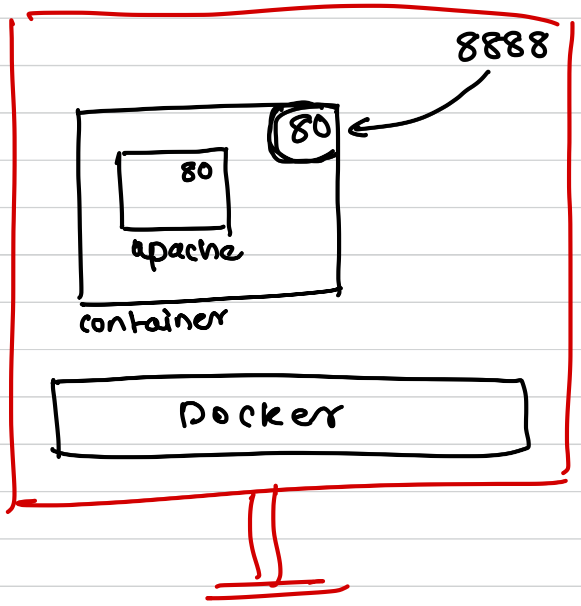
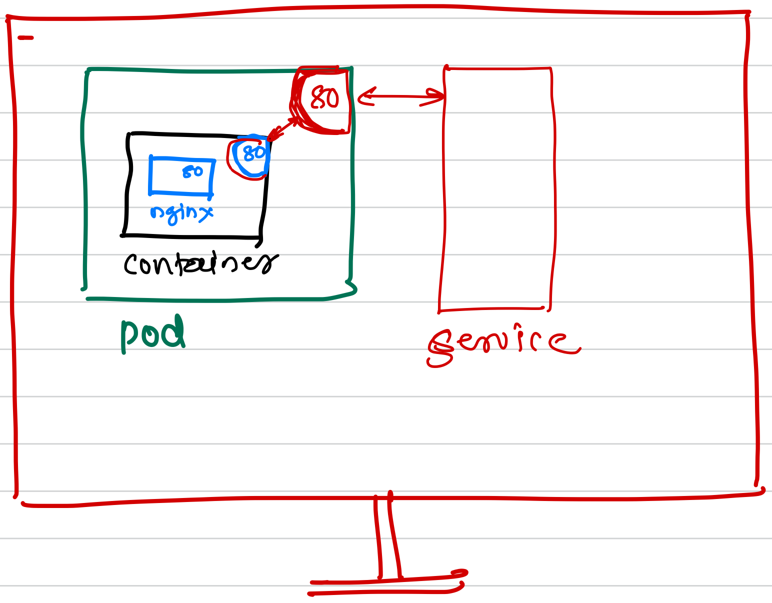
kubernetes

Multi-Container Pods (10%)

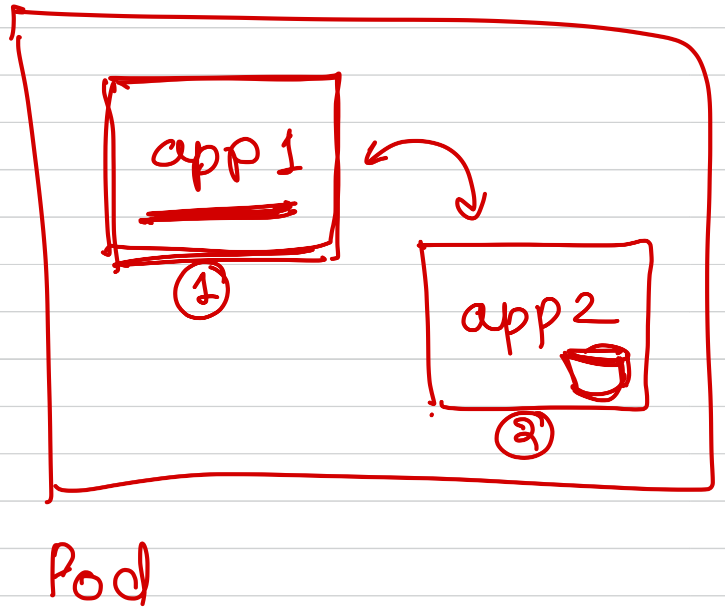


8888
80
80
apache
container
Docker



80
80
80
nginx
container
pod
service

multi-container

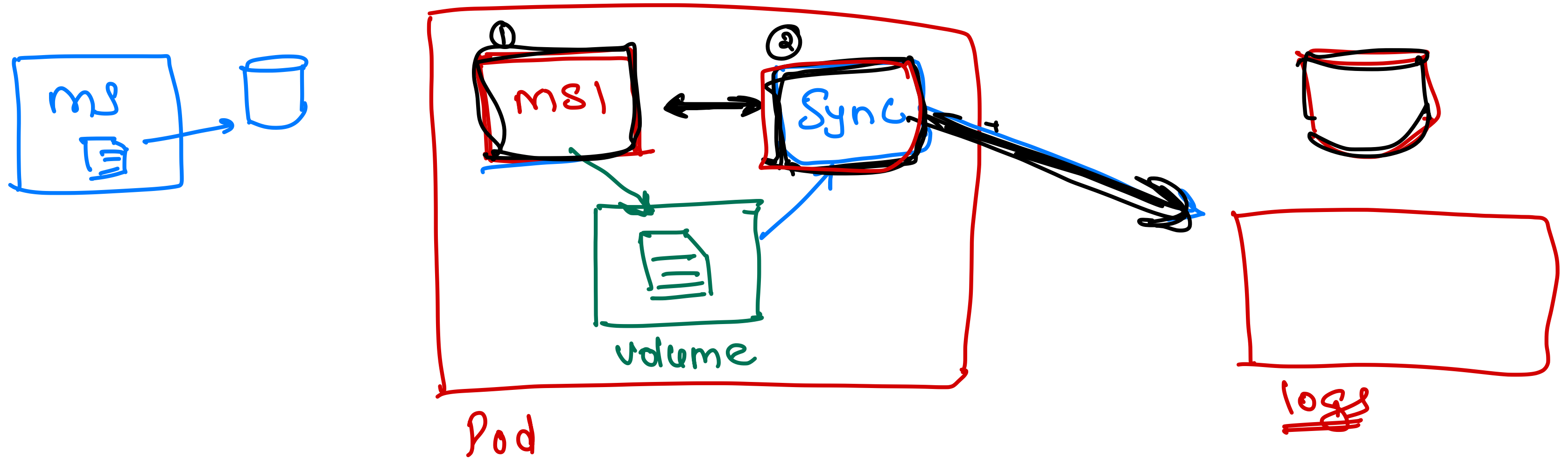


Sidecar

Init Containers

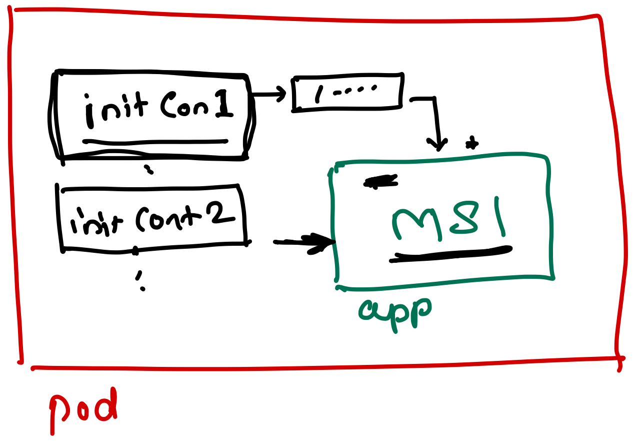
Sidecar

- Design pattern where a pod may run multiple containers
- These containers share resources like the IP address assigned to the pod and a volume
- Tasks
  - Create a pod with two containers
  - Create a volume and mount it in both the containers
  - Write data to the volume from one container and format the result in another container



Init Containers

- These are the specialized containers that run before the application containers in a Pod
- A pod can have one or more init containers
- Init containers are similar to regular containers except
  - The init containers always run before the application container
  - The init containers always run to completion
  - Each init container must complete successfully before the next one starts
  - If a pod's init container fails, kubernets repeatedly restarts the pod until the init container succeeds
- Tasks
  - Create a pod with init container



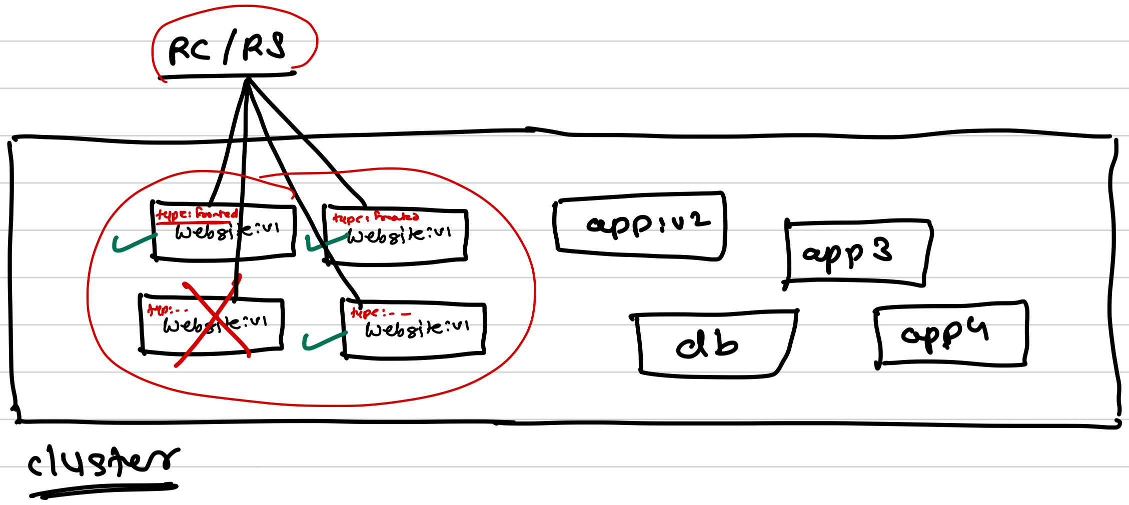
Pod Design (20%)

① Coding

② Deployment

- containerize
  - ✓ ↳ create docker image
  - ✓ ↳ push image to docker hub

- kubernetes
  - ✓ ↳ run app inside kube cluster



Replication Controller & ReplicaSet ✓

- Both of them do the similar job
- You specify the number of pods (desired count) you want to run, and RC or RS will make sure that those many pods are running at any given time
- If pod crashes RC/RS will start new pods to match the desired count
- To create the new pod, RC/RS will use the template specified in the yml definition
- Replication Controller supports only equality based selectors
- ReplicaSet supports both equality based as well as set based selectors

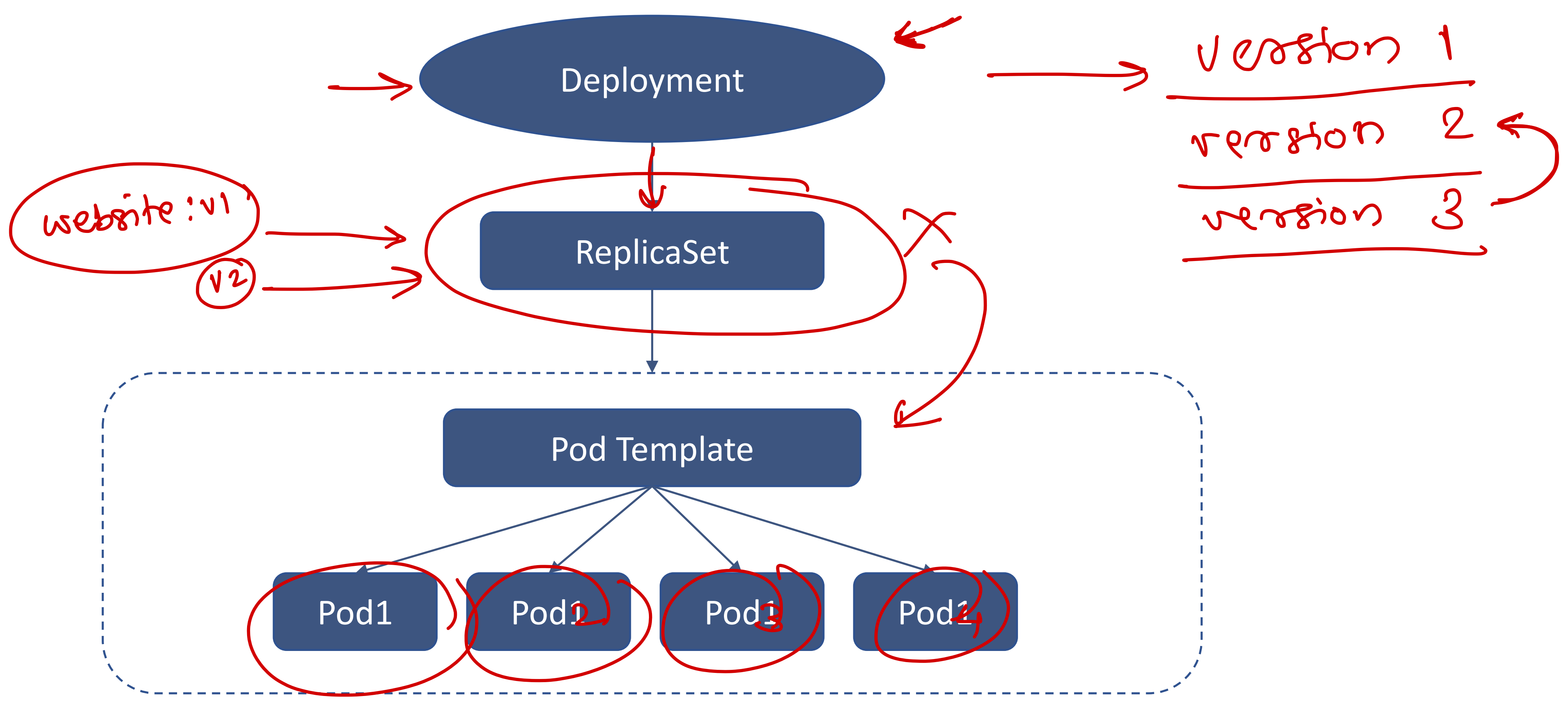
- Tasks
  - Create a equality based selector replication controller
  - Create a equality based selector replica set
  - Create a set based selector replica set

Deployments

- Describes the desired state of a component of the application
- In a simplest case, deployment involves one more more ReplicaSets to create pod replicas
- User does not require to use ReplicaSet separately as deployment will use it internally
- A single deployment is similar to a single microservice in the application
- Result of each deployment is a ReplicaSet which then manage the pods in the microservice
- You can create deployment
  - Using command - imperative
  - Using yml definition file
- Tasks:
  - Create deployment using command line
  - Create deployment using yml definition file

Deployment

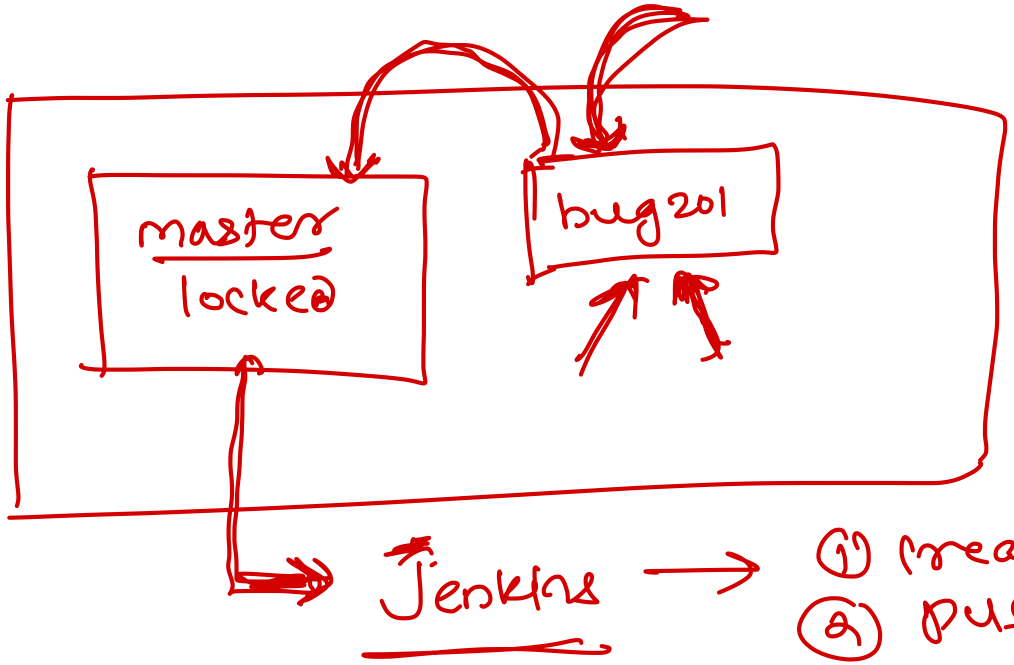
scaling



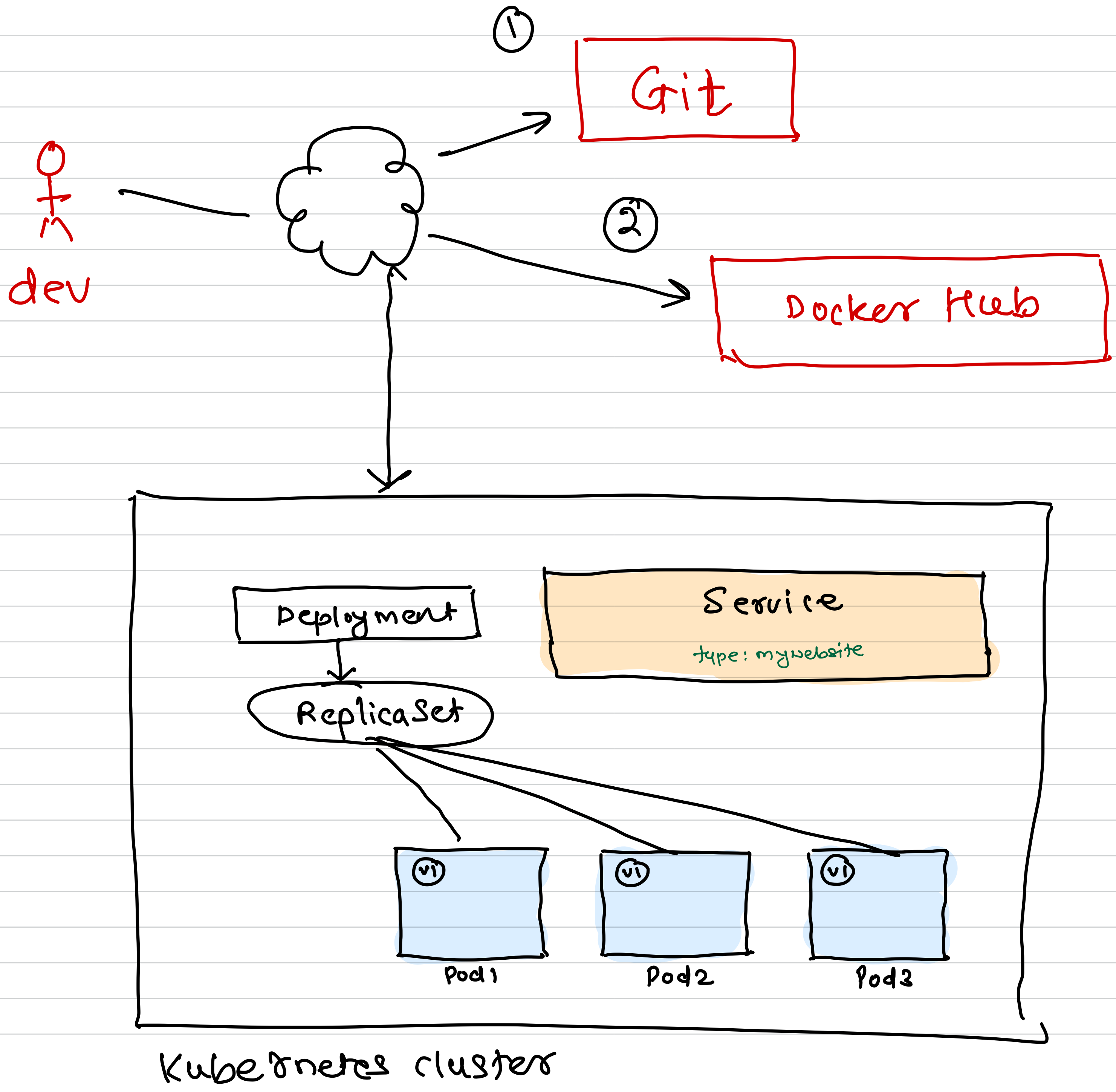
Deployment functionalities

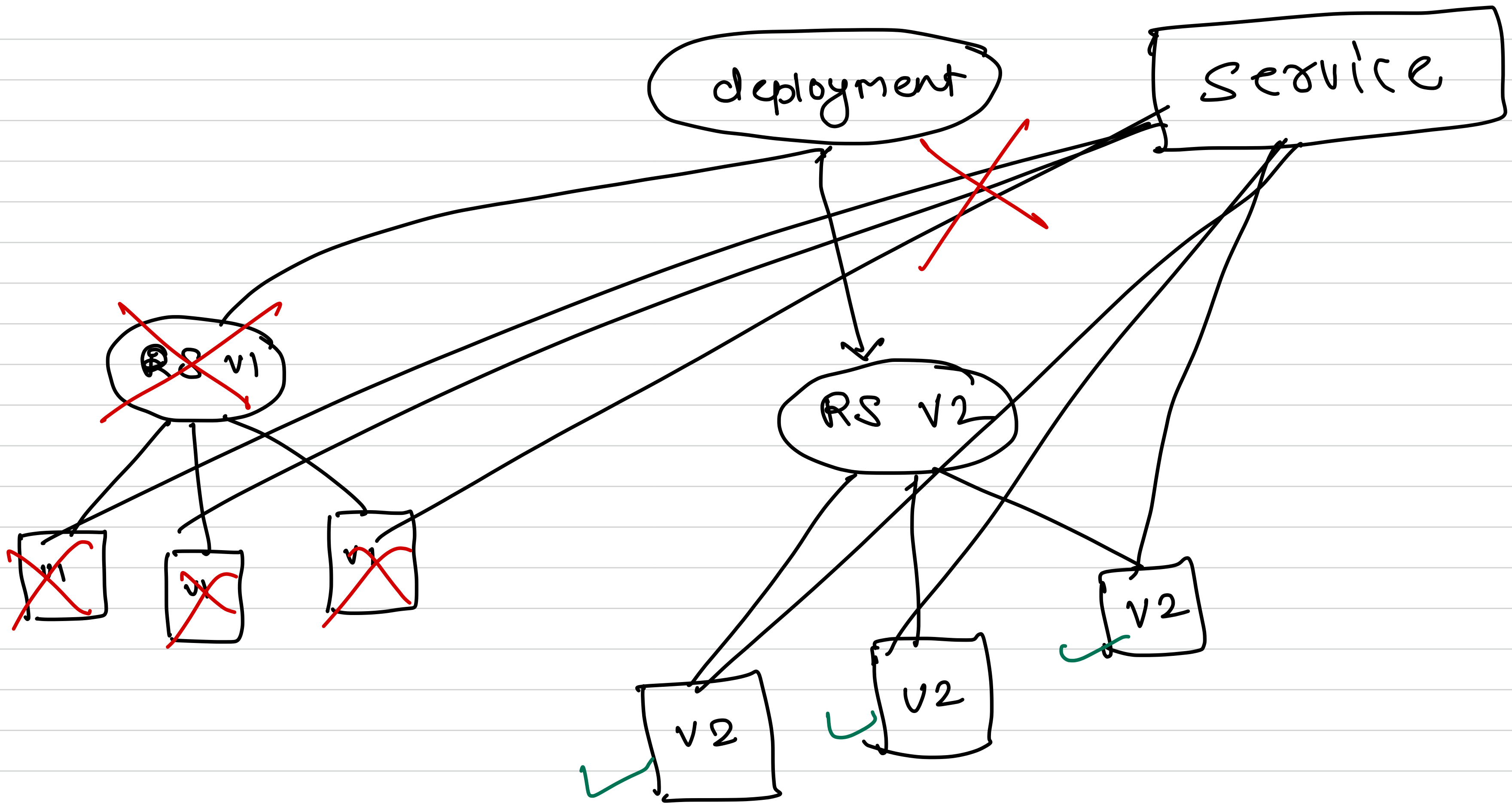
- Updating
- Rolling Back
- Scaling
- Pausing and Resuming

① update yaml
② k edit deployment <..>
③ k set image deploy <name> \
<container name> = <image>



→ ① create image
② push image
③ testing
④ deploy


1
Git
dev
2
Docker Hub
Deployment
Service
type: mywebsite
ReplicaSet
v1
v1
v1
Pod1
Pod2
Pod3
Kubernetes cluster


deployment
service
RS v1
RS v2
v1
v1
v1
v2
v2
v2

Configuration (18%)

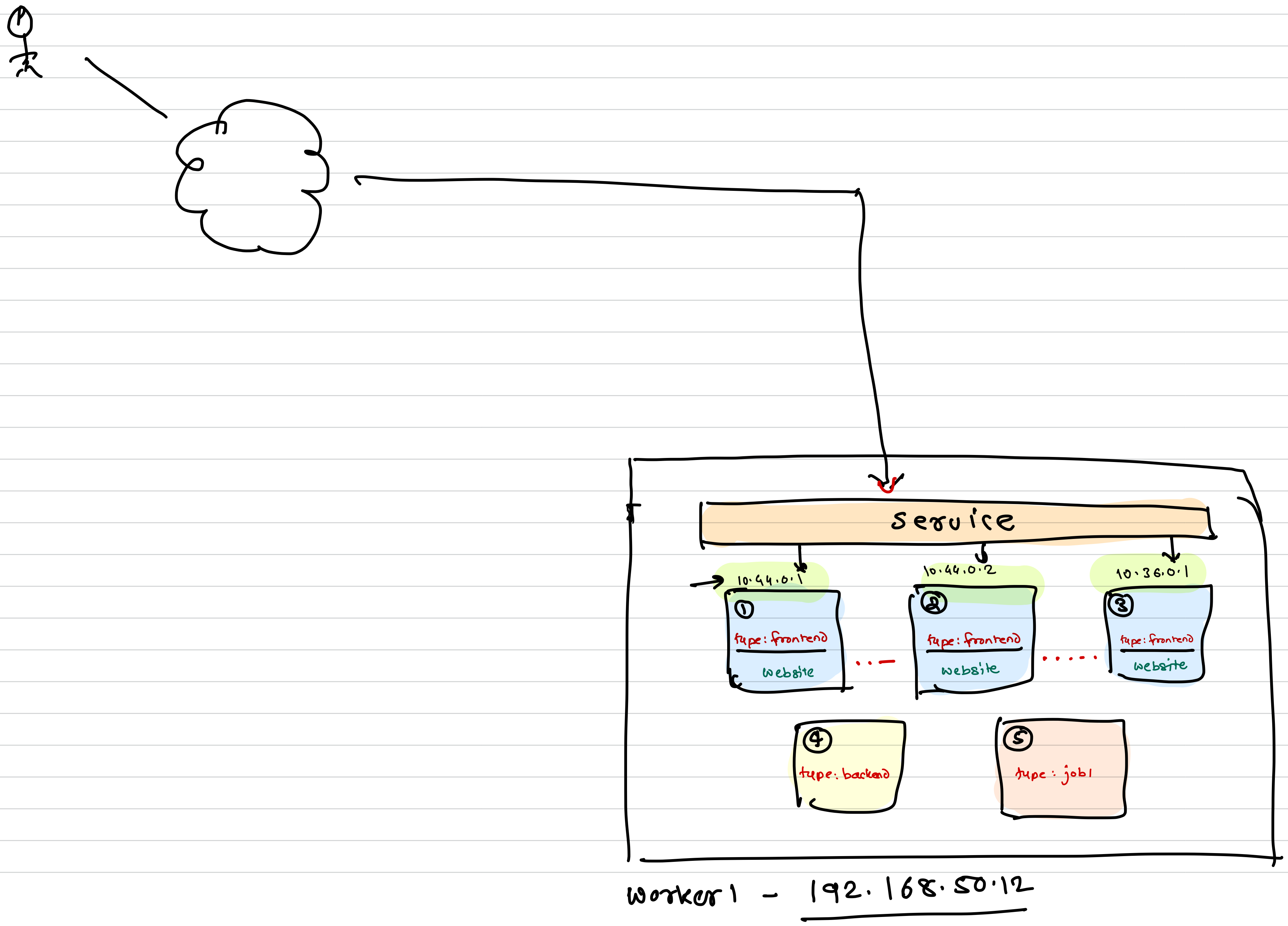
Secrets

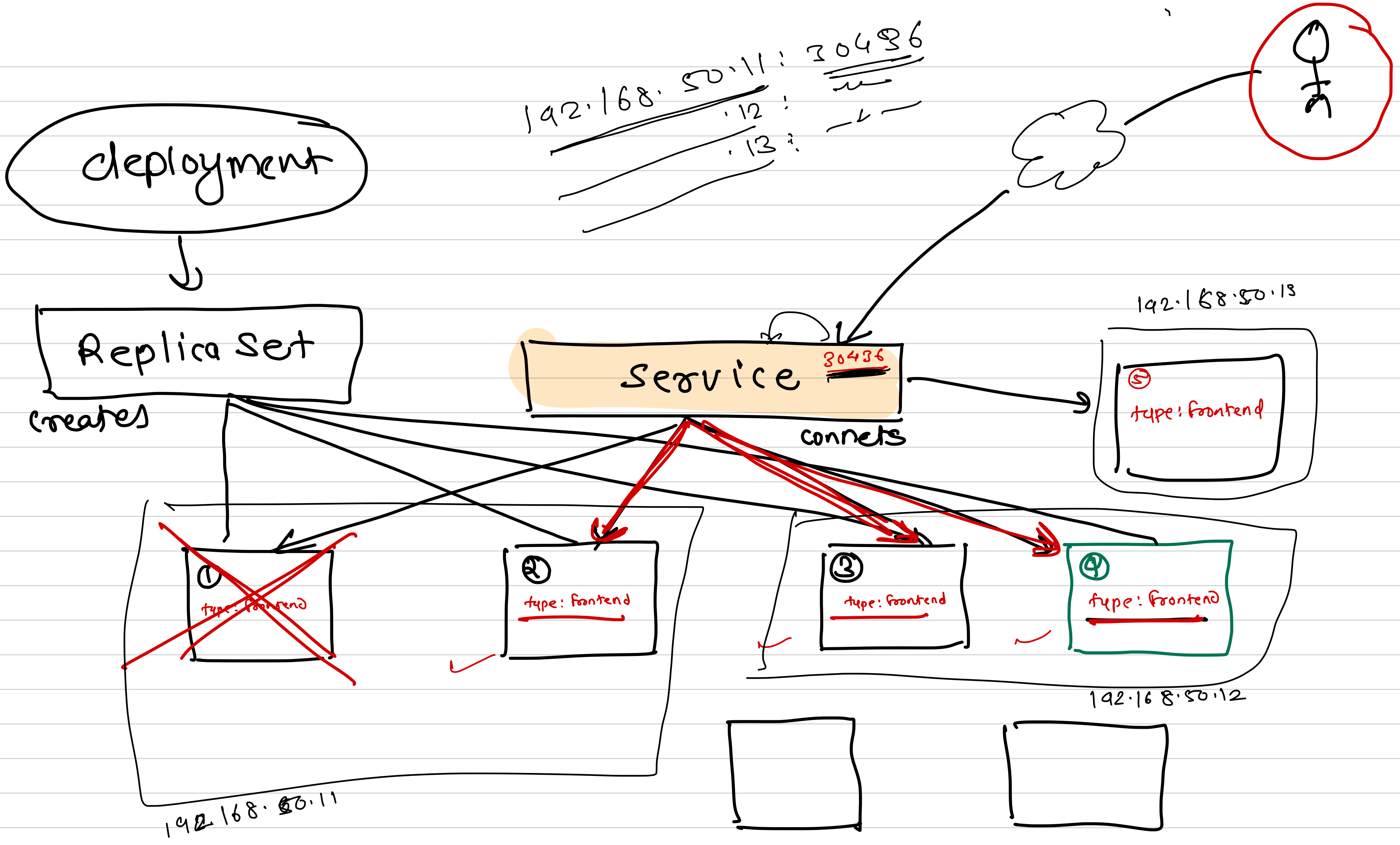
- Usually used to pass initial data for application
- Contains a small amount of sensitive data such as a password or a token
- Represented by a Kubernetes object
- Sensitive data in a Secret object allows for more control and reduces the risk of accidental exposure
- Can be created administrators or developers
- Needs to be created before the pod that depends on it
- Individual secrets are limited to 1MB in size
- Tasks
  - Create secret with key-value pairs
  - Create a pod which consumes the secret

ConfigMaps

- Used to store non-confidential data in key-value pairs
- Pod can consume the ConfigMaps as environment variables, command line arguments or as configuration files in a volume
- Tasks
  - Create a config map with key-value pairs
  - Create a pod that consumes the config map

Services & Networking (13%)


service
10.44.0.1
10.44.0.2
10.36.0.1
1
type: frontend
website
2
type: frontend
website
3
type: frontend
website
4
type: backend
5
type: job1
worker1 - 192.168.50.12



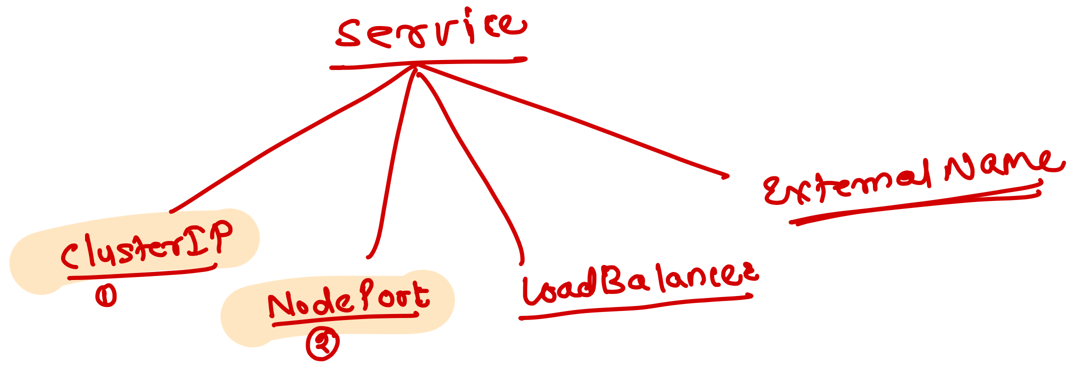
192.168.50.11 : 30436
.12 :
.13 :
deployment
Replica Set
creates
Service
30436
connects
192.168.50.13
5
type: frontend
1
type: frontend
2
type: frontend
3
type: frontend
4
type: frontend
192.168.50.12
192.168.50.11

Need of Services

- Kubernetes pods are mortal, means they can die because of any reason
- Kubernetes provide different controllers to manage the lifecycle of these pods.
- Kubernetes gives a pod its own unique IP address, but if the pod dies then the IP address changes
- It becomes very difficult to keep track of which IP address to connect to access the application
- The problem increases in multi-tier applications

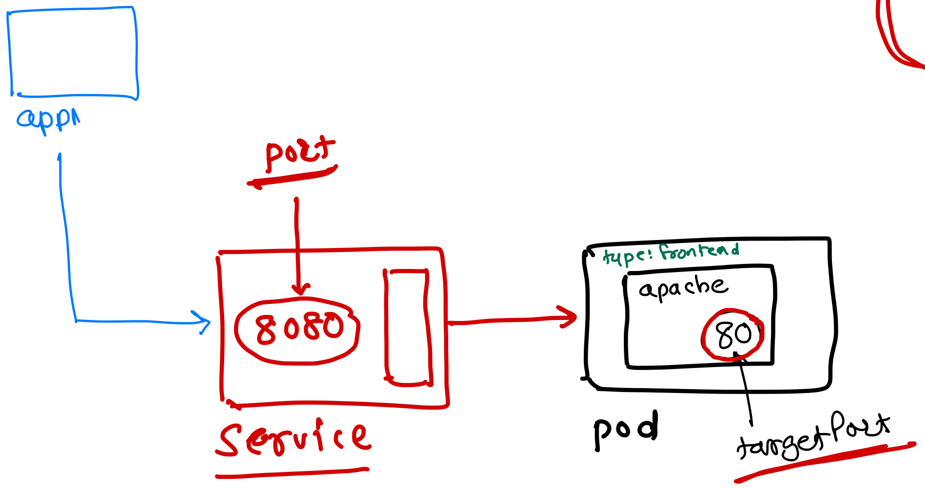
Service

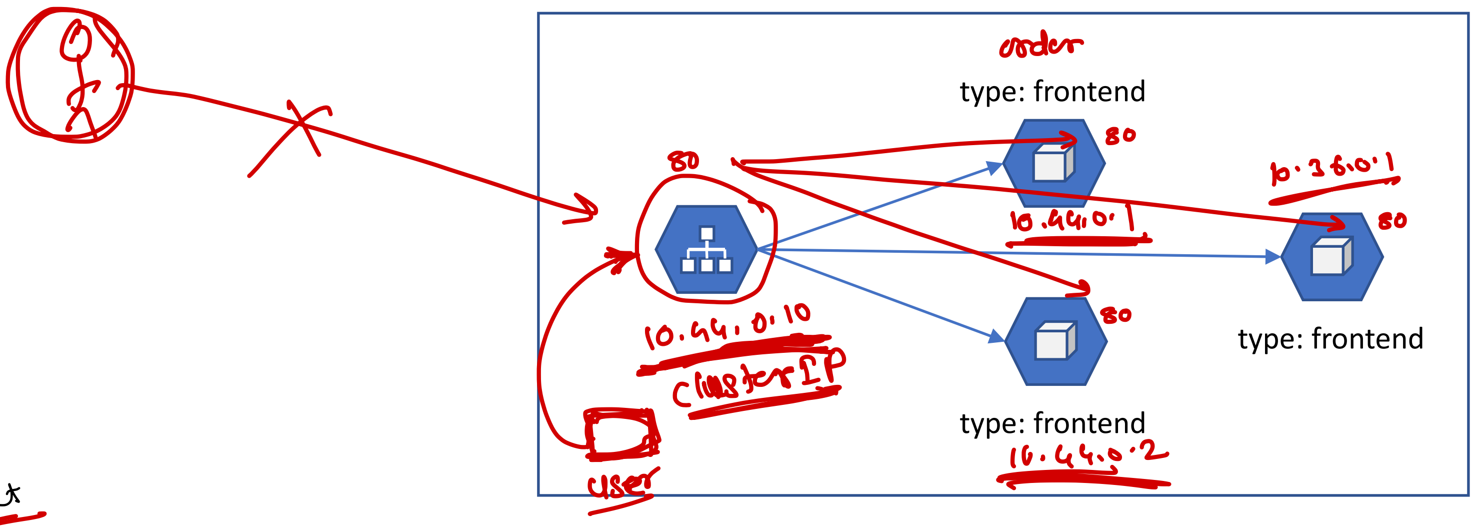
- Service provides an abstraction for a set of Pods and a policy to access them
- The set of Pods targeted by a service are determined by selector
- Service is used to expose an application running in set of pods
- It provides a single DNS name and can load balance across them
- Represented by Kubernetes API object and it is namespaced



Service Type: ClusterIP

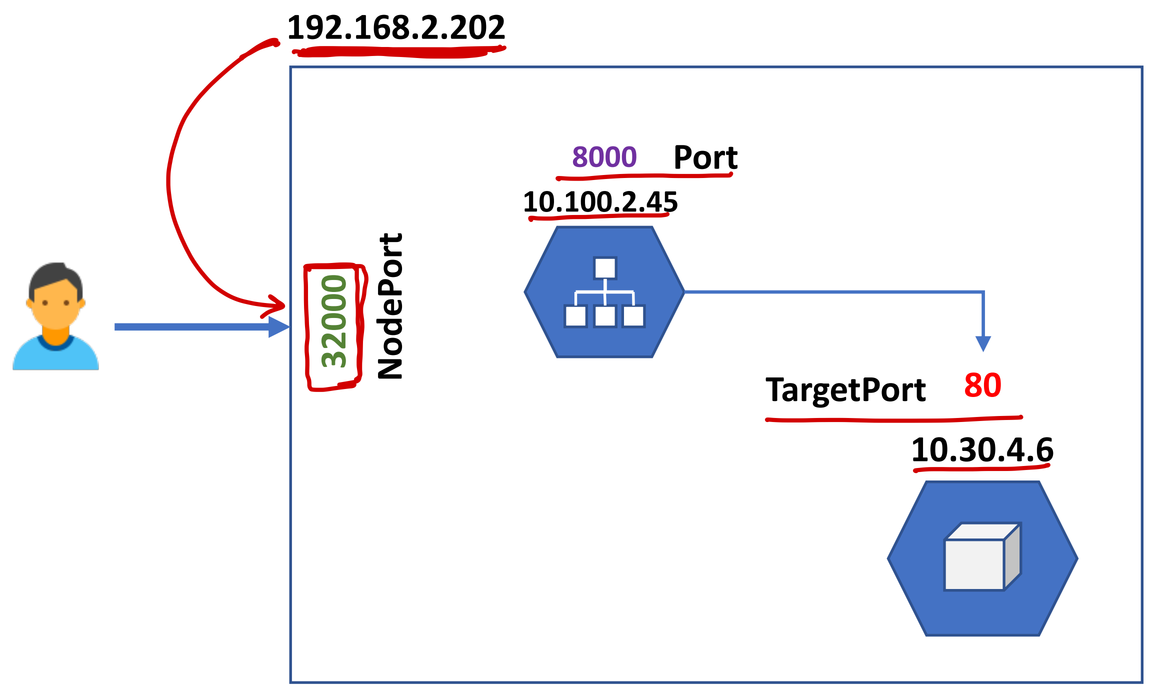
- Exposes the service within the cluster
- It uses cluster internal IP which is not reachable from external network
- Service maps any incoming port to a targetPort





Service Type: NodePort

- Exposed on each node's IP address
- NodePort service can be accessible from external network
- Service provides the NodePort on which the application is accessible
- ClusterIP service gets created automatically



Service Types

- LoadBalancer
  - Exposes the service externally using a cloud provider's load balancer
  - Along with the LoadBalancer, ClusterIP and NodePort services are created automatically
  - External load balancer routes the traffic through these pods
- ExternalName
  - The service does not contain the selectors, instead it uses DNS names
  - It maps the pod to the external name like test.sunbeaminfo.com
  - Mainly used to expose and object using cname record

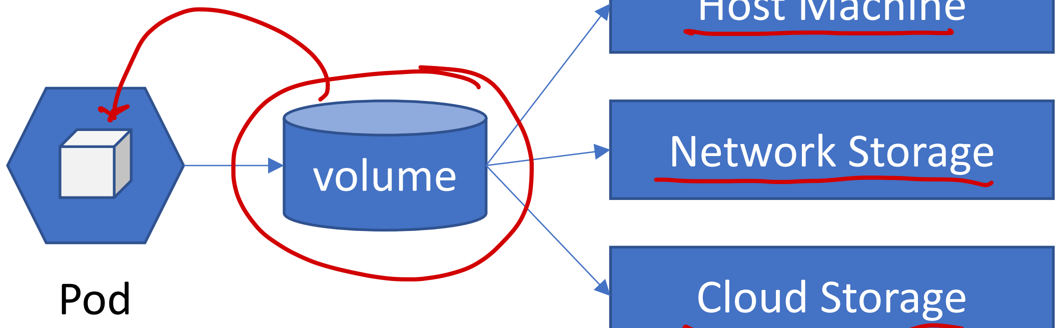
State Persistence (8%)

Storage

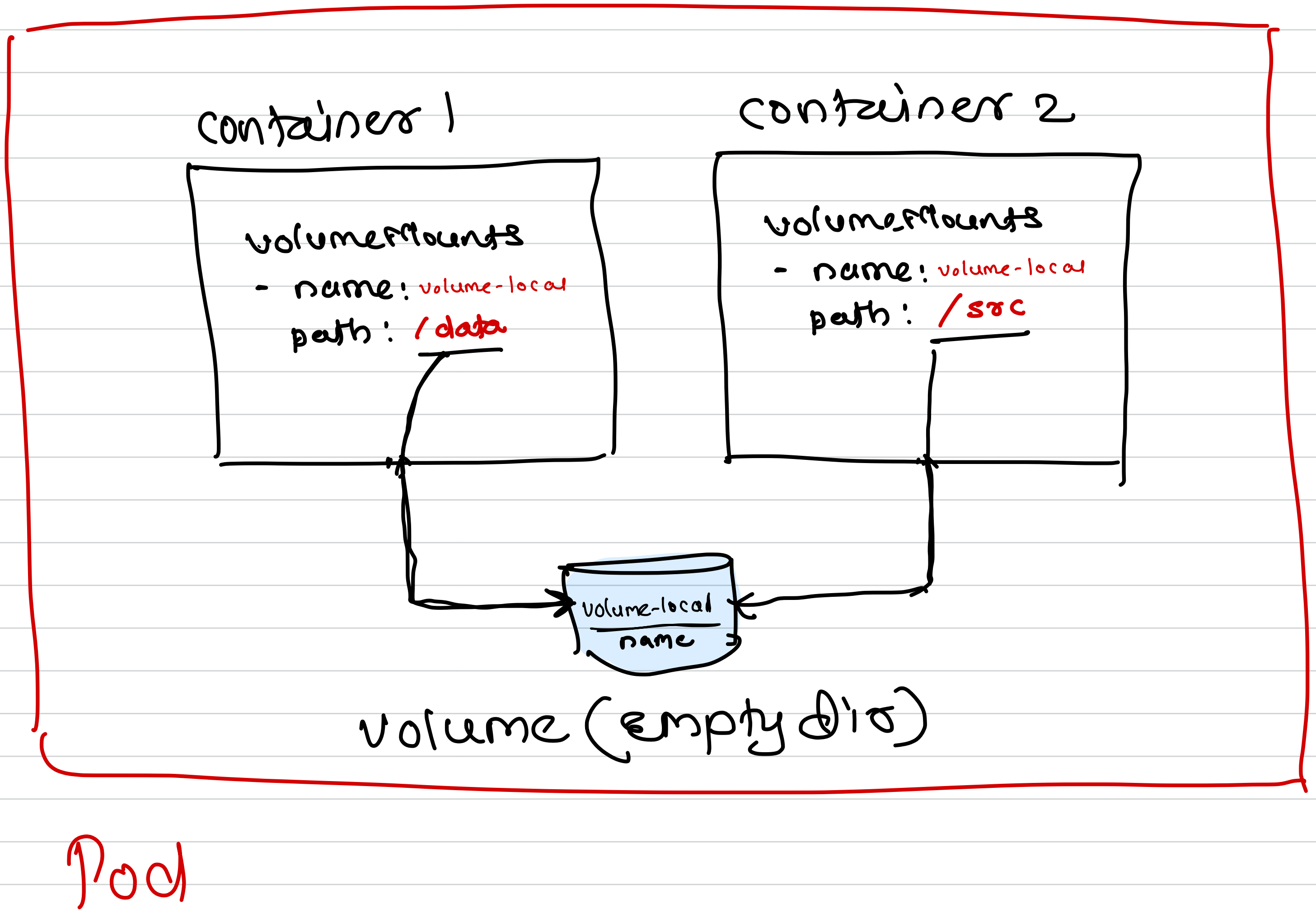
- The Pod and container running in Pod are ephemeral
- That is the data can not be persisted inside the container or Pod
- If you want to persist the data, then it has to be stored outside of the pod
- Also the initial data to start the pod need to be supplied from outside as it is not practical to send this data inside the yml file or in the image file
- To solve this problem, Kubernetes provides the storage

Volumes

- Used to persist the data outside the pod
- Volumes are mounted on the containers
- There are different ways the volumes can store the data
  - Host machine
  - Network Storage
  - Cloud Storage
- Volume types
  - emptyDir
  - hostPath
- Volume Drivers
  - Local
  - AzuerFileStorage
  - Flocker



SideCar

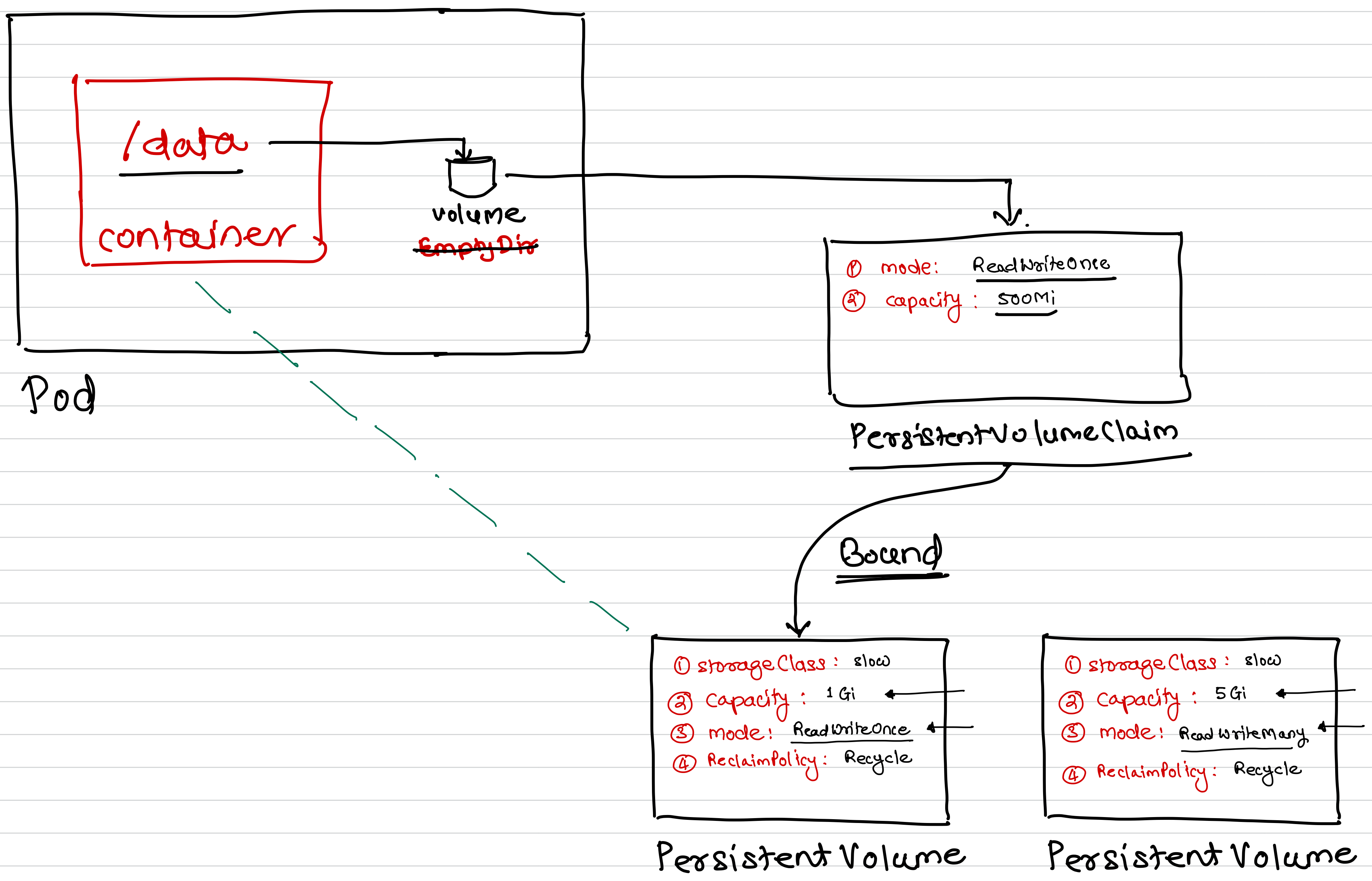

container 1
volumeMounts
- name: volume-local
path: /data
container 2
volumeMounts
- name: volume-local
path: /src
volume-local
name
volume (emptyDir)
Pod

PersistentVolume

- A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using storage classes
- It is a resource in the cluster just like a node is a cluster resource
- It has a lifecycle independent of any individual Pod that uses the PV
- Task
  - Create a PersistentVolume

PersistentVolumeClaim

- A PersistentVolumeClaim (PVC) is a request for storage by a user
- It is similar to a Pod: Pods consume node resources and PVCs consume PV resources
- Claims can request specific size and access modes
- Task
  - Create PersistentVolumeClaim
  - Create a Pod and consume PV using PVC


/data
container
volume
~~EmptyDir~~
Pod
① mode: ReadWriteOnce
② capacity : 500Mi
PersistentVolumeClaim
Bound
① storageClass : slow
② capacity : 1Gi
③ mode: ReadWriteOnce
④ ReclaimPolicy : Recycle
Persistent Volume
① storageClass : slow
② capacity : 5Gi
③ mode: ReadWriteMany
④ ReclaimPolicy : Recycle
Persistent Volume