadata:
  creationTimestamp: "2019-11-20T03:53:22Z"
  name: redis-creds
  namespace: default
  resourceVersion: "3437"
  selfLink: /api/v1/namespaces/default/secrets/redis-creds
  uid: aee28dcf-f1f4-40ba-83fc-8b775c208264
type: Opaque
```



Now we can Deploy Our Image

- You can use kubectl apply with the files under /vagrant/deploy

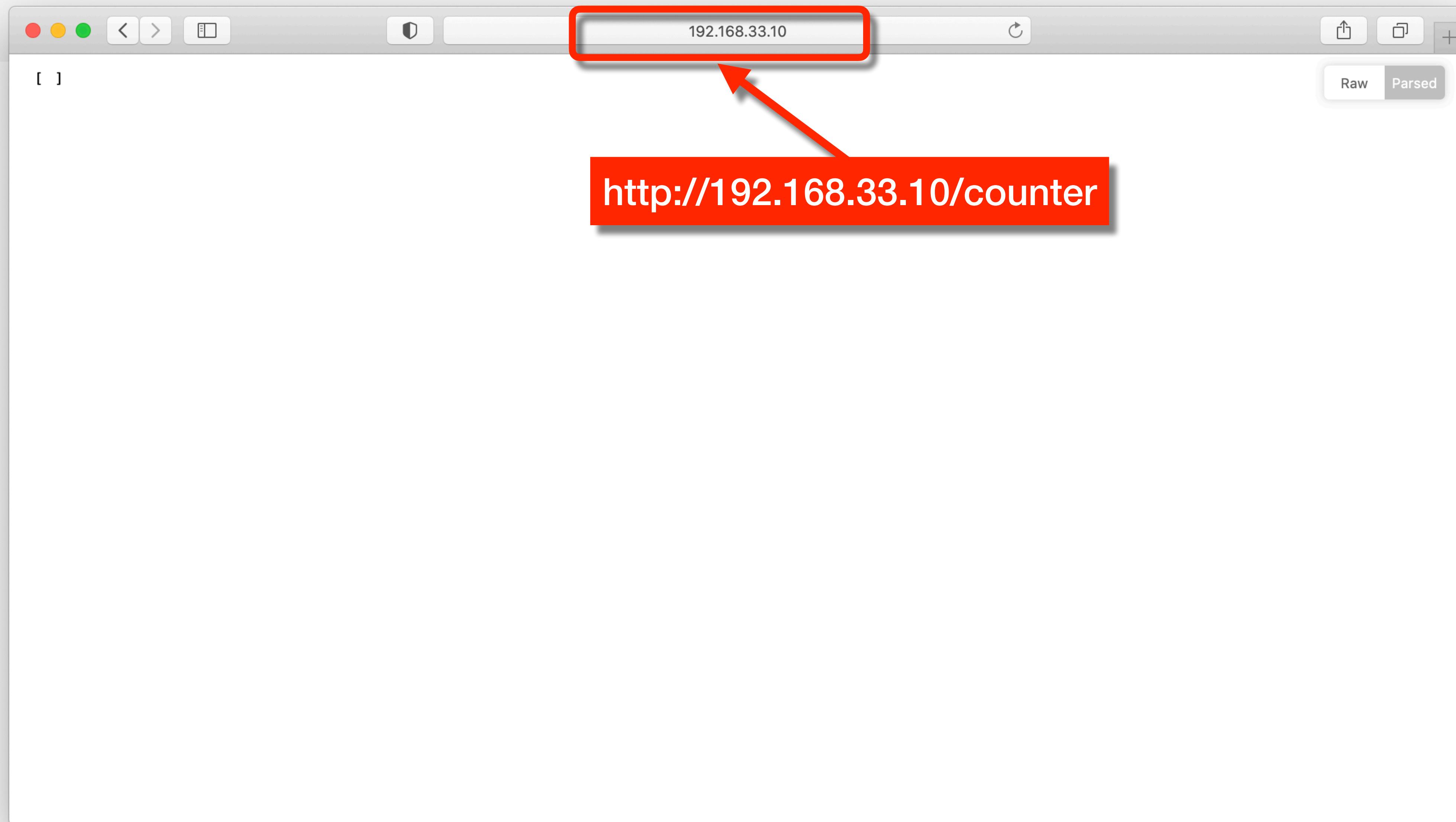
```
$ cd /vagrant/deploy
```

```
$ kubectl apply -f deployment.yaml
deployment.apps/hitcounter created
```

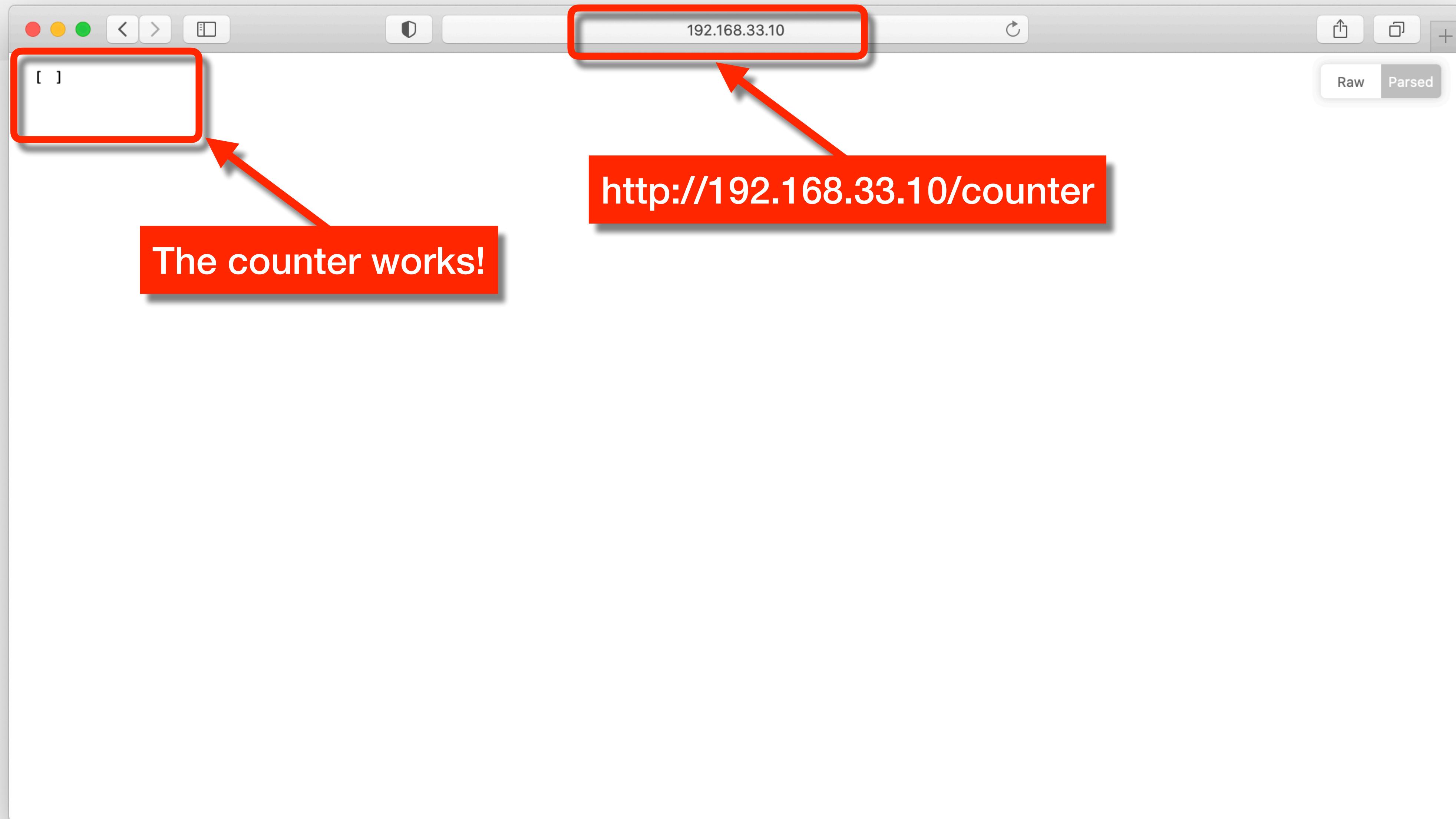
```
$ kubectl apply -f service.yaml
service/hitcounter-service created
```

```
$ kubectl apply -f ingress.yaml
ingress.extensions/hitcounter-ingress created
```

Your App on Kubernetes



Your App on Kubernetes

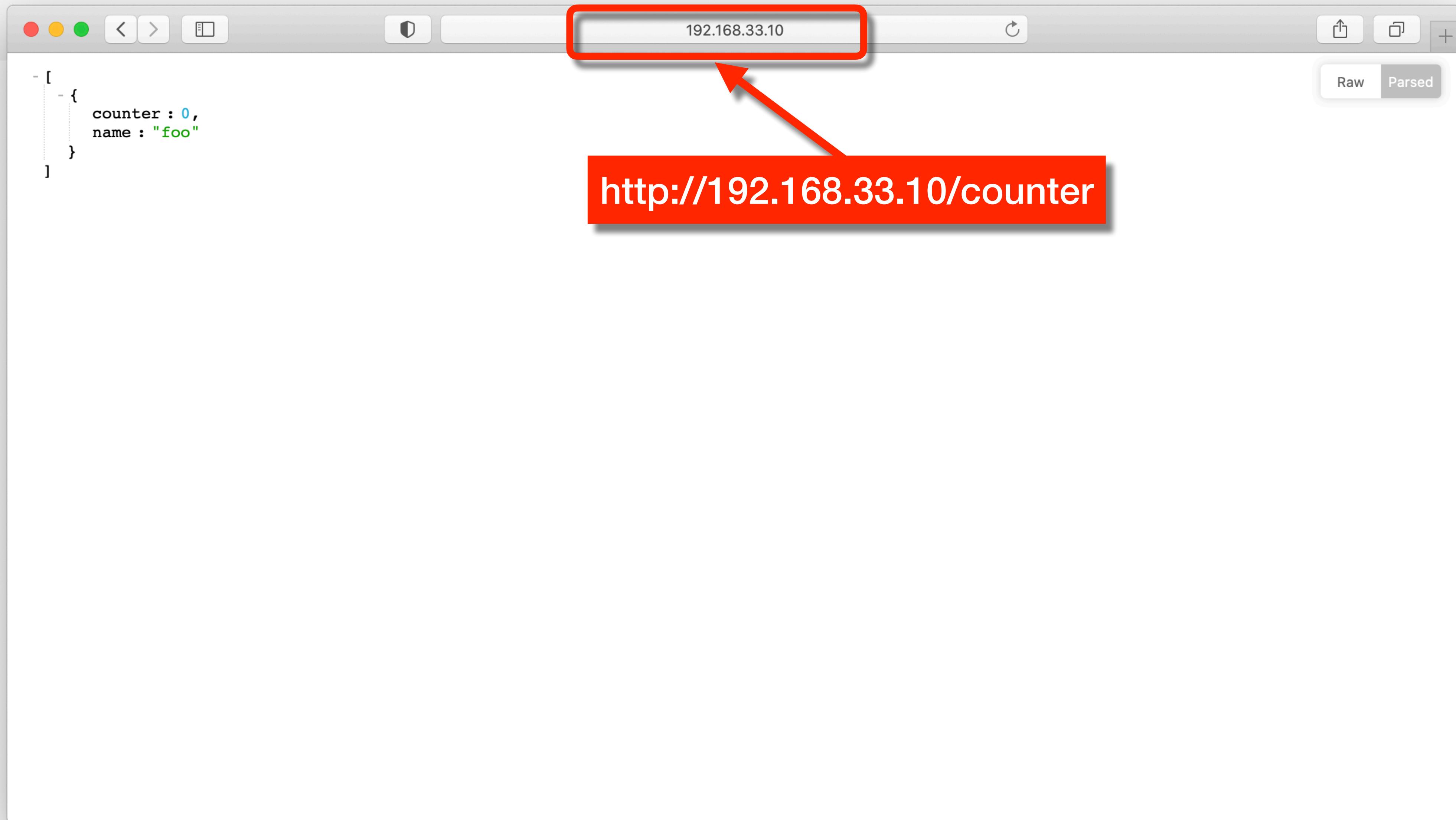


Use HTTP command to increment the counter

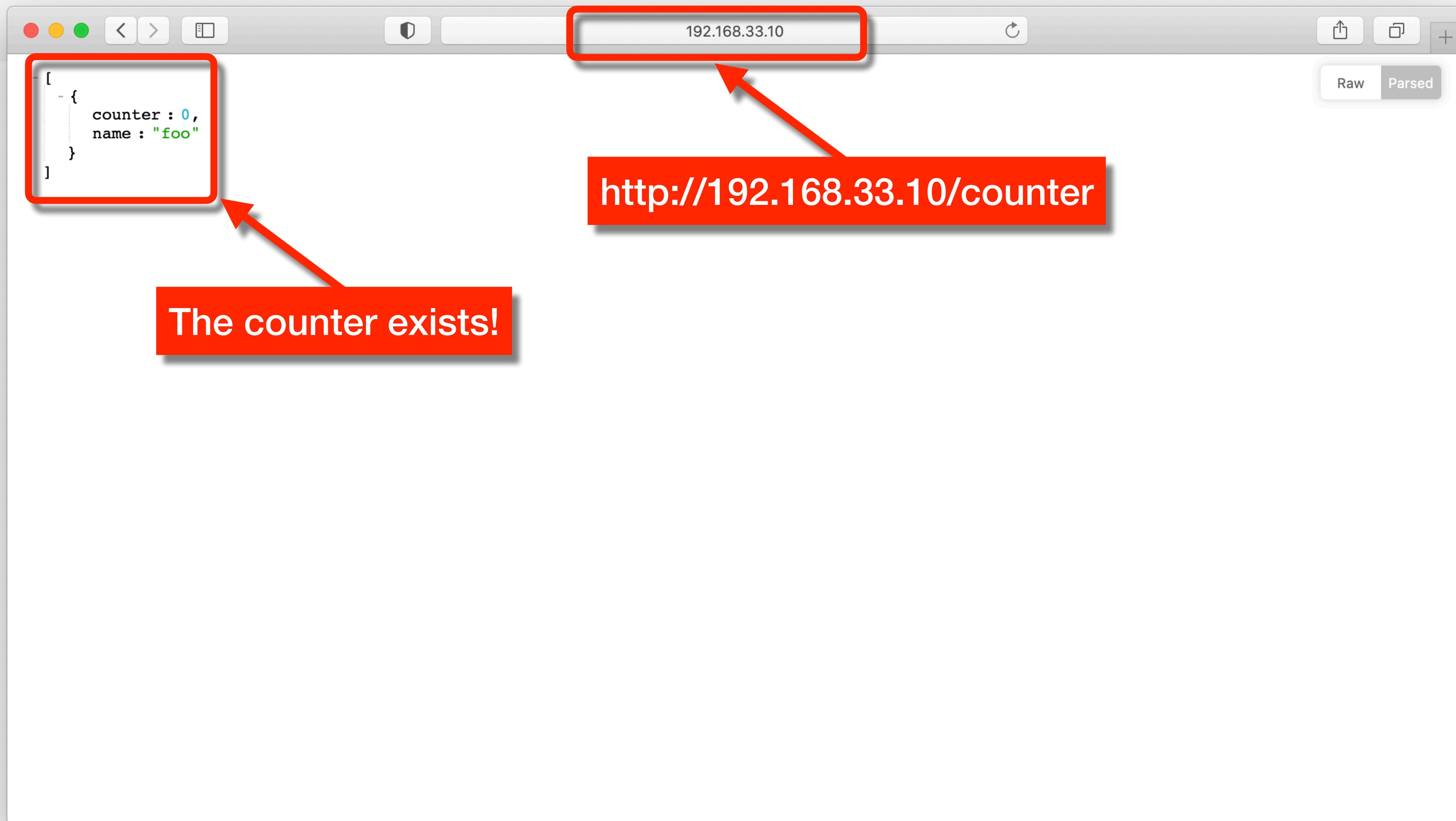
```
$ http post localhost/counters/foo
HTTP/1.1 201 CREATED
Connection: keep-alive
Content-Length: 27
Content-Type: application/json
Date: Mon, 23 Nov 2020 18:49:45 GMT
Location: http://localhost/counters/foo
Server: nginx/1.19.0

{
    "counter": 0,
    "name": "foo"
}
```

Check again



Check again



Let's Scale our Service

- Kubernetes will scale your service horizontally for you by creating more pods

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
hitcounter-5dbfb848b4-h5xsk	1/1	Running	0	11m
redis-784d69fbfd-qp5p5	1/1	Running	0	19m

```
$ kubectl scale --replicas 3 deployment/hitcounter  
deployment.apps/hitcounter scaled
```

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
hitcounter-5dbfb848b4-7zmqc	1/1	Running	0	7s
hitcounter-5dbfb848b4-9xrhj	1/1	Running	0	7s
hitcounter-5dbfb848b4-h5xsk	1/1	Running	0	12m
redis-784d69fbfd-qp5p5	1/1	Running	0	20m

Let's Scale our Service

Now there are three (3) instances running

- Kubernetes will scale your service horizontally for you by creating more pods

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
hitcounter-5dbfb848b4-h5xsk	1/1	Running	0	11m
redis-784d69fbfd-qp5p5	1/1	Running	0	19m

```
$ kubectl scale --replicas 3 deployment/hitcounter
```

deployment.apps/hitcounter scaled

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
hitcounter-5dbfb848b4-7zmqc	1/1	Running	0	7s
hitcounter-5dbfb848b4-9xrhj	1/1	Running	0	7s
hitcounter-5dbfb848b4-h5xsk	1/1	Running	0	12m
redis-784d69fbfd-qp5p5	1/1	Running	0	20m

Watch the Rollout

- You can use this command to watch the rollout:

```
kubectl rollout status deployment <name>
```

```
$ kubectl scale --replicas 10 deployment/hitcounter
deployment.apps/hitcounter scaled

$ kubectl rollout status deployment hitcounter
Waiting for deployment "hitcounter" rollout to finish: 1 of 10 updated replicas are available...
Waiting for deployment "hitcounter" rollout to finish: 2 of 10 updated replicas are available...
Waiting for deployment "hitcounter" rollout to finish: 3 of 10 updated replicas are available...
Waiting for deployment "hitcounter" rollout to finish: 4 of 10 updated replicas are available...
Waiting for deployment "hitcounter" rollout to finish: 5 of 10 updated replicas are available...
Waiting for deployment "hitcounter" rollout to finish: 6 of 10 updated replicas are available...
Waiting for deployment "hitcounter" rollout to finish: 7 of 10 updated replicas are available...
Waiting for deployment "hitcounter" rollout to finish: 8 of 10 updated replicas are available...
Waiting for deployment "hitcounter" rollout to finish: 9 of 10 updated replicas are available...
deployment "hitcounter" successfully rolled out
```

- You can roll back with:

```
kubectl rollout undo deployment <name>
```

Filter by Labels

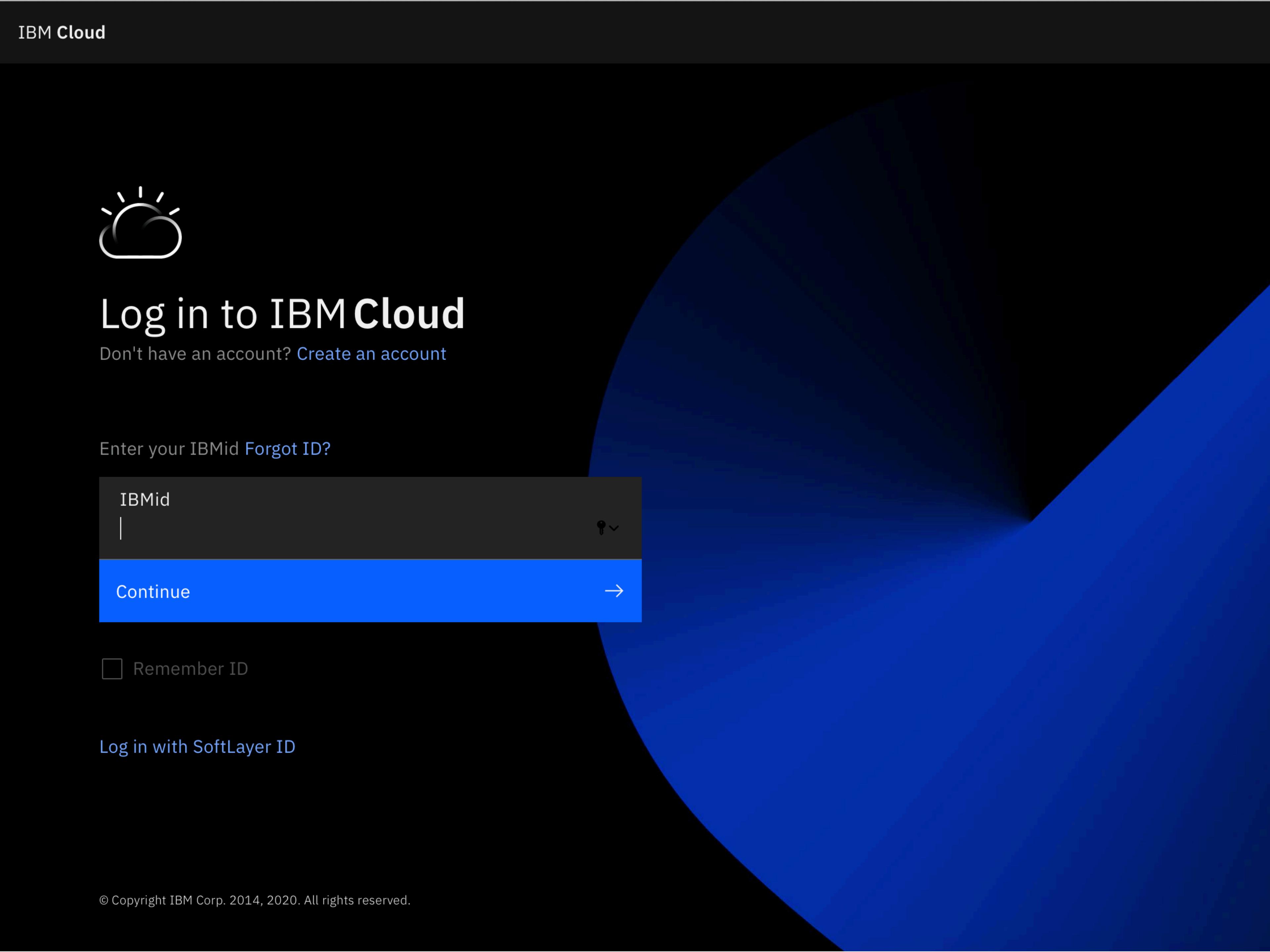
- You can use labels to filter command results

```
kubectl get pods -l <label>=<value>
```

\$ kubectl get pods					
NAME	READY	STATUS	RESTARTS	AGE	
hitcounter-5dbfb848b4-h5xsk	1/1	Running	0	27m	
hitcounter-5dbfb848b4-lts5s	1/1	Running	0	6m18s	
redis-784d69fbfd-qp5p5	1/1	Running	0	36m	

\$ kubectl get pods -l app=hitcounter					
NAME	READY	STATUS	RESTARTS	AGE	
hitcounter-5dbfb848b4-h5xsk	1/1	Running	0	28m	
hitcounter-5dbfb848b4-lts5s	1/1	Running	0	6m27s	

Deploy to IBM Cloud



The image shows the IBM Cloud login interface. At the top, it says "IBM Cloud". Below that is a cloud icon with a sun. The main title is "Log in to IBM Cloud". A link "Don't have an account? [Create an account](#)" is present. A large input field is labeled "IBMid" with a placeholder and a password icon. A blue "Continue" button with a right arrow is below it. To the left of the "Continue" button is a checkbox for "Remember ID". Below the input field is a link "Log in with SoftLayer ID". At the bottom, a copyright notice reads "© Copyright IBM Corp. 2014, 2020. All rights reserved." The background features a dark blue and black abstract design.

IBM Cloud

Log in to IBM Cloud

Don't have an account? [Create an account](#)

Enter your IBMid [Forgot ID?](#)

IBMid

Continue →

Remember ID

[Log in with SoftLayer ID](#)

© Copyright IBM Corp. 2014, 2020. All rights reserved.

How do we Deploy Our Images?

- You must use the IBM Cloud command line interface to push your images to the IBM Cloud Docker Registry
- Then you can deploy containers from those images
- The IBM Cloud CLI is already installed in your Vagrant VM

Create Kubernetes Cluster

<https://cloud.ibm.com/>

The screenshot shows the IBM Cloud dashboard interface. On the left is a sidebar with icons for various services: Resource summary, Clusters (1), Cloud Foundry apps (2), Cloud Foundry services (3), Services (4), and Developer tools (2). The main area displays a 'Resource summary' with 12 resources. A prominent blue button at the top right says 'Create resource +'. The URL 'cloud.ibm.com' is visible in the browser's address bar.

Dashboard

Resource summary

12 Resources

Clusters 1

Cloud Foundry apps 2

Cloud Foundry services 3

Services 4

Developer tools 2

Add resources +

Upgrade account

Create resource +

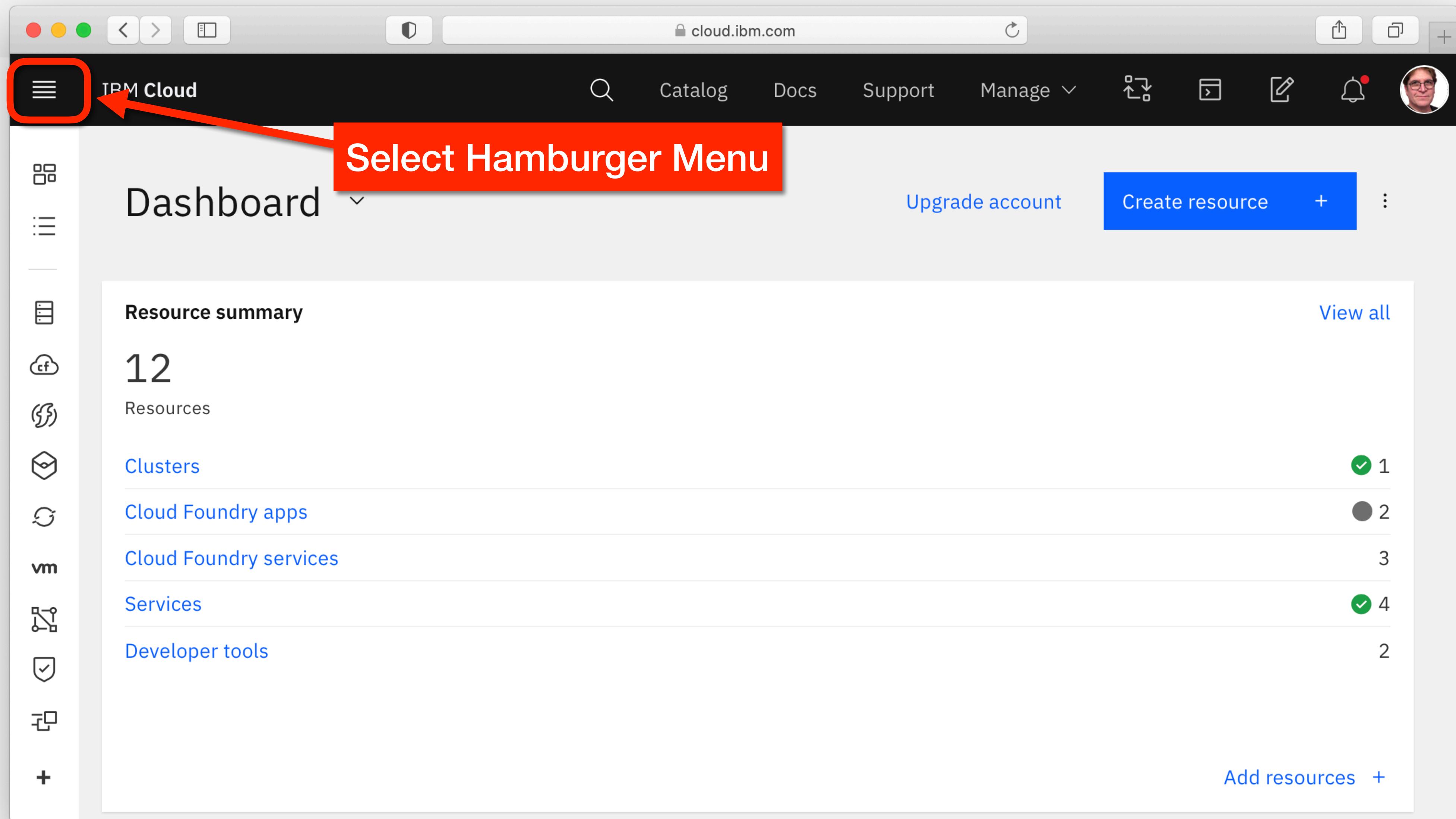
IBM Cloud

Catalog Docs Support Manage

cloud.ibm.com

Create Kubernetes Cluster

<https://cloud.ibm.com/>



Create Kubernetes Cluster

The screenshot shows the IBM Cloud dashboard with a sidebar on the left and a main content area on the right.

Left Sidebar:

- Dashboard
- Resource List
- Classic Infrastructure
- Cloud Foundry >
- Functions >
- Kubernetes > Overview** (This item is highlighted with a red box and an arrow points to it from a callout box.)
- OpenShift >
- VMware >
- VPC Infrastructure >
- Security and Compliance >
- Code Engine >
- API Management

Main Content Area:

Deploy, scale, and manage your containerized application workloads

Deploy native Kubernetes clusters with the latest upstream versions on hardened master and worker nodes.

Select Kubernetes | Overview

Clusters
Registry
Helm Catalog

Cluster capabilities

Create a Cluster

The screenshot shows the IBM Cloud interface for creating a Kubernetes cluster. The left sidebar has a dark theme with blue highlights for 'Kubernetes'. The main content area has a white background with a central blue button labeled 'Create a cluster'.

cloud.ibm.com

IBM Cloud

Kubernetes

Catalog Docs Support Manage

Clusters Registry Helm catalog

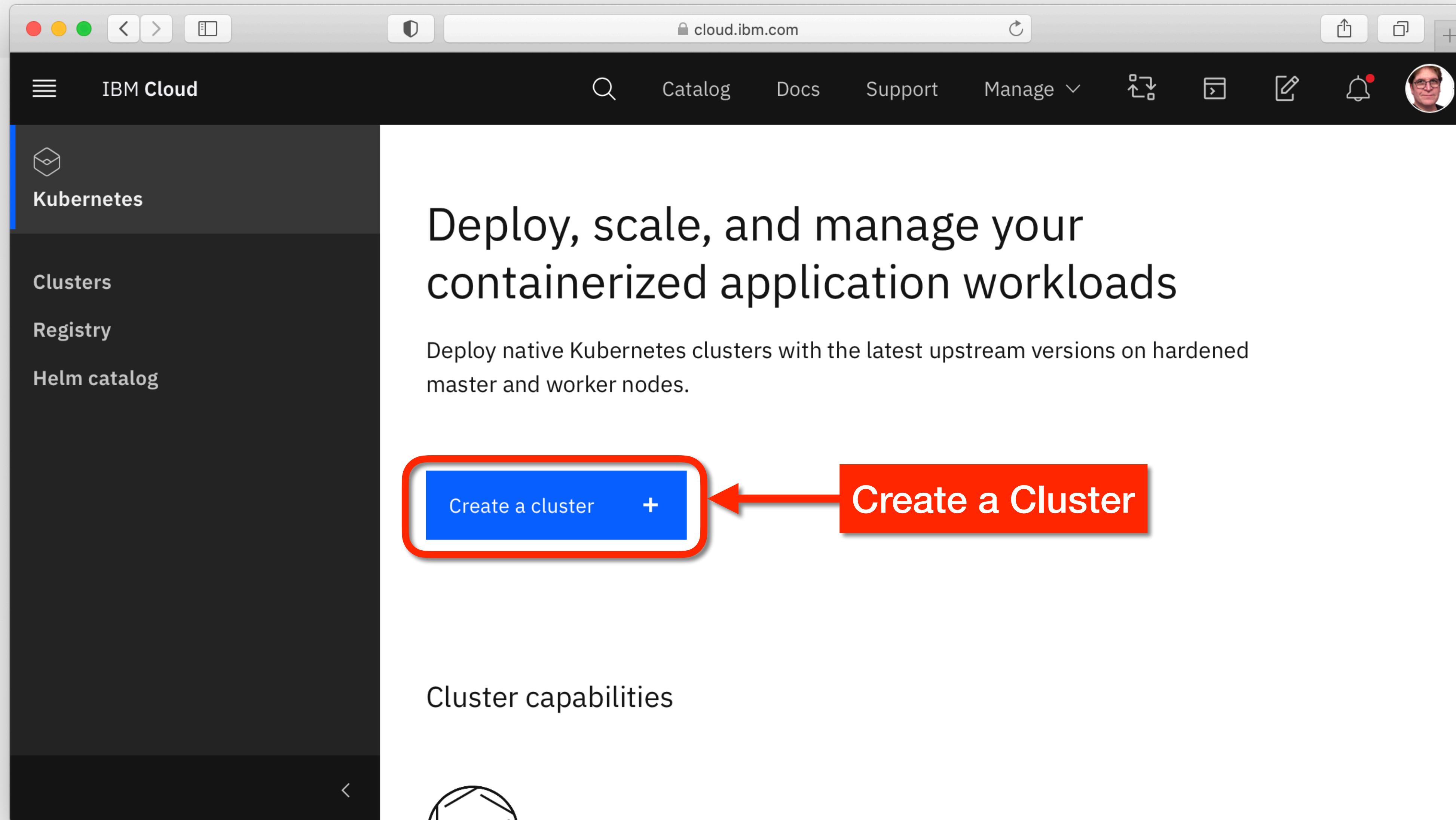
Deploy, scale, and manage your containerized application workloads

Deploy native Kubernetes clusters with the latest upstream versions on hardened master and worker nodes.

Create a cluster +

Cluster capabilities

Create a Cluster



Create a Cluster

The screenshot shows the IBM Cloud web interface for creating a Kubernetes cluster. The top navigation bar includes links for View all, Catalog, Docs, Support, Manage, and a user profile. The main left sidebar has a 'Create' button highlighted with a blue border, and an 'About' button. The central area displays a 'Summary' section with a tree view showing a 'Kubernetes cluster' node expanded, revealing a 'Worker node' with a value of '1'. The 'Worker node' row shows 'Free - 2 vCPUs 4GB RAM'. Below this, a 'Total estimated cost' is listed as 'Free/mo'. A note states: 'Additional charges for networking and bandwidth might apply. Actual monthly total will vary with tiered pricing.' At the bottom right is a large blue 'Create' button.

View all /

Kubernetes cluster

Create About

Select a plan

Learn more about the differences between plans in our [docs](#).

Free

Orchestration service

Select the [container platform type](#) and version for your cluster. For more information about versions, including links to the container platform community release notes, [see the docs](#).

Summary

Kubernetes cluster

1 Worker node Free

Free - 2 vCPUs 4GB RAM

Total estimated cost Free/mo

Additional charges for *networking and bandwidth* might apply.
Actual monthly total will vary with *tiered pricing*.

Create

Add to estimate

Create a Cluster

The screenshot shows the IBM Cloud web interface for creating a Kubernetes cluster. The URL in the browser is `cloud.ibm.com`. The top navigation bar includes links for Catalog, Docs, Support, Manage, and a user profile. On the left, there's a sidebar with a 'Create' button and an 'About' link. The main content area is titled 'Kubernetes cluster' and features a prominent red callout box with the text 'Select FREE (not standard)'. Below this, a dropdown menu is highlighted with a red border and arrow, showing the option 'Free'. To the right, a 'Summary' panel details the cluster configuration: 1 Worker node (Free - 2 vCPUs 4GB RAM). The total estimated cost is listed as 'Free/mo'. A note states that additional charges for networking and bandwidth might apply. At the bottom, there are 'Create' and 'Add to estimate' buttons.

Select FREE (not standard)

Free

Orchestration service

Select the [container platform type](#) and version for your cluster. For more information about versions, including links to the container platform community release notes, [see the docs](#).

Summary

Kubernetes cluster

1 Worker node Free

Free - 2 vCPUs 4GB RAM

Total estimated cost Free/mo

Additional charges for [networking](#) and [bandwidth](#) might apply.

Actual monthly total will vary with [tiered pricing](#).

Create

Add to estimate

Create a Cluster

The screenshot shows the IBM Cloud web interface for creating a Kubernetes cluster. The top navigation bar includes links for Catalog, Docs, Support, Manage, and a user profile. The main page title is "Kubernetes cluster". A prominent red callout box in the center-left says "Select FREE (not standard)". Below it, a dropdown menu is highlighted with a red border and arrow, showing "Free" as the selected option. To the right, a "Summary" section details the cluster configuration: "1 Worker node" (Free - 2 vCPUs 4GB RAM). The "Total estimated cost" is listed as "Free/mo". A note states that additional charges for networking and bandwidth might apply. A large blue "Create" button is at the bottom right. Red arrows point from the text "Select FREE (not standard)" to the dropdown menu, and from the text "Scroll Down" to the bottom of the page.

View all /

Kubernetes cluster

Create About

Select a plan **Select FREE (not standard)**

Learn more about the differences between plans in our [docs](#).

Free

Orchestration service

Select the [container platform type](#) and version for your cluster. For more information about versions, including links to the container platform community release notes, see [the docs](#).

Summary

Kubernetes cluster

1 Worker node **Free**

Free - 2 vCPUs 4GB RAM

Total estimated cost **Free/mo**

Additional charges for [networking](#) and [bandwidth](#) might apply.
Actual monthly total will vary with [tiered pricing](#).

Scroll Down

Create

Add to estimate

Create a Cluster

The screenshot shows the IBM Cloud Orchestration service interface for creating a Kubernetes cluster. The top navigation bar includes links for Catalog, Docs, Support, Manage, and a user profile. The main left panel is titled "Orchestration service" and displays a summary of the selected container platform type and version. A large "Create" button is prominently displayed at the bottom right.

Orchestration service

Select the [container platform type](#) and version for your cluster. For more information about versions, including links to the container platform community release notes, [see the docs](#).

Kubernetes ✓

1.18.12

Resource details

Cluster name

mycluster-free

Resource group

Default

Summary

Kubernetes cluster

1 Worker node Free

Free - 2 vCPUs 4GB RAM

Total estimated cost Free/mo

*Additional charges for networking and bandwidth might apply.
Actual monthly total will vary with tiered pricing.*

Create

Add to estimate

Create a Cluster

The screenshot shows the IBM Cloud Orchestration service interface for creating a Kubernetes cluster. The top navigation bar includes links for Catalog, Docs, Support, Manage, and a user profile. The main left panel is titled "Orchestration service" and lists "Kubernetes" as the selected container platform type, version 1.18.12. A red callout box with the text "Give it a name" points to the "Cluster name" input field in the "Resource details" section, which contains the value "mycluster-free". The right panel displays a "Summary" of the cluster configuration, showing one worker node (Free - 2 vCPUs 4GB RAM) and a total estimated cost of Free/mo. Additional charges for networking and bandwidth might apply. A large blue "Create" button is at the bottom of the summary panel.

cloud.ibm.com

IBM Cloud

Catalog Docs Support Manage

Orchestration service

Select the [container platform type](#) and version for your cluster. For more information about versions, including links to the container platform community release notes, [see the docs](#).

Kubernetes 1.18.12

Give it a name

Resource details

Cluster name: mycluster-free

Resource group: Default

Summary

Kubernetes cluster

1 Worker node **Free**
Free - 2 vCPUs 4GB RAM

Total estimated cost **Free/mo**

Additional charges for [networking](#) and [bandwidth](#) might apply.
Actual monthly total will vary with [tiered pricing](#).

Create

Add to estimate

Create a Cluster

The screenshot shows the IBM Cloud Orchestration service interface for creating a Kubernetes cluster. The left panel displays the selection of the container platform type (Kubernetes) and version (1.18.12). The right panel shows the summary of the cluster creation, including a single worker node with 2 vCPUs and 4GB RAM, categorized under 'Free'. A red box highlights the 'Create' button in the summary panel. Red arrows point from the 'Give it a name' callout to the cluster name input field ('mycluster-free') and from the 'Create a Cluster' callout to the 'Create' button.

Orchestration service

Select the [container platform type](#) and version for your cluster. For more information about versions, including links to the container platform community release notes, [see the docs](#).

Kubernetes 1.18.12

Resource details

Cluster name mycluster-free

Resource group Default

Summary

Kubernetes cluster

1 Worker node Free

Free - 2 vCPUs 4GB RAM

Total estimated cost Free/mo

Additional charges for [networking](#) and [bandwidth](#) might apply.
Actual monthly total will vary with [tiered pricing](#).

Create

Add to estimate

Wait until state is normal

The screenshot shows the IBM Cloud web interface for managing Kubernetes clusters. The left sidebar is dark-themed and includes options for 'Kubernetes' (selected), 'Clusters' (highlighted with a blue bar), 'Registry', and 'Helm catalog'. The main content area has a light background and displays the title 'Kubernetes clusters'. At the top of the main area is a search bar labeled 'Filter table' and a blue button labeled 'Create cluster'. Below this is a table with the following columns: Name, State, Location, Worker Count, Created, and Version. A single row is visible for a cluster named 'nyu-devops', which is in 'Normal' state, located in 'Milan 01', has a 'Worker Count' of 1, was created 'Expires in 25 days', and is running 'Version 1.18.12'. At the bottom of the table are pagination controls showing 'Items per page: 25' and '1–1 of 1 item'.

Name	State	Location	Worker Count	Created	Version
nyu-devops	Normal	Milan 01	1	Expires in 25 days	1.18.12

Wait until state is normal

The screenshot shows the IBM Cloud interface for managing Kubernetes clusters. The left sidebar has 'Kubernetes' selected under 'Clusters'. The main area is titled 'Kubernetes clusters' and displays a table of clusters. One cluster, 'nyu-devops', is listed with the following details:

Name	State	Location	Worker Count	Created	Version
nyu-devops	Normal	Milan 01	1	Expires in 25 days	1.18.12

A red box highlights the 'Normal' status of the 'nyu-devops' cluster. A red arrow points from the text 'State must be Normal before you can use it' at the bottom of the slide to the 'Normal' status cell in the table.

State must be Normal before you can use it

Login to IBM Cloud

- Login using your API key for Kubernetes

```
$ ibmcloud login -a https://cloud.ibm.com --apikey @~/.bluemix/apiKey.json -r us-south
```

Login

```
ibmcloud login \
-a https://cloud.ibm.com \
--apikey @~/.bluemix/apiKey.json \
-r us-south
```

Setup for Kubernetes

- Initialize environment for Kubernetes

```
$ ibmcloud ks cluster config --cluster nyu-devops
```

This should be your cluster name

Login to IBM Cloud

```
ibmcloud login -a https://cloud.ibm.com --apikey @~/.bluemix/apiKey.json -r us-south
```

```
(venv) vagrant@kubernetes:~$ ibmcloud login -a https://cloud.ibm.com --apikey @~/.bluemix/apiKey.json -r us-south
API endpoint: https://cloud.ibm.com
Authenticating...
OK
```

Targeted account NYU (21caa03e2981e94f56ea98f347b995a5)

Targeted region us-south

```
API endpoint:      https://cloud.ibm.com
Region:           us-south
User:              jjr12@nyu.edu
Account:          NYU (21caa03e2981e94f56ea98f347b995a5)
Resource group:   No resource group targeted, use 'ibmcloud target -g RESOURCE_GROUP'
CF API endpoint:
Org:
Space:
(venv) vagrant@kubernetes:~$
```

A few More Commands

```
ibmcloud ks cluster config --cluster <cluster_name>
```

```
(venv) vagrant@kubernetes:~$ ibmcloud ks cluster config --cluster nyu-devops
OK
```

The configuration for `nyu-devops` was downloaded successfully.

```
ibmcloud ks cluster config --cluster nyu-devops
```

Added context for `nyu-devops` to the current kubeconfig file.

You can now execute '`kubectl`' commands against your cluster. For example, run '`kubectl get nodes`'.

If you are accessing the cluster for the first time, '`kubectl`' commands might fail for a few seconds while RBAC synchronizes.

Confirm it Works

kubectl cluster-info

```
(venv) vagrant@kubernetes:~$ kubectl cluster-info
```

Kubernetes master is running at <https://127.0.0.1:16443>

CoreDNS is running at <https://127.0.0.1:16443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy>

Metrics-server is running at <https://127.0.0.1:16443/api/v1/namespaces/kube-system/services/https:metrics-server:/proxy>

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

kubectl get nodes

```
(venv) vagrant@kubernetes:~$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
kubernetes	Ready	<none>	5d3h	v1.19.3-34+a56971609ff35a

Login to Container Registry

ibmcloud cr login

```
(venv) vagrant@kubernetes:~$ ibmcloud cr login
```

```
Logging in to 'registry.ng.bluemix.net'...
```

```
Logged in to 'registry.ng.bluemix.net'.
```

```
Logging in to 'us.icr.io'...
```

```
Logged in to 'us.icr.io'.
```

```
OK
```

Set Your Namespace

- The first time you use `ibmcloud cr login` you will need to set a namespace with: `ibmcloud namespace-add <namespace>`

```
$ ibmcloud cr login
Retrieving client certificates for IBM Containers...
{
  "code": "IC5076E",
  "description": "The namespace has not yet been specified. A namespace must be set for your organization before you can work with IBM Containers and cannot be changed once set. You can use 'cf ic namespace set NAMESPACE' to set the namespace.",
  "host_id": "217",
  "incident_id": "267-1476848744.843-1127076",
  "name": "NamespaceNotSpecified",
  "rc": "404",
  "type": "Infrastructure"
}
$ ibmcloud cr namespace-add nyu_edu
nyu_edu
```

Tag an Image for IBM Cloud Container Registry

- To send an image to IBM Cloud first we must tag it with the registry name

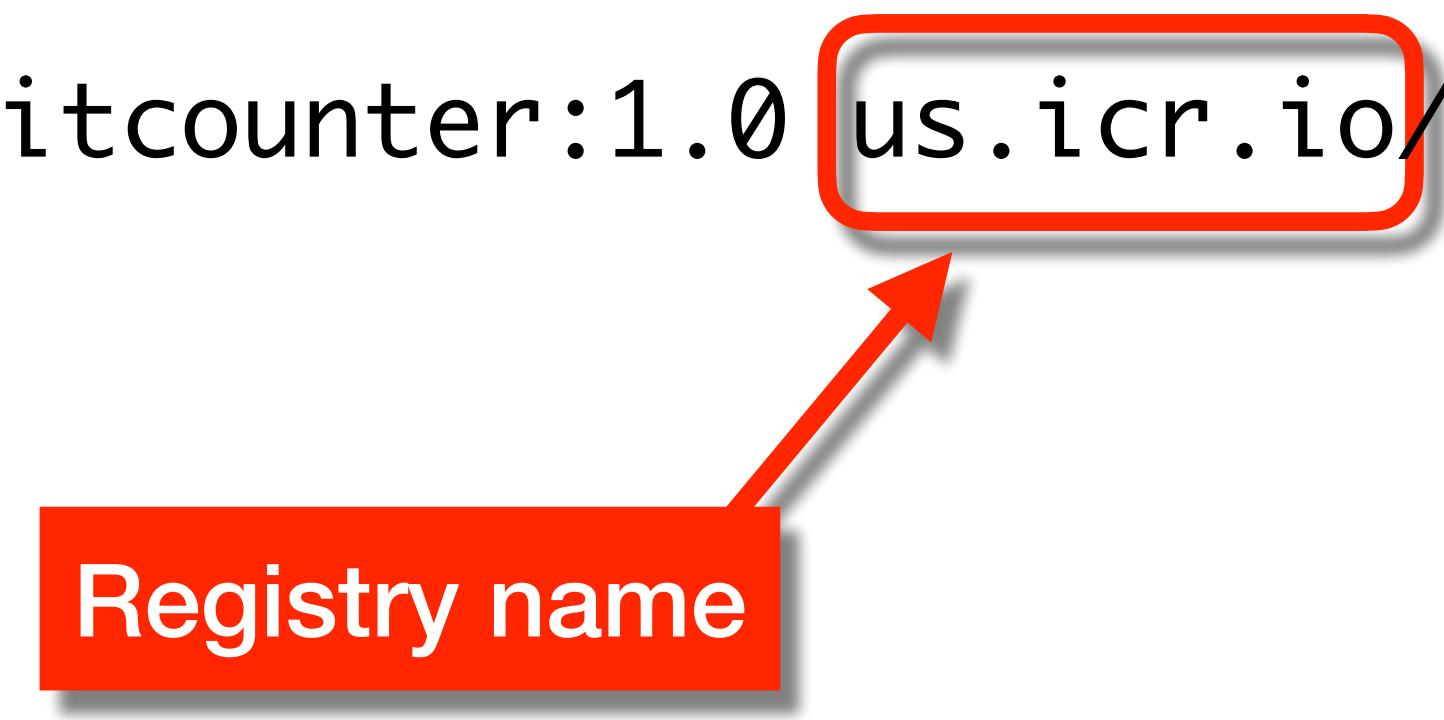
```
docker tag hitcounter:1.0 us.icr.io/nyu_edu/hitcounter:1.0
```

Tag an Image for IBM Cloud Container Registry

- To send an image to IBM Cloud first we must tag it with the registry name

```
docker tag hitcounter:1.0 us.icr.io/nyu_edu/hitcounter:1.0
```

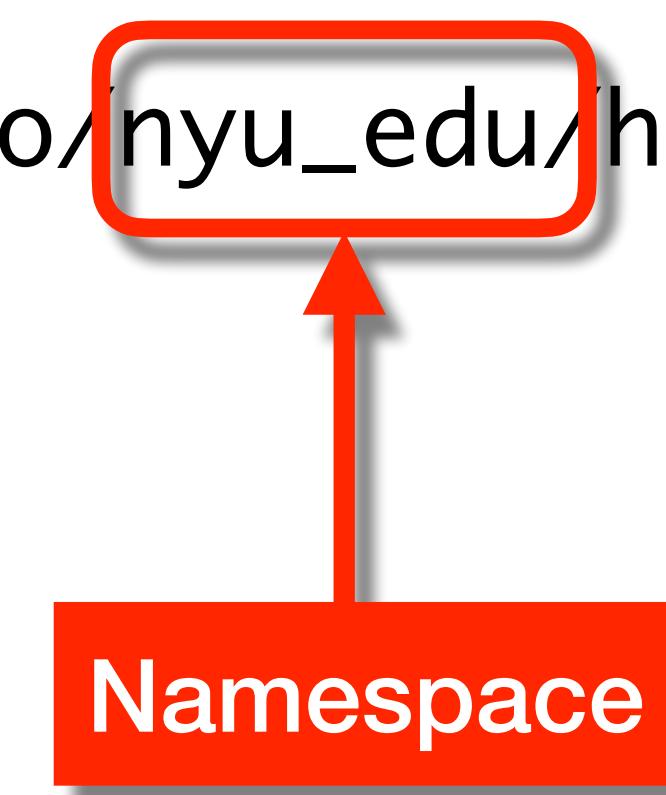
Registry name



Tag an Image for IBM Cloud Container Registry

- To send an image to IBM Cloud first we must tag it with the registry name

```
docker tag hitcounter:1.0 us.icr.io/nyu_edu/hitcounter:1.0
```



Tag an Image for IBM Cloud Container Registry

- To send an image to IBM Cloud first we must tag it with the registry name

```
docker tag hitcounter:1.0 us.icr.io/nyu_edu/hitcounter:1.0
```

Image name

Tag an Image for IBM Cloud

- To send an image to IBM Cloud first we must tag it with the registry name

```
docker tag hitcounter:1.0 us.icr.io/nyu_edu/hitcounter:1.0
```

```
(venv) vagrant@kubernetes:~$ docker tag hitcounter:1.0 us.icr.io/nyu_edu/hitcounter:1.0
```

```
(venv) vagrant@kubernetes: $ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hitcounter	1.0	fbb485930343	4 days ago	140MB
us.icr.io/nyu_edu/hitcounter	1.0	fbb485930343	4 days ago	140MB
mynginx	latest	b315e433ee20	4 days ago	21.8MB
<none>	<none>	4288a1d6b3a8	4 days ago	21.8MB
python	3.8-slim	dbd6cd1a5404	5 days ago	113MB
nginx	alpine	e5dc7aa4b5e	2 weeks ago	21.8MB
redis	6-alpine	c1949ec48c51	3 weeks ago	31.2MB
alpine	latest	d6e46aa2470d	4 weeks ago	5.57MB

Tag an Image for IBM Cloud

- To send an image to IBM Cloud first we must tag it with the registry name

```
docker tag hitcounter:1.0 us.icr.io/nyu_edu/hitcounter:1.0
```

```
(venv) vagrant@kubernetes:~$ docker tag hitcounter:1.0 u
```

These images are the same with
two different names

```
(venv) vagrant@kubernetes: $ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hitcounter	1.0	fbb485930343	4 days ago	140MB
us.icr.io/nyu_edu/hitcounter	1.0	fbb485930343	4 days ago	140MB
mynginx	latest	b315e433ee20	4 days ago	21.8MB
<none>	<none>	4288a1d6b3a8	4 days ago	21.8MB
python	3.8-slim	dbd6cd1a5404	5 days ago	113MB
nginx	alpine	e5dc7aa4b5e	2 weeks ago	21.8MB
redis	6-alpine	c1949ec48c51	3 weeks ago	31.2MB
alpine	latest	d6e46aa2470d	4 weeks ago	5.57MB

Push an Image to IBM Cloud

- Use docker push to send to IBM Cloud Container Registry

```
(venv) vagrant@kubernetes:~$ docker push us.icr.io/nyu_edu/hitcounter:1.0
```

The push refers to repository [us.icr.io/nyu_edu/hitcounter]

```
48e4096c5a26: Pushed
487b2567edc8: Pushing [=====>] 28.9MB
997671074b5b: Pushed
8898e0e667b9: Pushed
05c305d92486: Pushed
2ca008983975: Pushed
053be95103bc: Pushing [=====>] 6.799MB/28.35MB
45a173e90fb4: Pushing [=====>] 1.643MB/7.026MB
f5600c6330da: Waiting
```

Push an Image to IBM Cloud

- Use docker push to send to IBM Cloud **(Layers are being push to the cloud registry)**

```
(venv) vagrant@kubernetes:~$ docker push us.icr.io/nyu_edu/hitcounter:1.0
```

```
The push refers to repository [us.icr.io/nyu_edu/hitcounter]
48e4096c5a26: Pushed
487b2567edc8: Pushing [=====>] 28.9MB
997671074b5b: Pushed
8898e0e667b9: Pushed
05c305d92486: Pushed
2ca008983975: Pushed
053be95103bc: Pushing [=====>] 6.799MB/28.35MB
45a173e90fb4: Pushing [=====>] 1.643MB/7.026MB
f5600c6330da: Waiting
```

Push an Image to IBM Cloud

- Use docker push to send to IBM Cloud

```
(venv) vagrant@kubernetes:~$ docker push us.icr.io/nyu_edu/hitcounter:1.0
The push refers to repository [us.icr.io/nyu_edu/hitcounter]
48e4096c5a26: Pushed
487b2567edc8: Pushed
997671074b5b: Pushed
8898e0e667b9: Pushed
05c305d92486: Pushed
2ca008983975: Pushed
053be95103bc: Pushed
45a173e90fb4: Pushed
f5600c6330da: Pushed
1.0: digest: sha256:bbc05196e4e585733fc308d5e3392c857d5a3de22de559cd0b1ed6e2e5e0ca23 size: 2204
```

```
(venv) vagrant@kubernetes:~$ ic cr image-list
Listing images...
```

Repository	Tag	Digest	Namespace	Created	Size	Security status
us.icr.io/nyu_edu/hitcounter	1.0	bbc05196e4e5	nyu_edu	4 days ago	52 MB	No Issues
us.icr.io/nyu_edu/nyu-hitcounter	0.1.0	81090b3ed00d	nyu_edu	6 months ago	67 MB	2 Issues
us.icr.io/nyu_edu/nyu-hitcounter	latest	81090b3ed00d	nyu_edu	6 months ago	67 MB	2 Issues

OK

Push an Image to IBM Cloud

- Use docker push to send to IBM Cloud

```
(venv) vagrant@kubernetes:~$ docker push us.icr.io/nyu_edu/hitcounter:1.0
The push refers to repository [us.icr.io/nyu_edu/hitcounter]
48e4096c5a26: Pushed
487b2567edc8: Pushed
997671074b5b: Pushed
8898e0e667b9: Pushed
05c305d92486: Pushed
2ca008983975: Pushed
053be95103bc: Pushed
45a173e90fb4: Pushed
f5600c6330da: Pushed
1.0: digest: sha256:bbc05196e4e58573 size: 2204
```

Here is our new image in IBM Cloud

```
(venv) vagrant@kubernetes:~$ ic cr image-list
Listing images...
```

Repository	Tag	Digest	Namespace	Created	Size	Security status
us.icr.io/nyu_edu/hitcounter	1.0	bbc05196e4e5	nyu_edu	4 days ago	52 MB	No Issues
us.icr.io/nyu_edu/nyu-hitcounter	0.1.0	81090b3ea00a	nyu_edu	6 months ago	67 MB	2 Issues
us.icr.io/nyu_edu/nyu-hitcounter	latest	81090b3ed00d	nyu_edu	6 months ago	67 MB	2 Issues

OK

Deploy Hitcounter:1.0

- Create a Deployment

```
$ kubectl create deployment hitcounter --image=us.icr.io/nyu_edu/hitcounter:1.0
deployment.apps/hitcounter created
```

- Check that the deployment and pods were created

```
$ kubectl get deployments
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
my-nginx   1/1     1           1           3h42m
```

```
$ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
my-nginx-b49cf867-q7mjn      1/1     Running   0          3h42m
```

Deploy lab-docker

- Create a Service

```
$ kubectl expose deploy hitcounter --type=NodePort --port=8080
service/hitcounter exposed
```

- Check that the service was created

```
$ kubectl get service
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
hitcounter	NodePort	172.21.164.102	<none>	8080:31629/TCP	18s
kubernetes	ClusterIP	172.21.0.1	<none>	443/TCP	24h

Deploy lab-docker

- Create a Service

```
$ kubectl expose deploy hitcounter --type=NodePort --port=8080
service/hitcounter exposed
```

- Check that the service was created

This is the port it is listening on

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
hitcounter	NodePort	172.21.164.102	<none>	8080:31629/TCP	18s
kubernetes	ClusterIP	172.21.0.1	<none>	443/TCP	24h

Your App on IBM Cloud Kubernetes



173.193.111.140:32598

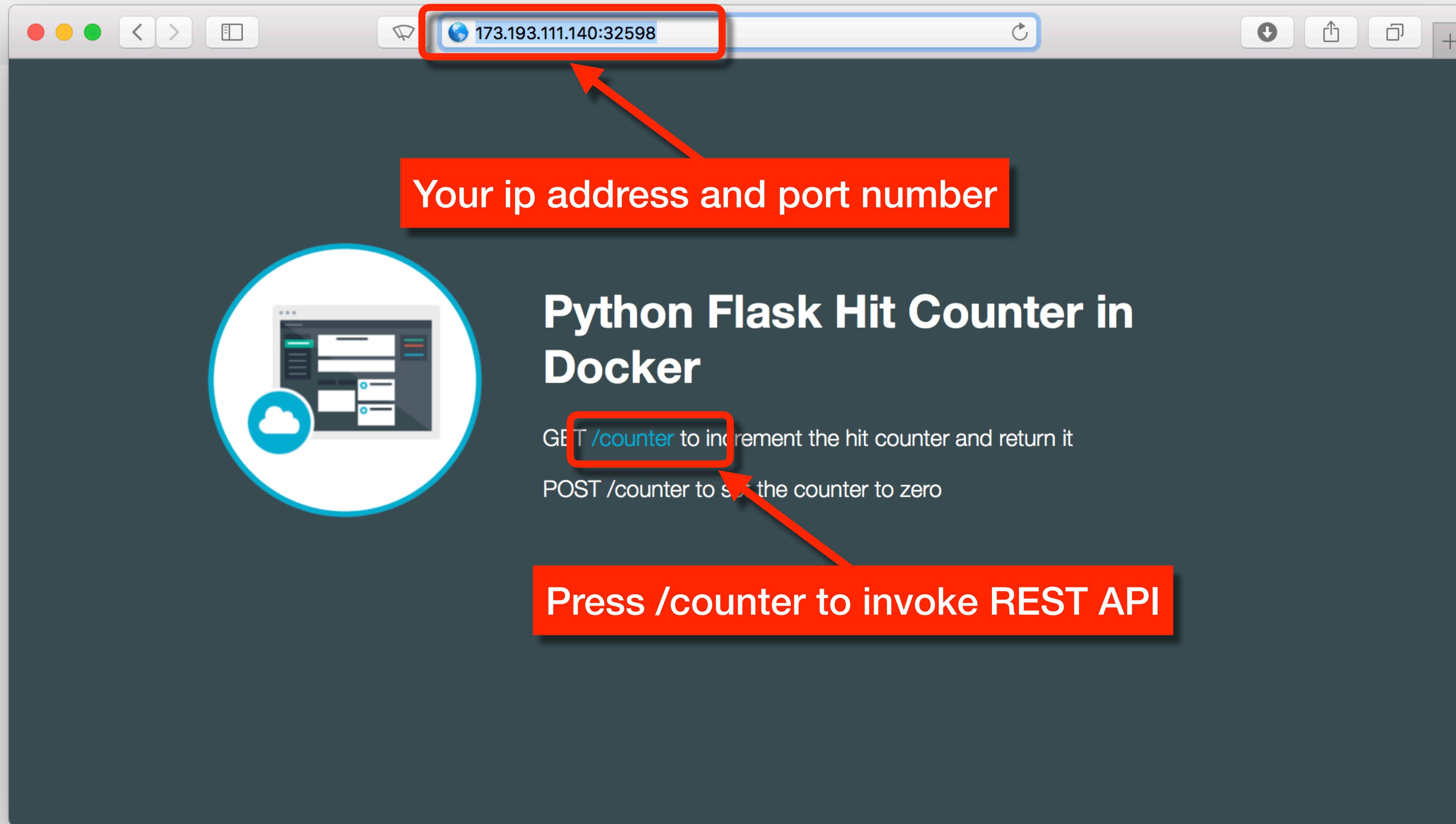
Your ip address and port number

Python Flask Hit Counter in Docker

GET `/counter` to increment the hit counter and return it
POST `/counter` to set the counter to zero

The screenshot shows a web browser window with a dark theme. The URL bar contains the IP address "173.193.111.140:32598". A red box highlights this URL, and a red arrow points from a red callout bubble containing the text "Your ip address and port number" to the highlighted URL.

Your App on IBM Cloud Kubernetes



Summary

You just created your first Kubernetes deployment

You learned how to create deployments

You learned how to expose deployments as services

You learned how to scale deployments

You can now deploy your microservices on any Kubernetes cloud

