Prerequisites

- For this workshop, you can use either the preconfigured VM environment or a user-supplied environment.

- For details on the VM option, see the file k8s-dev-setup-vm-environment-option.pdf in github.com/skilldocs/k8s-dev
  - For this option, you will need VirtualBox and the .ova file
    - » https://www.dropbox.com/s/d8bjycx07v9is4i/k8s-dev.ova?dl=0
      - OR -
    - » https://bclconf.s3.us-west-2.amazonaws.com/k8s-dev.ova

- For details on the user-supplied option, see the file k8s-dev-setup-user-supplied-environment-option.pdf in github.com/skilldocs/k8s-dev

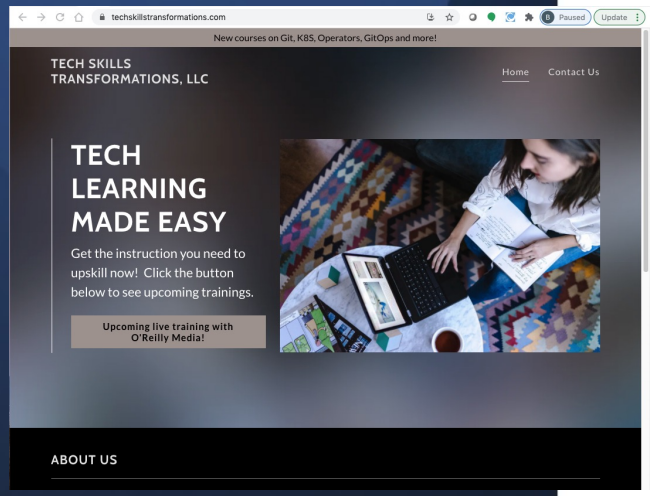
- You will also need the labs doc for the workshop from https://github.com/skilldocs/k8s-dev/blob/main/k8s-dev.pdf

Kubernetes for Developers Deep Dive

Brent Laster

Tech Skills Transformations

About me



- Founder, Tech Skills Transformations LLC
- R&D DevOps Director
- Global trainer – training (Git, Jenkins, Gradle, CI/CD, pipelines, Kubernetes, Helm, ArgoCD, operators)
- Author -
  - OpenSource.com
  - Professional Git book
  - Jenkins 2 – Up and Running book
  - Continuous Integration vs. Continuous Delivery vs. Continuous Deployment mini-book on O'Reilly
- https://www.linkedin.com/in/brentlaster
- @BrentCLaster
- GitHub: brentlaster

Professional Git Book

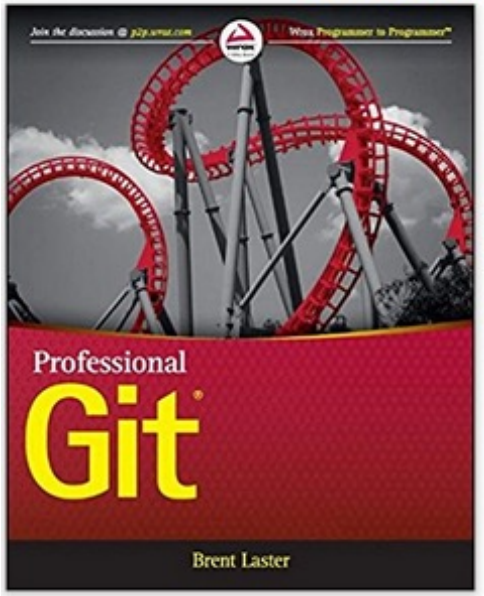
- Extensive Git reference, explanations,
- and examples
- First part for non-technical
- Beginner and advanced reference
- Hands-on labs

Professional Git 1st Edition

by Brent Laster (Author)

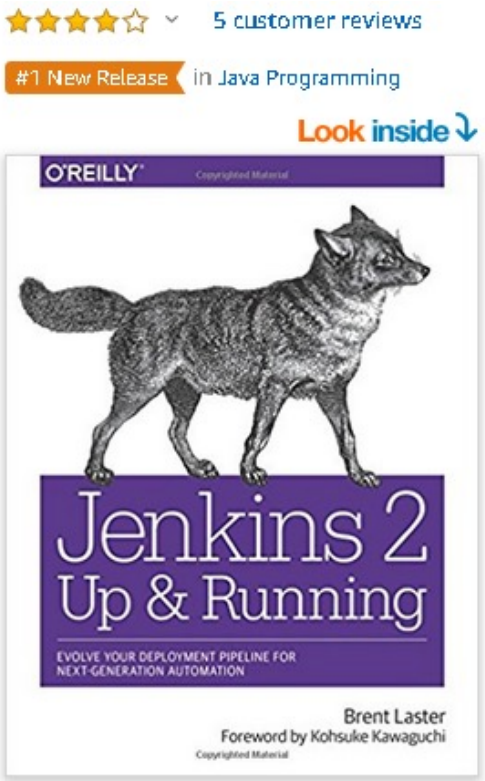
7 customer reviews

Look inside



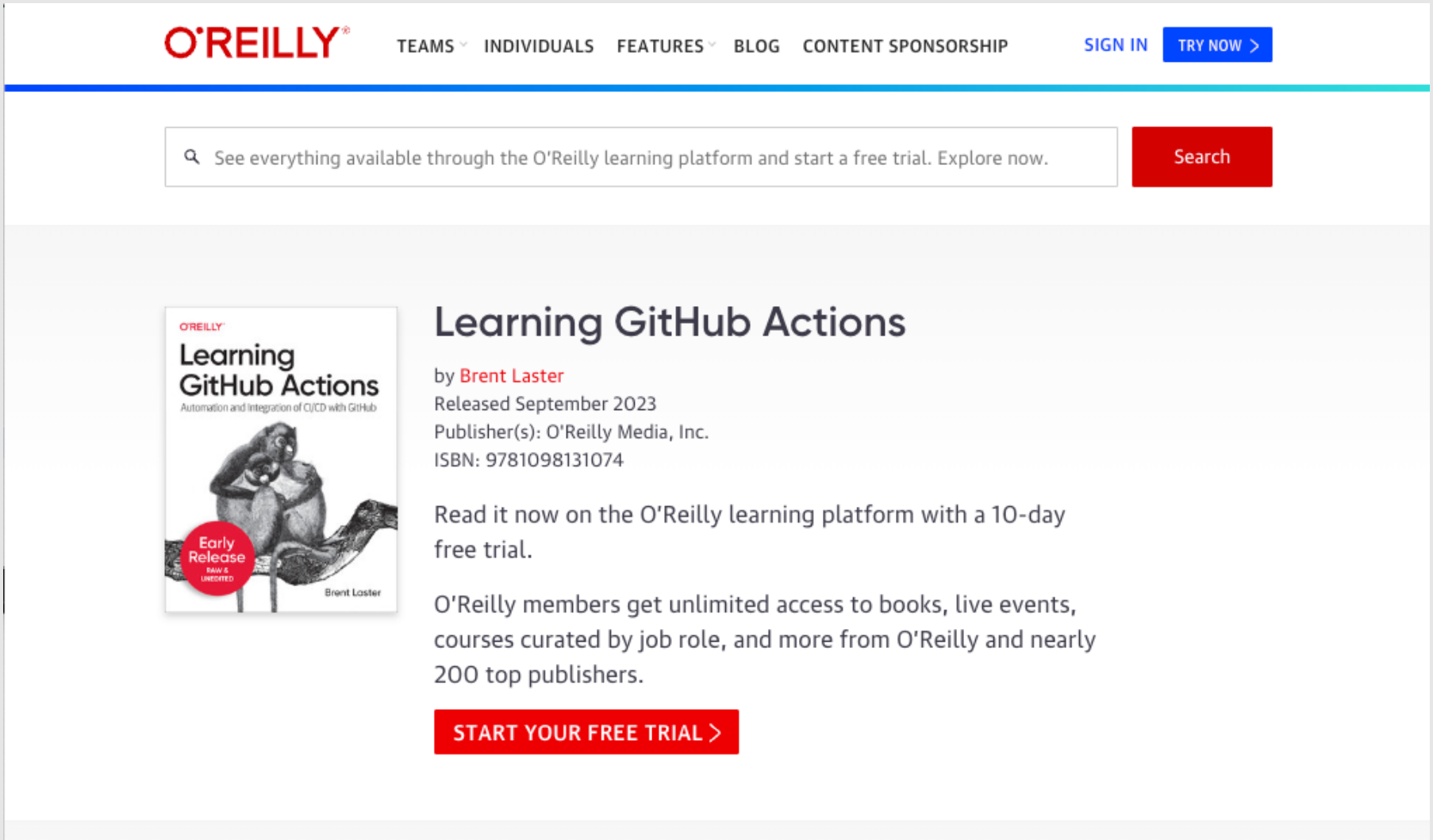
Jenkins 2 Book

- Jenkins 2 – Up and Running
- “It’s an ideal book for those who are new to CI/CD, as well as those who have been using Jenkins for many years. This book will help you discover and rediscover Jenkins.” ***By Kohsuke Kawaguchi, Creator of Jenkins***



Learning GitHub Actions - Early Release

- Early release - Chapters 1-6 on learning.oreilly.com



O'Reilly Training

April 26 & 27, 2021

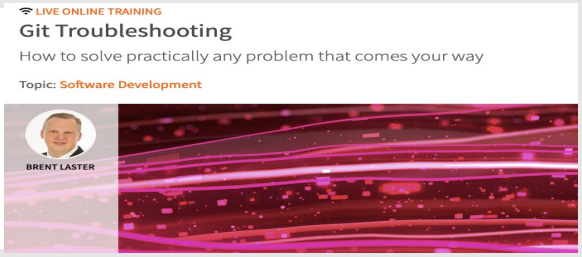
Git Fundamentals

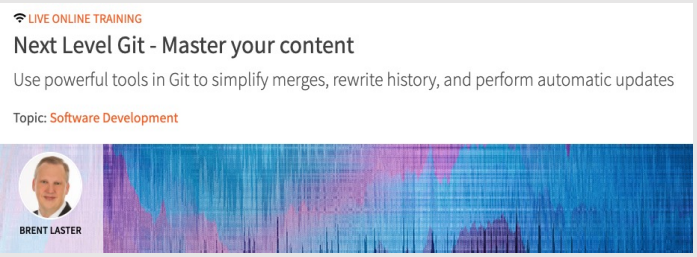
Join Brent Laster to gain the knowledge and skills you need to leverage Git to greatly simplify and speed up managing all of the changes in your source code. Once you ...

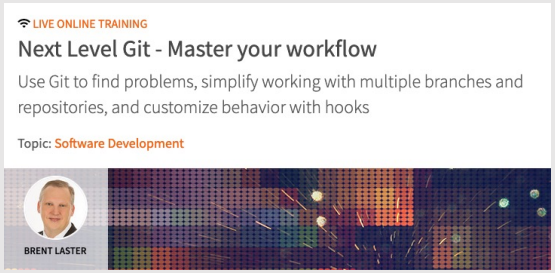
June 3, 10, 17 & 24, 2021

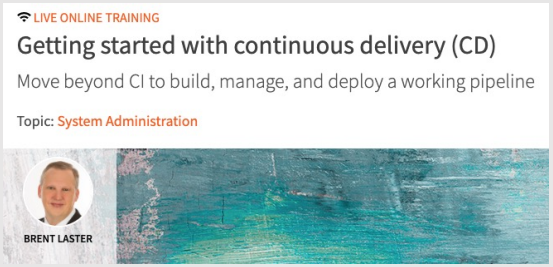
Git in 4 Weeks

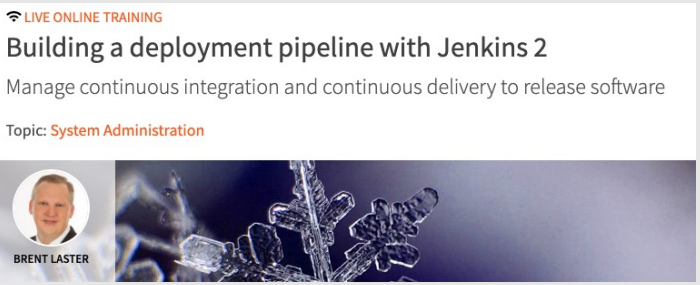
Join author, trainer, and DevOps director Brent Laster to learn how Git works—and discover all the things you can do with it. Over four sessions, Brent walks you through everything you ...







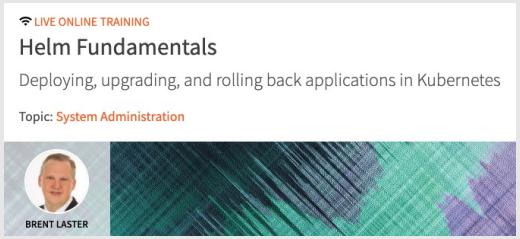


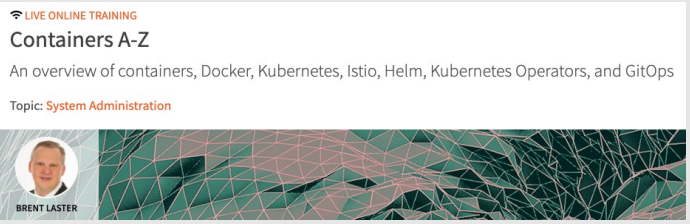


June 28, 2021

Troubleshooting Kubernetes

In this 3-hour course, global trainer, author, and DevOps director Brent Laster will show you how to respond to the most common problem situations you may encounter with Kubernetes. You'll learn ...





May 24, 2021

Continuous Delivery in Kubernetes with ArgoCD

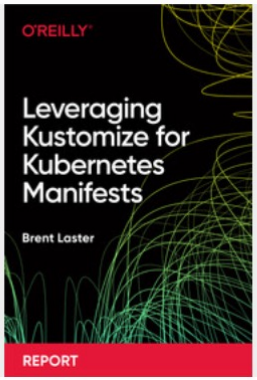
Join expert Brent Laster to explore GitOps and learn how to use Argo CD to implement GitOps in your Kubernetes deployments. APAC time friendly - You're a Kubernetes admin who wants ...

May 17, 2021

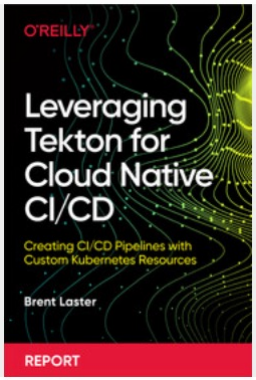
Building a Kubernetes Operator

Join expert Brent Laster to learn how the Operator pattern helps address these kinds of situations by allowing you to create custom controllers that extend the functionality of the Kubernetes API ...

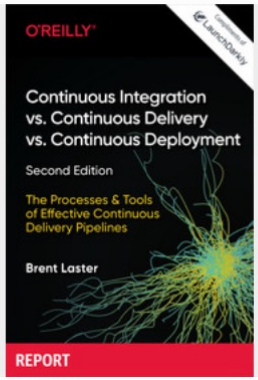
Other References



Leveraging Kustomize for Kubernetes Manifests



Leveraging Tekton for Cloud Native CI/CD



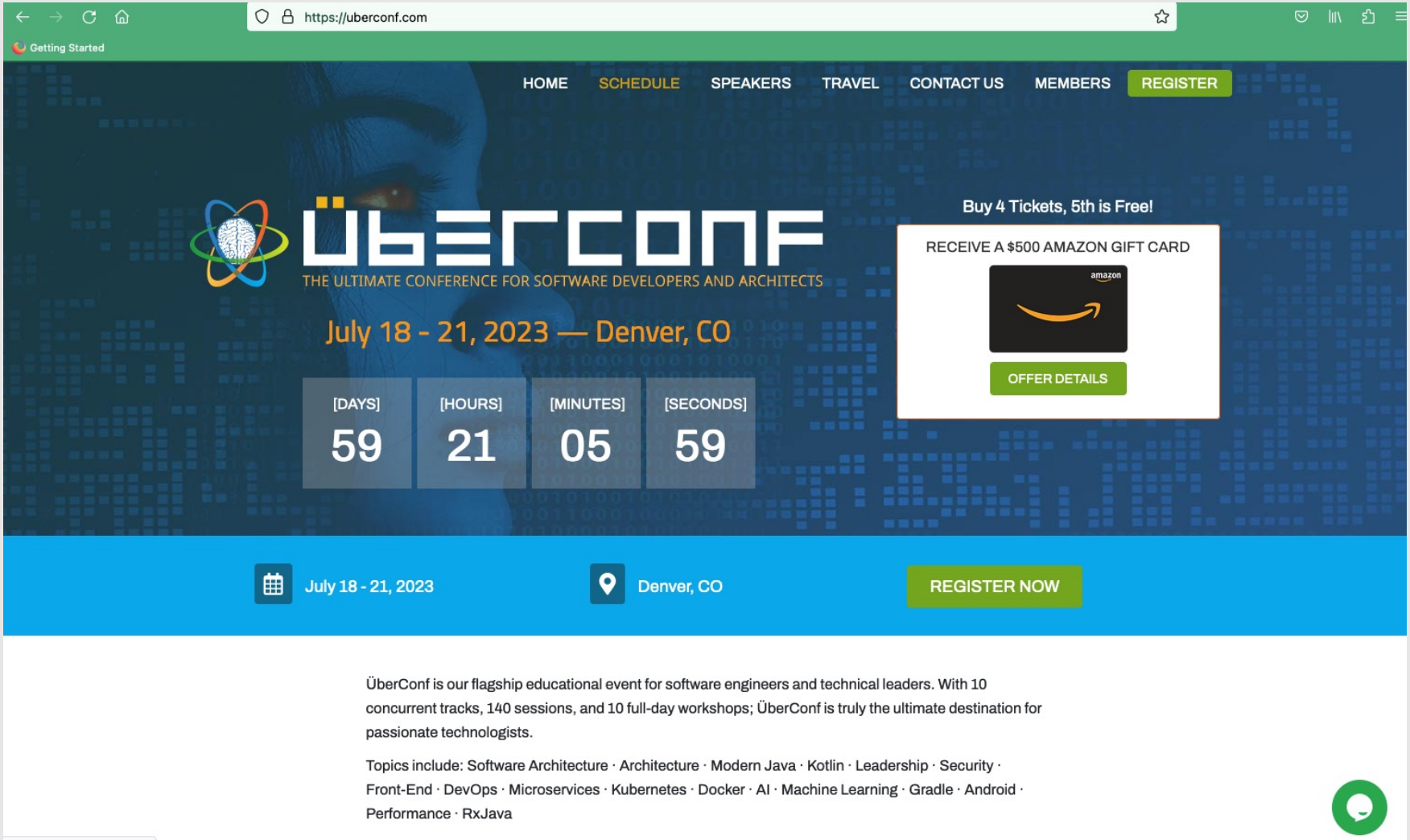
Continuous Integration vs. Continuous Delivery vs. Continuous Deployment, 2nd Edition

Continuous Integration vs. Continuous Delivery vs. Continuous Deployment

Navigating the Continuous Delivery Pipeline

NFJS Training

Webinar sponsored by Uberconf



Containers Overview

What are Containers?

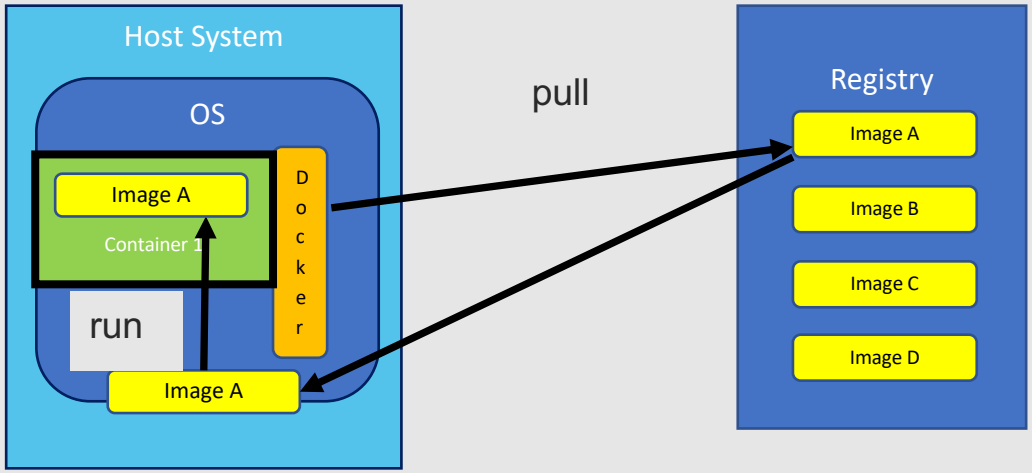
- A container is a standard unit of software that functions like a fully provisioned machine installed with all the software needed to run an application.
- It's a way of packaging software so that applications and their dependencies have a self-contained environment to run in, are insulated from the host OS and other applications - and are easily ported to other environments.
- A container is NOT a VM. A container leverages several features of the Linux OS to "carve out" a self-contained space to run in.

Containers are running instances of images.
Images define what goes into a container.
Containers are built from images.

Benefits of containers

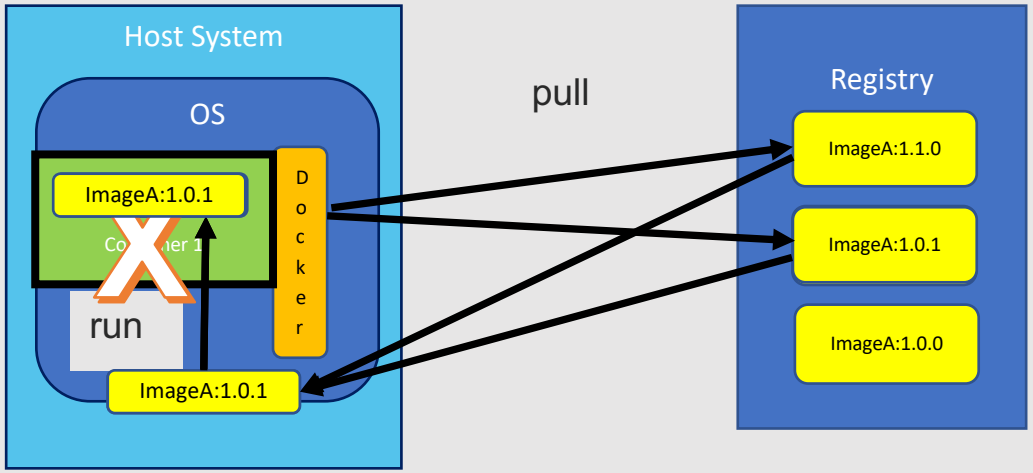
Easy to startup

- **Get image**
- **Run image as a container**



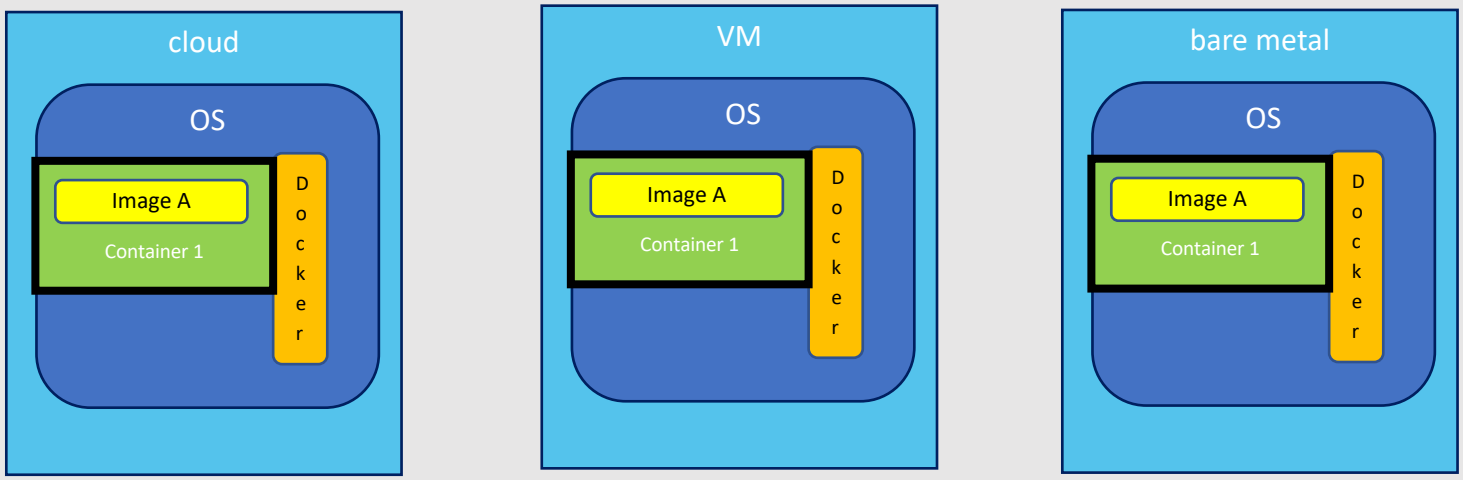
Quick, easy rollbacks

- **Get image**
- **Run image as a container**



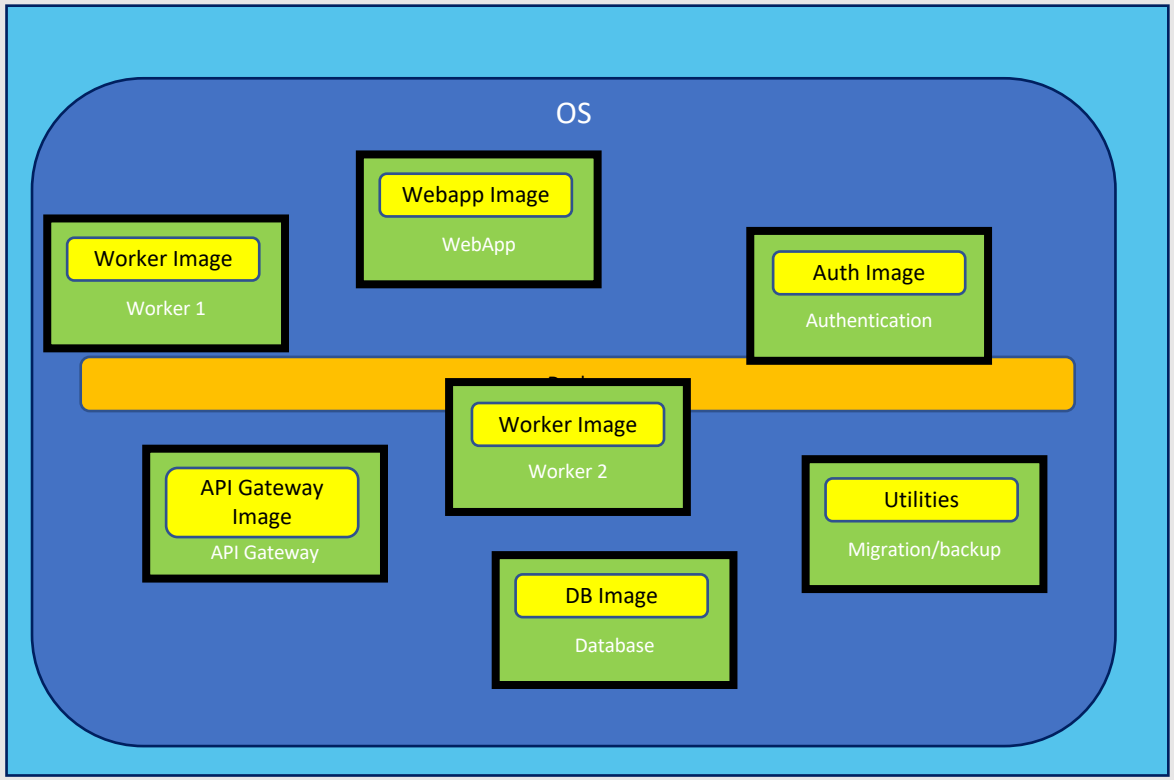
Portability

- **Run nearly anywhere without rebuilding**
- **Works same way since container is the environment**



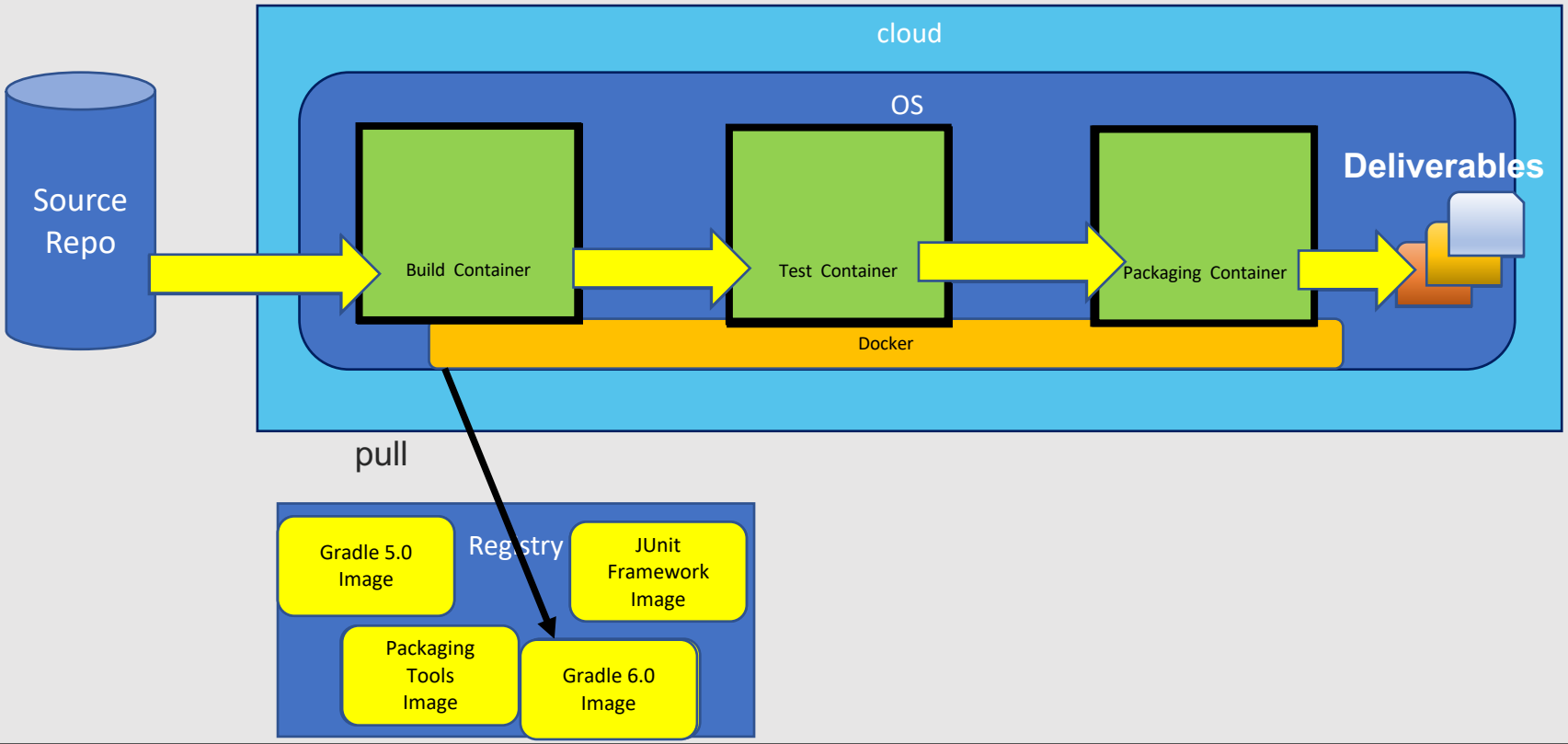
Supports Distributed/Decoupled Architecture

- **Loosely coupled services**
- **Removes need for monolithic stacks**



Agility

- **Define containers well-suited for a specific task**
- **Spin up/down containers quickly/easily to address needs**
  - **such as devops workflow**



How do we work with multiple containers?

- Docker compose tool allows us to start containers together
- Uses service specifications in docker-compose.yml file
- Simplifies running multiple containers that need to be “linked” together

- Composing containers together is essentially a 3-step process
  - Create needed Dockerfiles for each part of app
  - Define services that make up the app in a docker-compose.yml file
  - Run docker-compose up and it starts and runs your app

- Docker run
  - Could also use docker run command in extended form with –link option

- Docker stack
  - Command in Docker CLI - lets you mange cluster of containers via Swarm (competitor to Kubernetes)
  - Can also leverage docker-compose.yml file

- Kubernetes or Kubernetes Environments

```
roar-web-container:
    image: roar-web
    ports:
        - "8089:8080"
    links:
        - "roar-db-container:mysql"
roar-db-container:
    image: roar-db
    ports:
        - "3308:3306"
    environment:
        MYSQL_USER: admin
        MYSQL_PASSWORD: admin
        MYSQL_DATABASE: registry
        MYSQL_ROOT_PASSWORD: root+1
```

Kubernetes Overview

What is Kubernetes?

- A portable, extensible platform for managing containerized workloads and services (cluster orchestration system)
- Name derived from Greek for helmsman, thus the icon
- Frequently abbreviated as “k8s”
- Formerly known as “borg” – an internal Google project
- Open-sourced by Google in 2014

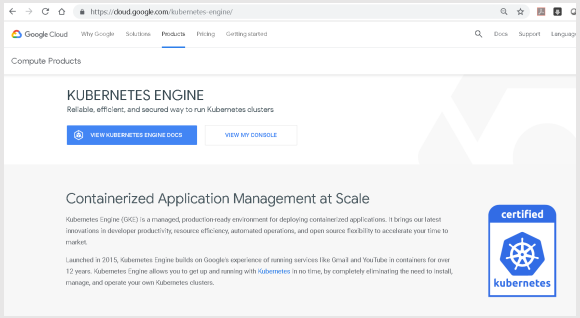
- Groups containers that make up an application into logical units for easy management and discovery.
- Goal is to provide a robust platform for running many containers.
- Allows automation of deployment, scaling, and managing containerized workloads.
- Kubernetes provides you with a framework to run distributed systems (of containers) resiliently.
- Takes care of
  - scaling requirements
  - failover
  - deployment patterns

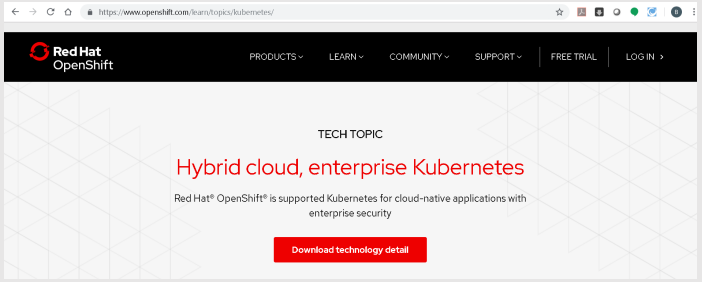
So how do we think about this?

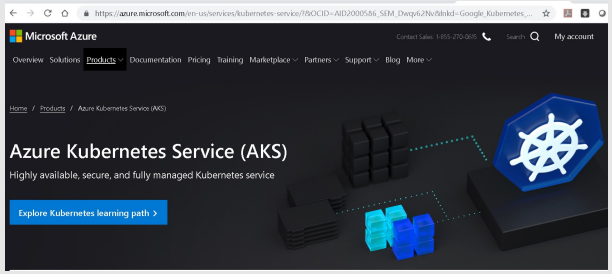
- Analogy: Datacenter for containers
  - If we think of images/containers as being like computers we stage and use
  - We can think of Kubernetes as being like a datacenter for those containers
  - Main jobs of datacenter
    - » Provide systems to service needs (regardless of the applications)
    - » Keep systems up and running
    - » Add more systems / remove systems depending on load
    - » Deal with systems that are having problems
    - » Deploy new systems when needed
    - Provide simple access to pools of systems
    - Etc..

Kubernetes is everywhere

- Has effectively “won the war” over other competitors
  - Docker swarm
  - Mesos
- Cloud providers all endorse it and provide ways to get a Kubernetes cluster
- Implementations on other enterprise platforms







Kubernetes Features

- Service discovery and load-balancing
- Automated rollouts and rollbacks
- Storage orchestration
- Batch execution
- Self healing
- Secret and configuration management
- Horizontal scaling
- Automatic bin packing

K8s Quick Terminology

- Cluster - an HA set of computers coordinated by k8s to work as a unit.
- Pods – object that contains and manages one or more containers and any attached volumes
- Service – abstraction that groups together pods based on identifiers called labels (or other characteristic)
- Deployment – defines a stateless app with a set number of pod replicas (scaled instances)
- Ingress – resource that lets cluster applications be exposed to external traffic
- Namespace - a logical area that groups k8s items like pods

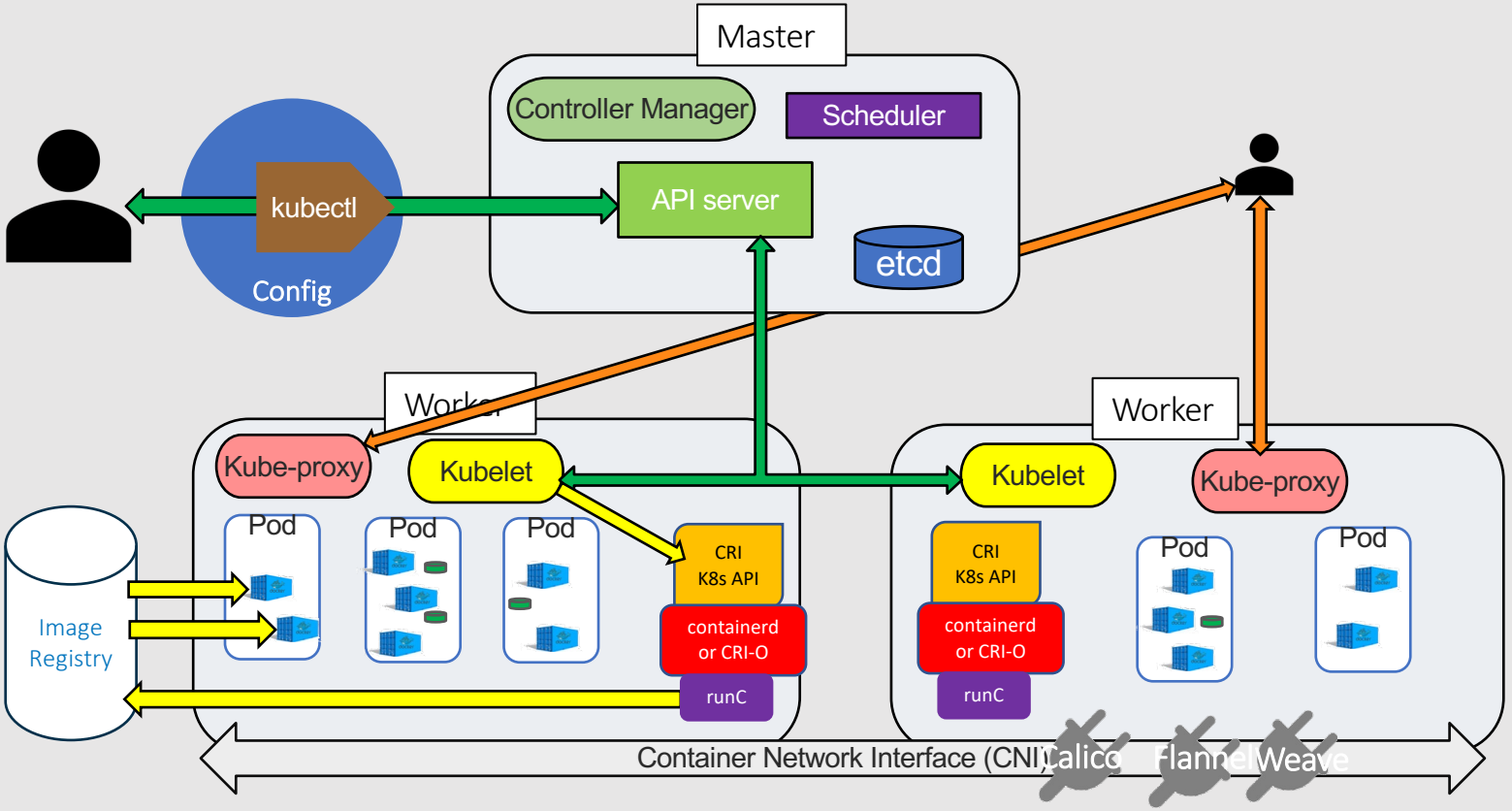
Cluster Overview

cluster = HA set of computers coordinated by K8s to work as a unit

K8s abstractions allow deploying containers w/o requiring tying to specific host

K8s automates distribution & scheduling of containers efficiently

Nodes are either master (coordinate the work) or worker (run the applications)

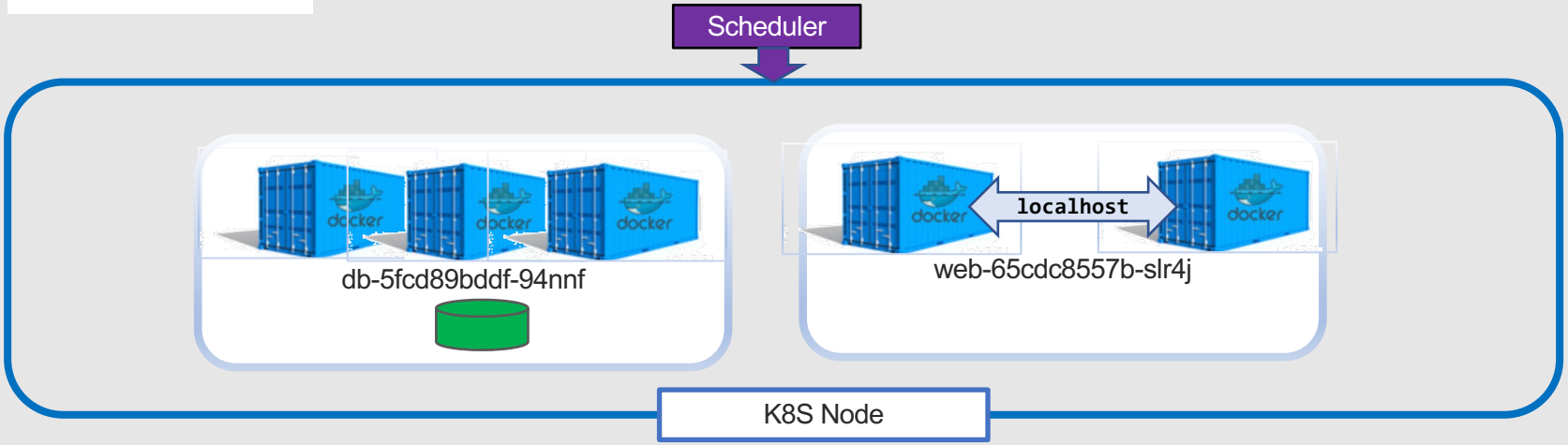


Pods

smallest deployable unit in Kubernetes

represents a group of 1 or more containers and any shared resources, such as storage volumes

scheduled on nodes by scheduler process



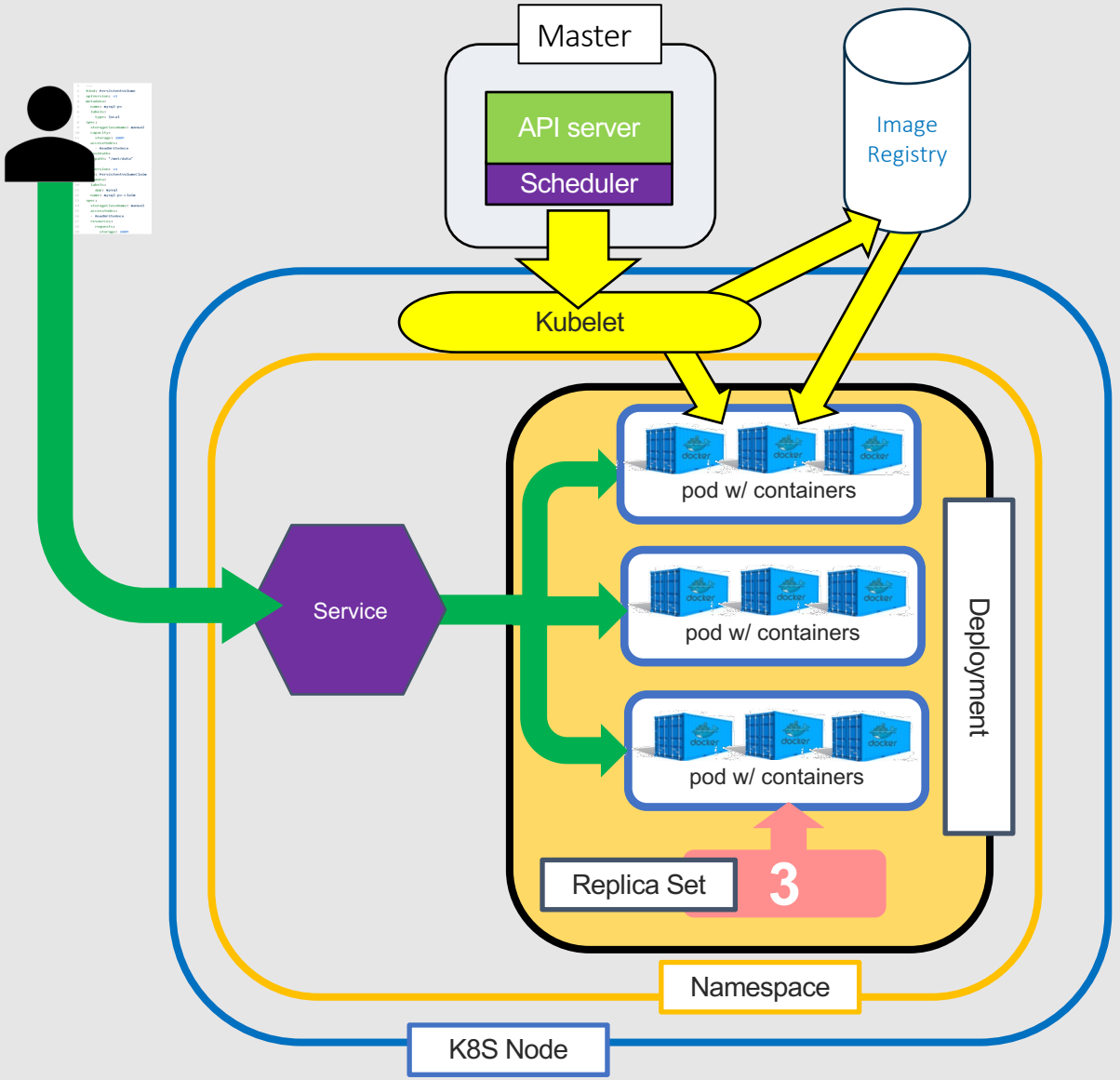
nodes in same pod can also talk to each other over "localhost"

containers should only be scheduled together if tightly coupled and need to share resources

can use node "selector" to specify node to run on

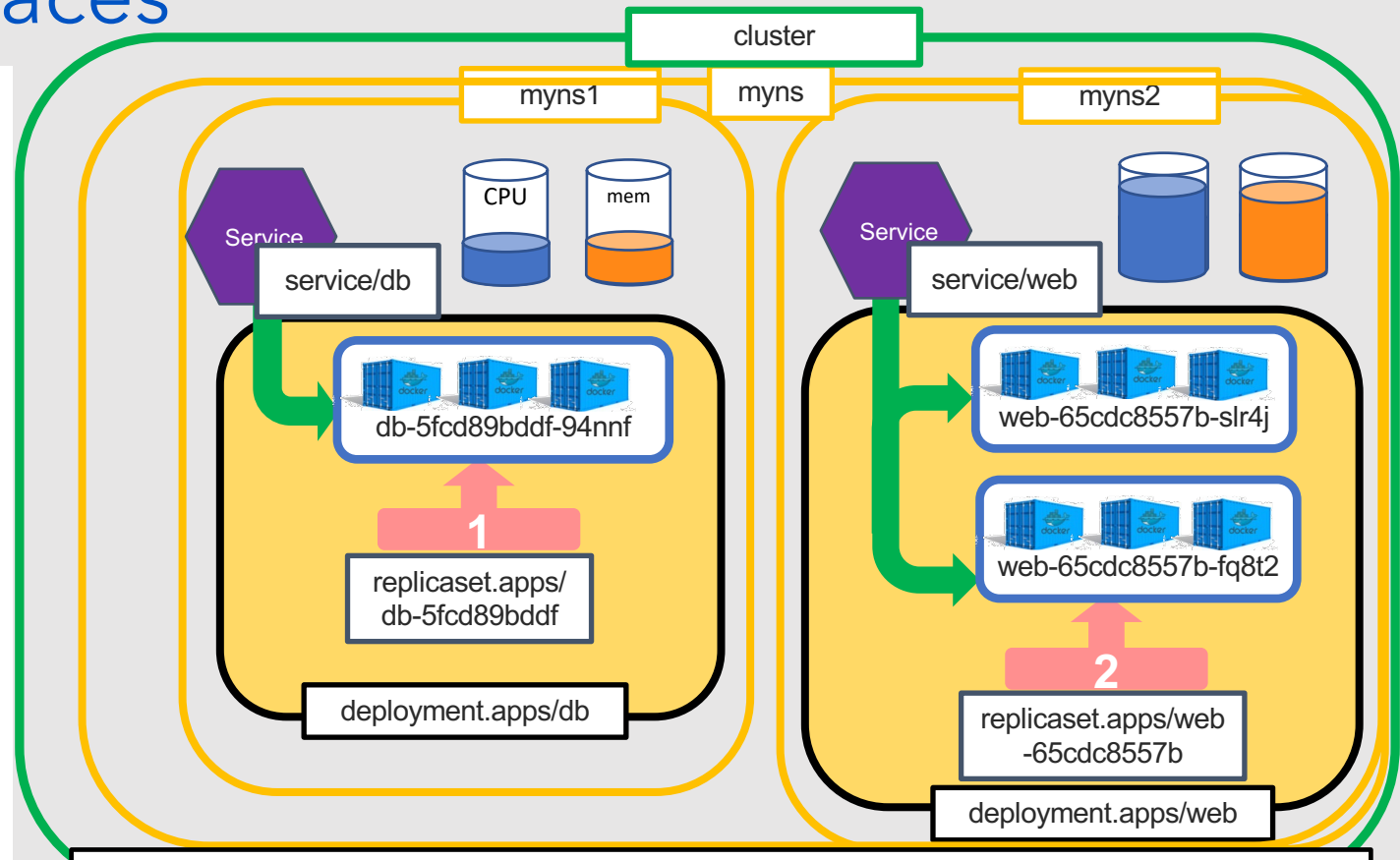
How are containers organized on K8s?

- K8s clusters have nodes
- Manifests (specs) define the objects
- Containers are created from images
- Pods wrap containers
- Pods are scaled (replicated) by deployments
- Pods are front-ended by services
- Namespaces group objects like pods, services, deployments together



Namespaces

- Logical, unique working area within a cluster
- Partitions cluster's resources
- Resources are associated/scoped to a namespace
- Namespaces can have assigned quotas for resource use/limits
- All resources within a namespace must be unique in name (Kubernetes can generate unique suffixes for similar resources)
- Namespace "default" is default
- Default namespace is the one used if another is not specified
- Object is abbreviated as "-n" when filtering by one
- Can set context to change current one




```
$ kubectl get pods -n myns
NAME                     READY   STATUS    RESTARTS   AGE
db-5fcd89bddf-94nnf      1/1     Running   0          4h18m
web-65cdc8557b-slr4j     1/1     Running   0          4h1m
web-65cdc8557b-fq8t2     1/1     Running   0          4h1m
```

```
NAME                     READY   STATUS    RESTARTS   AGE
pod/db-5fcd89bddf-94nnf  1/1     Running   0          4h23m
```

Kubectl Command Line

- Kubectl is the command line interface for running commands against k8s clusters
- Different pronunciations – usually either “kube control” or “kube cuttle” or “kube cuddle”
- Uses a “kube config” file to understand which cluster to work against and any necessary configuration information (kube config file may be sharable to other instantiations)
- Basic syntax is : kubectl [command] [TYPE] [NAME] flags
  - Examples:
    - **kubectl get** - get basic information about existence of objects
    - **kubectl apply** – create/update applications from files defining resources
    - **kubectl create** - create a new object
    - **kubectl describe** – get detailed info about current state of an object
    - **kubectl delete** – delete an object
- Resources can be referred to with resource/name syntax
  - kubectl describe pod/mypod-name
- Options can be in different locations
  - kubectl -n <namespace> get all
  - kubectl get all –n <namespace>
- Can use different forms of resource type names
  - singular, plural, abbreviations (ex: pod, pods, po)
- Abbreviations exist for most resources
  - po = pod, svc = service, deploy = deployment, etc.

```
diyuser3@training1:~$ kubectl get service -n roar2
NAME       TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
mysql      ClusterIP   10.109.196.123   <none>        3306/TCP
roar-web   NodePort    10.106.132.164   <none>        8089:30318
diyuser3@training1:~$
diyuser3@training1:~$ kubectl -n roar2 get svc
NAME       TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
mysql      ClusterIP   10.109.196.123   <none>        3306/TCP
roar-web   NodePort    10.106.132.164   <none>        8089:30318
diyuser3@training1:~$
diyuser3@training1:~$ kubectl get -n roar2 all | grep service


service/mysql      ClusterIP   10.109.196.123   <none>        33
service/roar-web   NodePort    10.106.132.164   <none>        80
```

kubernetes

Configs and Contexts

- kubeconfig files
  - organizes info about clusters, users, namespaces, and authentication mechanisms
  - used by kubectl to find info needed to pick cluster and talk to cluster's API server
  - default name/location is $HOME/.kube/config
  - can override default with KUBECONFIG env var or --kubeconfig flag on command line

- context
  - in kubeconfig file, groups together access parameters under name
  - access parameters are cluster, namespace (optional), and user
  - command line access to contexts is through "kubectl config <context>"

```
% kubectl config
Modify kubeconfig files using subcommands like "kubectl config set
current-context my-context"


Available Commands:
  current-context Display the current-context
  delete-cluster  Delete the specified cluster from the kubeconfig
  delete-context  Delete the specified context from the kubeconfig
  delete-user     Delete the specified user from the kubeconfig
  get-clusters    Display clusters defined in the kubeconfig
  get-contexts    Describe one or many contexts
  get-users       Display users defined in the kubeconfig
  rename-context  Rename a context from the kubeconfig file
  set             Set an individual value in a kubeconfig file
  set-cluster     Set a cluster entry in kubeconfig
  set-context     Set a context entry in kubeconfig
  set-credentials Set a user entry in kubeconfig
  unset           Unset an individual value in a kubeconfig file
  use-context     Set the current-context in a kubeconfig file
  view            Display merged kubeconfig settings or a specified
kubeconfig
file

Usage:
  kubectl config SUBCOMMAND [options]
```

YAML and K8S specifications

- YAML is a type of markup language to define Kubernetes specs for resources
- Stored in .yaml or .yml text file
- Kubectl apply can take such a file as input and update cluster based on specs
  - Turns yaml specs into resources/objects running in cluster
- Kubectl get –o yaml can be used to dump out spec and status as yaml from running object

- Conventional block format uses a hyphen + space to denote a new item in a list

```
--- # Favorite movies
- Casablanca
- North by Northwest
- The Man Who Wasn't There
```

- Keys are separated from values by a colon + space; indented blocks use indentation and newlines to separate key-value pairs
- Strings do not (generally) require quotation marks
- Data structure hierarchy is maintained by outline indentation

```
--- # Indented Block
  name: John Smith
  age: 33
```

```
---
receipt:     Oz-Ware Purchase Invoice
date:        2012-08-06
customer:
    first_name:   Dorothy
    family_name:  Gale

items:
    - part_no:   A4786
      descrip:   Water Bucket (Filled)
      price:     1.47
      quantity:  4

    - part_no:   E1628
      descrip:   High Heeled "Ruby" Slippers
      size:      8
      price:     133.7
      quantity:  1
```

Understanding Kubernetes Objects

- To work with k8s objects, you use the k8s API
  - Kubectl command-line tool makes calls to API for you
  - Could also use k8s api client libraries
- K8s objects are persistent entities in the k8s system.
- K8s uses these entities to represent the state of the cluster.
  - They can describe:
    - What application containers are running and on which nodes.
    - Resources available to applications.
    - Policies around how those applications behave.
- Kubernetes object is “record of intent”
  - After creation, k8s will work to ensure object exists
  - Creating an object declares what you want cluster workload to look like
  - Known as cluster’s “desired state”
- Declarative model vs. imperative model

Defining a Kubernetes Object in text

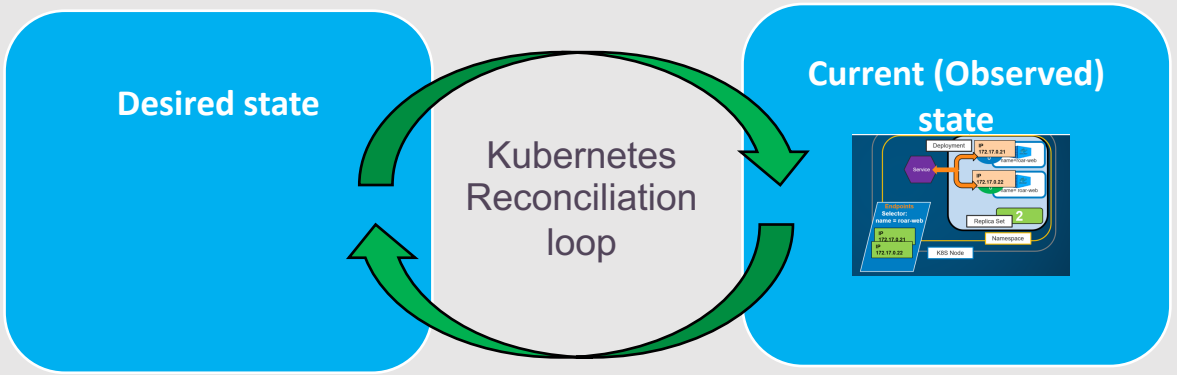
- When creating an object in k8s, have to provide
  - object spec to describe desired
  - basic info, such as a name
- K8s API expects info as JSON in body of request
- But, usually provide it from YAML file
- Kubectl command line converts to JSON for you
- Required fields
  - apiVersion – which version of the k8s API is being used to create the object
  - kind - what kind of object to create
  - metadata - data that helps uniquely identify the object, such as name, UID, namespace (optional)
  - Ojbect's spec
    - Format different for every k8s object
    - Contains nested fields specific to the object
    - Can find details on specs in the API reference

https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: roar-web
  labels:
    name: roar-web
  namespace: roar
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: roar-web
    spec:
      containers:
      - name: roar-web
        image: localhost:5000/roar-web:v1
        ports:
        - name: web
          containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: roar-web
  labels:
    name: roar-web
  namespace: roar
spec:
  type: NodePort
  ports:
  - port: 8089
```

Kubernetes is a Desired-State System

- User supplies desired state via declaring it in manifests
- Kubernetes works to balance the current state and the desired state
  - Desired state – what you want your production environment to be
  - Current (observed) state – current status of your production environment



Kubernetes Objects - Spec and Status

- Every running k8s object includes two nested object fields
  - spec
    - User-provided
    - Describes desired state for the object
  - status
    - Provided and updated by k8s system
    - Describes the actual state of the object

```
diyuser3@training1:~$ kubectl edit -n roar2 pod roar-web-74bb47bdb8-56l4n
```

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2019-07-21T20:04:34Z"
  generateName: roar-web-74bb47bdb8-
  labels:
    app: roar-web
    pod-template-hash: 74bb47bdb8
  name: roar-web-74bb47bdb8-56l4n
  namespace: roar2
  ownerReferences:
  - apiVersion: apps/v1
    blockOwnerDeletion: true
    controller: true
    kind: ReplicaSet
    name: roar-web-74bb47bdb8
    uid: 89d00855-abf0-11e9-b30c-080027c4188a
  resourceVersion: "346085"
  selfLink: /api/v1/namespaces/roar2/pods/roar-w
  uid: c2b50783-abf2-11e9-b30c-080027c4188a
spec:
  containers:
  - image: localhost:5000/roar-web:v2
    imagePullPolicy: Always
    name: roar-web
    ports:
    - containerPort: 8080
      name: web
      protocol: TCP
```

```
    secret:
      defaultMode: 420
      secretName: default-token-dxx7t
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2019-07-21T20:04:
    status: "True"
    type: Initialized
  - lastProbeTime: null
    lastTransitionTime: "2019-07-21T20:04:
    status: "True"
    type: Ready
  - lastProbeTime: null
    lastTransitionTime: "2019-07-21T20:04:
    status: "True"
    type: ContainersReady
  - lastProbeTime: null
    lastTransitionTime: "2019-07-21T20:04:
    status: "True"
    type: PodScheduled
  containerStatuses:
  - containerID: docker://41c2d31ef46b8ea1
    image: localhost:5000/roar-web:v2
    imageID: docker-pullable://localhost:5
    lastState:
      terminated:
        containerID: docker://c0ddd9686971
        exitCode: 143
```

kubectl edit

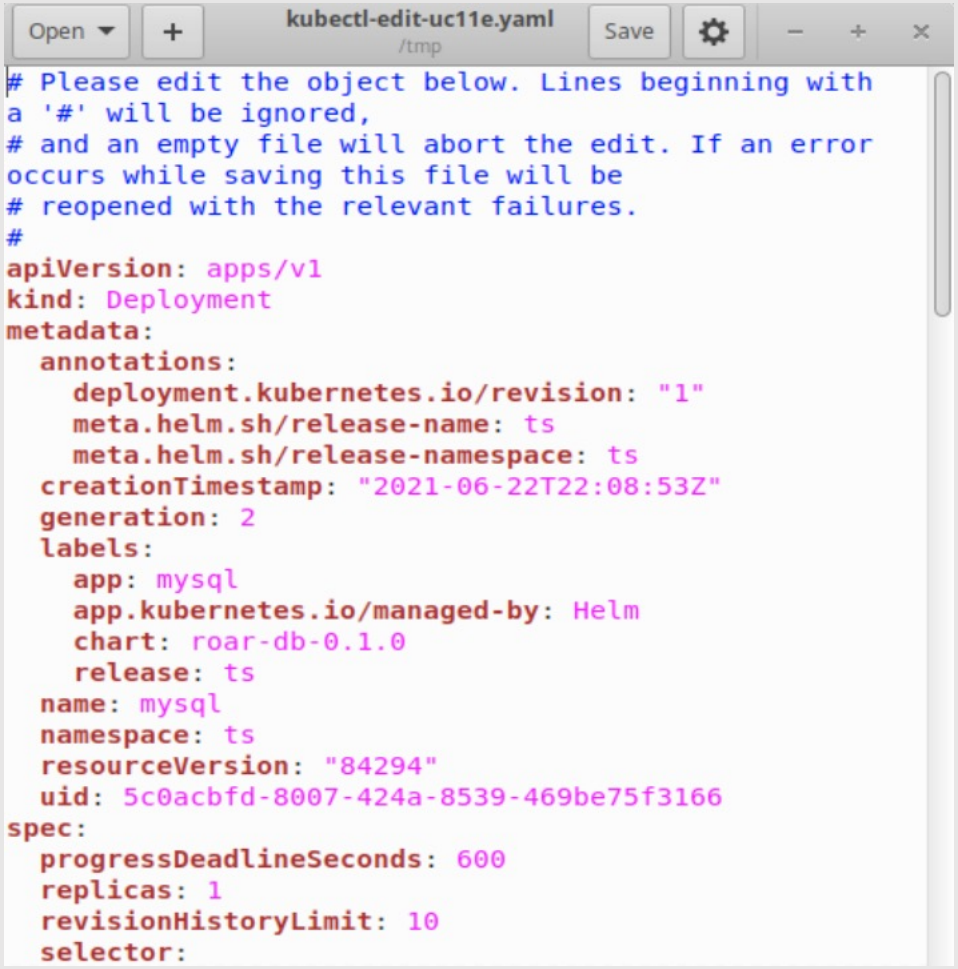
- Allows editing of resources in place with defined editor

Examples:
  # Edit the service named 'docker-registry':
  kubectl edit svc/docker-registry

  # Use an alternative editor
  KUBE_EDITOR="nano" kubectl edit svc/docker-registry

  # Edit the job 'myjob' in JSON using the v1 API format:
  kubectl edit job.v1.batch/myjob -o json

  # Edit the deployment 'mydeployment' in YAML and save the modified config in its annotation:
  kubectl edit deployment/mydeployment -o yaml --save-config




```
$ k edit deploy/mysql

deployment.apps/mysql edited
```

Labels

- Labels are the main mechanisms used in Kubernetes to select/organize/associate obects
- Label is a key-value pair without any pre-defined meaning
- Kubectl get has option “--show-labels” to show labels from objects

```
$ kubectl get pods -n istio1 --show-labels

NAME                            READY   STATUS    RESTARTS   AGE     LABELS
roar-current-6c75d49d78-bztfd   3/3     Running   6          6d21h   app=roar,version=current
roar-new-559997dc54-7qhjh       3/3     Running   6          6d21h   app=roar,version=new
```

- Can add label to object with “kubectl label” command

```
$ kubectl label deploy roar-new -n istio1 type=experimental
deployment.extensions/roar-new labeled
```

- We can filter based on a label
  - --selector is long form of option
  - -l is short form
- Most k8s objects support set-based selectors (choosing which items based on a set to se… …n do:

```
$ kubectl get pods -n istio1 -l 'version in (current, new)'

NAME                            READY   STATUS    RESTARTS   AGE
roar-current-6c75d49d78-bztfd   3/3     Running   6          6d21h
roar-new-559997dc54-7qhjh       3/3     Running   6          6d21h
```

- Labels can apply to other objects, such as nodes and services and be used in other operations

```
$ kubectl delete pods -l 'version in (current, new)'
```

- Types: “Equality-based” (=, !=) “Set-based” (in ())

```
apiVersion: v1
kind: Pod
metadata:
  name: roar-current
  labels:
    app: roar
    version: current
spec:
  containers:
  - image: localhost:5000/roar-web:v1
    imagePullPolicy: Always
    name: roar-web
    ports:
    - containerPort: 8080
      name: roar-web
      protocol: TCP
    resources: {}
```

```
apiVersion: v1
kind: Pod
metadata:
  name: roar-new
  labels:
    app: roar
    version: new
spec:
  containers:
  - image: localhost:5000/roar-web:v2
    imagePullPolicy: Always
    name: roar-web
    ports:
    - containerPort: 8080
      name: roar-web
      protocol: TCP
    resources: {}
```

Deployments

- Spinning Up Pods
  - A deployment is like a pod “supervisor”.
  - After initiating a deployment, we have the deployment, the replicaset, and the pods
  - Naming of the pods and replicaset are derived from the deployment name

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: roar-web
  labels:
    name: roar-web
  namespace: roar
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: roar-web
    spec:
      containers:
      - name: roar-web
        image: localhost:5000/roar-web:v1
        ports:
        - name: web
          containerPort: 8080
```

```
diyuser3@training1:~$ k get deploy -n roar  | grep web
roar-web   1/1     1            1           6d23h
diyuser3@training1:~$ k get rs  -n roar  | grep web
roar-web-f95cf6574   1         1         1       6d23h
diyuser3@training1:~$ k get pod  -n roar  | grep web
roar-web-f95cf6574-ndt4k   1/1     Running   0          9m44s
```

Deployments

- Updating Pods
  - What happens if we make a significant change (update image to v2) and apply it to the running instance?
  - kubectl apply –f roar-complete.yaml –n roar
  - Pods with old versions are terminated and new ones created with the updated version

```
$ kubectl get pods -n roar -w

NAME                        READY   STATUS    RESTARTS   AGE
mysql-8559ccc47d-crvhv      1/1     Running   0          4h22m
roar-web-f95cf6574-wcm7z    1/1     Running   0          15s
roar-web-7bccc45c79-pp9hs   0/1     Pending   0          0s
roar-web-7bccc45c79-pp9hs   0/1     Pending   0          0s
roar-web-f95cf6574-wcm7z    1/1     Terminating   0          50s
roar-web-7bccc45c79-pp9hs   0/1     ContainerCreating   0          0s
roar-web-f95cf6574-wcm7z    0/1     Terminating         0          53s
roar-web-7bccc45c79-pp9hs   1/1     Running             0          3s
```

  - And new replicaset is created

```
$ k get rs -n roar

NAME                  DESIRED   CURRENT   READY   AGE
mysql-8559ccc47d      1         1         1       6d22h
roar-web-7bccc45c79   1         1         1       55s
roar-web-f95cf6574    0         0         0       7d5h
```

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: roar-web
  labels:
    name: roar-web
    type: experimental
  namespace: roar
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: roar-web
    spec:
      containers:
      - name: roar-web
        image: localhost:5000/roar-web:v2
        ports:
        - name: web
          containerPort: 8080
```

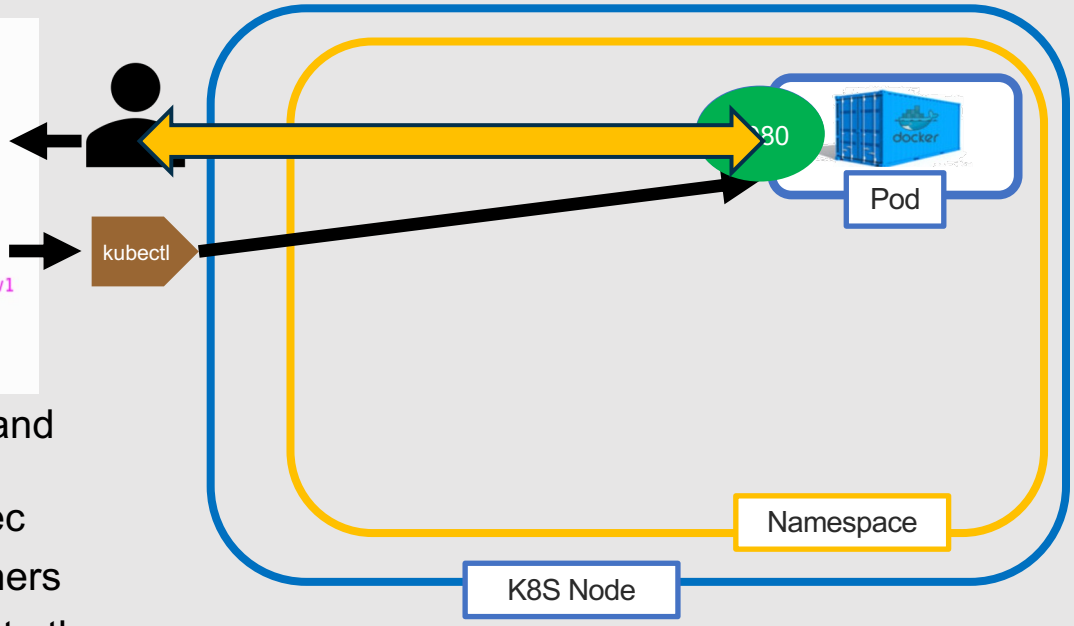
Lab 1 - Exploring and Deploying into Kubernetes

Purpose: In this lab, we'll start to learn about Kubernetes and its object types, such as nodes and namespaces. We'll also deploy a version of our app that has had Kubernetes yaml files created for it. And we'll see how to do some simple debugging when Kubernetes deployments don't go as planned.

Deployments vs. pods-only

- Using Pods without Deployments

```
apiVersion: v1
kind: Pod
metadata:
  name: roar-web
  labels:
    name: roar-web
  namespace: roar
spec:
  containers:
  - name: roar-web
    image: localhost:5000/roar-web:v1
    ports:
    - name: web
      containerPort: 8080
```

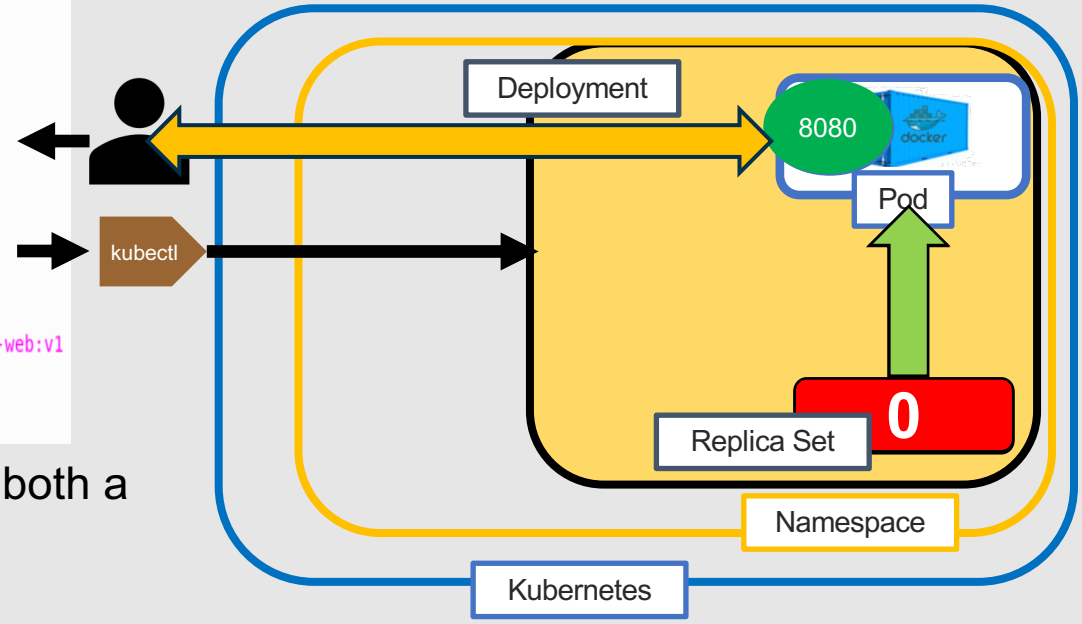


- Objects run within node and namespace
- Objects created from spec
- Pods encompass containers
- Ways to connect directly to them
- If pod goes away (and still need functionality), must manually start a new one
- Different ip address on new one

Deployments vs. pods-only

- Pods with Deployments

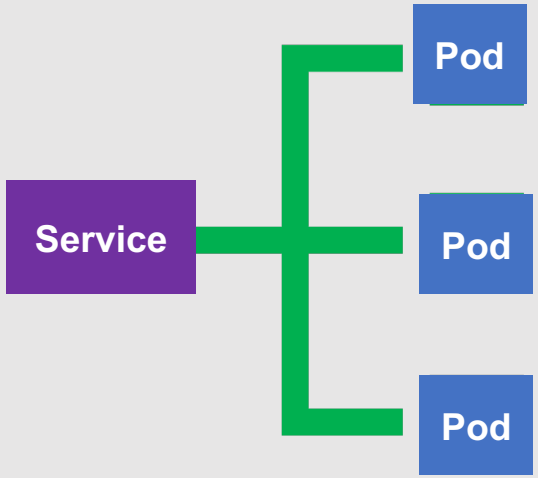
```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: roar-web
  labels:
    name: roar-web
  namespace: roar
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: roar-web
    spec:
      containers:
      - name: roar-web
        image: localhost:5000/roar-web:v1
        ports:
        - name: web
          containerPort: 8080
```



- Deployments include both a replicaset and pods
- Specs for both
- If pod goes away, replicaset starts up a new one automatically – up to number of replicas specified
- Different ip address on new one

Services

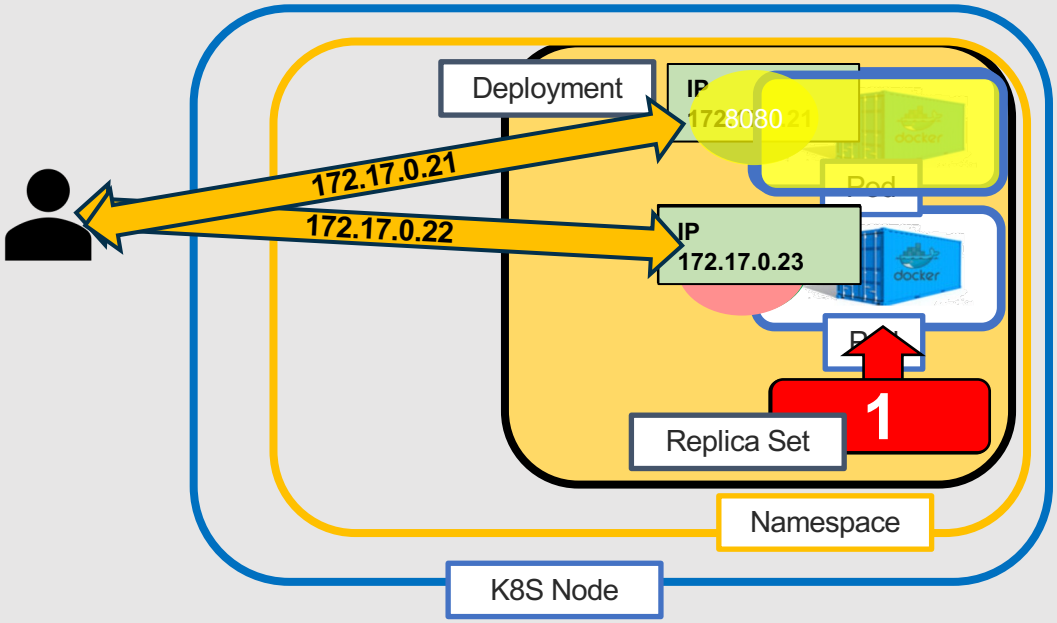
- A single virtual, stable connection for multiple volatile pods
  - Kubernetes service provides a level of abstraction for pods
  - Pods may be “volatile” so can’t rely on their respective IP addresses to connect to long-term
  - A service provides a virtual, consistent IP to connect to for a set of pods
  - Pods can be selected by labels
  - Virtual IP of a service is not connected to an actual network interface
  - Virtual IP exists to forward traffic to one or more pods
  - Kube-proxy process (running on every node) is responsible for keeping the mapping between the virtual IP and the pods up-to-date
  - Kube-proxy queries the API server to learn about new services



Services

- Without Services

- We can connect to individual pods (if allowed)
- If in deployment and pod goes away, we lose access to that pod
- ReplicaSet in deployment will spin up another one but won't have same address
- Similar situation if a pod becomes unusable
- - may have other pods that could handle needs, but no way to automatically discover or connect

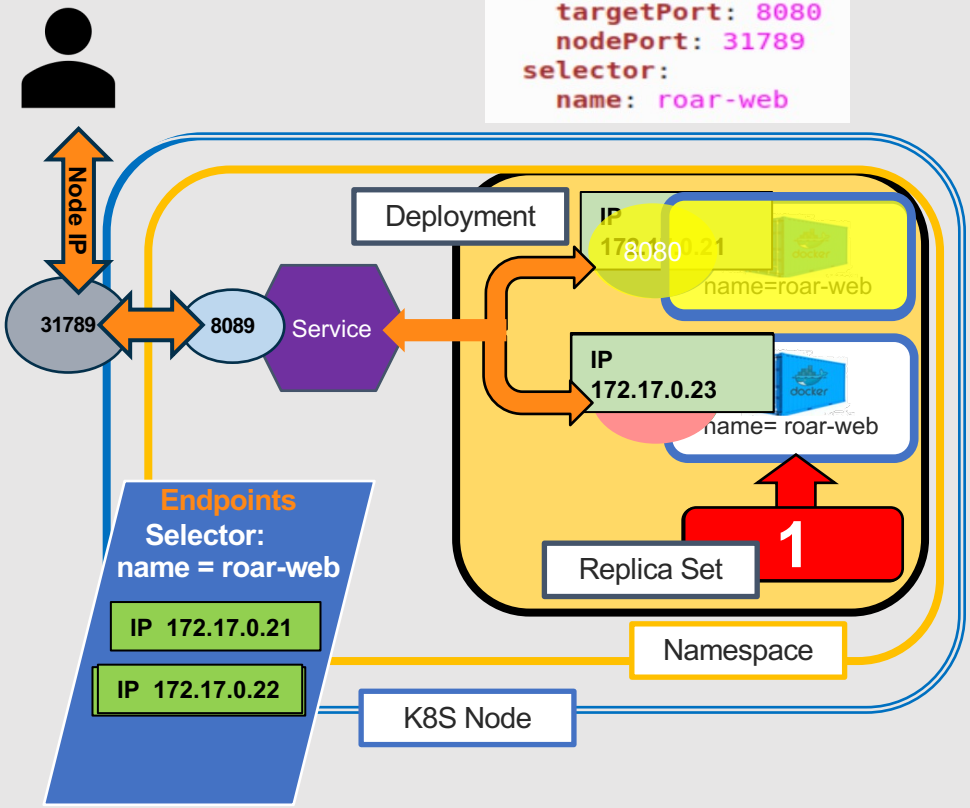


Services

- With a service

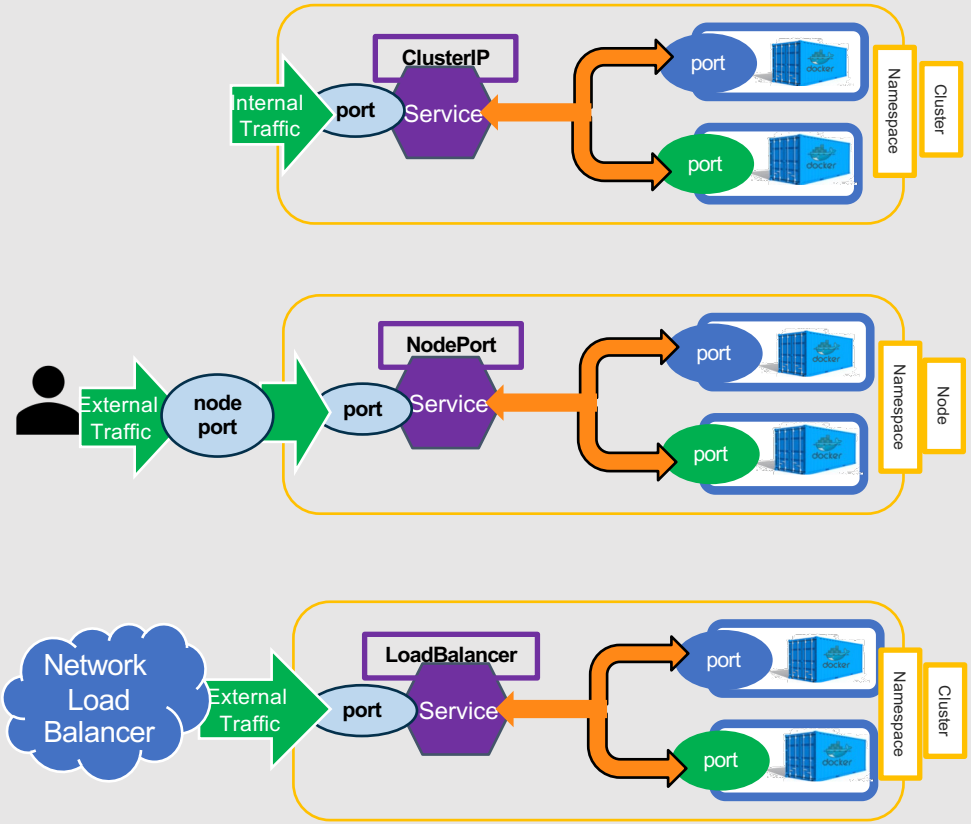
- Service provides virtual address for us to connect to for frontend
- Uses labels to select “pool” of backend pods to map to
- Endpoints for a service are list of available ip addresses for usable pods
- Example service here is NodePort
- If one pod goes down or becomes unavailable, service can connect to another pod
- Spec is shown below

```
apiVersion: v1
kind: Service
metadata:
  name: roar-web
  labels:
    name: roar-web
  namespace: roar2
spec:
  type: NodePort
  ports:
  - port: 8089
    targetPort: 8080
    nodePort: 31789
  selector:
    name: roar-web
```



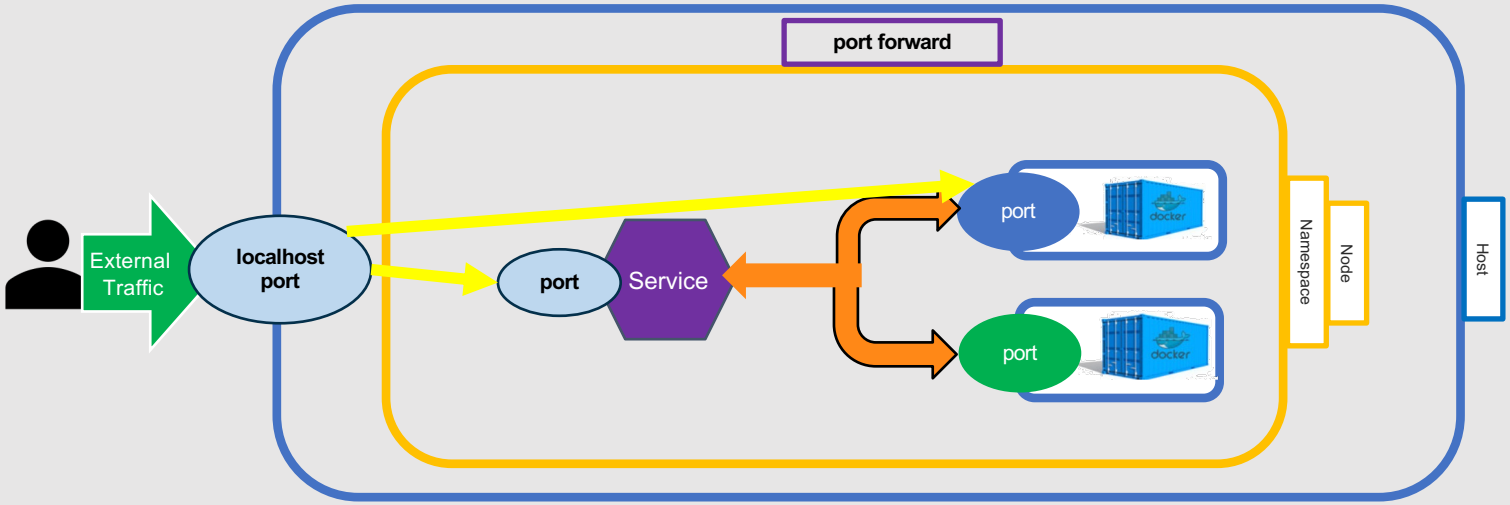
Types of Services

- ClusterIP
  - Default K8S service
  - Provides service inside cluster for other apps in cluster to access
  - No external access (except via kubectl proxy for debugging, etc.)
- NodePort
  - Most basic way to get external traffic to service
  - Opens up a port on the K8S node
  - Any traffic going to that port on the node is forwarded to the service
  - Uses node IP address
  - Only ports in range 30000-32767
- LoadBalancer
  - Gives you Load Balancer with single IP that forwards all traffic to service
  - Default method for directly exposing a service
  - No filtering, no routing
  - Cost factor on clouds
- ExternalName
  - Maps service to contents of an external name field (foo.bar.com)
  - Returns a CNAME record with that value



Forwarding Ports in Kubernetes

- Allows you to access and interact with internal K8s cluster processes from your localhost
- Used to investigate and validate services locally
- Command in K8s is "kubectl port-foward"
  - Arguments include cluster resource name
  - port number to port-forward to
- Result is K8s API server establishes a single http connection between localhost and resource on the cluster
- Allows connection to any pod in cluster
  - if pod fails, connect and fix
- Especially useful for back-end services not usually intended for remote exposure



Port Forward Examples

Specific pod - *Selects specific pod with port number for container = 8080 and listening port = 30001*

```
$ kubectl port-forward pod/roar-web-f5cbff465-vvr8f 30001:8080
```

Random local port - *Listen on a random port locally, and forward to port 8089 within pod*

```
$ kubectl port-forward svc/roar-web :8089
```

Corresponding local and remote port - *Listen and forward data using identical port (8080) on localhost and in pod*

```
$ kubectl port-forward pod/roar-web-f5cbff465-v8f 8080
```

Specific Local IP Address for Port Forwarding - *Listen on port 8080 on localhost using defined IP; forward to port 5762 in the pod*

```
$ kubectl port-forward --address 10.0.2.15 pod/roar-web-f5cbff465-vvr8f  8080
```

Use deployment to select port-forward pod - *Listen and forward data using the same port 8080 both locally and within the pod. The deployment defines which pod to use*

```
$ kubectl port-forward deploy/roar-web 8080
```

Allow service to define port-forward pod - *Listen and forward data using the same ports (8080 5762) both locally and within the pod. The service selects which pod to use*

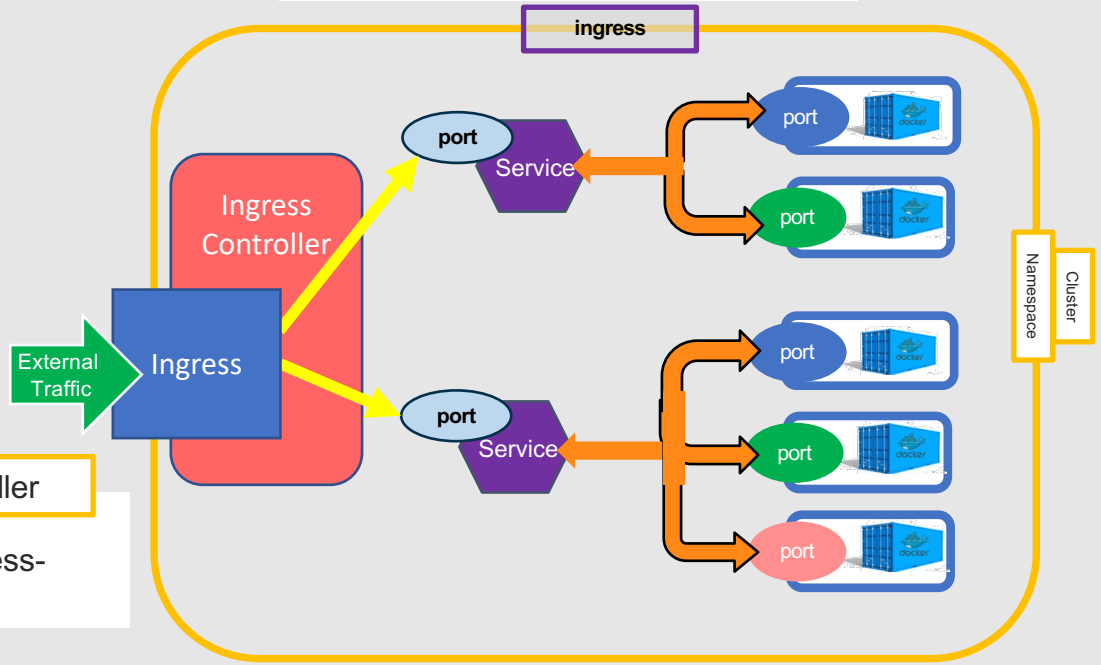
```
$ kubectl port-forward svc/roar-web 8089
```

Ingresses

- Allows you to expose services outside of cluster
- Allows consolidation/configuration of routing rules in a single resource
- NodePort and LoadBalancer let you expose a service via the manifest
- Ingress is completely independent resource
- Declared, created, and destroyed independently
- Must have ingress controller for cluster

Ingress YAML

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
        - path: /current
          backend:
            serviceName: v1-service
            servicePort: 8089
        - path: /beta
          backend:
            serviceName: v2-service
            servicePort: 8089
```



Install Ingress Controller

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/provider/cloud-generic.yaml
```

kubectl logs

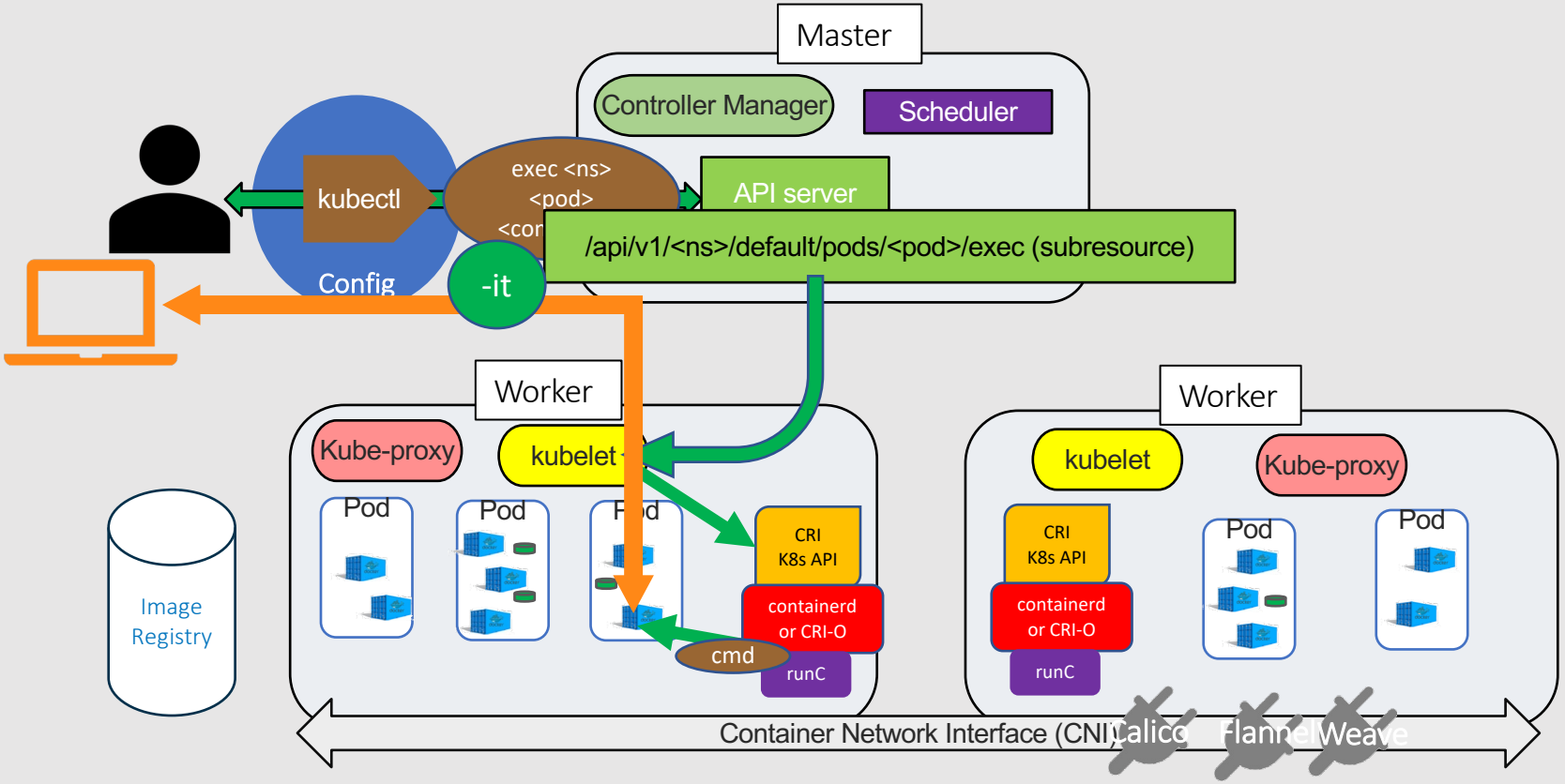
Print the logs for a container in a pod or specified resource.
Examples:

```
  # Return snapshot logs from pod nginx with only one container
  kubectl logs nginx

  # Return snapshot logs from all containers in pods defined by label app=nginx
  kubectl logs -lapp=nginx --all-containers=true

  # Return snapshot of previous terminated ruby container logs from pod web-1
  kubectl logs -p -c ruby web-1

  # Begin streaming the logs of the ruby container in pod web-1
  kubectl logs -f -c ruby web-1

  # Display only the most recent 20 lines of output in pod nginx
  kubectl logs --tail=20 nginx

  # Show all logs from pod nginx written in the last hour
  kubectl logs --since=1h nginx

  # Show logs from a kubelet with an expired serving certificate
  kubectl logs --insecure-skip-tls-verify-backend nginx

  # Return snapshot logs from first container of a job named hello
  kubectl logs job/hello

  # Return snapshot logs from container nginx-1 of a deployment named nginx
  kubectl logs deployment/nginx -c nginx-1
```

```
$ k logs mysql-5c97d489dd-z5pp2

2021-06-22 23:43:46+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5.7.32-1debian10 started.
2021-06-22 23:43:46+00:00 [Note] [Entrypoint]: Initializing database files
2021-06-22T23:43:46.950512Z 0 [Warning] TIMESTAMP with implicit DEFAULT value is deprecated. Please use --explicit_defaults_for_timestamp server option (see documentation for more details).
2021-06-22T23:43:47.138897Z 0 [Warning] InnoDB: New log files created, LSN=45790
2021-06-22T23:43:47.211492Z 0 [Warning] InnoDB: Creating foreign key constraint system tables.
2021-06-22T23:43:47.270535Z 0 [Warning] No existing UUID has been found, so we assume that this is the first time that this server has been started. Generating a new UUID: b02442ec-d3b3-11eb-9519-0242ac110006.
2021-06-22T23:43:47.274194Z 0 [Warning] Gtid table is not ready to be used. Table 'mysql.gtid_executed' cannot be opened.
2021-06-22T23:43:47.625895Z 0 [Warning] CA certificate ca.pem is self signed.
2021-06-22T23:43:47.848450Z 1 [Warning] root@localhost is created with an empty password ! Please consider switching off the --initialize-insecure option.
2021-06-22 23:43:52+00:00 [Note] [Entrypoint]: Database files initialized
2021-06-22 23:43:52+00:00 [Note] [Entrypoint]: Starting temporary server
```

kubectl exec

- Allows for inspecting and debugging applications
- Allows executing commands inside our containers
- Can be interactive



Lab 2 - Working with services and ports

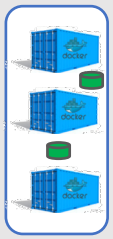
Purpose: In this lab, we'll learn more about how Kubernetes services and ports work and how to debug basic issues with them.

Data Center Analogy

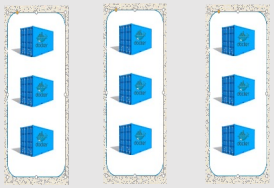
- Functions: Uptime, scaling, redundancy…

- Container in a pod ~ server in a rack

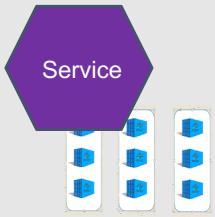

- Pod ~ rack of servers




- Deployment ~ multiple racks (replicas)




- Service ~ central control / login server



- Namespace ~ server room

kubectl describe

```
 # Describe a node
  kubectl describe nodes kubernetes-node-
emt8.c.myproject.internal

  # Describe a pod
  kubectl describe pods/nginx

  # Describe a pod identified by type and name in
"pod.json"
  kubectl describe -f pod.json

  # Describe all pods
  kubectl describe pods

  # Describe pods by label name=myLabel
  kubectl describe po -l name=myLabel

  # Describe all pods managed by the 'frontend'
replication controller
(rc-created pods
  # get the name of the rc as a prefix in the pod the
name).
  kubectl describe pods frontend
```

```
$ k describe pod mysql-7bf6b7fc5-2tdkw
Name:           mysql-7bf6b7fc5-2tdkw
Namespace:      ts
Priority:       0
Node:           <none>
Labels:         app=mysql
                pod-template-hash=7bf6b7fc5
Annotations:    <none>
Status:         Pending
IP:
IPs:            <none>
Controlled By:  ReplicaSet/mysql-7bf6b7fc5
Containers:
  roar-db:
    Image:      quay.io/techupskills/roar-db:1.0.1
    Port:       3306/TCP
    Host Port:  0/TCP
    Limits:
      cpu:     1
      memory:  10Gi
    Requests:
      cpu:      1
      memory:   10Gi
    Readiness:  exec [mysql] delay=5s timeout=1s period=10s #success=1 #failure=3
    Environment:
      MYSQL_DATABASE:       registry
Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-j2lcx (ro)
Conditions:
  Type           Status
  PodScheduled   False
Volumes:
  kube-api-access-j2lcx:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   Guaranteed
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type     Reason            Age                From               Message
  ----     ------            ----               ----               -------
  Warning  FailedScheduling  15s (x23 over 42m)  default-scheduler  0/1 nodes are
available:
1 Insufficient memory.
```

Configmaps and Secrets

- Approaches for configuring applications
- Could specify environment variables in Dockerfile or k8s yaml file
- Bad part of environment variables is they are tied to container or deployment.
- To change them, must modify container or redo deployment.
- Data must also be duplicated across objects.
- Alternative is secrets for confidential data and configmaps for non-confidential data
- Difference between Secrets and ConfigMaps are that Secrets are obfuscated with a Base64 encoding.
- Can be created like any other objects in Kubernetes.

Secrets

- Examples
  - Environment variables set up in spec for admin and root passwords
  - Create secrets for both
    - Create via command line

```
k create secret generic mysqlsecret2 --
from-literal=mysqlpassword=admin --
from-literal=mysqlrootpassword=root+1
-n roar
```

    - Create via yaml file

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysqlsecret
type: Opaque
data:
  mysqlpassword: YWRtaW4=
  mysqlrootpassword: cm9vdCsx
```

  - Update spec to use those

```yaml
        env:
        - name: MYSQL_DATABASE
          value: registry
        - name: MYSQL_PASSWORD
          value: admin
        - name: MYSQL_ROOT_PASSWORD
          value: root+1
        - name: MYSQL_USER
          value: admin
```

```yaml
        env:
          - name: MYSQL_DATABASE
            value: registry
          - name: MYSQL_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mysqlsecret
                key: mysqlpassword
          - name: MYSQL_ROOT_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mysqlsecret
                key: mysqlrootpassword
          - name: MYSQL_USER
            value: admin
```

Configmaps

- Examples
  - Environment variables set up in spec for database and user
  - Create configmap for both
    - Create via command line

k create configmap mysql-configmap --from-literal=mysql.database=registry --from-literal=mysql.user=admin -n roar To change them, must modify container

OR

- Create via yaml file

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-configmap
data:
  mysql.database: registry
  mysql.user: admin
```

- Update spec to use those

```
58        env:
59          - name: MYSQL_DATABASE
60            value: registry
61          - name: MYSQL_PASSWORD
62            valueFrom:
63              secretKeyRef:
64                name: mysqlsecret
65                key: mysqlpassword
66          - name: MYSQL_ROOT_PASSWORD
67            valueFrom:
68              secretKeyRef:
69                name: mysqlsecret
70                key: mysqlrootpassword
71          - name: MYSQL_USER
72            value: admin
```

```
58        env:
59          - name: MYSQL_DATABASE
60            valueFrom:
61              configMapKeyRef:
62                name: mysql-configmap
63                key: mysql.database
64          - name: MYSQL_PASSWORD
65            valueFrom:
66              secretKeyRef:
67                name: mysqlsecret
68                key: mysqlpassword
69          - name: MYSQL_ROOT_PASSWORD
70            valueFrom:
71              secretKeyRef:
72                name: mysqlsecret
73                key: mysqlrootpassword
74          - name: MYSQL_USER
75            valueFrom:
76              configMapKeyRef:
77                name: mysql-configmap
78                key: mysql.user
```

Lab 3 – Working with Kubernetes secrets and configmaps

Purpose: In this lab, we'll get some practice storing secure and insecure information in a way that is accessible to k8s but not stored in the usual deployment files.

Kubernetes Storage Concepts

- Container filesystem
  - Storage at the level of a container
    - » Ephemeral – only exists while container is running
    - » Any changes made are gone when container is gone
- Volume
  - Storage at the level of a pod
    - » Data is preserved if a container crashes
    - » Allows sharing of data across containers in a pod
    - » Only exists while pod exists
    - » Any changes made are gone when pod is gone (deleting pod deletes the volume)
- Persistent Volume
  - Storage outside the pod – cluster level
    - » Data is preserved if pod goes away
- Persistent Volume Claim
  - Way of saying what is needed for storage
    - » Defines specific amount of storage requested, access mode, etc.
  - Kubernetes looks for a specific matching Persistent Volume and matches it to claim
  - PVC's have different access modes they can specify
    - » ReadWriteOnce – one bound Pod with R/W access
    - » ReadOnlyMany – many bound Pods with R only access
    - » ReadWriteMany – many bound Pods with R/W access

```yaml
---
kind: PersistentVolume
apiVersion: v1
metadata:
  name: mysql-pv
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 100M
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  labels:
    app: mysql
  name: mysql-pv-claim
spec:
  storageClassName: manual
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 100M
```

```yaml
    spec:
      containers:
      - name: mysql
        image: localhost:5000/roar-db:v1
        ports:
        - name: mysql
          containerPort: 3306
        env:
          - name: MYSQL_DATABASE
            valueFrom:
              configMapKeyRef:
                name: mysql-configmap
                key: mysql.database
          - name: MYSQL_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mysqlsecret
                key: mysqlpassword
          - name: MYSQL_ROOT_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mysqlsecret
                key: mysqlrootpassword
          - name: MYSQL_USER
            valueFrom:
              configMapKeyRef:
                name: mysql-configmap
                key: mysql.user
        volumeMounts:
        - mountPath: /var/lib/mysql
          name: mysql-pv-claim
      volumes:
      - name: mysql-pv-claim
        persistentVolumeClaim:
          claimName: mysql-pv-claim
---
```

Kubernetes Storage Objects

- Objects
  - PV's
    - physical volume on the host machine or network that stores the data to be persisted
  - PVC's
    - request for PV's
    - claim check for resource
  - PV's are attached to pods via PVC's
  - Storageclasses define the type of PVCs that a user can request
  - Think of it like:

Pod -> PVC -> PV -> (Storage Class, If applicable) -> Host machine

Storage Classes and Provisioners

- Storage classes provides way for provider to define classes of storage they offer
- May define QOS levels, backup policies, etc.
- Storage classes have a Provisioner to determine what kind of PV
- Storage classes can dynamically provision a PV if so configured
- Admins can specify StorageClass for PVCs

Volume Plugin	Internal Provisioner	Config Example
AWSElasticBlockStore	✓	AWS EBS
AzureFile	✓	Azure File
AzureDisk	✓	Azure Disk
CephFS	-	-
Cinder	✓	OpenStack Cinder
FC	-	-
Flexvolume	-	-
Flocker	✓	-
GCEPersistentDisk	✓	GCE PD
Glusterfs	✓	Glusterfs
iSCSI	-	-
Quobyte	✓	Quobyte
NFS	-	-
RBD	✓	Ceph RBD
VsphereVolume	✓	vSphere
PortworxVolume	✓	Portworx Volume
ScaleIO	✓	ScaleIO
StorageOS	✓	StorageOS
Local	-	Local

Lab 4 – Working with persistent storage - Kubernetes Persistent Volumes and Persistent Volume Claims

Purpose: In this lab, we'll see how to connect pods with external storage resources via persistent volumes and persistent volume claims.

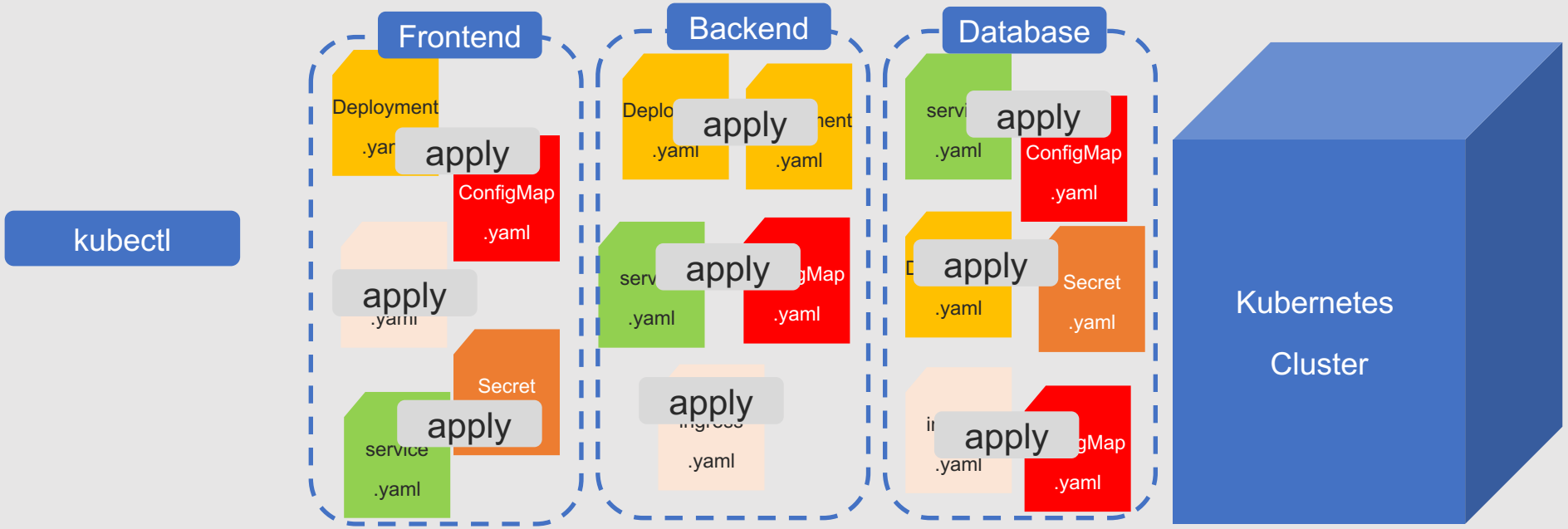
Helm

What is Helm?

- Package Manager and Lifecycle Manager for K8s
  - Like yum, apt but for K8s
  - Bundles related manifests (such as deployment.yaml, sevice.yaml, etc.) into a “chart”
  - When installing chart, Helm creates a “release”
  - Lifecycle management
    - Create, Install, Upgrade, Rollback, Delete, Status, Versioning
  - Benefits
    - Templating, Repeatability, Reliability, Multiple Environment, Ease of collaboration

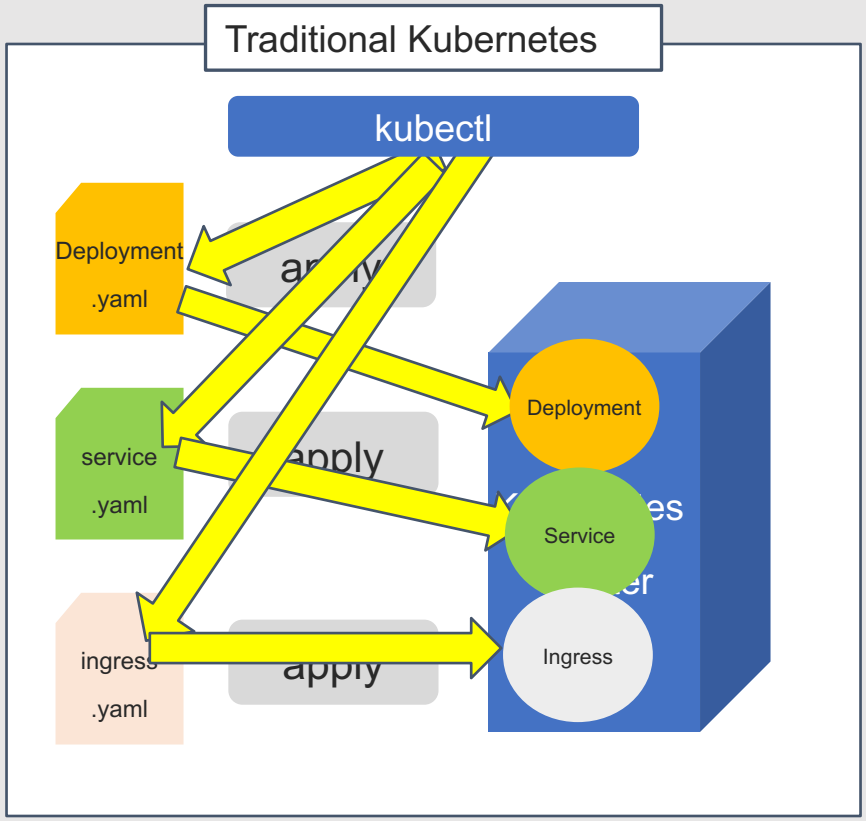
Why do we need something like this?

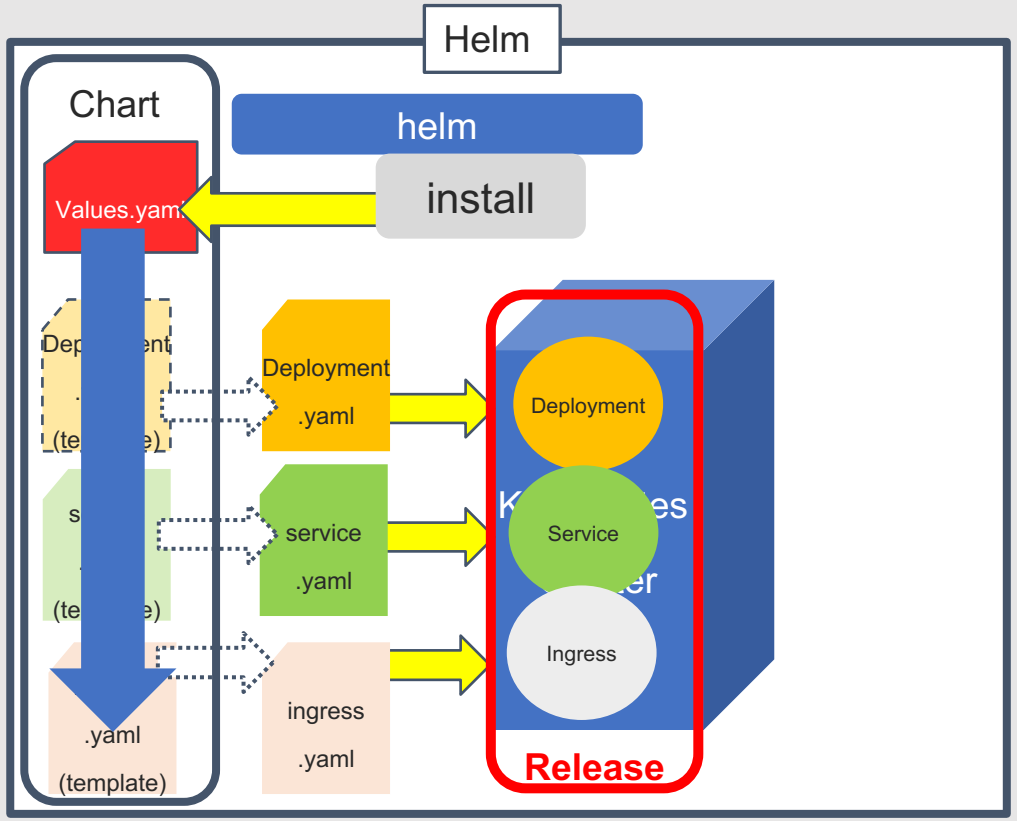
- Scale and complexity
- Microservice = Pod + Deployment+ ReplicationSet + Ingress + Service times # of microservices
- Duplication of values across objects
- Hard to override values (no parameterization)
- Managing lifecycle of all the objects is challenging



How does Helm simplify things?

- Traditional deployment in Kubernetes is done with kubectl across files into separately managed items
- Helm deploys units called charts as managed releases





What are the advantages of using Helm?

- Saves having to deploy multiple individual manifests (for multiple Kubernetes objects)
- Allows for reuse via parameterizing (templates)
- Manages releases of Helm packages
- Simplified mechanism for finding and deploying popular software (packaged as Helm charts)
- Makes it easier to share your applications (via charts)

Helm Topology

- Chart – a package; a bundle of K8s resources
- Release – a chart instance loaded into K8s
  - Same chart can be installed several times into the same cluster
  - Each such chart will have its own release
- Repository – a repository of published charts
- Template - a K8s configuration file mixed with Go/Sprig templates

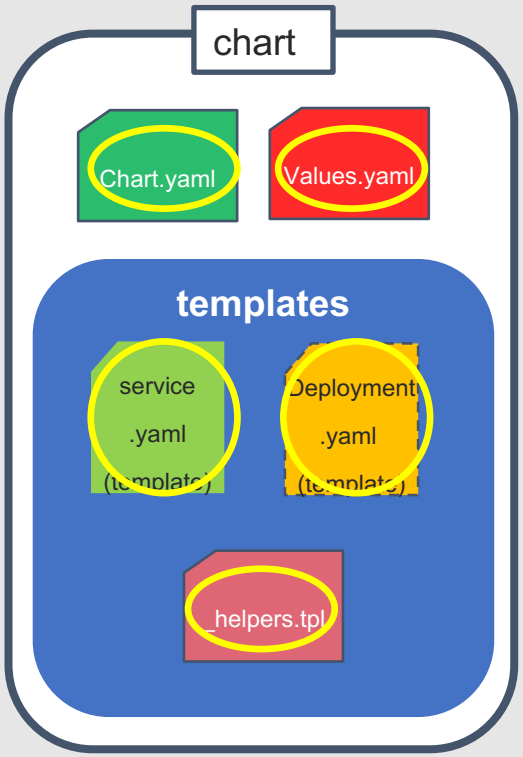
NAME	REVISION	UPDATED	STATUS	CHART	APP VERSION	NAMESPACE
istio	2	Sat Jul 20 22:09:38 2019	DEPLOYED	istio-1.2.0	1.2.0	istio-system
istio-init	1	Thu Jun 27 13:34:49 2019	DEPLOYED	istio-init-1.2.0	1.2.0	istio-system
istio1	5	Sun Oct 27 17:56:53 2019	DEPLOYED	roar-web-0.1.0		istio1
jenkins-x	1	Thu Jun 6 07:53:23 2019	DEPLOYED	jenkins-x-platform-2.0.330		jx
roar2	2	Sun Oct 27 17:31:35 2019	DEPLOYED	roar-helm-0.1.0		roar2

What is a Chart?

- Packaging format for Helm
- Define a way to compose a set of K8S resources and values to make up a deployment
- Deployable unit
  - Can be installed, updated, removed
- “Source code”
  - Can be versioned and managed in source control and compressed in packages
- Charts can include other charts as dependencies

Example Files in a Helm Chart

- Main chart directory
- Chart.yaml – chart description
- Values.yaml – values to be used in chart
- Templates – templated files that will form K8s manifests
- _helpers.tpl – helper functions

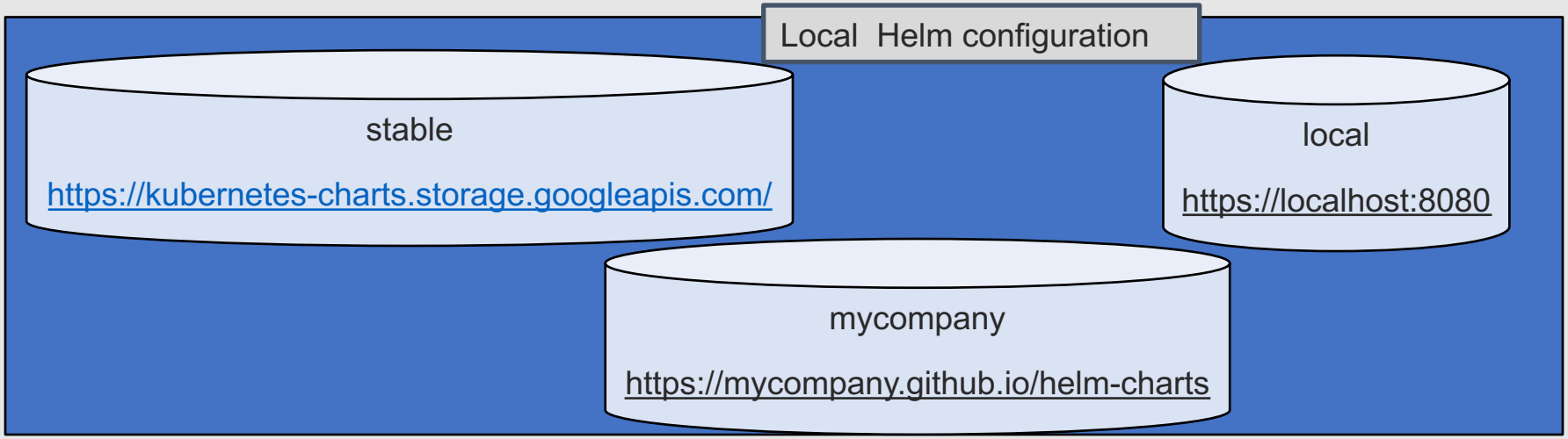


deployment.yaml
~/helm-ws/roar-db/templates

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: {{ template "roar-db.name" . }}
  labels:
    app: {{ template "roar-db.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
  namespace: {{ .Values.namespace }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    metadata:
      labels:
        app: {{ template "roar-db.name" . }}
    spec:
      containers:
      - name: {{ .Chart.Name }}
        image: bclaster/roar-db-image:v1
        imagePullPolicy: Always
        ports:
        - name: {{ .Values.deployment.ports.name }}
          containerPort: {{ .Values.deployment.ports.containerPort }}
        env:
          {{- include "roar-db.environment-values" . | indent 10 }}
```

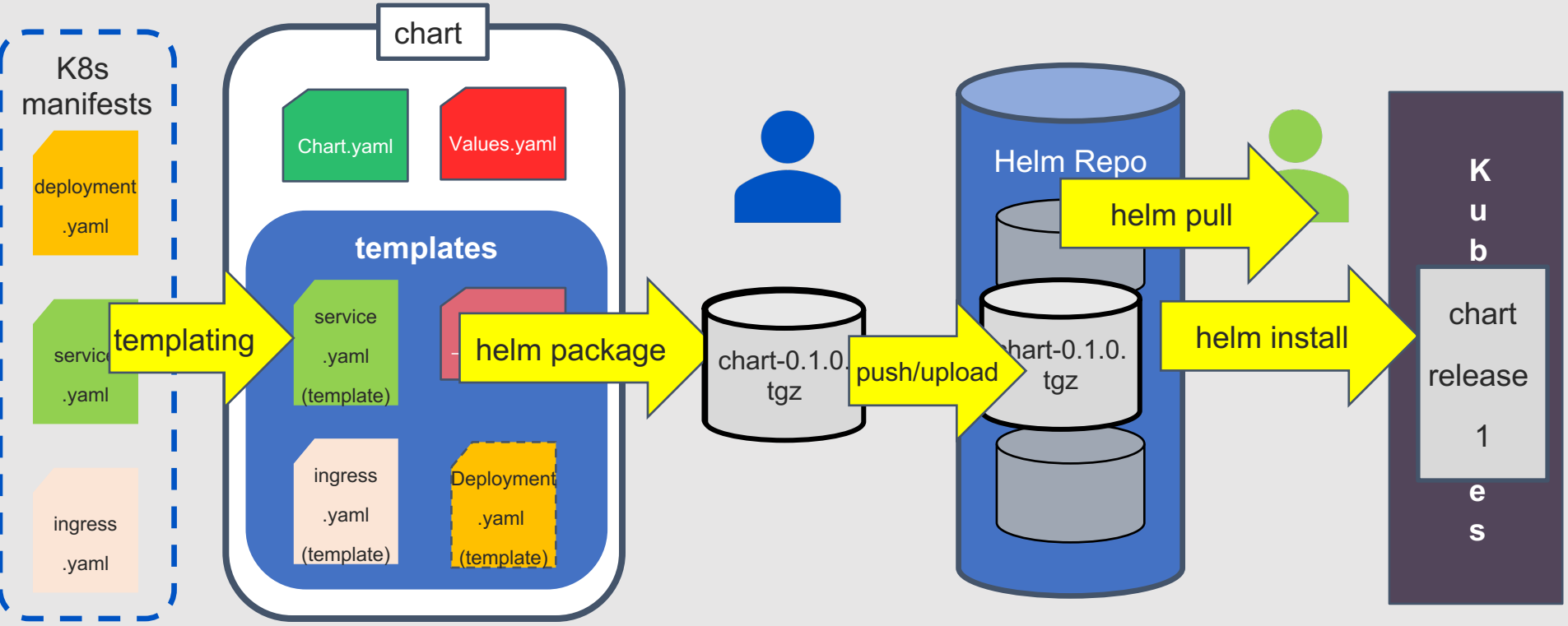
What is a chart repository?

- Chart repo
  - Location where Helm charts can be stored and shared
  - Able to serve yaml and tar files
    - » Helm package format is tar file
    - » Index file for repo is yaml
  - Responds to REST API GET requests to get packages
  - Many storage options available – cloud buckets, local storage, GH Pages, etc.
- Helm instance can have many repos defined/added



Helm as Package Manager

- Kubernetes files are “templated”
- Templates and related files are structured as a chart
- Charts are packaged
- Packages are stored in repos for easy use by others
- Charts can be pulled (downloaded) and optionally unpacked (untar)
- Charts are installed from repos (with values) as releases into Kubernetes



Helm Repo Operations

- Show all repos
  - $ helm repo list
- Add a repo
  - $ helm repo add <repo> url
  - $ helm repo add stable https://kubernetes-charts.storage.googleapis.com/
- Search a repo
  - $ helm search repo <repo> <chartname>
  - $ helm search hub
- Remove a repo
  - $ helm repo rm <repo>

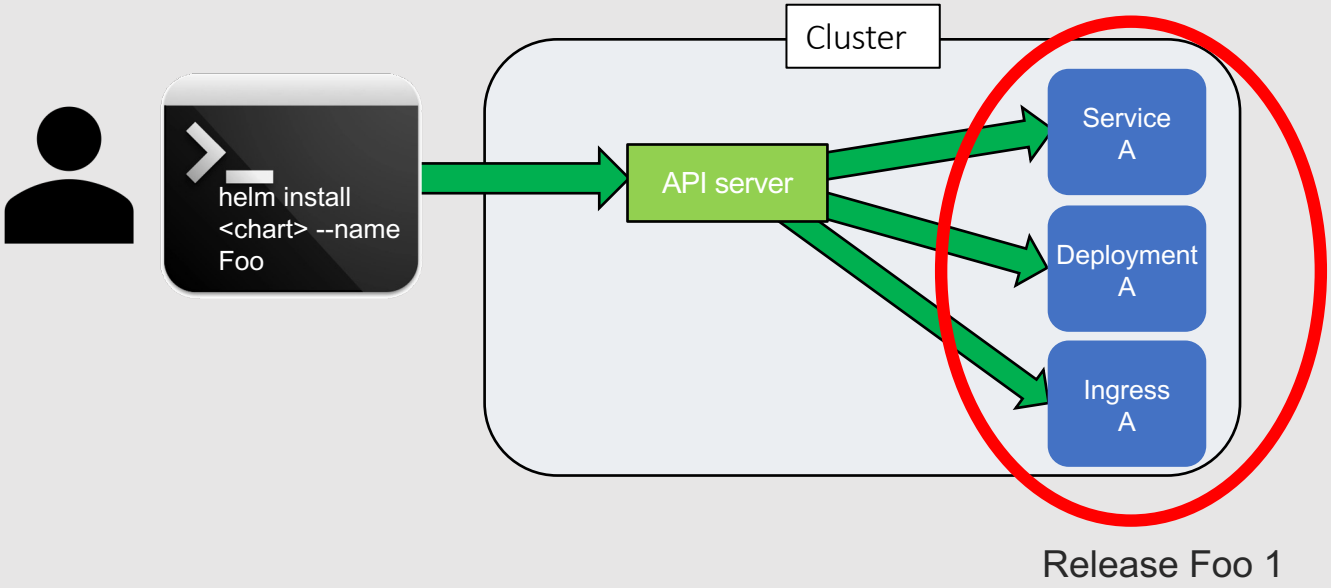
```
$ helm search repo stable | head
NAME                            CHART VERSION   APP VERSION     DESCRIPTION
stable/acs-engine-autoscaler    2.2.2           2.1.1           DEPRECATED Scales worker nodes within agent pools
stable/aerospike                0.3.2           v4.5.0.5        A Helm chart for Aerospike in Kubernetes
stable/airflow                  7.1.5           1.10.10         Airflow is a platform to programmatically autho...
stable/ambassador               5.3.2           0.86.1          DEPRECATED A Helm chart for Datawire Ambassador
stable/anchore-engine           1.6.9           0.7.2           Anchore container analysis and policy evaluatio...
stable/apm-server               2.1.5           7.0.0           The server receives data from the Elastic APM a...
stable/ark                      4.2.2           0.10.2          DEPRECATED A Helm chart for ark
```

Helm Operations

- completion generate autocompletions script for the specified shell (bash or zsh)
- create create a new chart with the given name
- dependency manage a chart's dependencies
- env helm client environment information
- get download extended information of a named release
- help Help about any command
- history fetch release history
- **install install a chart**
- lint examine a chart for possible issues
- **list list releases**
- package package a chart directory into a chart archive
- plugin install, list, or uninstall Helm plugins
- pull download a chart from a repository and (optionally) unpack it in local directory
- **repo add, list, remove, update, and index chart repositories**
- rollback roll back a release to a previous revision
- **search search for a keyword in charts**
- **show show information of a chart**
- status display the status of the named release
- template locally render templates
- test run tests for a release
- uninstall uninstall a release

Helm Install a Chart

- helm install <chart>



Getting Information about Releases

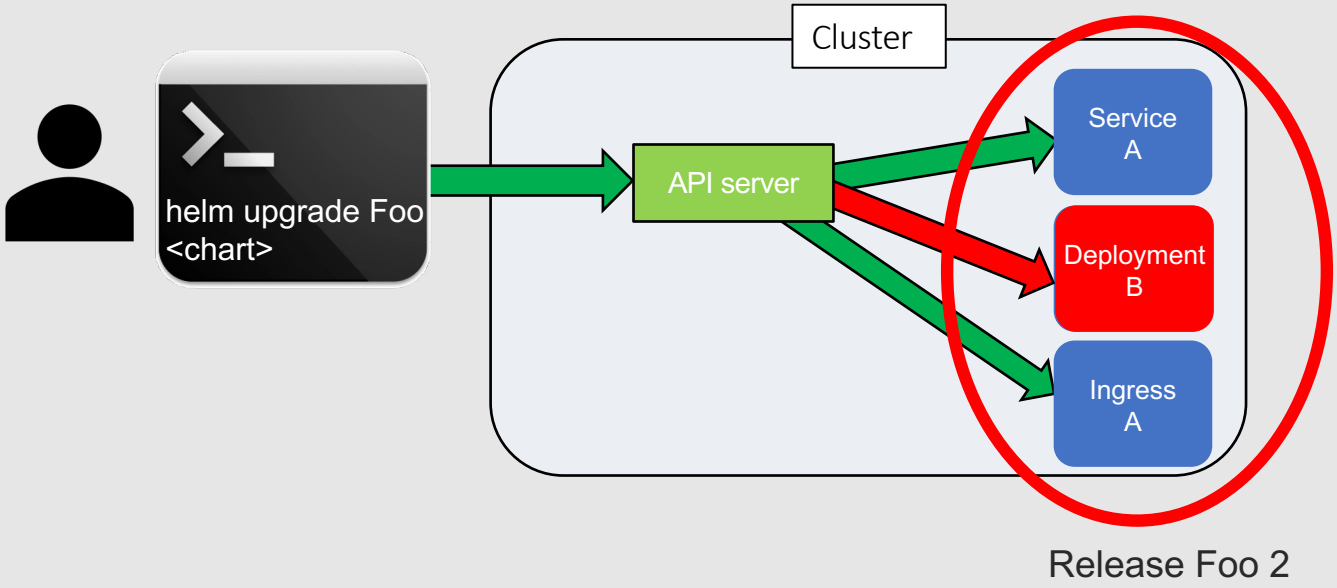
- A release is an instance of a Helm chart deployed in Kubernetes
- Has a release name that can be different from chart name
- See list of releases
  - $ helm list
- See current status
  - $ helm status RELEASE_NAME [flags]
- See history of release
  - $ helm history RELEASE_NAME [flags]

```
$ helm status local-chartmuseum
NAME: local-chartmuseum
LAST DEPLOYED: Sun Jun 14 15:55:32
 2020
NAMESPACE: default
STATUS: deployed
REVISION: 2
TEST SUITE: None
NOTES:
** Please be patient while the chart is
being deployed **

Get the ChartMuseum URL by running:
```

```
$ helm history local-chartmuseum
REVISION UPDATED                  STATUS     CHART                   APP VERSION DESCRIPTION
1        Sun Jun 14 15:45:38 2020 superseded chartmuseum-2.13.0 0.12.0 Install complete
2        Sun Jun 14 15:55:32 2020 deployed   chartmuseum-2.13.0 0.12.0 Upgrade complete
```

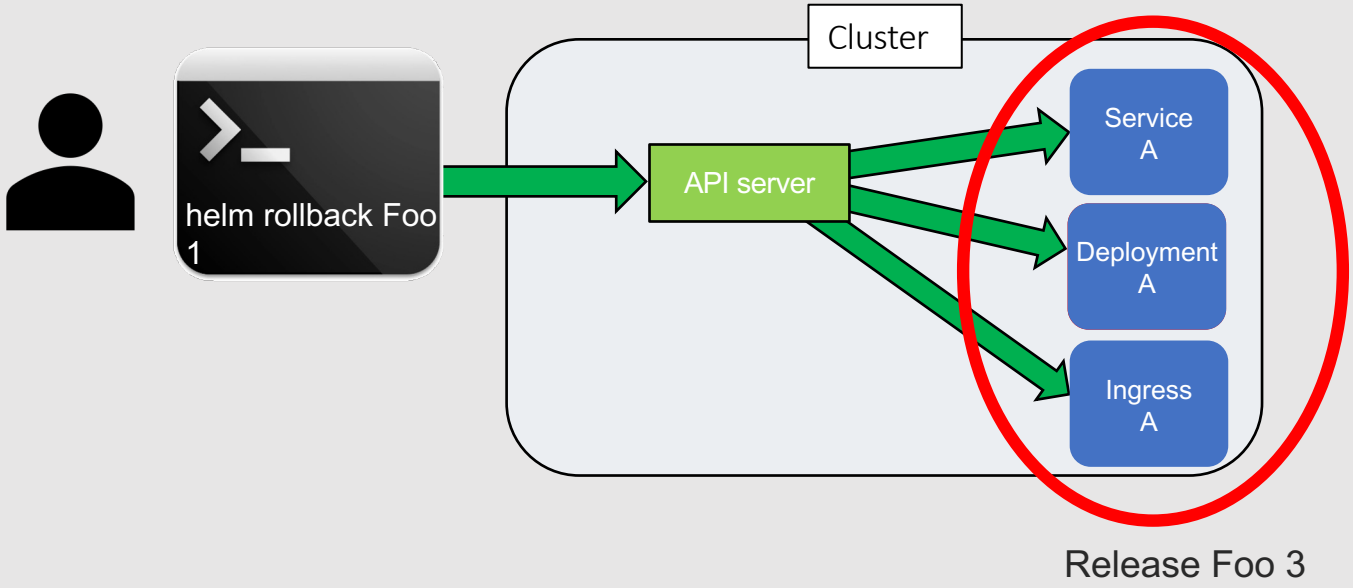
Helm Upgrading a Chart

- helm upgrade <release> <chart>
- use --set to override chart settings



Helm Rollback

- helm history <release> (to see revision numbers)
- helm rollback <release> <revision>



Our Example App

- R.O.A.R. (Registry of Animal Responders)
  - **Simple app with two pieces**
    - **Webapp on the front**
    - **Mysql database on the backend**
  - **Written in Java**
  - **Simple REST API**
  - **War file is produced for webapp**
  - **Managed in Tomcat**

Lab 5 - Working with Helm

Purpose: In this lab, we'll compare a Helm chart against standard Kubernetes manifests and then deploy the Helm chart into Kubernetes

Traditional K8s yaml vs Helm template

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: roar-web
  labels:
    app: roar-web
  namespace: roar
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: roar-web
    spec:
      containers:
      - name: roar-web
        image: localhost:5000/roar-web-v1
        ports:
        - name: web
          containerPort: 8080
```

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: {{ template "roar-web.name" . }}
  labels:
    app: {{ template "roar-web.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
  namespace: {{ .Values.namespace }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    metadata:
      labels:
        app: {{ template "roar-web.name" . }}
    spec:
      containers:
      - name: {{ .Chart.Name }}
        image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
      {{- if .Values.image.pullPolicy }}
        imagePullPolicy: {{ toYaml .Values.image.pullPolicy }}
      {{- end }}
        ports:
        - name: {{ .Values.deployment.ports.name }}
          containerPort: {{ .Values.deployment.ports.containerPort }}
```


How Values are resolved in Helm

templates/deployment.yaml

```
spec:
    containers:
    - name: {{ .Chart.Name }}
      image: bclaster/roar-db-image:v1
      imagePullPolicy: Always
      ports:
      - name: {{ .Values.deployment.ports.name }}
        containerPort: {{ .Values.deployment.ports.containerPort }}
      env:
        {{- include "roar-db.environment.values" . | indent 10 }}
```

values.yaml

```
# Default values for roar-db-chart.
# This is a YAML-formatted file.
# Declare variables to be passed
# into your templates.
replicaCount: 1
nameOverride: mysql
deployment:
  ports:
    name: mysql
    containerPort: 3306
```

deployment.yaml

```
…

containerPort: 3306
```

- string indicates hierarchial path
- dot notation separates levels
- .Values refers to top-level values from values.yaml

Template Functions

- Allows you to transform data passed into a template
- Function syntax
  - functionname arg1 arg2 …
- Over sixty functions available to use; provided by
  - Go template language
  - Go sprig template library
- Examples
  - upper, quote, eq
  - custom functions

Pipelines

- Similar to the idea of Unix pipelines
- Allows chaining together multiple template commands
- Simplify doing multiple things in sequence
- When pipelining arguments, result of previous evaluation is sent as last argument to next one

{{ first evaluation | function arg1 }}

equivalent to

{{ function arg1 arg2 }} where “arg2” = “first evaluation”

- Example

```
name:  {{ .Values.user.name | upper | quote }}
```

Lab 6 - Templating with Helm

Purpose: In this lab, you'll get to see how we can change hard-coded values into templates, override values, and upgrade releases through Helm.

Kustomize

Reuse from Customization

- Kubernetes yaml files are often reused
- Approaches for reuse
  - Modify the original to suit your needs
  - Make multiple copies and edit them separately for different needs
- Challenges
  - Divergence from original “source” files
  - Having multiple disparate copies means it is more challenging to leverage a common “base” set
  - Making consistent changes across multiple, independent copies for new versions, etc. becomes challenging
- Another approach is parameterizing templates – i.e. Helm

Challenges with Helm

- Templates and value files are not usable Kubernetes specifications
- They are more like a new kind of “programming” that wraps around the Kubernetes spec
- Parameterized templates change the original specification
- Many things become parameters for many different users
- Counters a typical goal of reuse – keep differences between specs small and simple

Another Option - Kustomize

- Kustomize creates modified Kubernetes manifests by “overlaying” declarative specifications on top of existing Kubernetes manifests
- Changes to make/overlay are stored in a separate file called *kustomization.yaml*
- Kustomize reads the kustomization.yaml file and the Kubernetes spec files (manifests) and dumps out completed resources to stdout with the changes from kustomization.yaml applied “on top of” the standard Kubernetes files
- *kustomize build* command causes files to be processed and output created
- Kustomize leverages the idea of “transformers”
- Transformers are functionality built into Kustomize that “transforms” your Kubernetes manifests given a simple set of declarative rules

Simple Example

Directory

roar-web-deploy.yaml
roar-web-svc.yaml
kustomization.yaml

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    app: roar-web
  name: roar-web
spec:
  replicas: 1
  template:
    metadata:
      labels:
```

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: roar-web
  name: roar-web
spec:
  ports:
  - name: frontend
```

```yaml
# common label to apply
commonLabels:
  level: test

# List of resource files that
# kustomize reads, modifies
# and emits as a YAML string
resources:
- roar-web-deploy.yaml
- roar-web-svc.yaml
```

kustomize build .

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: roar-web
    level: test
  name: roar-web
spec:
  ports:
  - name: frontend
    port: 8089
    targetPort: 8080
  selector:
    app: roar-web
    level: test
  type: NodePort
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    app: roar-web
    level: test
  name: roar-web
spec:
  replicas: 1
  selector:
    matchLabels:
      level: test
  template:
```

"Common" changes

- commonLabel – adds a common label (name: value) to each K8S resource
- commonAnotations – adds an annotation to all K8S resources
- namePrefix – adds a common prefix to all resource names

```
# common annotation to apply
commonAnnotations:
  build: latest
  branch: master

# List of resource files
# kustomize reads, modifies
# and emits as a YAML string
resources:
- roar-web-deploy.yaml
- roar-web-svc.yaml
```

Kustomize build .

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    branch: master
    build: latest
  labels:
    app: roar-web
  name: roar-web
spec:
  ports:
  - name: frontend
    port: 8089
    targetPort: 8080
  lector:
    p: roar-web
  pe: NodePort
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    branch: master
    build: latest
  labels:
    app: roar-web
  name: roar-web
```

```
# name prefix to apply
namePrefix:
 betaTest-

# List of resource files that
# kustomize reads, modifies
# and emits as a YAML string
resources:
- roar-web-deploy.yaml
- roar-web-svc.yaml
```

Kustomize build .

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: roar-web
  name: betaTest-roar-web
spec:
  ports:
  - name: frontend
    port: 8089
    targetPort: 8080
  ector:
    : roar-web
  e: NodePort
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    app: roar-web
  name: betaTest-roar-web
```

Lab 7 - Run a basic Kustomize example

Purpose: In this lab, we'll see how to make a set of manifests usable with Kustomize and how to use Kustomize to add additional changes without modifying the original files.

Variants

- Common use case is needing multiple variations (variants) of a set of common resources (i.e. dev, stage, prod)
- To do this, Kustomize has concept of “overlay” and “base”
  - Both represented via kustomization.yaml file (includes resources + customizations)
  - Base – declares things variants have in common
  - Overlays – declare the differences
- Variants can also apply “patches”
  - Patch – partial deployment spec – used to update some existing spec as opposed to adding new information

Kustomize Base

```
diyuser3@training1:~/ato-ws/roar-kz/base/db$ kustomize build .
apiVersion: v1
kind: Service
metadata:
  labels:
    app: roar-db
```

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    labels:
5      app: roar-db
6    name: roar-db
7  spec:
8    ports:
9    - name: mysql
10     port: 3306
11     targetPort: 3306
12   selector:
13     app: roar-db
```

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    labels:
5      app: roar-db
6    name: roar-db
7  spec:
8    ports:
9    - name: mysql
       port: 3306
       targetPort: 3306
       app:
     path: "/metadata/name"
     value: "mysql"
   target:
     kind: Service
     name: "roar-db"
```

sions/v1beta1

hat kustomize reads, modifies
ng

Resources to include

from "roar-db" to "mysql"
ted with the webapp

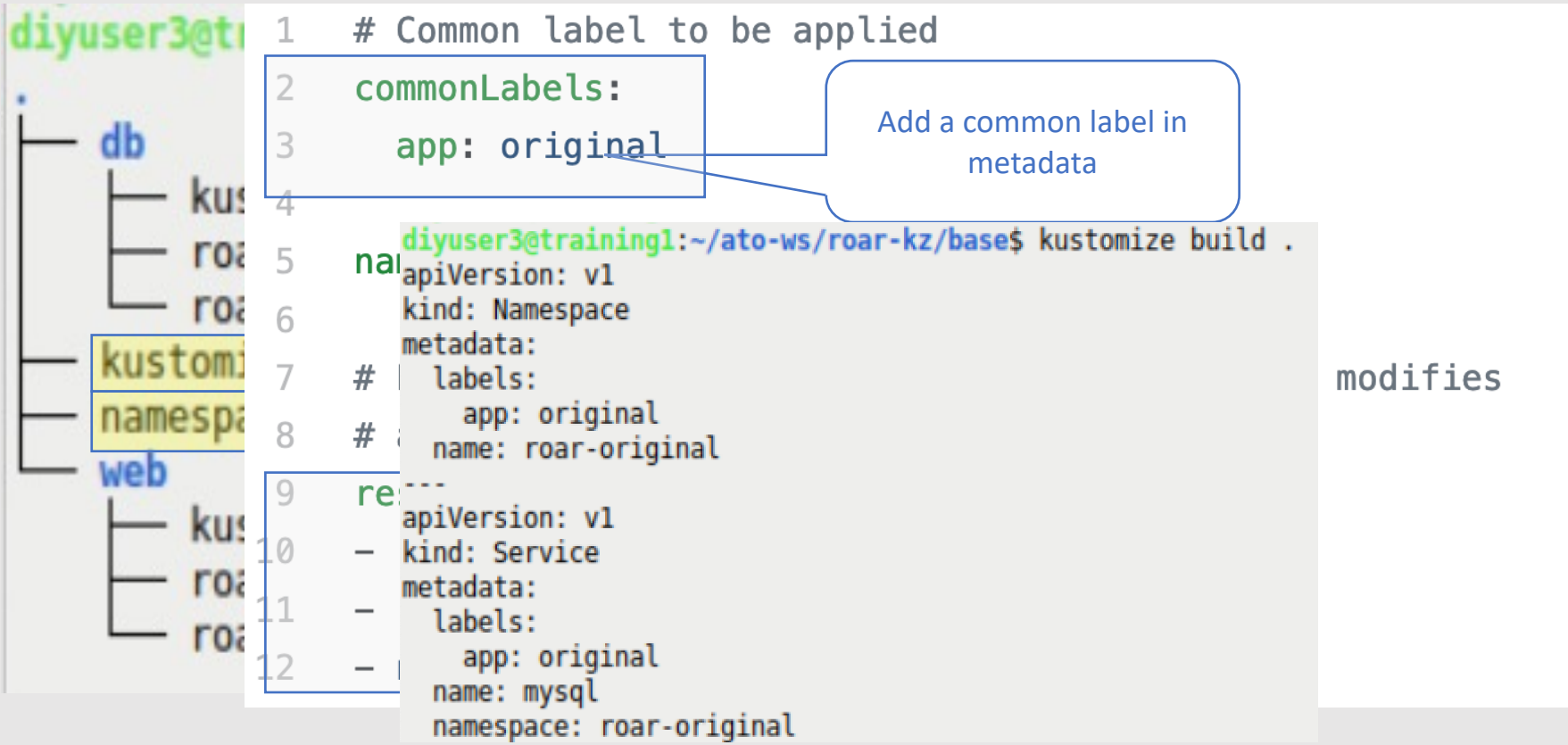
Patch block = defines patch in existing text

Type of patch = location in YAML hierarchy and replacement value

Item to search for

Kustomize Base

- Original resources + kustomization



Kustomize Base

```
diyuser3@training1:~/ato-ws/roar-kz/base/web$ kustomize build .
apiVersion: v1
kind: Service
metadata:
  labels:
    app: roar-web
  name: roar-web
spec:
  ports:
  - name: frontend
    port: 8089
    targetPort: 8080
  selector:
    app: roar-web
  type: NodePort
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    app: roar-web
  name: roar-web
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: roar-web
    spec:
      containers:
      - image: localhost:5000/roar-web:v1
        imagePullPolicy: Always
        name: roar-web
        ports:
        - containerPort: 8080
          name: web
```

```
piVersion: extensions/v1beta1
apiVersion: v1
kind: Service
metadata:
  labels:
    app: roar-web
  name: roar-web
1
2   # List of resource files that kustomize reads, modifies
3   # and emits as a YAML string
4   resources:
5   - roar-web-deploy.yaml
6   - roar-web-svc.yaml
  selector:
ays
    app: roar-web
  type: NodePort
      - containerPort: 8080
        name: web
```

Resources to include

Kustomize Feature List

Field	Type	Explanation
namespace	string	add namespace to all resources
namePrefix	string	value of this field is prepended to the names of all resources
nameSuffix	string	value of this field is appended to the names of all resources
commonLabels	map[string]string	labels to add to all resources and selectors
commonAnnotations	map[string]string	annotations to add to all resources
resources	[]string	each entry in this list must resolve to an existing resource cfg file
configmapGenerator	[]ConfigMapArgs	Each entry in this list generates a ConfigMap
secretGenerator	[]SecretArgs	Each entry in this list generates a Secret
generatorOptions	GeneratorOptions	Modify behaviors of all ConfigMap and Secret generator
bases	[]string	Each entry in this list should resolve to a directory containing a kustomization.yaml file
patchesStrategicMerge	[]string	Each entry in this list should resolve a strategic merge patch of a Kubernetes object
patchesJson6902	[]Json6902	Each entry in list should resolve to a Kubernetes obj and a Json Patch
vars	[]Var	Each entry is to capture text from one resource's field
images	[]Image	Each entry is to modify the name, tags and/or digest for one image without creating patches
configurations	[]string	Each entry in this list should resolve to a file containing Kustomize transformer configurations
crds	[]string	Each entry in this list should resolve to an OpenAPI definition file for Kubernetes types

Kustomize Option in Kubernetes

- Kubectl has an option to reference Kustomize resources
  - “-k” or ”--kustomize”
- Option needs to point to a kustomization directory
  - example: kubectl apply –k <kustomization directory>
- Multiple operations understand –k option
  - kubectl apply –k ./
  - kubectl get –k ./
  - kubectl describe –k ./
  - kubectl diff –k ./
  - kubectl delete –k ./

Lab 8 - Creating Variants

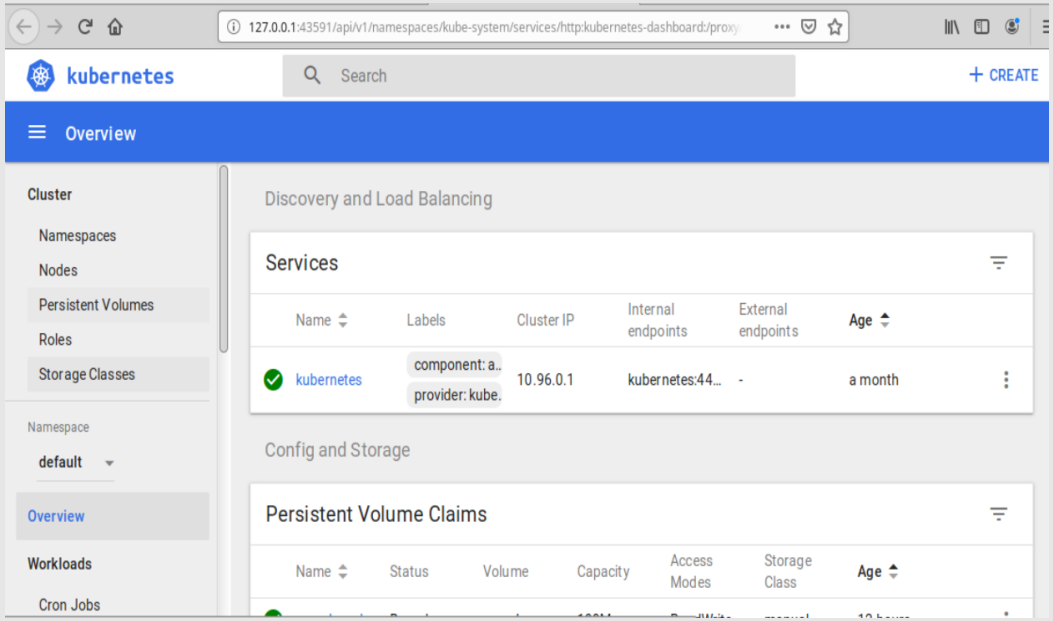
Purpose: In this lab, we'll see how to create production and stage variants of our simple application

Monitoring

Kubernetes Dashboard

- Visual interface to the cluster

- General-purpose, web-based UI for clusters
- Can get overall status about objects in the cluster
- Invoke via “minikube dashboard” or “kubectl apply” from location and then “kubectl proxy”
- Can also create or modify objects via “+ CREATE” link
  - Can type in json or yaml
  - Can upload a spec file

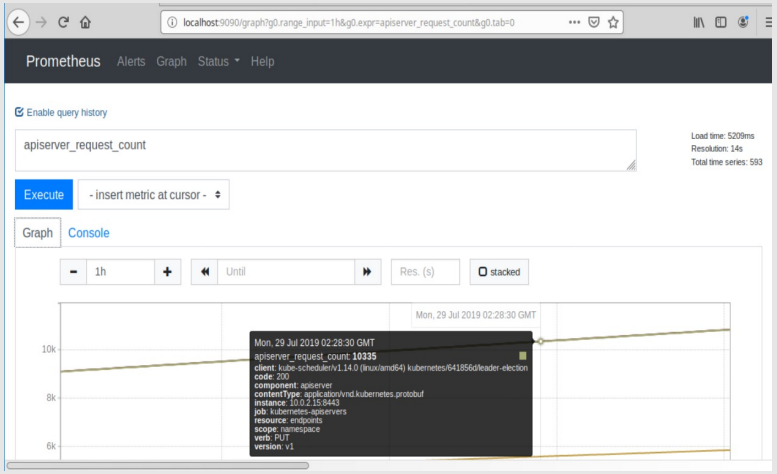


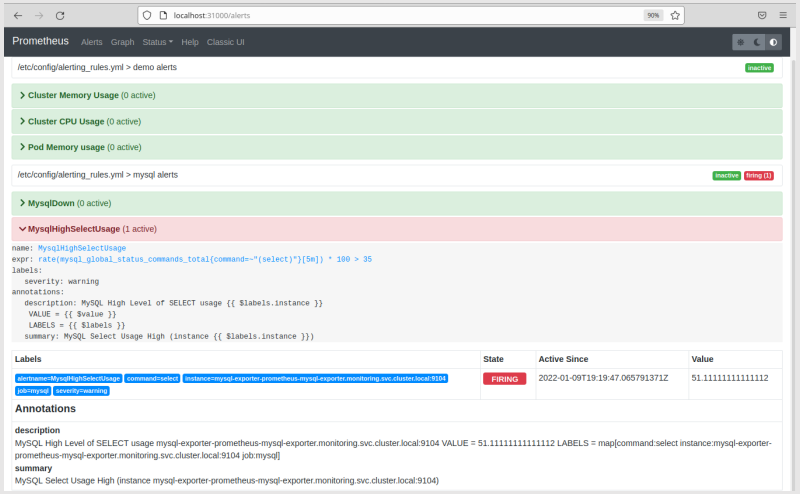
Prometheus

- Event Monitoring and Alerting

- Open-source application for event monitoring and alerting
- Records realtime metrics in a time-series database
- Powerful query language - PromQL
- Prometheus “monitoring platform” usually has
  - Multiple exporters that typically run on the monitored host to export metrics
  - Prometheus to centralize and store the metrics
  - Alertmanager to trigger alerts based on the metrics
  - Grafana to produce dashboards

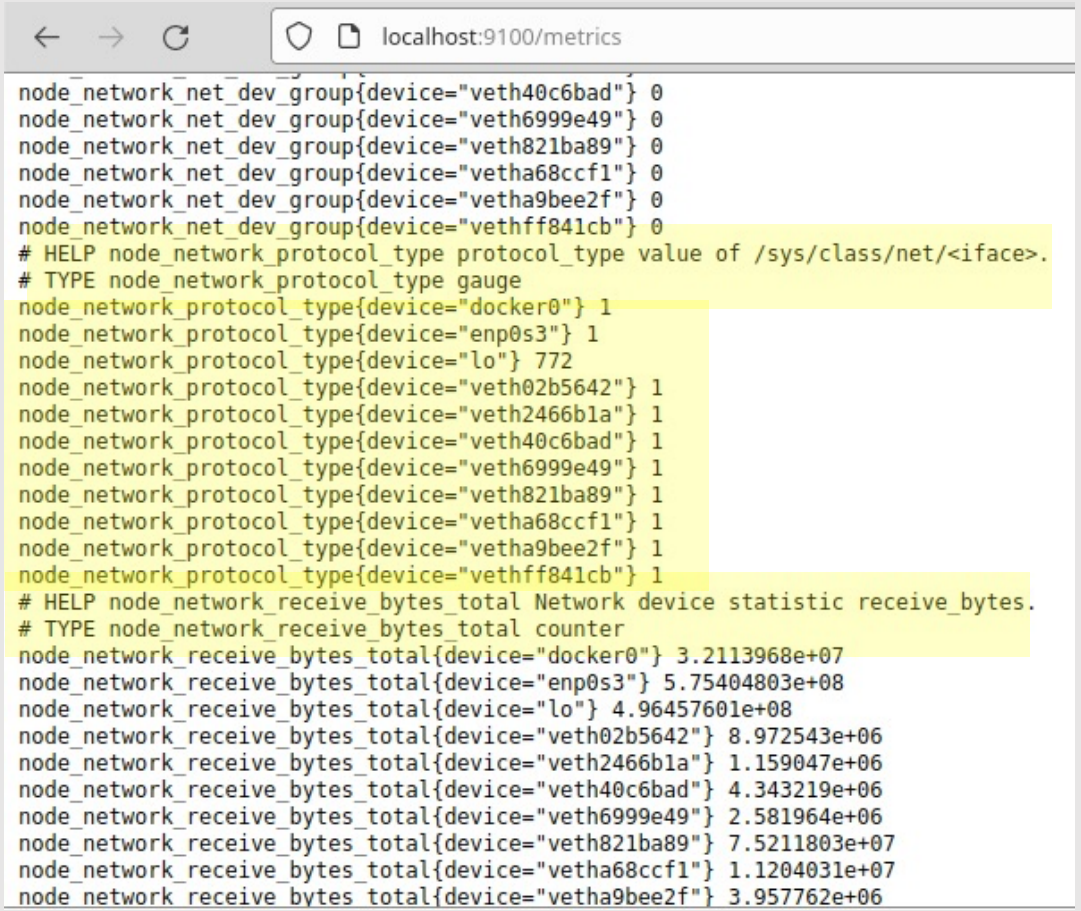
https://en.wikipedia.org/wiki/Prometheus_(software)





Example Node Exporter Metrics

- Gauge
- Counter
- Multiple interfaces shown



Metric types

Metric Type	Description
Counter	Cumulative metric representing a single increasing value
Gauges	Single numerical value which can either increase or decrease
Histograms	Samples values and counts them in configurable buckets
Summaries	Samples values and also provides sum and total counts

How Prometheus is intended to be used

- Can be used to monitor most anything and analyze performance
  - Databases, VMs, servers, apps
- Works well for collecting and evaluating metrics
  - Numeric time series
- Good for dynamic service-oriented architectures and machine-centric monitoring
- Supports multi-dimensional data collection and querying
- Designed for reliability
- Each Prometheus server is standalone
  - Not dependent on remote services or network storage

Prometheus query examples

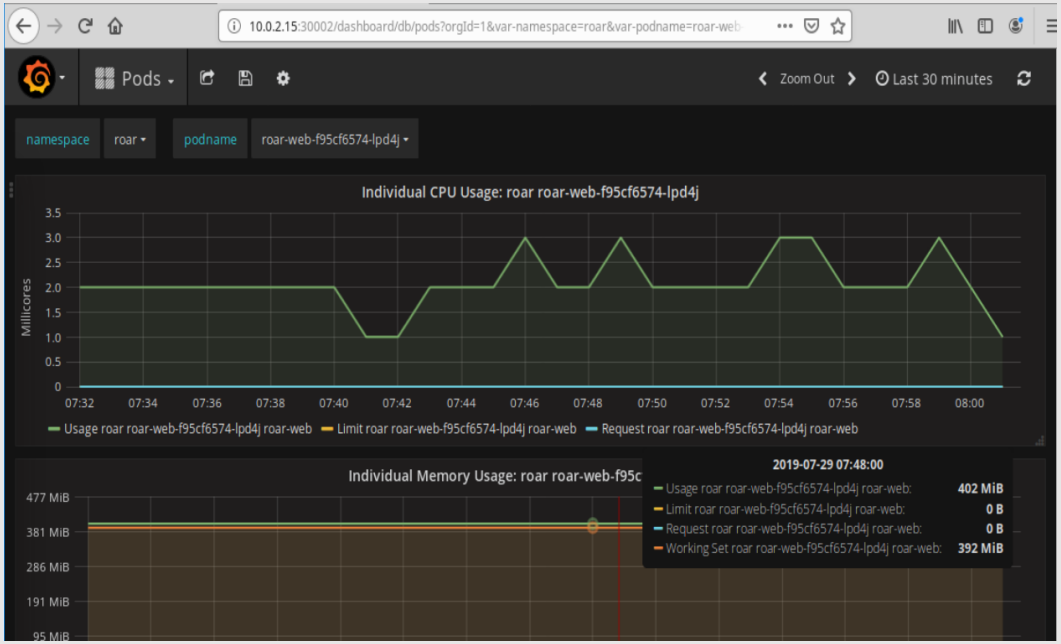
- count of pods per cluster and namespace
  - sum by (namespace) (kube_pod_info)
- number of containers by cluster and namespace without CPU limits (detects containers w/ no cpu limits)
  - count by (namespace)(sum by (namespace,pod,container)(kube_pod_container_info{container!=""}) unless sum by (namespace,pod,container)(kube_pod_container_resource_limits{resource="cpu"}))
- pod restarts by namespace
  - sum by (namespace)(changes(kube_pod_status_ready{condition="true"}[5m]))
- pods not ready
  - sum by (namespace) (kube_pod_status_ready{condition="false"})
- cpu overcommit
  - sum(kube_pod_container_resource_limits{resource="cpu"}) - sum(kube_node_status_capacity_cpu_cores)
- memory overcommit
  - sum(kube_pod_container_resource_limits{resource="memory"}) - sum(kube_node_status_capacity_memory_bytes)
- number of ready nodes per cluster
  - sum(kube_node_status_condition{condition="Ready", status="true"}==1)
- nodes readiness flapping
  - sum(changes(kube_node_status_condition{status="true",condition="Ready"}[15m])) by (node) > 2
- cpu idle by cluster
  - sum((rate(container_cpu_usage_seconds_total{container!="POD",container!=""}[30m]) - on (namespace,pod,container) group_left avg by (namespace,pod,container)(kube_pod_container_resource_requests{resource="cpu"})) * -1 >0)
- memory idle by cluster
  - sum((container_memory_usage_bytes{container!="POD",container!=""} - on (namespace,pod,container) avg by (namespace,pod,container)(kube_pod_container_resource_requests{resource="memory"})) * -1 >0) / (1024*1024*1024)

- abs()
- absent()
- absent_over_time()
- ceil()
- changes()
- clamp()
- clamp_max()
- clamp_min()
- day_of_month()
- day_of_week()
- days_in_month()
- delta()
- deriv()
- exp()
- floor()
- histogram_quantile()
- holt_winters()
- hour()
- idelta()
- increase()
- irate()
- label_join()
- label_replace()
- ln()
- log2()
- log10()
- minute()
- month()
- predict_linear()
- rate()
- resets()
- round()
- scalar()
- sgn()
- sort()
- sort_desc()
- sqrt()
- time()
- timestamp()
- vector()
- year()
- <aggregation>_over_time()
- Trigonometric Functions

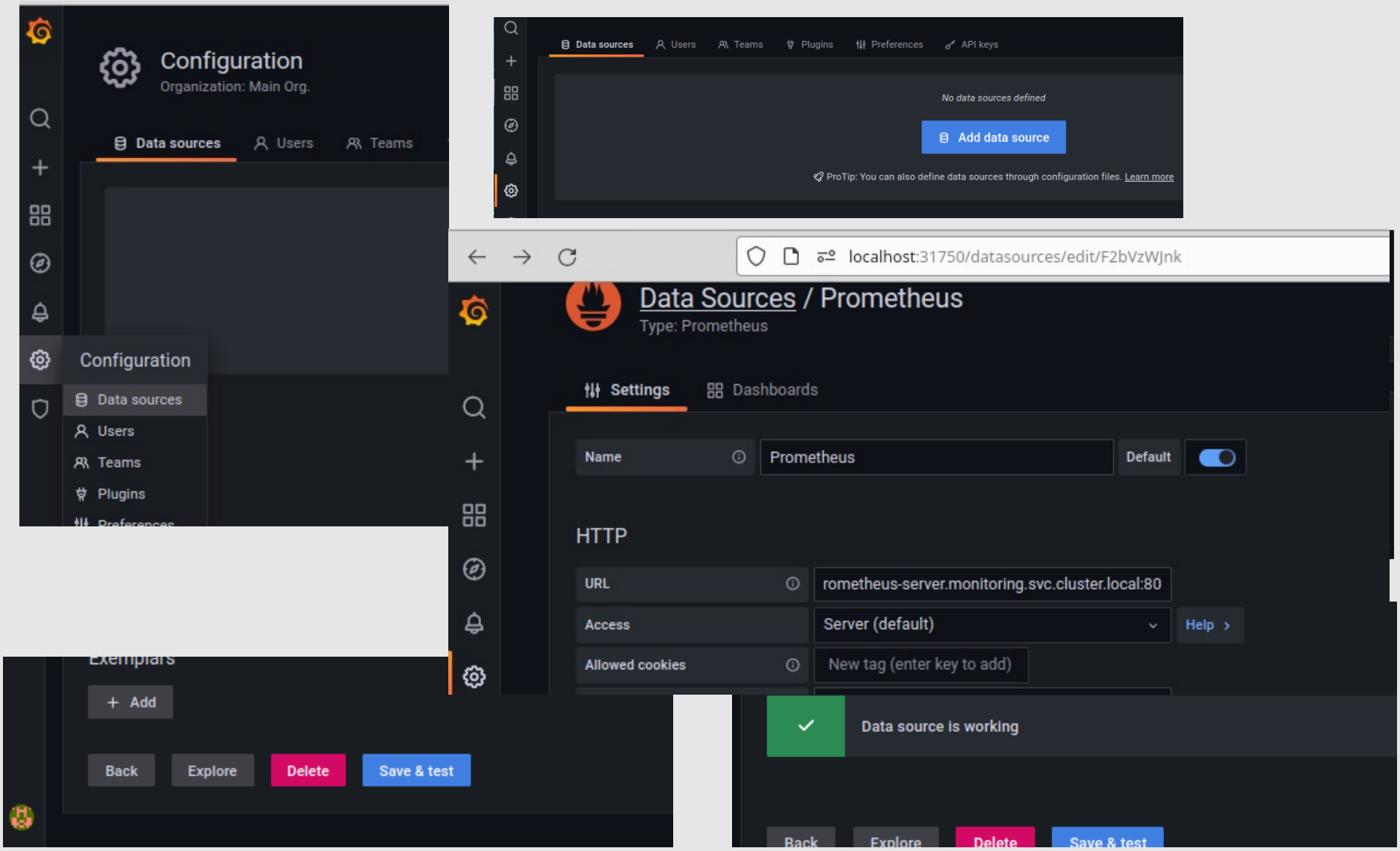
Grafana

- Metric analytics and visualization suite

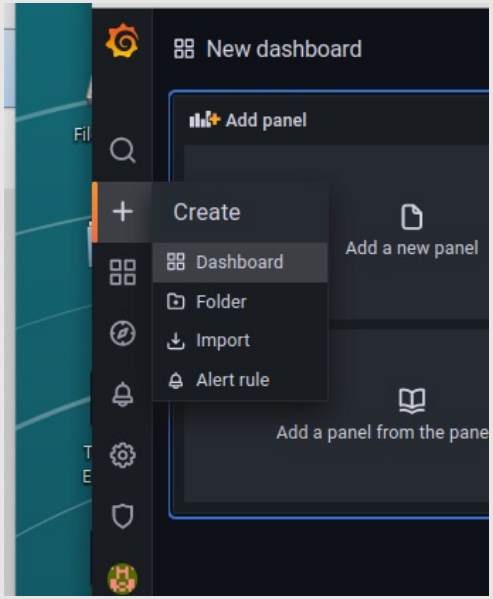
- Way to visualize time series for infrastructure and application analytics
- Allows users to build dashboards with graphs, panels, etc.
- Accepts data from many data sources including:
  - Graphite
  - Elasticsearch
  - InfluxDB
  - Prometheus
  - MySQL
  - Postgres
  - Cloudwatch

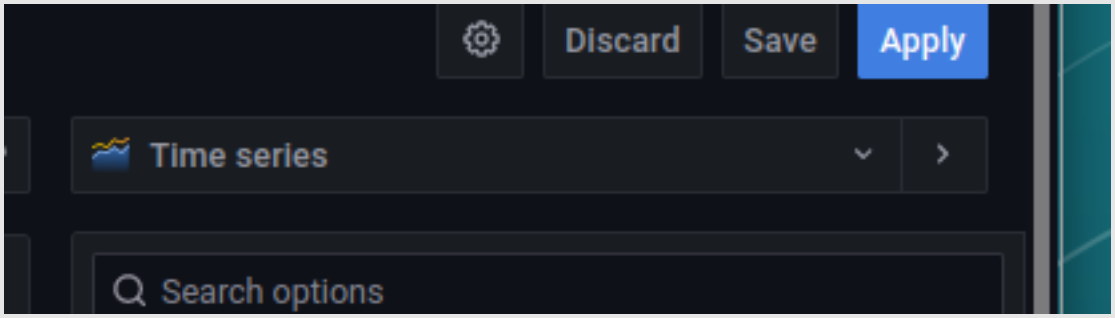


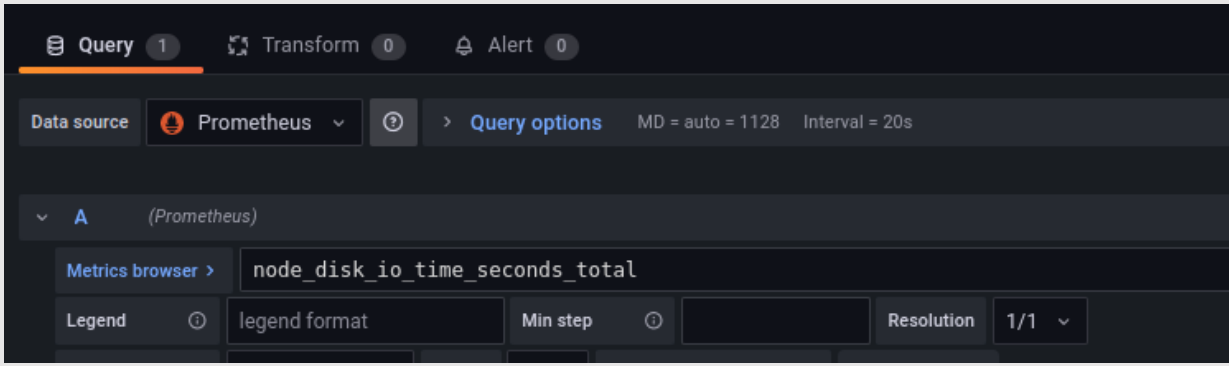
Adding a Data Source

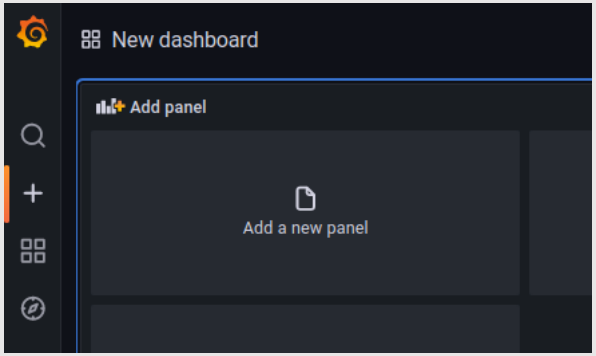


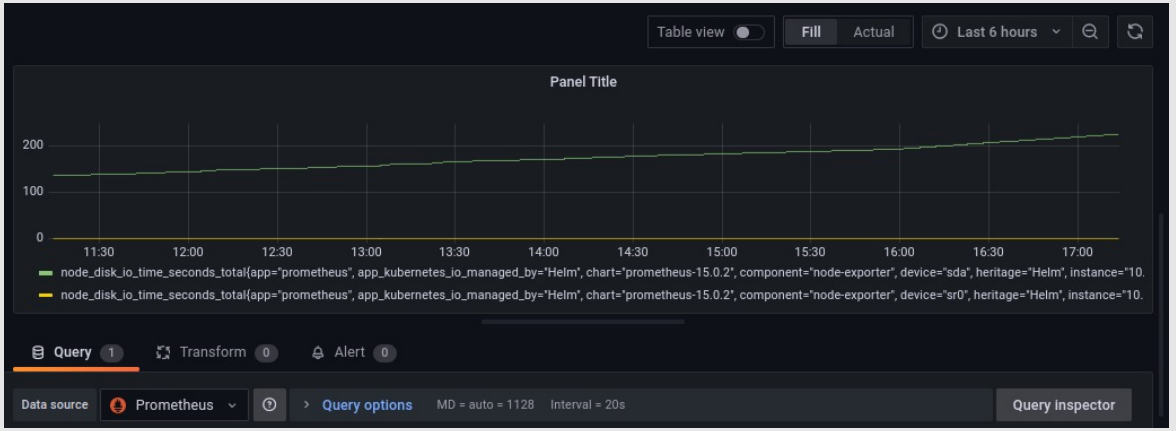
Creating a Dashboard



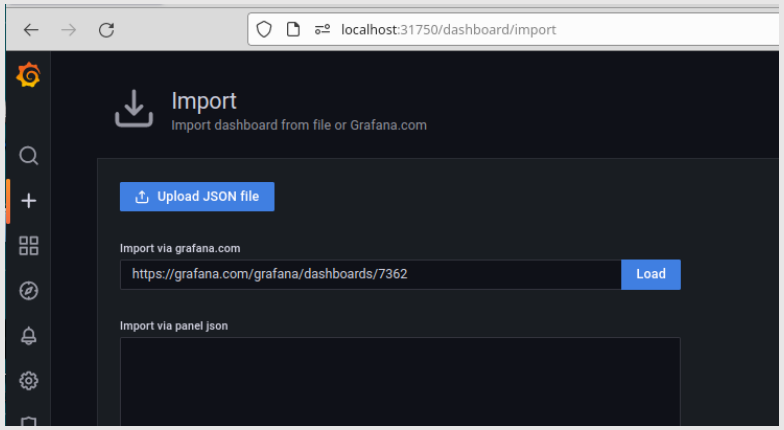


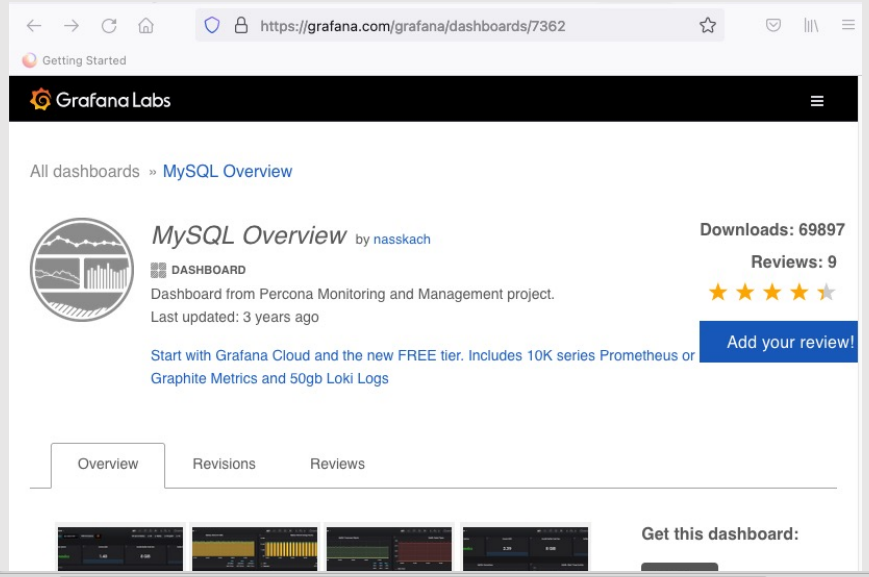


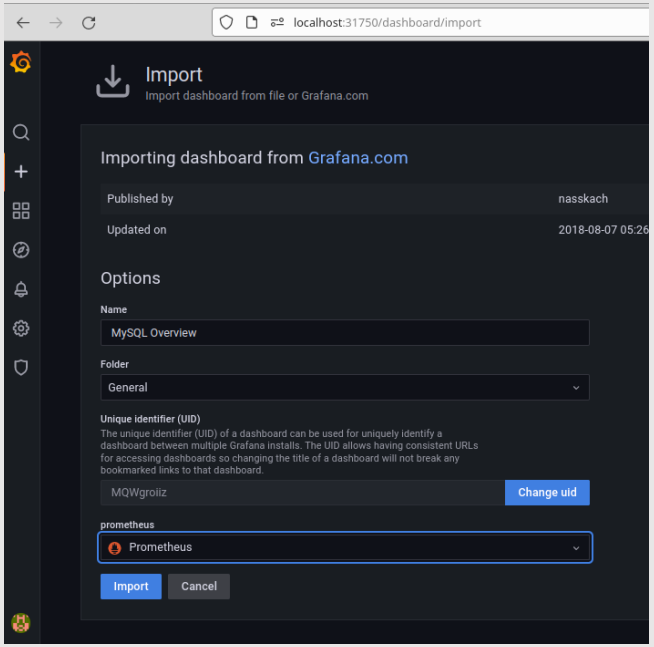


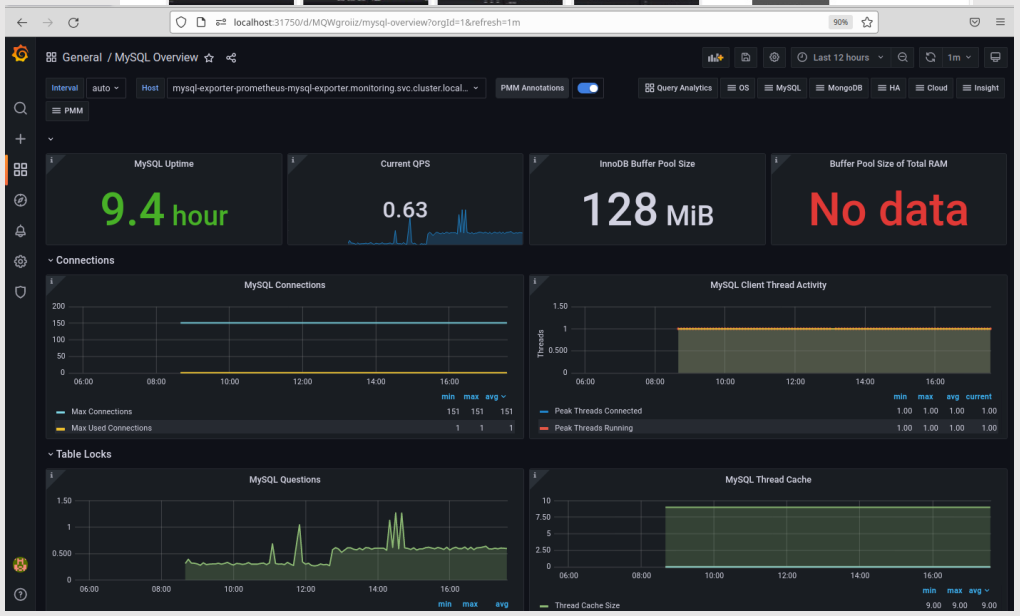


Importing a Dashboard



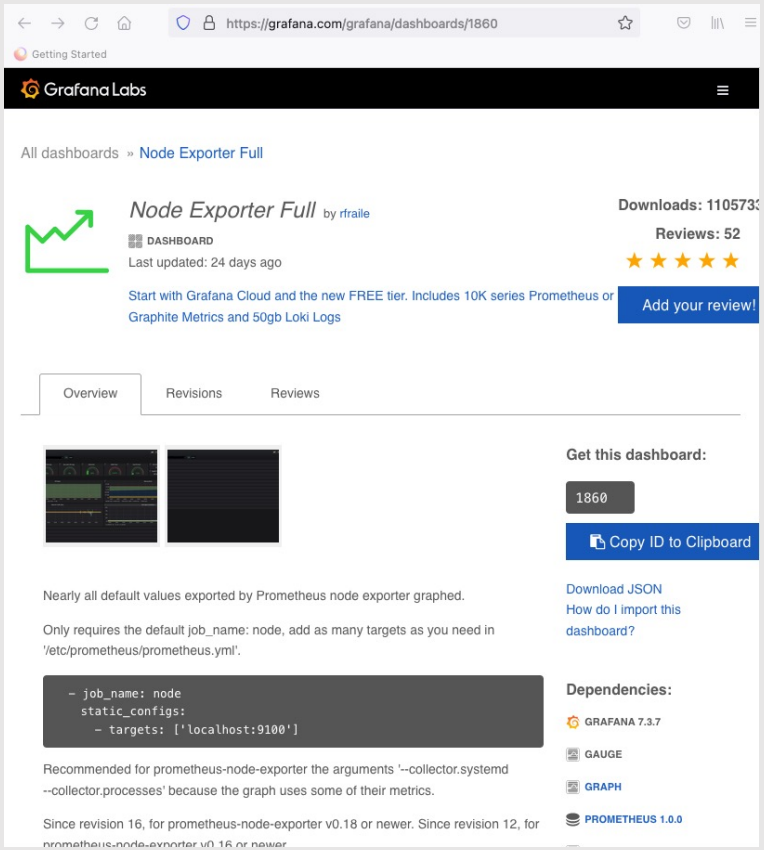


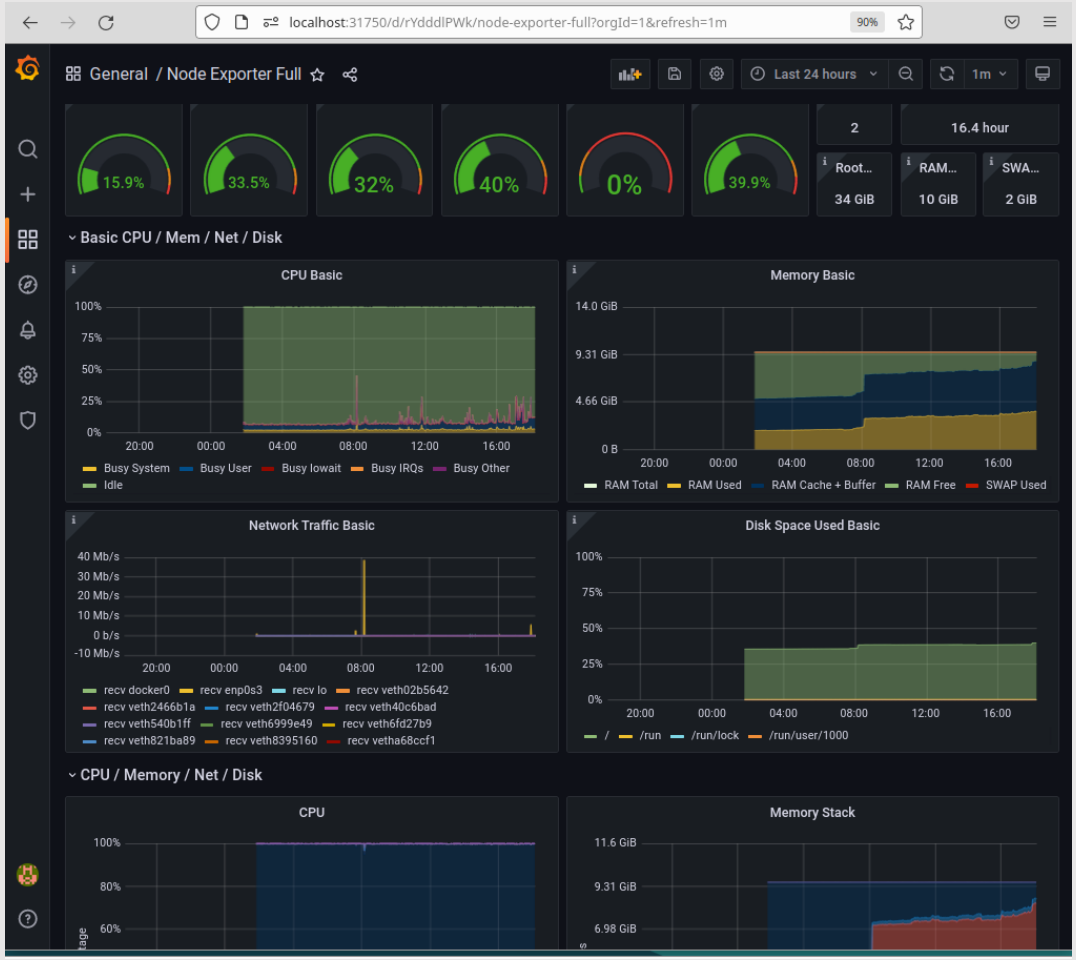




Additional Dashboards Available

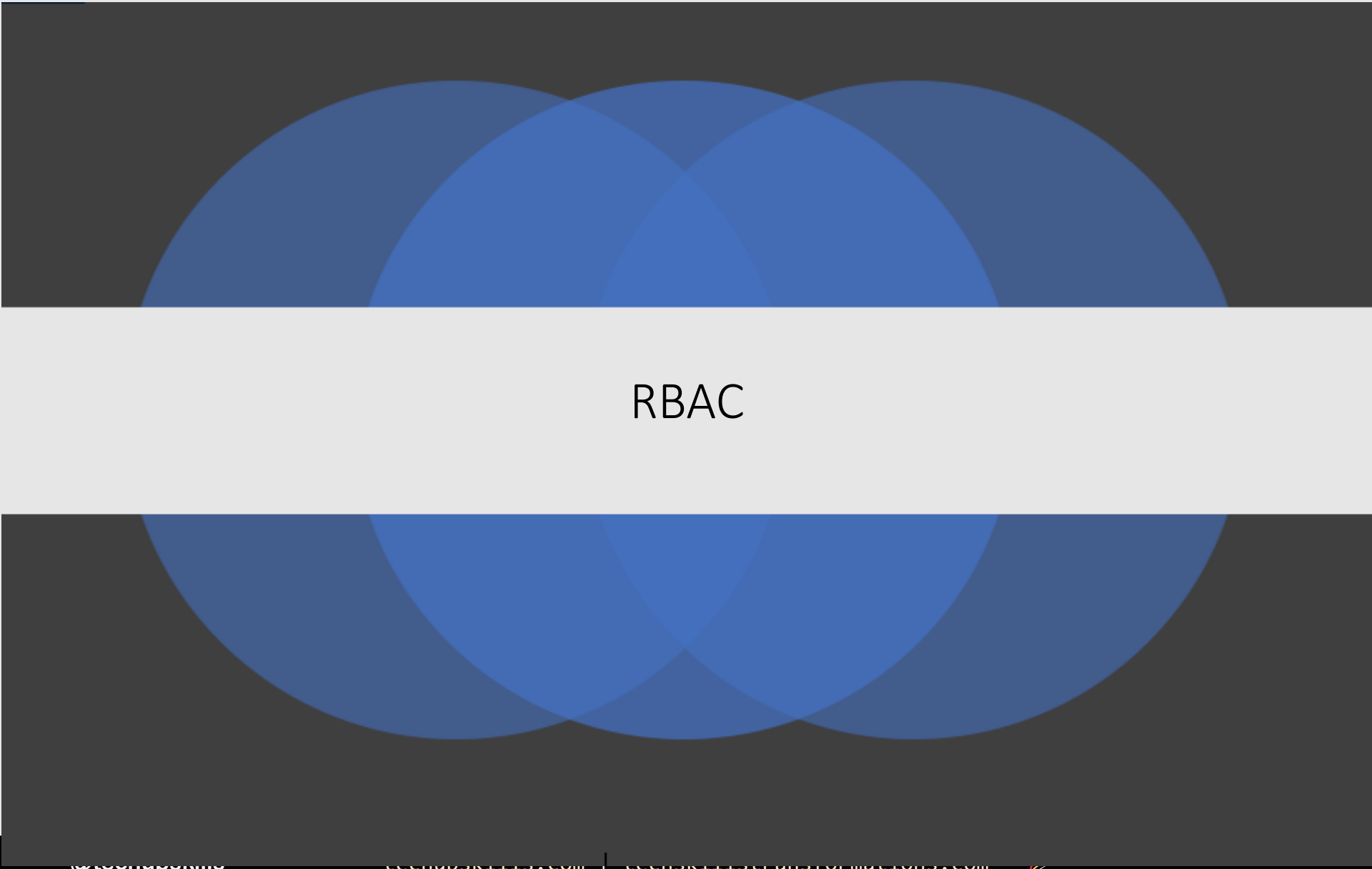
- Node Exporter Full





Lab 9 - Monitoring

Purpose: This lab will introduce you to a few of the ways we can monitor what is happening in our Kubernetes cluster and objects.

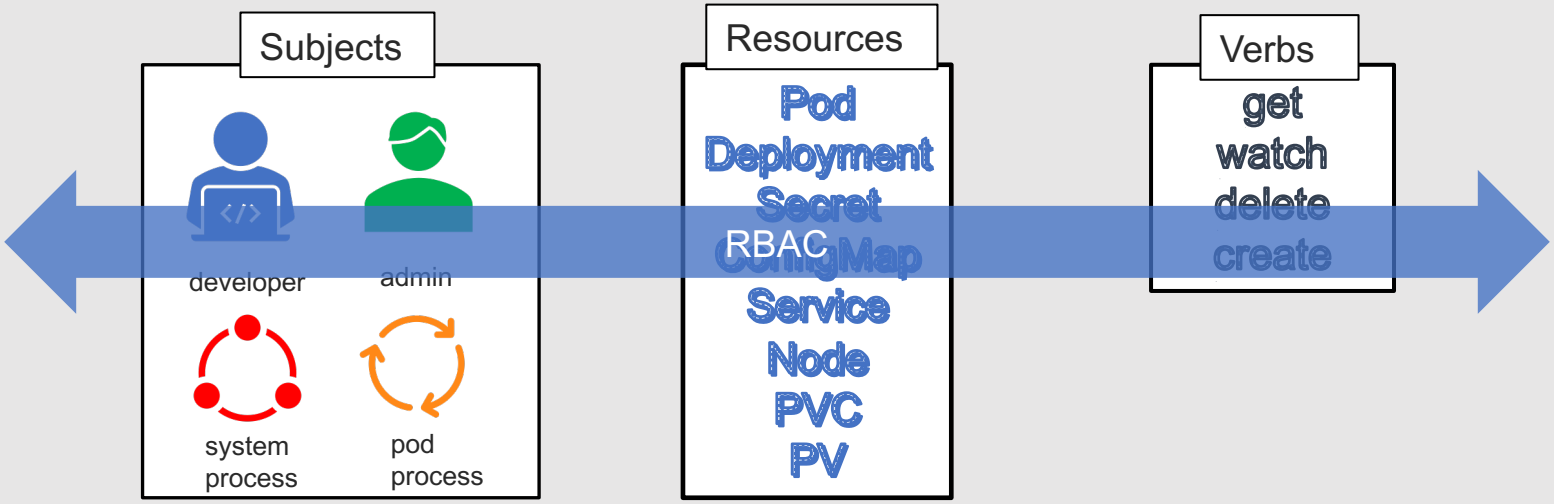

RBAC

Kubernetes RBAC

- Role-based Access Control
  - Why do we need RBAC ? Because in a real-world k8s environment, we may need to:
    - Have multiple users with different properties, establishing a proper authentication mechanism.
    - Have full control over which operations each user or group of users can execute.
    - Have full control over which operations each process inside a pod can execute.
    - Limit the visibility of certain resources of namespaces.

Kubernetes RBAC

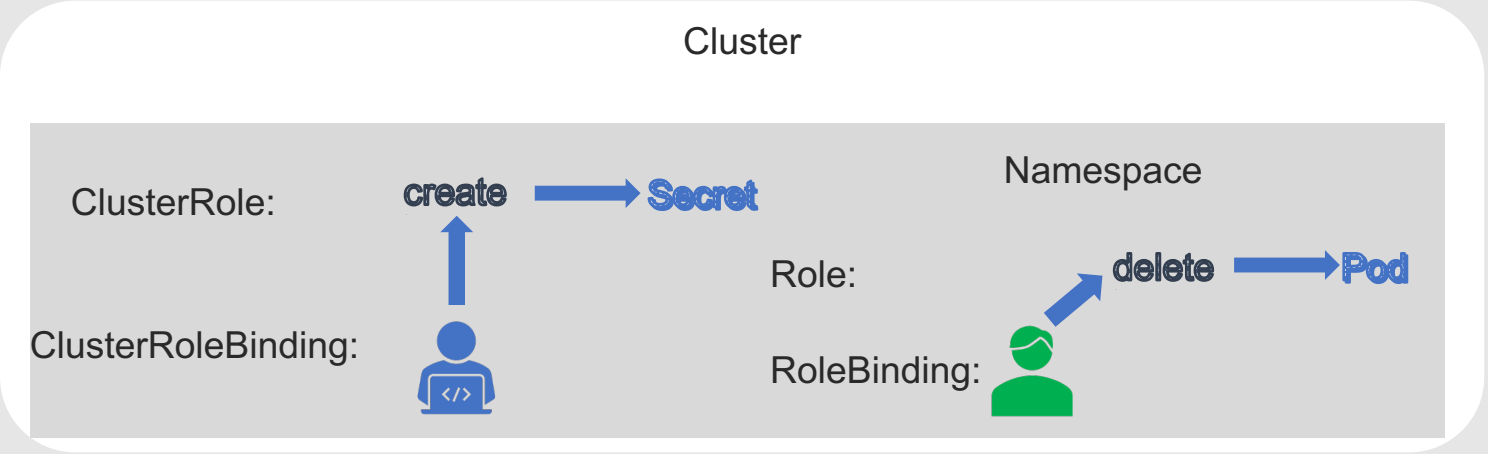
- The pieces
  - 3 types of elements involved
    - Subjects: users or processes that need to access the API
    - Resources: Kubernetes API Objects available in the cluster. (Examples include Pods, Deployments, Services, Nodes, and Persistent Volumes, etc.)
    - Verbs: The Kubernetes operations that can be executed on the resources above. Multiple kinds of verbs available – all are CRUD operations (Examples include get, watch, create, delete, etc.)



Goal: Connect subjects, API resources, and verbs. Specify, given a user, which operations can be executed over which resources.

Kubernetes RBAC

- The mechanisms
  - Object types
    - k8s Roles: Connect API Resources and Verbs; can be reused for different Subjects; bound to a namespace
    - k8s RoleBinding: Connects Subjects to Roles; Given a role that combines API objects and verbs, defines which subjects can use it.
    - k8s ClusterRole: a role to be applied across a cluster
    - k8s ClusterRoleBinding: Connects Subjects to ClusterRole.

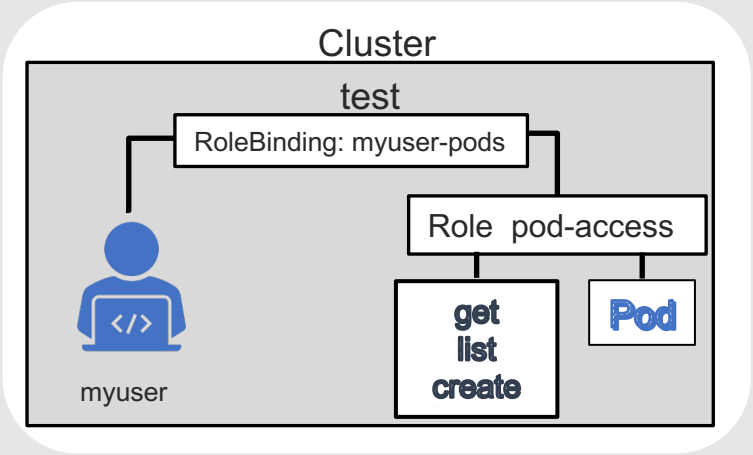


Kubernetes RBAC

- Example: give user limited access to work with pods in test ns
  - Specs define role and rolebindings

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-access
  namespace: test
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - get
  - list
  - create
```

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: myuser-pods
  namespace: test
subjects:
- kind: User
  name: myuser
roleRef:
  kind: Role
  name: pod-access
  apiGroup: rbac.authorization.k8s.io
```



  - User can do these operations:
    - kubectl get pods –n test
    - kubectl describe pod –n test <name of pod>
    - kubectl create –f <pod-spec.yaml> -n test
  - But not these
    - kubectl get p[illegible]efault
    - kubectl get p[illegible]n test

Kubernetes RBAC - Service Accounts

- Users: These are global, and meant for humans or processes living outside the cluster.
- ServiceAccounts: These are namespaced and meant for intra-cluster processes running inside pods.
- Both authenticate against the API to get access to resources
- But there is no kubernetes “User” object
- So you can do “kubectl create serviceaccount <name>” but not “kubectl create user <name>”
- Bottom line: cluster does not store any information about users, so must be managed outside of cluster with certificates, tokens, etc.

Create ServiceAccount

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: myapp
  labels:
    app: myapp
```

Create Role

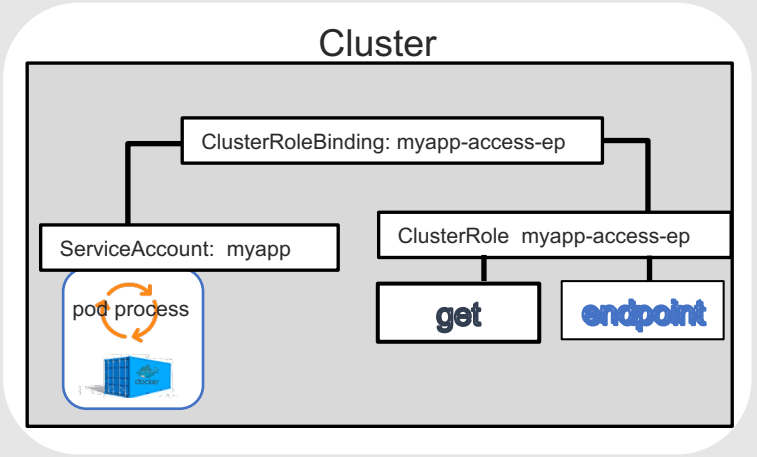
```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: myapp-access-ep
  labels:
    app: myapp
rules:
- apiGroups: [""]
  resources: ["endpoints"]
  verbs: ["get"]
```

Add ServiceAccount to Pod

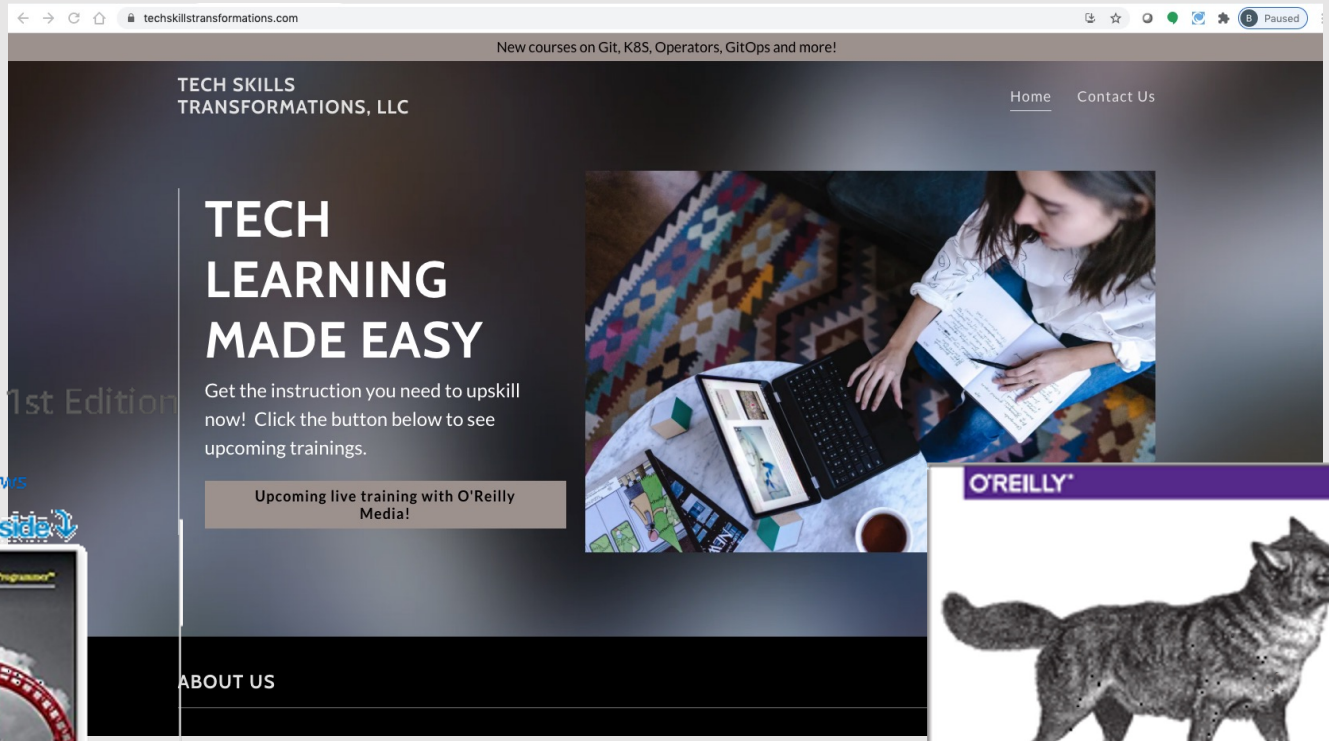
```yaml
[...]
spec:
  template:
      metadata:
        name: myapp
        labels:
          app: myapp
      spec:
        serviceAccountName: myapp
        containers:
[...]
```

Tie Role to ServiceAccount

```yaml
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: myapp-access-ep
  labels:
    app: myapp
subjects:
- kind: ServiceAccount
  name: myapp
roleRef:
  kind: ClusterRole
  name: myapp-access-ep
  apiGroup: rbac.authorization.k8s.io
```



That's all - thanks!



techskillstransformations.com
getskillsnow.com