NON ROCKET SCIENCE KUBERNETES

Manuel Calvo

OBJECTIVES

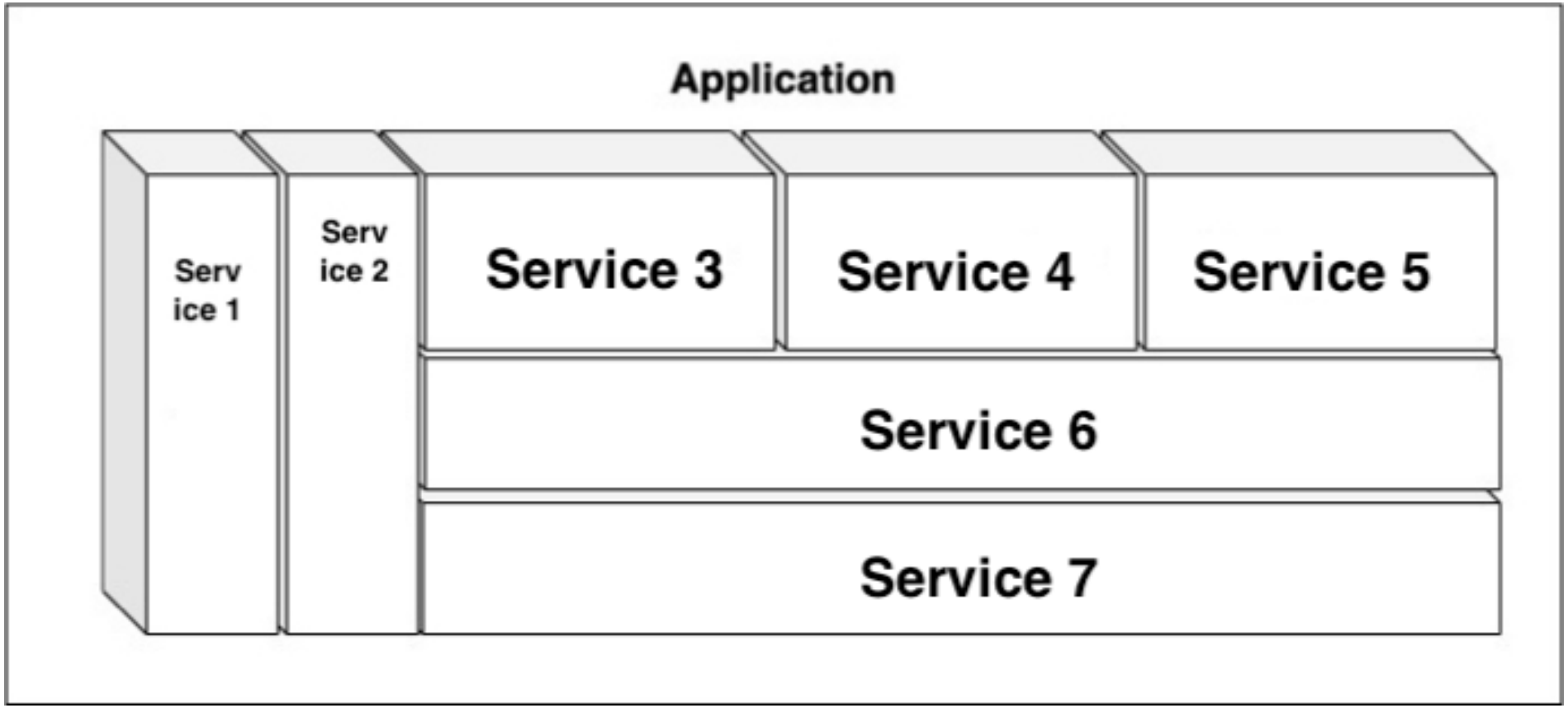
We are not time constrained, we will cover as much as we can in 5 weeks

We are not here just to say I did the course, we are here to help you get started using K8s

Will try to make it hands on as possible

Please interrupt me if you have a question

THE MONOLITH



THE MONOLITH / ADVANTAGES

Only one piece of software: easier testing, trouble shoot, monitoring and management

Need less network traffic

Easier to achieve data consistency

THE MONOLITH / DISADVANTAGES

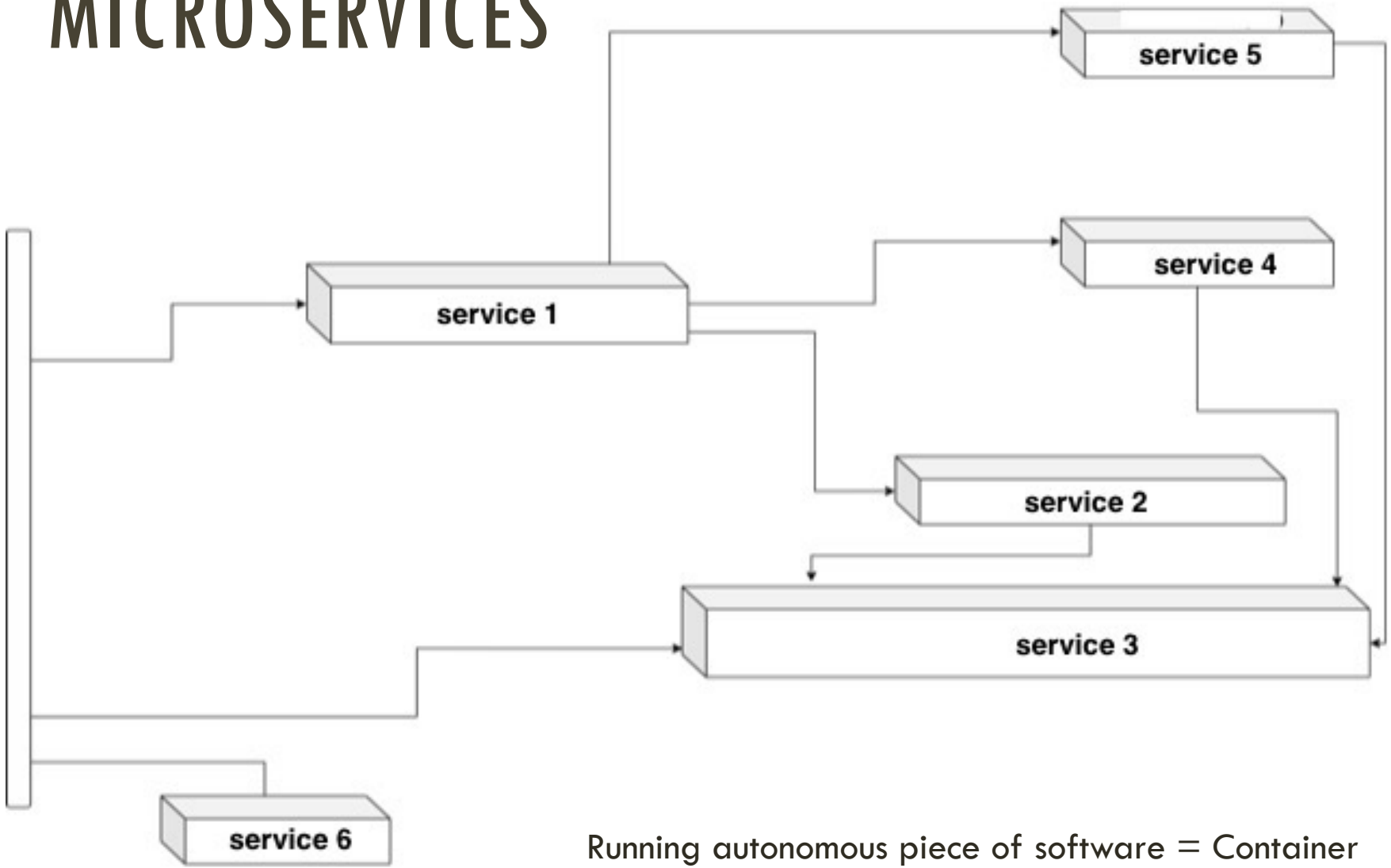
Can’t scale individual components

Managing a single piece of code by a large team can become complex and prompt for bottle necks

Harder to mix / incorporate different technologies

Slower time to market

MICROSERVICES



"Microservices are small, autonomous services that work well together"

- Sam Newman

These autonomous pieces of software run in what is known as containers

Running autonomous piece of software = Container

MICROSERVICES / ADVANTAGES

Can scale components individually

Easier to adopt new technologies

Easier to distribute workload

Faster time to market

MICROSERVICES / DISADVANTAGES

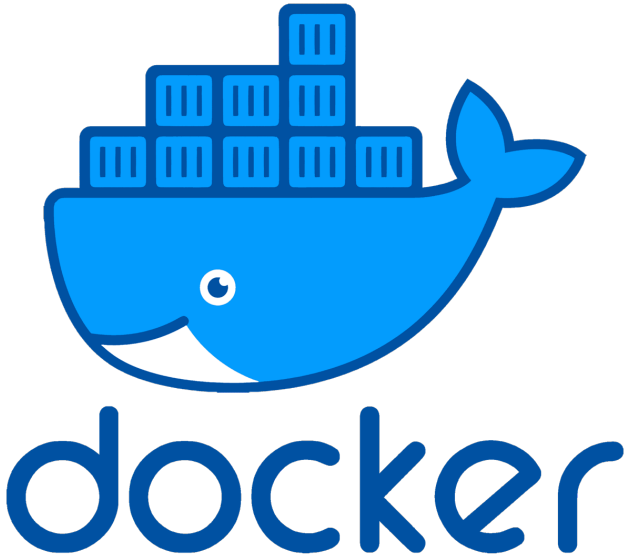
Harder to test, trouble shoot, monitor and manage

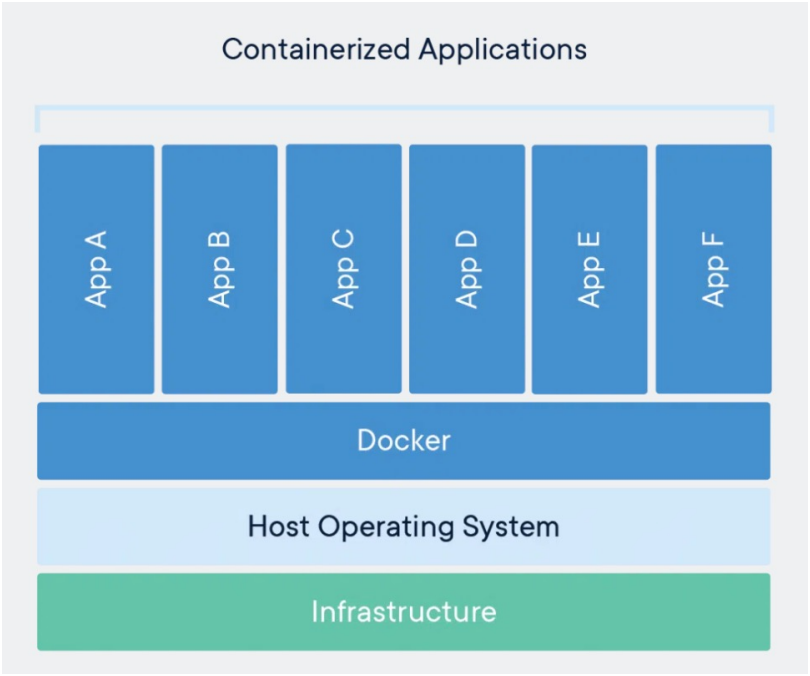
Require additional network traffic

Harder to achieve data consistency

More expensive to run

IMAGES AND CONTAINERS





Docker: the most famous tool to work with containers, but not the only option in the market

OCI (Open Container Inititiave) = creating open industry standards around container formats and runtimes

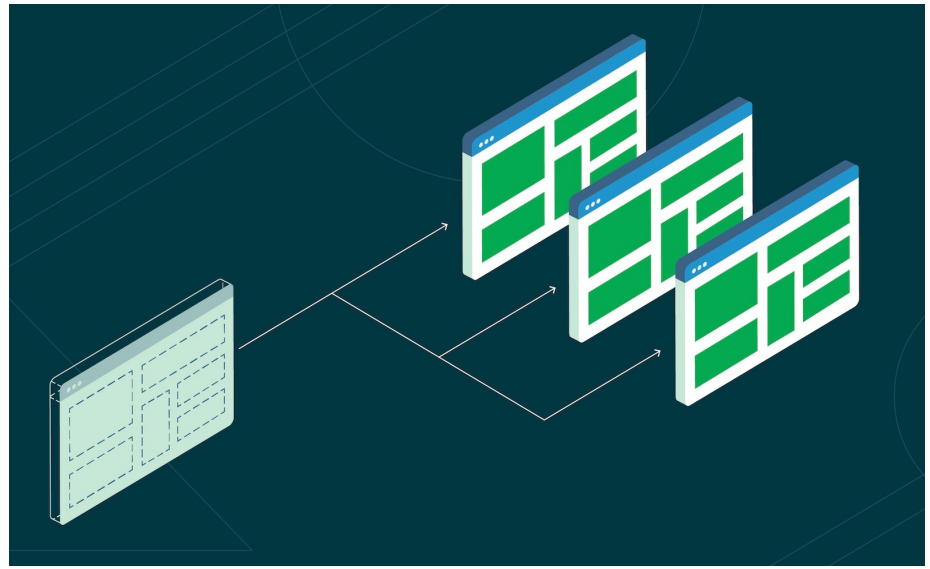
IMAGES AND CONTAINERS

Image: is the actual sofware we run, it needs to be build in a format containers can manage (OCI standard format)

Container: an actual running instance of the sofware packaged as an image.

"Think of a container as a shipping container for software — it holds important content like files and programs so that an application can be delivered efficiently from producer to consumer"

- https://circleci.com/blog/docker-image-vs-container



Why this picture ?

IMAGES AND CONTAINERS

Containers registries

Some are public some are private

All software that runs in a K8s cluster, runs inside a container and its downloaded from a container registry

https://hub.docker.com/search

- ie: mysql

- 192.168.99.100:5000/my-image
- myregistry.azurecr.io/marketing/campaign10-18/email-sender:v2
- hello-world:latest

LAB: SEARCH DOCKER HUB

https://hub.docker.com/search

Search docker hub and give us an example of a container image that caught your attention

Why that image ?

What version you think you will need ?

What is its image URI ?

WHY K8S ?

Besides just running the software in the containers, K8s adds additional features:

Self healing (ie: if a containers goes down or unresponsive, it will automatically restart it for us, will reclaim resources based on priorities)

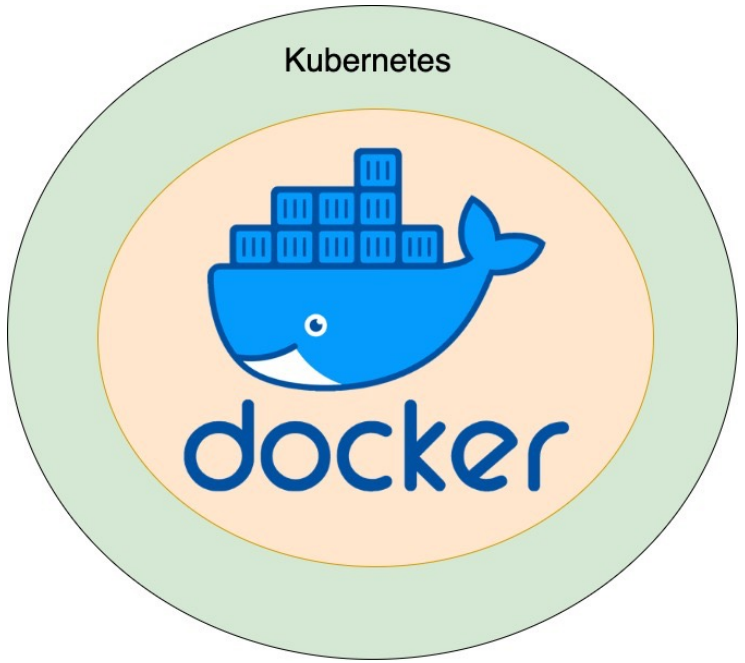
Cluster, multi node (unlike Docker, where a single host manages all the containers, Kubernetes clusters distribute the workload across multiple machines)

Security (defines who and what can be accessed)

Sharing information

Provide ways to organize large deployments (multiple cluster, partition clusters, distribute loads)

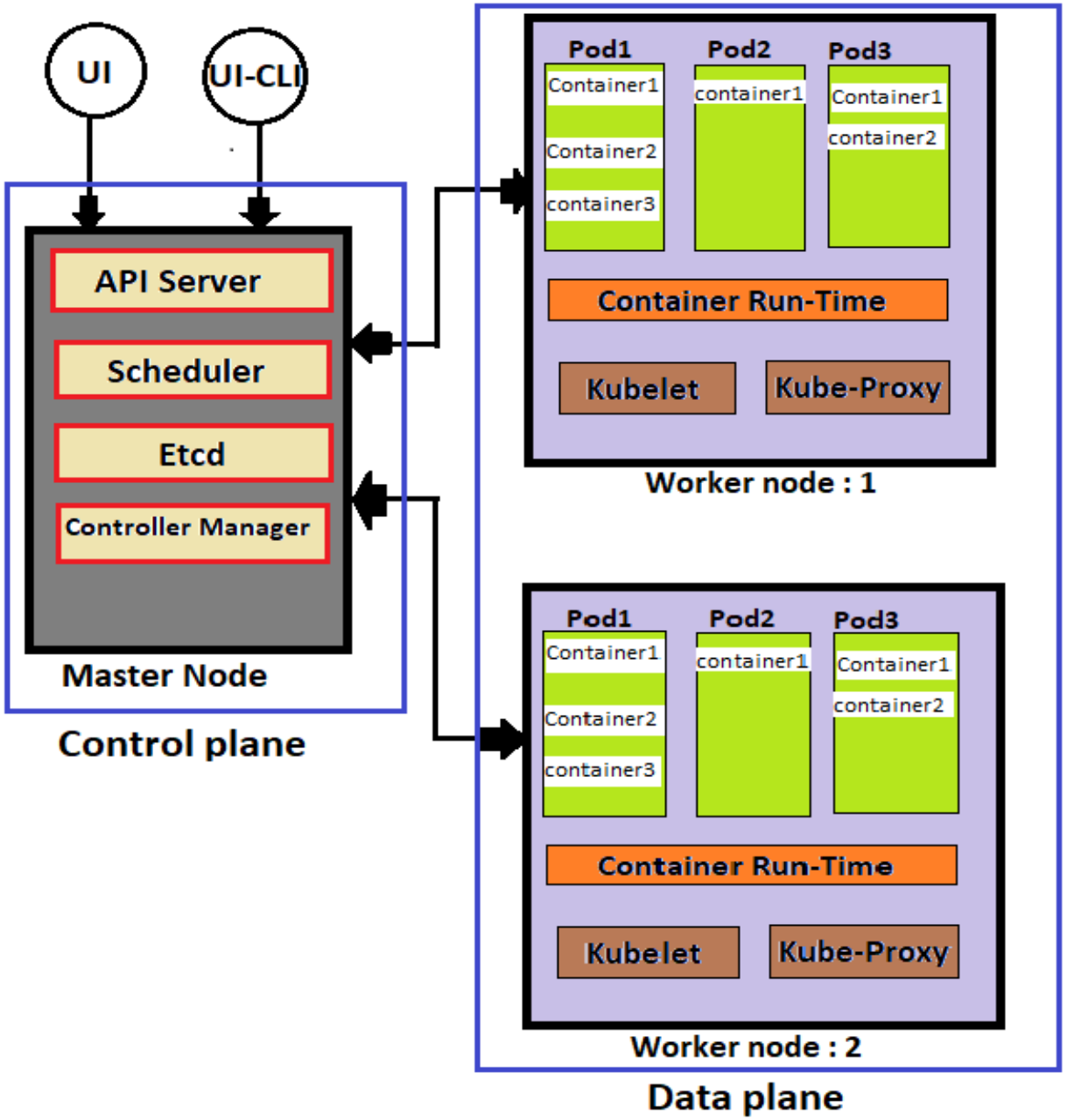
Declarative (you just tell K8s what you want, and it will determine what is the best way to achieve that request)



More than just running containers, K8s is a container orquestrator

K8S ARCHITECTURE

Kubernetes consists of two main parts: the **control plane (master node)** and the **data plane (worker node)**. Each part consists of multiple components



https://devopscube.com/kubernetes-architecture-explained/

K8S ARCHITECTURE / CONTROL PLANE

API Server: acts as the central communication hub within the Kubernetes cluster. End users, and other cluster components, talk to the cluster via the API server. When you use *kubectl* you are actually communicating with the API server through HTTP REST APIs. It is the only component that communicates with etcd

Etcd: serves as the backing store for the entire cluster's information. It stores the configuration and state data of the cluster

Scheduler: is responsible for assigning tasks, such as pods to worker nodes based on available resources and constraints. It ensures optimal resource utilization and load balancing across the cluster. It is a controller

Controller Manager: ensures that the cluster remains in the desired configuration at all times. Monitors the state of the cluster and takes corrective actions if anything goes wrong. Some built-in controllers: replicaset, daemonset, cronjob. You can extend kubernetes with custom controllers associated with a custom resource definition (CRDs, Operator pattern)

K8S ARCHITECTURE / WORKER NODE

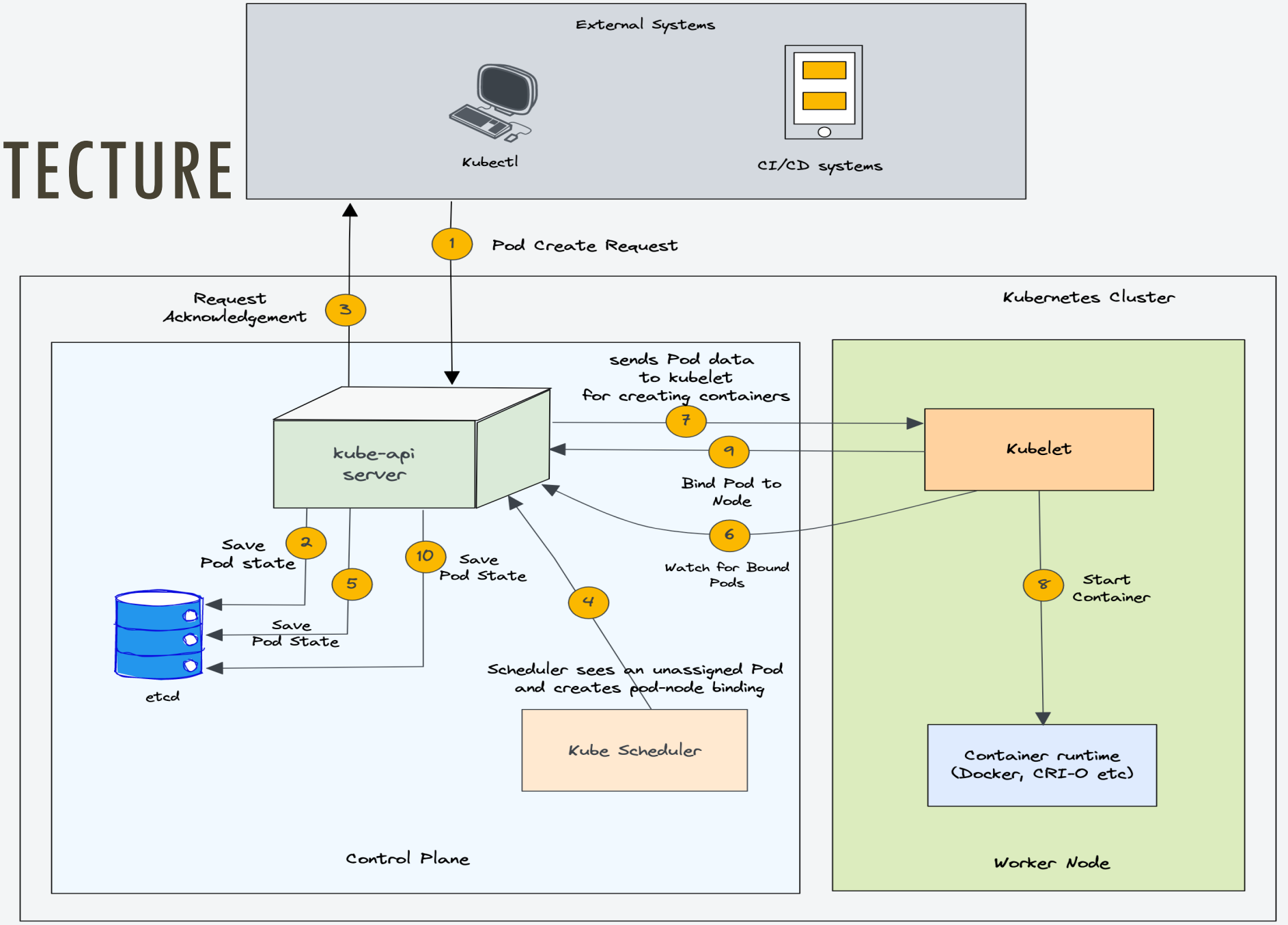
Kubelet: manages the state of individual worker nodes. It ensures that pods are created and running on the node. If a pod fails, the kubelet informs the API server, which then instructs the controller manager to regenerate the container. Mounts volumes. It manages Pods networking through the Container Networking Interface (CNI)

Kube-proxy: handles networking tasks within *services* in the cluster. It generates IP addresses and performs load balancing to enable communication. It utilizes IP tables on the Linux machine to configure networking

Container Runtime: is the software responsible for running and managing containers. It provides the environment in which containers can execute, including managing the lifecycle of containers, resource allocation, and isolation

- **Container Runtime Interface (CRI):** It is a set of APIs that allows Kubernetes to interact with different container runtimes. It allows different container runtimes to be used interchangeably with Kubernetes

K8S ARCHITECTURE



LAB: INSTALL MINIKUBE

Minikube is a lightweight Kubernetes implementation that allows to run a simple cluster on your local machine. There are other alternatives like: k3s, kind

https://minikube.sigs.k8s.io/docs/start/

macOS

- brew install minikube

Windows

- winget install minikube

LAB: INSTALL MINIKUBE

minikube config set memory 8g

minikube config set cpus 4

minikube config set driver hyperkit (only MacOs)

minikube config set container-runtime docker

minikube -p learn start **--kubernetes-version=v1.18.6** --nodes=2

LAB: KUBECTL / CHECKING INSTALLATION

What is kubectl?

- alias k=kubectl
- nano ~/.zshrc or ~/.bashrc
- source ~/.zshrc

```
// Gets K8s version number
kubectl version --output=json

// Gets list of types of resources installed in the cluster
kubectl api-resources
```

LAB: KUBECTL / CHECKING INSTALLATION

```
// Gets list of resources in default namespace
kubectl get all
```

```
// List of resources of a specific type
kubectl get <resource_type or short_name> -o wide --watch
```

- kubectl get nodes –o wide
- Kubectl get ev –o wide --watch
- kubectl get po –o wide --watch // List pods. Why po?

```
// Gets details about a specific resource
kubectl describe <resource_type or short_name> <resource_name>
```

- kubectl describe node learn

LAB: KUBECTL / CHECKING INSTALLATION

```
// Gets logs
kubectl logs <pod_name or service_name>
kubectl logs --tail=20 <pod_name or service_name> // last 20 entries
kubectl logs -f <pod_name or service_name> // monitor for new entries

Difference between logs and events
```

LAB: HELLO WORLD

// This creates a **pod** with the name *first* and passes some env variables

// That pod is running your image in a container

```
kubectl run first --image=cloudnatived/demo:hello --env="TEXT=Hi" --env="OTHER=value2"
```

https://hub.docker.com/search

Tell us:

- Is it running?
- In which node it runs?
- What is its IP?
- Can you ping it?

https://github.com/shiriuki/k8s-training/blob/main/kubectl-commands.yaml

LAB: HELLO WORLD

Why are you not able to ping your pod?

How to ping your pod:

- `m ssh -p learn -n learn-m02` `// Connects through ssh to the "node" that is running the pod.`
- `ping 10.224.1.4`
- `curl http://10.244.1.4:8080` `// If it exposes a web port`

A simpler way, open a temporary channel through the kubectl to access the pod's containers ports (advantages don't need to the pod's IP or machine where it is running):

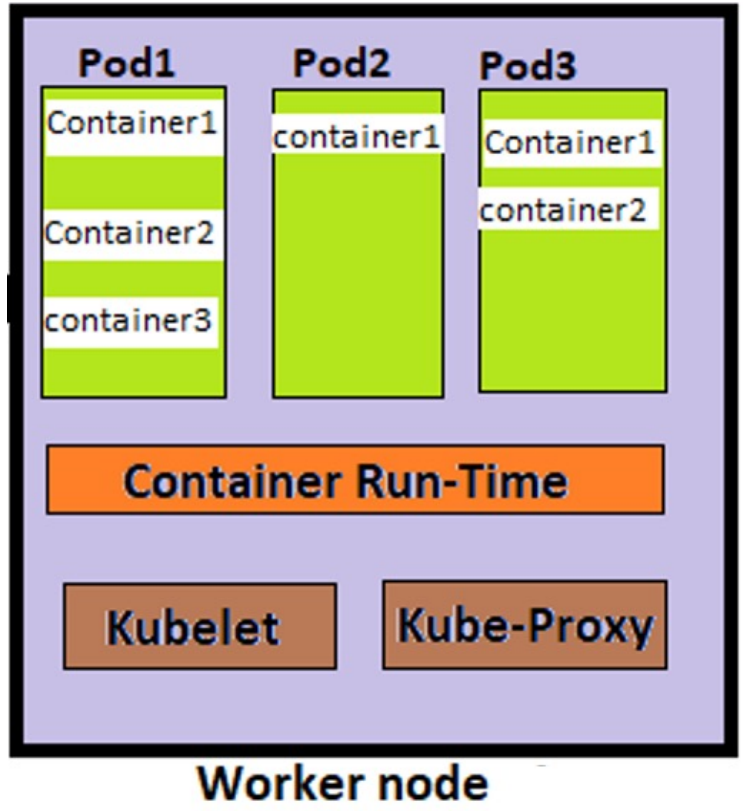
```
kubectl port-forward <pod_name> <local_port>:<pod_container_port>
kubectl port-forward first 7777:8080
curl http://localhost:7777
```

PODS

Pod is most granular unit of execution in k8s. Anything that you want to run in K8s, runs as an image in a container, and that container belongs to a pod. You can't just run a container (as in Docker), the container needs to be part of a pod

Each pod has a unique IP address and the port space is shared by all the containers in that pod. This means that different containers inside a pod can communicate with each other using their corresponding ports on localhost. They also share volumes.

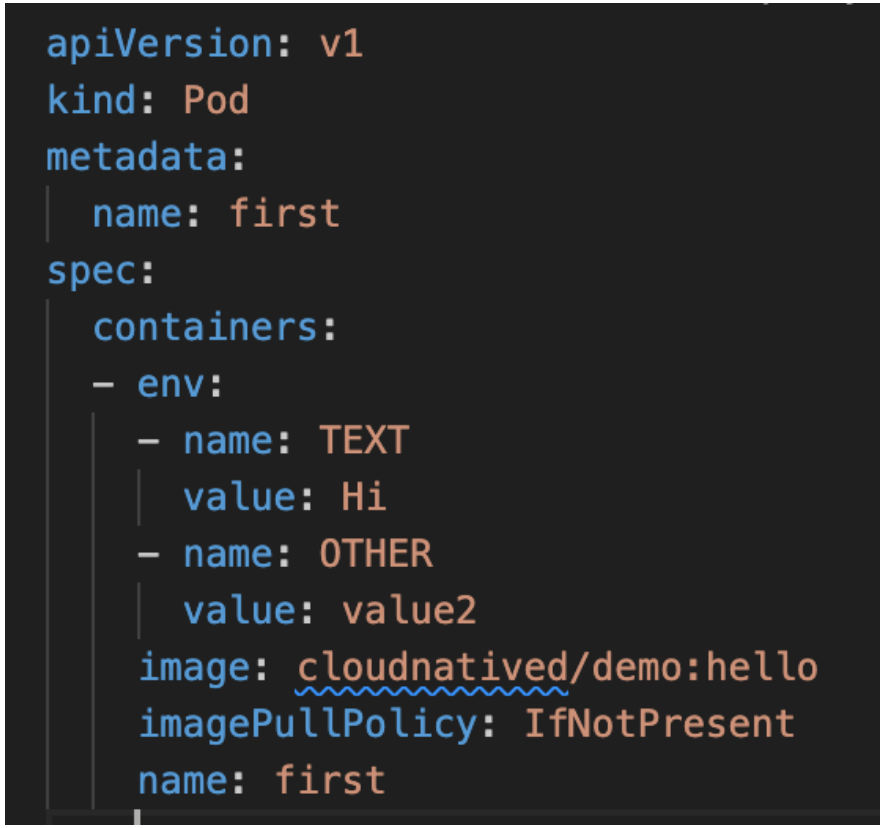
```
// connects to the pod's container
kubectl exec -it pod_name -c container_name bash
```



https://devops4solutions.medium.com/kubernetes-pods-explained-with-examples-1478288e07a4

PODS

```
// Generates the yaml representation of a K8s resource
kubectl get pod first -o yaml > firstPod.yaml
```



LAB: PODS

kubectl explain pod // Gives information about a K8s resource and its fields

kubectl explain pod.spec.containers.imagePullPolicy // Gives info about a field

kubectl apply –f secondPod.yaml // deploys a K8s resource described in yaml

Deploy a new Pod:

- **name**: second
- **image**: wardviaene/http-echo
- should display a customized message (how ?)
- should always download the image (how ?)
- What is your approach to do it?
- Run **kubectl get pods -o wide –watch** in one tab, and the command to deploy the new pod in another tab

https://github.com/shiriuki/k8s-training/blob/main/kubectl-commands.yaml

POD PATTERNS

Remember a Pod can run 1 or more containers

```yaml
kind: Pod
apiVersion: v1
metadata:
  name: multi
spec:
  containers:
  - name: container-1
    image: nginx
  - name: container-2
    image: ubuntu
    command:
    - /bin/bash
    - -ec
    - while :; do echo '.'; sleep 5; done
```

https://betterprogramming.pub/understanding-kubernetes-multi-container-pod-patterns-577f74690aee

POD PATTERNS / INIT CONTAINER

Init container

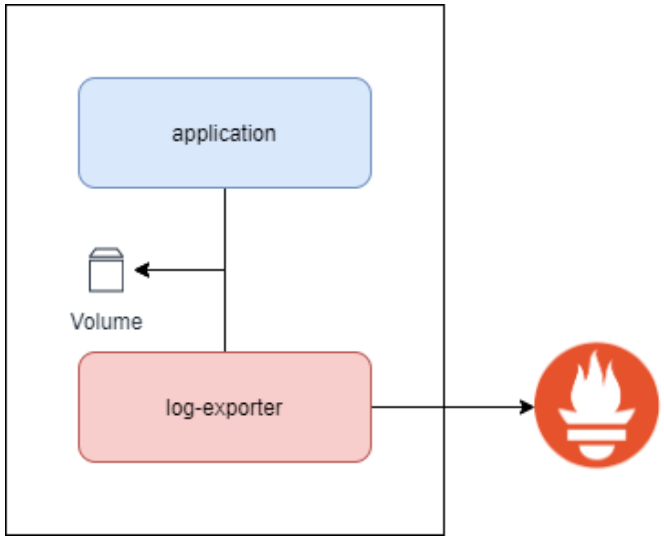
The init container must run successfully before the the regular containers can run. If that doesn't happen K8s won't proceed to execute the regular containers

```yaml
kind: Pod
apiVersion: v1
metadata:
  name: example
spec:
  initContainers:
  - name: create-db
    image: postgres:12
    command:
    - /bin/bash
    - -c
    - psql -h postgresHost -U postgres -c "CREATE DATABASE MYDB"
  containers:
  - name: activity-sender
    image: gcr.io/radix/activity-sender
```

POD PATTERNS / SIDECAR

Sidecar

Is used to enhance or extend the existing functionality of the main container. Your container works perfectly well without the sidecar, but with it, it can perform some extra functions.

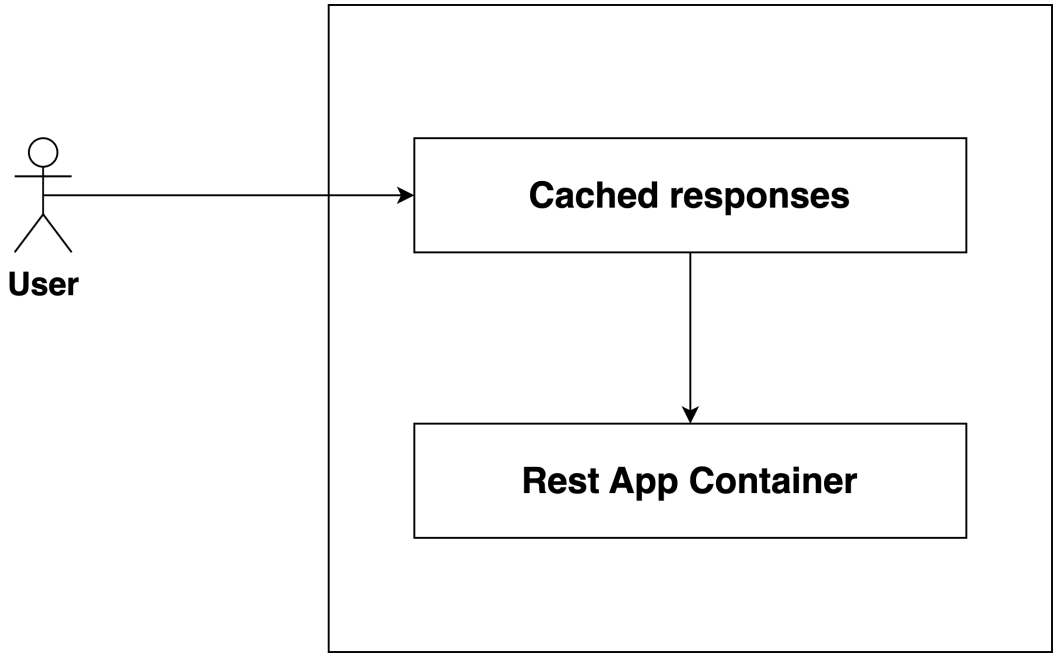



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: sidecar-pod
spec:
  volumes:
  - name: logs
    emptyDir: {}
  containers:
  - name: app-container
    image: alpine
    command: ["/bin/sh"]
    args: ["-c", "while true; do date >> /var/log/app.log; sleep 2;done"]
    volumeMounts:
    - name: logs
      mountPath: /var/log
  - name: log-exporter-sidecar
    image: nginx
    ports:
      - containerPort: 80
    volumeMounts:
    - name: logs
      mountPath: /usr/share/nginx/html
```

POD PATTERNS / AMBASSADOR

Ambassador

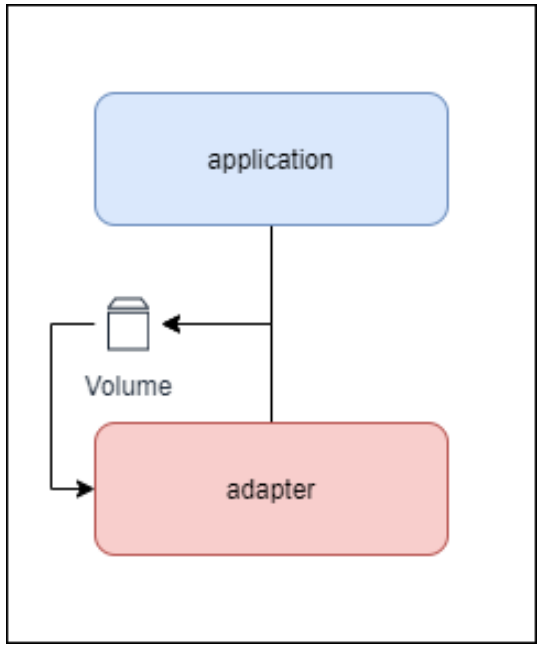
A proxy to the external world. We create another container that can act as a proxy to the main container. For example a cache of responses of the main container



POD PATTERNS / ADAPTER

Adapter

The adapter pattern helps you standardize something. For example, transforming the logs of a legacy app to a newer standard format.




```yaml
apiVersion: v1
kind: Pod
metadata:
  name: adapter-pod
  labels:
    app: adapter-app
spec:
  volumes:
  - name: logs
    emptyDir: {}
  containers:
  - name: app-container
    image: alpine
    command: ["/bin/sh"]
    args: ["-c", "while true; do date >> /var/log/app.log; sleep 2;done"]
    volumeMounts:
    - name: logs
      mountPath: /var/log
  - name: log-adapter
    image: alpine
    command: ["/bin/sh"]
    args: ["-c", "tail -f /var/log/app.log|sed -e 's/^/Date /' > /var/log/out.log"]
    volumeMounts:
    - name: logs
      mountPath: /var/log
```

LABELS

With them you can categorize your K8s resources, so later you can filter them out. They are key/value pairs

Ie: **google.com/customer**=ibm, google.com/customer=common

// Gets all resources and their current labels

kubectl get all --show-labels -A

// Filters resources by label

kubectl get all -A -l tier=control-plane

kubectl get pods -A -l google.com/customer=ibm, google.com/bu**!=**finance

LABELS

Valid key has 2 segments:
optional prefix and name, separated by a slash (/). If specified,
the prefix must be a DNS not longer than 253 characters
le: google.com/app

Rules for valid label value and name segment:
no more than 63 characters
must begin and end with an alphanumeric character ([a-z0-9A-Z])
could contain dashes (-), underscores (_), dots (.)

```
// Adds labels to an existing K8s resource
kubectl label <resource_type> <resource_id> google.com/nodetype=db
kubectl label node learn-m02 google.com/nodetype=db –overwrite
kubectl label all -l k8s-app=kube-dns -A mylabel=value
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: declarative-nginx
  labels:
    radixiot.com/app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
```

ANNOTATIONS

Used by third party apps to attach metadata to K8 resources. K8s doesn´t care or know about that metadata or how it is used. The keys & values used in annotations have no restrictions

```yaml
kind: Pod
apiVersion: v1
metadata:
  name: pod-with-annotation
  annotations:
    velero.io/dontBackup: "true"
spec:
  containers:
  - name: first-container
    image: nginx
```

```
// Adds annotations to an existing K8s resource
kubectl annotate pod nginx \
  pre.hook.backup.velero.io/command='["/sbin/fsfreeze", "--freeze", "/var/log/nginx"]' \
  post.hook.backup.velero.io/command='["/sbin/fsfreeze", "--unfreeze", "/var/log/nginx"]'

// To remove annotation
kubectl annotate pod nginx post.hook.backup.velero.io/command-
```

SERVICES

A service associates a set of pods with an abstract service name

The need for services arises from the fact that pods are short lived and can be replaced at any time. This means that pods that need to communicate with another pod cannot rely on the IP address of the underlying single pod. Instead, they connect to the service name, which relays them to a relevant, currently-running pod

Uses labels and selectors to match the pods it represents

It will distribute the load between those pods

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: nginx
```

https://komodor.com/learn/kubernetes-service-examples-basic-usage-and-troubleshooting/

SERVICES / TYPES

ClusterIP

It receives a cluster-internal IP address, making its pods only accessible from within the cluster

Internal K8s DNS creates an entry that allows to resolve:

- http://my-cip-service.default:7777
- http://my-cip-service.default.svc.cluster.local:7777
- http://my-cip-service:7777
- k exec -it pod1 sh
- nslookup my-cip-service 8.8.8.8

TIPS:

`kubectl port-forward service/myservice 80:7777` doesn't distribute the load, also a gRPC channel doesn't distribute the load

you can't ping a service IP

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-cip-service
spec:
  selector:
    app: msg
  type: ClusterIP
  ports:
  - protocol: TCP
    port: 7777
    targetPort: 8080
```

SERVICES / TYPES

Behind the scenes K8s creates (and automatically adjust) a list of pods that match the service's selector

`kubectl get endpoints`

```
[manuelcalvo [Downloads] # k get endpoints
NAME              ENDPOINTS                                         AGE
kubernetes        192.168.205.77:8443                               11d
my-cip-service    10.244.0.3:8080,10.244.1.4:8080,10.244.1.5:8080   79m
```

TIP:

You can create a service without a selector, and manually define its endpoitns. This can be used to access services outside the K8s cluster that can't be discovered through DNS

```yaml
apiVersion: v1
kind: Service
metadata:
  name: minio-service
  namespace: velero
spec:
  type: ClusterIP
  ports:
  - protocol: TCP
    port: 9000
    targetPort: 9000
---
apiVersion: v1
kind: Endpoints
metadata:
  name: minio-service
  namespace: velero
subsets:
  - addresses:
    - ip: 192.168.2.193
    ports:
    - port: 9000
```

SERVICES / TYPES

NodePort

Builds on top of the ClusterIP service, exposing it to a port accessible from outside the cluster

The kube-proxy component on each node is responsible for listening on the node's external ports and forwarding client traffic from the NodePort to the ClusterIP

By default, all nodes in the cluster listen on the service's NodePort, even if they are not running a pod that matches the service selector

The external port must in the range: 30000-32767

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-np-service
spec:
  selector:
    app: msg
  type: NodePort
  ports:
  - protocol: TCP
    nodePort: 32000
    port: 7777
    targetPort: 8080
```

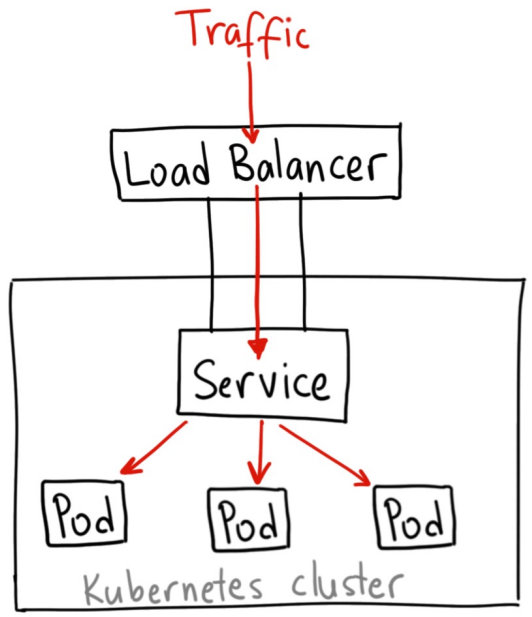
SERVICES / TYPES

LoadBalancer

Builds on top of the NodePort service

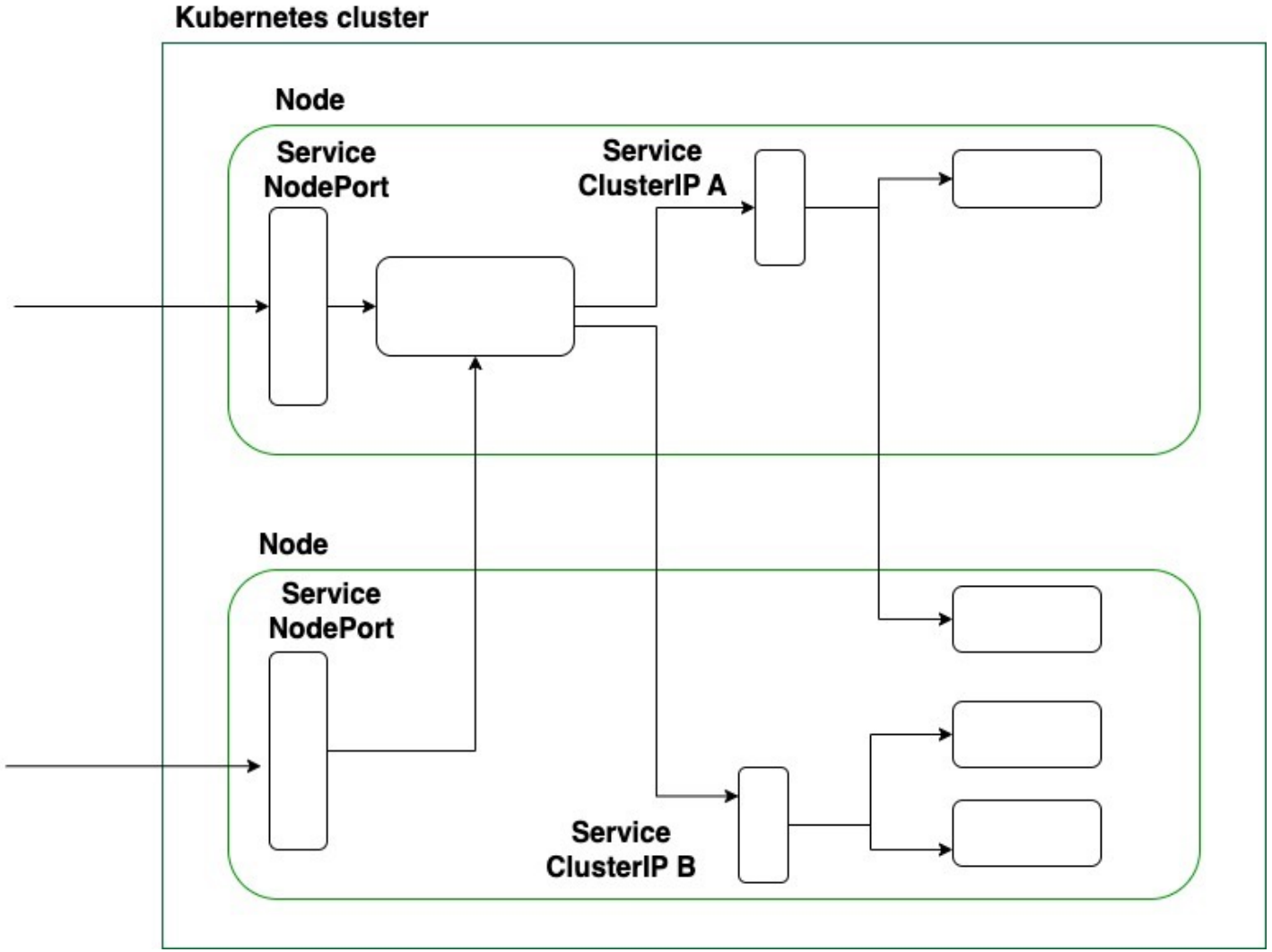
To be used in cloud environments for them to provide a load balancer that will forward requests to the K8s cluster

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-lb-service
spec:
  selector:
    app: msg
  type: LoadBalancer
  ports:
  - protocol: TCP
    nodePort: 30000
    port: 7777
    targetPort: 8080
```




```
$ kubectl get svc nginx-service
NAME          TYPE         CLUSTER-IP    EXTERNAL-IP                                                              PORT(S)      AGE
nginx-service LoadBalancer 172.20.54.138 ac8415de24f6c4db9b5019f789792e45-443260761.us-east-2.elb.amazonaws.com 80:30968/TCP 21h

$ curl ac8415de24f6c4db9b5019f789792e45-443260761.us-east-2.elb.amazonaws.com
<!DOCTYPE html>
<html>]
<head>
<title>Welcome to nginx!</title>
...
```

SERVICES / EXAMPLES



LAB: SERVICES

Create 2 pod instances using this image: wardviaene/http-echo

Each pod should display a different message

Then create a service of type NodePort that redirects requests to those pods

Probe load balancing is happening

https://github.com/shiriuki/k8s-training/blob/main/kubectl-commands.yaml
https://github.com/shiriuki/k8s-training/blob/main/service-cip.yaml
https://github.com/shiriuki/k8s-training/blob/main/service-np.yaml

NAMESPACES

Provide a way to partition a cluster

By default, a Kubernetes cluster has a default namespace that is used to deploy resources if no other namespace is specified

They help to improve resource management and enhance security

```
// Create a namespace
kubectl create namespace infra

// Gets the existing namespaces
kubectl get ns

// Gets list of all resources in all namespaces
kubectl get all -A

// Get list of resources of a type in a specific namespace
kubectl get <resource_type> -n <namespace>
kubectl get pod -n kube-system
```

NAMESPACES

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: infra
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  namespace: infra
...
```

LAB: NAMESPACES

In a **single** yaml file (use --- to indicate the start of a new k8s resource)

- Create a namespace with name: infra
- Deploy Postgres behind a NodePort service in namespace infra
- https://hub.docker.com/_/postgres#:~:text=have%20found%20useful.-,Environment%20Variables,-The%20PostgreSQL%20image
- Postgres default container port 5432

Connect to your Postgres using pgAdmin

- https://www.pgadmin.org/download/

List all resources available in all namespaces

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: other
---
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  namespace: other
...
---
apiVersion: v1
kind: Service
metadata:
  name: myservice
  namespace: other
spec:
  selector:
    mylabel: value
...
```

https://github.com/shiriuki/k8s-training/blob/main/kubectl-commands.yaml
https://github.com/shiriuki/k8s-training/blob/main/service-np.yaml
https://github.com/shiriuki/k8s-training/blob/main/namespace.yaml

CONFIG MAPS

They provide a way to store data in key-value pairs separately from the application. They can be consumed by pods as environment variables or files in a volume

Should not be used to store confidential or sensitive data because they are stored as plain text

Its data can be shared between pods

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config-map
data:
  currency: USD
  headerColor: black
  welcomeMsg: Hi there
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: podm
spec:
  containers:
  - name: echo
    image: wardviaene/http-echo
    env:
    - name: TEXT
      valueFrom:
        configMapKeyRef:
          name: my-config-map
          key: welcomeMsg
```

https://yuminlee2.medium.com/kubernetes-configmaps-fee26f4c3ccf

SECRETS

Should be used to store sensitive information like passwords, keys, and tokens in **base64 format**

They can be consumed by pods as environment variables or files in a volume

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
data:
  <key>: <base64_encoded_value>
  mongoPassw: dGhlcGFzcw==
```

```
apiVersion: v1
kind: Pod
metadata:
  name: podm
spec:
  containers:
  - name: db
    image: mongo
    env:
    - name: MONGO_INITDB_ROOT_PASSWORD
      valueFrom:
        secreteRef:
          name: my-secret
          key: mongoPassw
```

```
echo -n "thepassw" | base64
echo -n "PHN0cmluZz4=" | base64 –d
k get secret my-secret -o yaml
```

https://yuminlee2.medium.com/kubernetes-secrets-4287b5a83606

LAB: SECRETS

Modify your postgres installation (previous lab) so its password comes from a secret

https://github.com/shiriuki/k8s-training/blob/main/kubectl-commands.yaml
https://github.com/shiriuki/k8s-training/blob/main/secret.yml

VOLUMES

A volume is a directory containing data, which can be accessed by containers in a pod. The location of the directory, the storage media that supports it, and its contents, depend on the specific type of volume being used

Processes running within containers in a pod see a file system view composed of:

- A root file system that matches the content of the container image
- Volumes mounted on the container (if defined). Each volume mounts on a specific path within the container file system

They are defined within the context of a pod, meaning that you can't create a volume by itself. This makes volumes unmanageable and their lifecycle bound directly to the pod. This means that deleting the pod will result in either deleting the volume or liberating it depending on the volume type used.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config-map
data:
  currency: USD
  headerColor: black
  welcomeMsg: Hi there
---
apiVersion: v1
kind: Pod
metadata:
  name: podv
spec:
  containers:
  - name: echo
    image: wardviaene/http-echo
    volumeMounts:
    - name: cm
      mountPath: /cm
  volumes:
    - name: cm
      configMap:
        name: my-config-map
```

https://medium.com/swlh/kubernetes-storage-explained-558e85596d0c

VOLUMES / EMPTYDIR

A temporary directory of the **node**, that shares a pod's lifetime. When the pod is destroyed, it will destroy the shared volume and all its contents

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: sharevol
spec:
  containers:
  - name: c1
    image: centos:7
    command:
      - "bin/bash"
      - "-c"
      - "sleep 10000"
    volumeMounts:
      - name: xchange
        mountPath: "/tmp/xchange"
  - name: c2
    image: centos:7
    command:
      - "bin/bash"
      - "-c"
      - "sleep 10000"
    volumeMounts:
      - name: xchange
        mountPath: "/tmp/data"
  volumes:
  - name: xchange
    emptyDir: {}
```

VOLUMES / HOSTPATH

A specific directory of the **node** is shared

Because it is a specific directory of the node it won't be destroyed when the pod is destroyed

The user running the container should have permissions to access that directory

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: sharevol
spec:
  containers:
  - name: c1
    image: centos:7
    command:
      - "bin/bash"
      - "-c"
      - "sleep 10000"
    volumeMounts:
      - name: xchange
        mountPath: "/tmp/xchange"
  - name: c2
    image: centos:7
    command:
      - "bin/bash"
      - "-c"
      - "sleep 10000"
    volumeMounts:
      - name: xchange
        mountPath: "/tmp/data"
  volumes:
  - name: xchange
    hostPath:
      path: /manuels/data
```

VOLUMES / CONFIGMAP / SECRET

The contents of config maps or secrets are mounted as virtual directory. Where each key is a file, and the value of the key is the content of the corresponding file

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config-map
data:
  currency: USD
  headerColor: black
  welcomeMsg: Hi there
---
apiVersion: v1
kind: Pod
metadata:
  name: podv
spec:
  containers:
  - name: echo
    image: wardviaene/http-echo
    volumeMounts:
    - name: cm
      mountPath: /cm
  volumes:
  - name: cm
    configMap:
      name: my-config-map

  volumes:
  - name: <volume_name>
    secret:
      secretName: <secret_name>
```

PERSISTENT VOLUMES

They are completely independent from the pod they are attached to and even independent from the host where it's running when it's not a local persistent volume

Persistent Volumes (PV) decouples storage from the Pod and makes them a first class citizen in kubernetes with their own separate manageable object and abstraction layer

PVs allow data sharing between pods by mounting the same PV to various pods

PVs will remain even after the pod is deleted or replaced

Can be provisioned in two ways:

- manually by an administrator
- dynamically using storage classes with a configured provisioner that specifies which volume plugin to be used for provisioning

PV / MANUAL PROVISIONING

An administrator has access to the underlying storage provider and have knowledge of the disk configuration. For this example in GKE, will create a persistent volume using the GCEPersistentDisk plugin

accessModes:

- ReadWriteOnce — the volume can be mounted as read-write by a single node
- ReadWriteOnly — the volume can be mounted as read-only by many nodes
- ReadWriteMany — the volume can be mounted as read-write by many nodes
- ReadWriteOncePod — the volume can be mounted as read-write by single pod

persistentVolumeReclaimPolicy:

- Retain — manual reclamation
- Delete — associated storage asset such as AWS EBS or GCE PD volume is deleted
- Recycle — basic scrub (rm -rf /thevolume/*)

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mongodb-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
    - ReadOnlyMany
  volumeMode: Filesystem
  storageClassName: manual
  persistentVolumeReclaimPolicy: Retain
  gcePersistentDisk:
    pdName: mongodb
    fsType: ext4
```

https://medium.com/geekculture/storage-kubernetes-92eb3d027282

PV / MANUAL PROVISIONING

Through a PersistentVolumeClaim (PVC) users can request storage by specifying the storage class, accessMode and the size of the volume needed

Binding criteria:

- accessMode must be matched between the PV and PVC.
- The requested storage by a PVC must be equal to or less than the storage capacity of a PV. An equal match will always get priority. But if there is no exact match, then PVC will claim a PV with a larger storage capacity than what was demanded by PVC

Should be " " for manual provisioning

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
  - ReadWriteOnce
  storageClassName: "fast"
---
apiVersion: v1
kind: Pod
metadata:
  name: mongodb
spec:
  containers:
  - name: mongo
    image: mongo
    volumeMounts:
    - name: mongodb-data
      mountPath: /data/db
  volumes:
  - name: mongodb-data
    persistentVolumeClaim:
      claimName: mongodb-pvc
```

https://medium.com/geekculture/storage-kubernetes-92eb3d027282

PV / DYNAMIC PROVISIONING

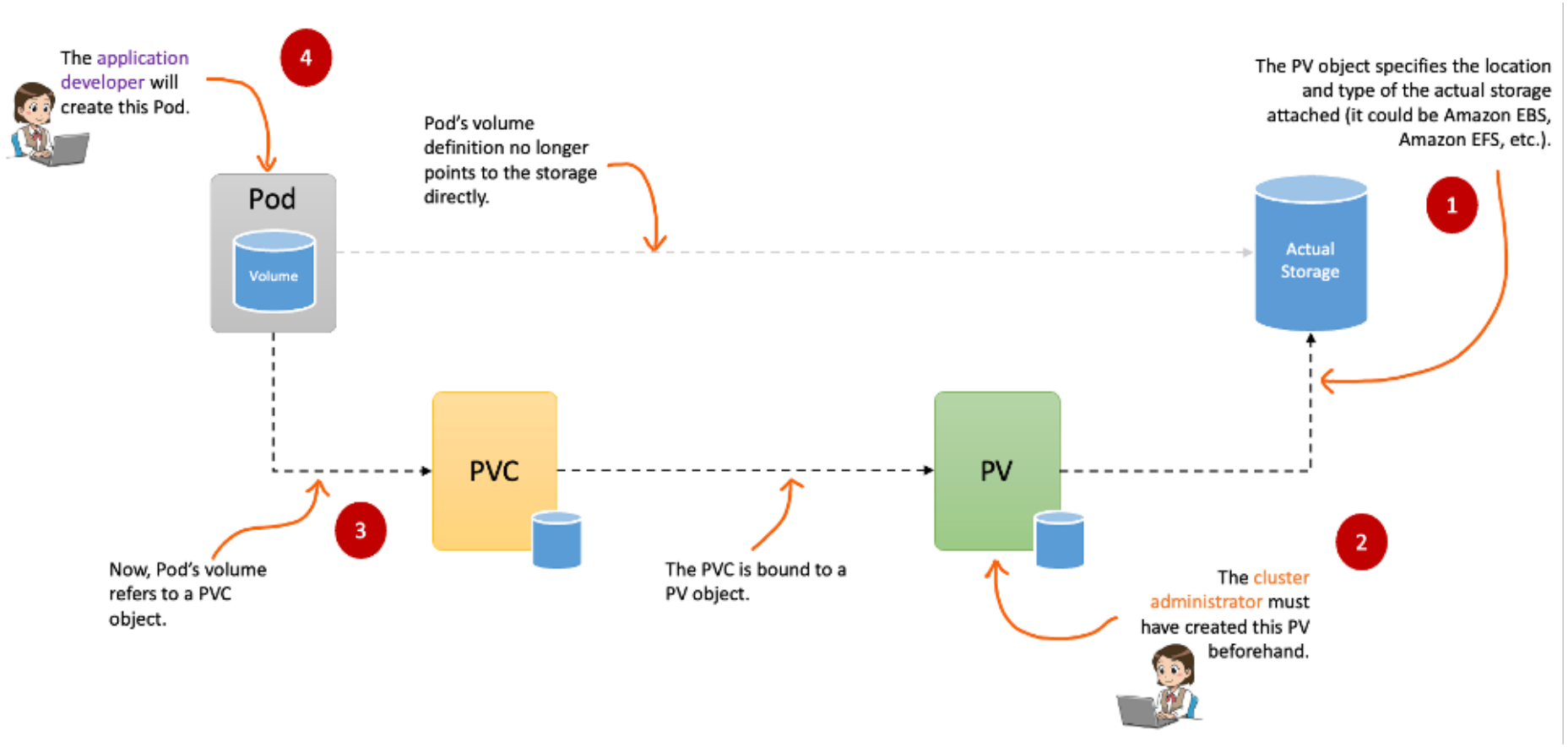
An K8S object called storage class (“StorageClass”) dynamically creates persistent volumes when requested using a PersistentVolumeClaim

The main goal of storage classes is to eliminate the need for cluster administrators to pre-provision storage and allow them to be created on-demand

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: slow
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ssd-local-path
provisioner: cluster.local/local-path-provisioner
parameters:
  nodePath: /data/ssd
```

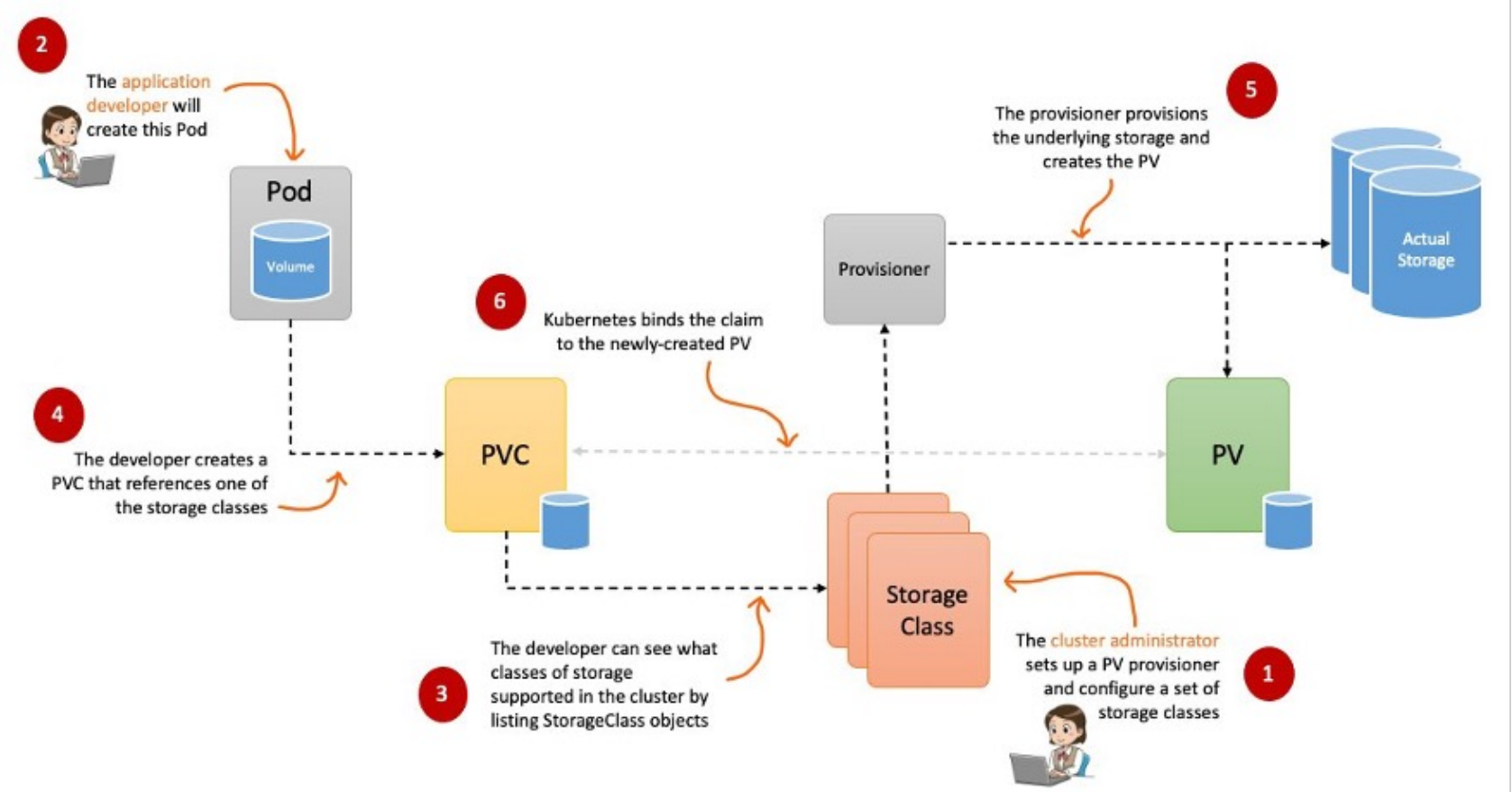
https://medium.com/geekculture/storage-kubernetes-92eb3d027282

PV / MANUAL PROVISIONING




```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-volume01
spec:
  capacity:
    storage: 50Mi
  accessModes:
  - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  hostPath:
      path: /var/local/data
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 50Mi
---
apiVersion: v1
kind: Pod
metadata:
  name: web-server
spec:
  containers:
  - name: nginx-webserver
    image: nginx
    volumeMounts:
    - name: webserver-vol01
      mountPath: "/usr/share/nginx/html"
  volumes:
  - name: webserver-vol01
    persistentVolumeClaim:
      claimName: myclaim
```

PV / DYNAMIC PROVISIONING




```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ebs-claim-01
spec:
  storageClassName: gp2-encrypted
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
---
apiVersion: v1
kind: Pod
metadata:
  name: webserver
spec:
  containers:
  - name: nginx-container
    image: nginx:latest
    volumeMounts:
    - mountPath: "/usr/share/nginx/html"
      name: test-volume
  volumes:
  - name: test-volume
    persistentVolumeClaim:
      claimName: ebs-claim-01
```

PV / LOCAL-PATH-PROVISIONER

https://github.com/rancher/local-path-provisioner

Install

kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-provisioner/v0.0.24/deploy/local-path-storage.yaml

// Set local-path as the default dynamic provisioner

// and the default volume type as 'local'

kubectl patch storageclass local-path -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true","defaultVolumeType":"**local**"}}}'

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: local-path-config
  namespace: local-path-storage
data:
  config.json: |-
   {
      "nodePathMap":[
      {
         "node":"DEFAULT_PATH_FOR_NON_LISTED_NODES",
         "paths":["/bbdata"]
      },
      {
         "node":"yasker-lp-dev1",
         "paths":["/opt/local-path-provisioner", "/data1"]
      },
      {
         "node":"yasker-lp-dev3",
         "paths":[]
      }
      ]
   }
```

PV / LOCAL-PATH-PROVISIONER

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim2
spec:
  storageClassName: "local-path"
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Mi
---
apiVersion: v1
kind: Pod
metadata:
  name: web-server
spec:
  containers:
    - name: nginx-webserver
      image: nginx
      volumeMounts:
        - name: webserver-vol01
          mountPath: "/usr/share/nginx/html"
  volumes:
    - name: webserver-vol01
      persistentVolumeClaim:
        claimName: myclaim2
```

LAB: VOLUMES

Knowing that Postgres stores its data in this folder:

/var/lib/pgsql/data

modify your Postgres pod so it stores Postgres data in a PersistentVolume (manually provisioned) so if the pod is deleted its data will persist

Using PgAdmin or DBBeaver probe the Postgres data was not lost after the pod was previously deleted

https://github.com/shiriuki/k8s-training/blob/main/kubectl-commands.yaml
https://github.com/shiriuki/k8s-training/blob/main/secret.yml

CONTROLLERS / DEPLOYMENT

Provide a way to define, update, and maintain desired replica counts of Pods

When deployed for the first time, after every update, or restart, behind the scenes it creates a new ReplicaSet k8s resource using the configuration specified in the template section and replicas value

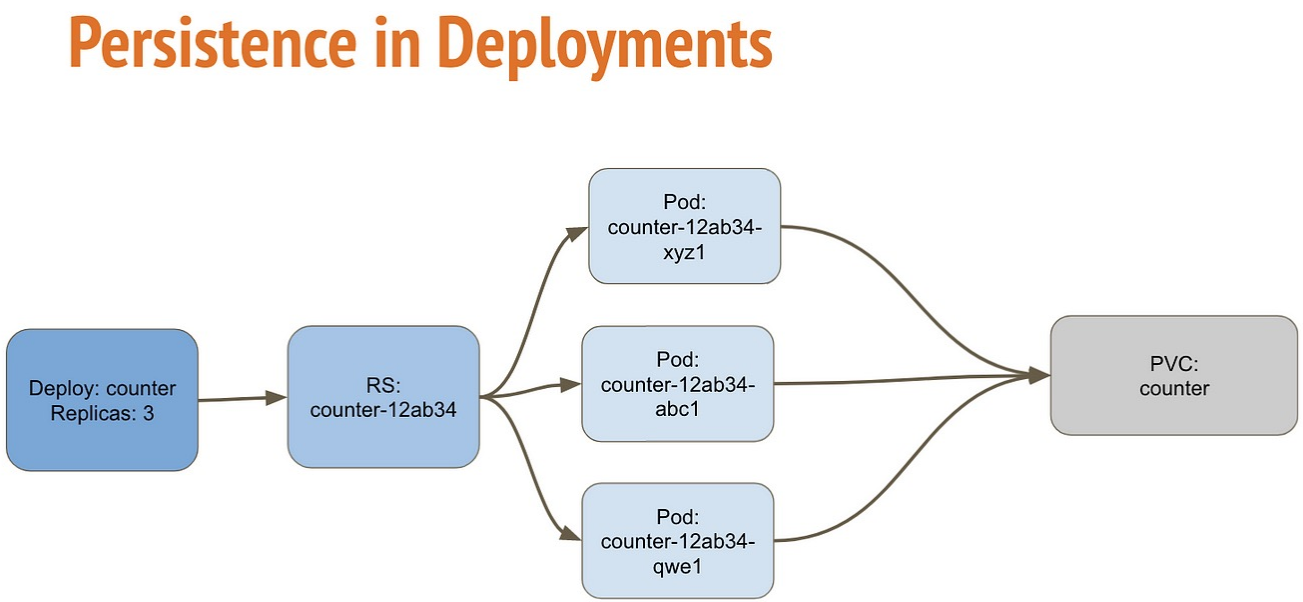
```
// Changes the number of replicas in a deployment
k scale deploy nginx-deployment --replicas 5

// Creates a new ReplicaSet and migrate pods to it
k rollout restart deployment/nginx-deployment

// Go back to the previous ReplicaSet
k rollout undo deployment/nginx-deployment
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx-container
          image: nginx
```

https://medium.com/@kadimasam/kubernetes-namespaces-deployments-replicasets-pods-and-containers-86239b785cf9

CONTROLLERS / DEPLOYMENT

They should not be used if your deployment directly writes to a persistence volume, as if you have more than 1 replica, that could create inconsistencies because them will share the same PersistentVolume



https://medium.com/stakater/k8s-deployments-vs-statefulsets-vs-daemonsets-60582f0c62d4

CONTROLLERS / STATEFULSET

Manages the deployment and scaling of a set of Pods, and provides guarantee about the ordering and uniqueness of these Pods

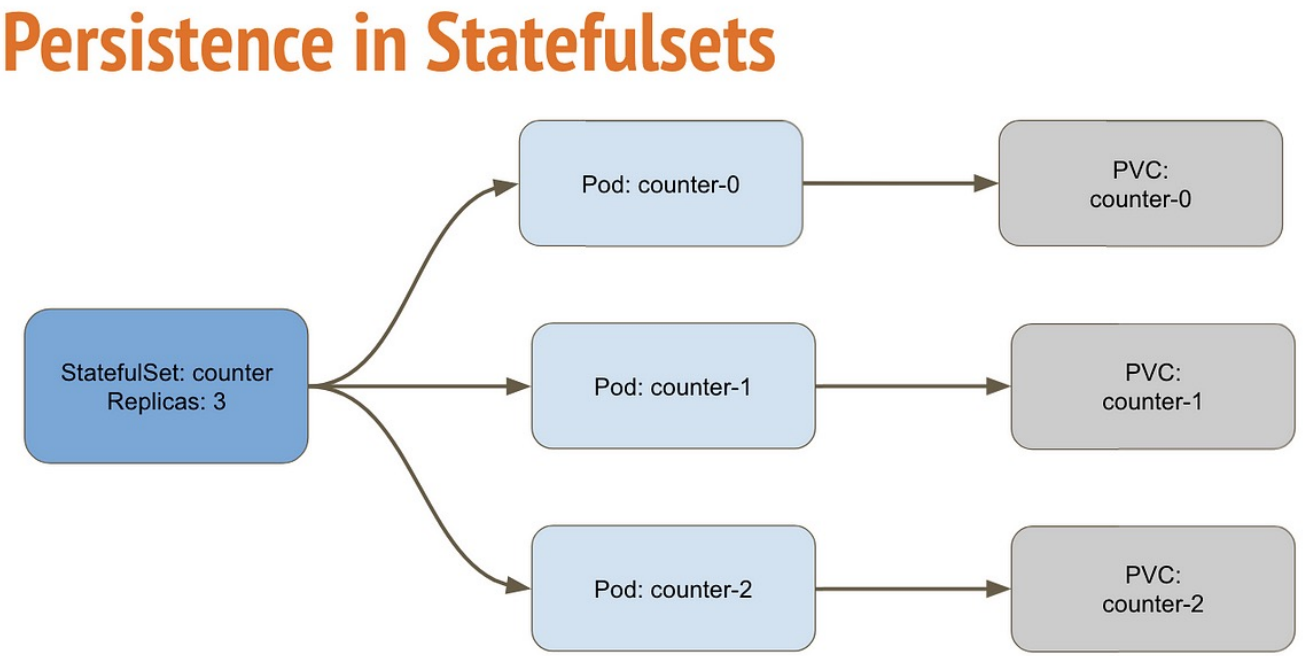
It doesn't create ReplicaSet rather itself creates the Pod with a unique naming convention. If you create a StatefulSet with name **counter,** it will create a pod with name **counter-0,** and for multiple replicas of a statefulset, their names will increment like **counter-0**, **counter-1**, **counter-2, etc**

```
// Changes the number of replicas in a statefulset
k scale sts/counter --replicas 5

// Creates restarts sts pods
k rollout restart sts/counter

// Go back to the previous configuration
k rollout undo sts/counter
```

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: counter
spec:
  replicas: 1
  selector:
    matchLabels:
      app: counter
  template:
    metadata:
      labels:
        app: counter
    spec:
      containers:
      - name: counter
        image: kahootali/counter:1.1
        volumeMounts:
        - name: counter
          mountPath: /app/
  volumeClaimTemplates:
  - metadata:
      name: counter
    spec:
      accessModes:
      - ReadWriteMany
      resources:
        requests:
          storage: 50Mi
      storageClassName: standard
```

CONTROLLERS / STATEFULSET

Every replica of a stateful set will have its own state, and each of the pods will be creating its own PVC. So a statefulset with 3 replicas will create 3 pods, each having its own Volume, so total 3 PVCs.

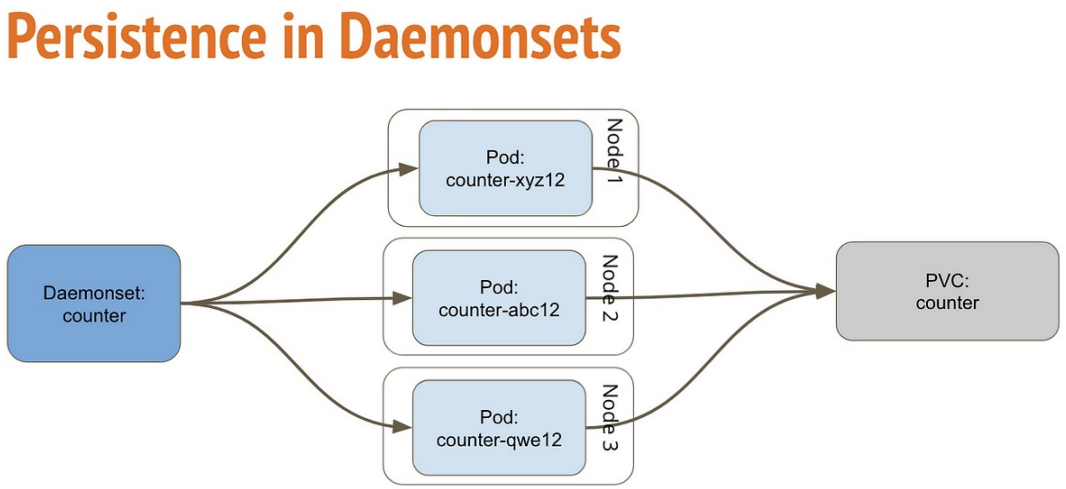


CONTROLLERS / DAEMONSET

Is a controller that ensures that the pod runs on all the nodes of the cluster. If a node is added/removed from a cluster, DaemonSet automatically adds/deletes the pod.

Typical use cases:

- Monitor exporters
- Log collectors

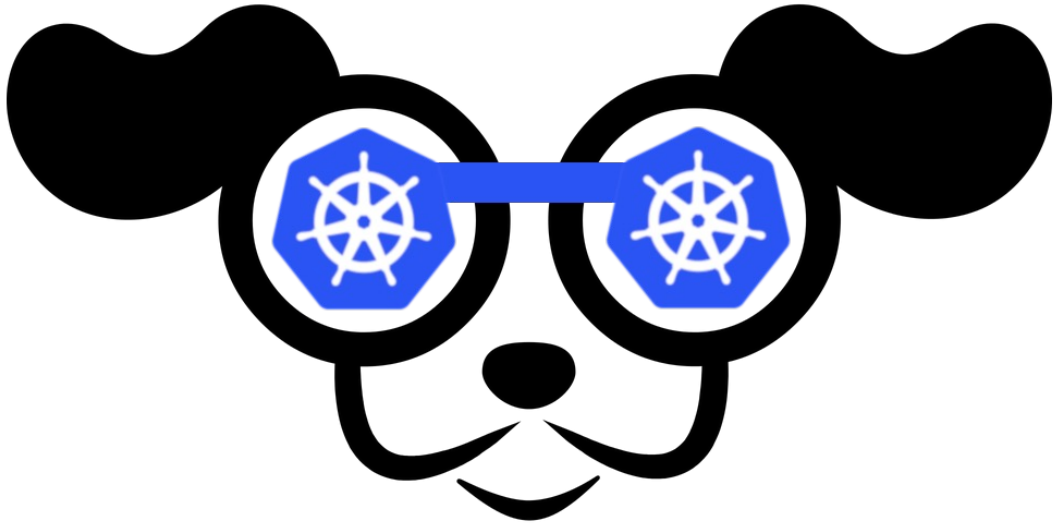


Each Replica running on each

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluent-bit
  namespace: monitoring
  labels:
    k8s-app: fluent-bit-logging
spec:
  selector:
    matchLabels:
      k8s-app: fluent-bit-logging
  template:
    metadata:
      labels:
        k8s-app: fluent-bit-logging
    spec:
      containers:
        - name: fluent-bit
          image: fluent/fluent-bit:2.1.7
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 2020
          volumeMounts:
            - name: varlog
              mountPath: /var/log
            - name: varlibdockercontainers
              mountPath: /var/lib/docker/containers
              readOnly: true
            - name: fluent-bit-config
              mountPath: /fluent-bit/etc/
      terminationGracePeriodSeconds: 10
      volumes:
        - name: varlog
          hostPath:
            path: /var/log
        - name: varlibdockercontainers
          hostPath:
            path: /var/lib/docker/containers
        - name: fluent-bit-config
          configMap:
            name: fluent-bit-config
```

TOOLS / K9S

https://k9scli.io/topics/install/



LAB: FINAL LAB

https://medium.com/@shubhangi.thakur4532/how-to-deploy-mysql-and-wordpress-on-kubernetes-8ea1260c27dd

Follow the steps of that article

- Use minikube (no need to use google cloud)
- Use a secret instead of plain text password
- Make mysql db persistent (hint: see this https://kubernetes.io/docs/tutorials/stateful-application/mysql-wordpress-persistent-volume/)

Did it work for you? Lets discuss what you did in it

ADDITIONAL TOPICS

Helm charts

Autoscaling (hpa)

Resources / limits

Affinities (pod / node)

QoS / Priorities

Jobs / Cronjobs

Probes

SOPS

Telepresence

Backup (Velero)

Cordon / Drain

Deployment strategies (rolling, blue-green, canary)

Ingress controller, API Gateway

RBAC & Service Accounts

Helmfile

Metrics / Monitoring (Prometheus / Grafana)

Autoscaling (vpa, keda)

Log aggregation (Graylog)

Taints & Tolerations

Service mesh (Istio)

http://medium.com

THANK YOU

I hope this training helped you to get start with K8s and was fun at the same time