Kubernetes Configurations and Advance Pod Concepts

- starts at 9:05 pm

AGENDA

1. Kubectl apply summary
2. Labels and Annotations
3. Multi-container Pods and use Cases
4. Init Containers
5. Replication Controller and ReplicaSets
6. Deployments Introduction
7. DaemonSets in Kubernetes
8. Configmaps
9. Secrets
10. Inject Data into Applications
11. Resource Settings for pods in Kubernetes
12. Service Accounts
13. Pod Scheduling and Placement
14. Taints and Tolerations
15. Node Affinity/ Anti Affinity
16. Node Selectors and Labels

Kubectl Apply summary. → kubectl apply -f pod.yaml

① Kubectl client.

↓

② API server.

↓

③ etcd (podspec)

↓

④ controller manager. → (API server create pod (pending))

↓

⑤ scheduler → updates podspec with the node it has decided. (etcd)

↓

⑥ Kubelet

The **Kubelet** on the assigned node continuously watches the API server for Pods scheduled to run on its node.

↓ Pulls pod spec from API server.

→ starts the pod.

⑦ CRI (pulls the image)
→ Run the containers.

⑧ Kube proxy. (Networking)

Summary: Flow of Events

1. **kubectl apply** → Reads `kubeconfig` to authenticate with the API server and sends the Pod spec.

2. **API Server** validates the request, writes the desired state to **etcd**, and notifies other components.

3. **Controller Manager** creates the Pod in the API server (Pending state).

4. **Scheduler** updates the Pod spec in **etcd** with the node it has selected for the pod, and informs the API server.

5. **Kubelet** on the assigned node fetches the Pod spec, prepares the environment, and starts the containers.

6. **Container Runtime** runs the containers, and **Kube-Proxy** ensures networking is properly configured.

keeps monitoring the pod.

Labels and Annotations

① Labels

pod1

env: prod

→ Use Case

① Selection.

→ Selectors (deployments, Replicaset)

② Filtering.

kubectl get pods -l env:prod.

③ grouping.

labels: → key

app: my-app → value

environment: production

tier: backend

Annotations.

key : value.

eg:→ giving description.

→ Automation.

Jenkins, argoCD.

annotations:

build: "v1.2.3"

description: "Pod running a production application"

maintainer: "team@mycompany.com"

pod.yaml

apiVersion: v1

kind: Pod

metadata:

name: demo-pod

labels:

app: demo-app

environment: dev

team: devops

annotations:

description: "This is a demo pod for labels and annotations"

build: "v1.0"

spec:

containers:

```
  - name: nginx-container
    image: nginx
```

Command to show labels of the pod

```
kubectl get pods --show-labels
```

You can use `kubectl label` to add labels to your resources.

```
kubectl label pod pod-name app=nginx
```

Filtering pods based on labels

```
kubectl get pods -l environment=production
```

Label nodes

```
kubectl label node node-name region=us-west
```

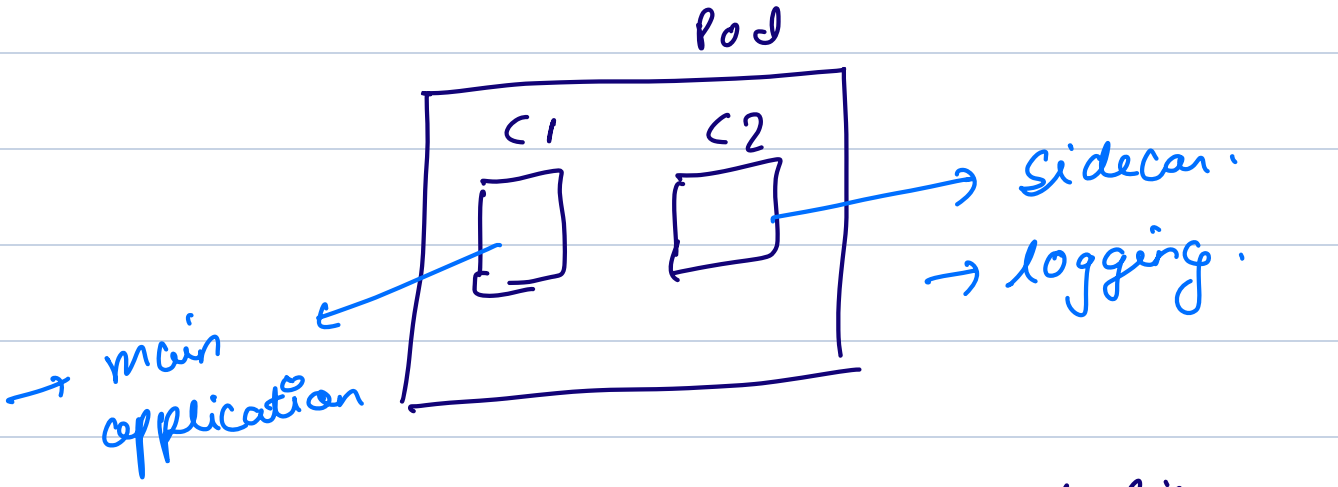
Get nodes based on labels

```
kubectl get nodes -l role=frontend
```

Adding Annotations to a Pod

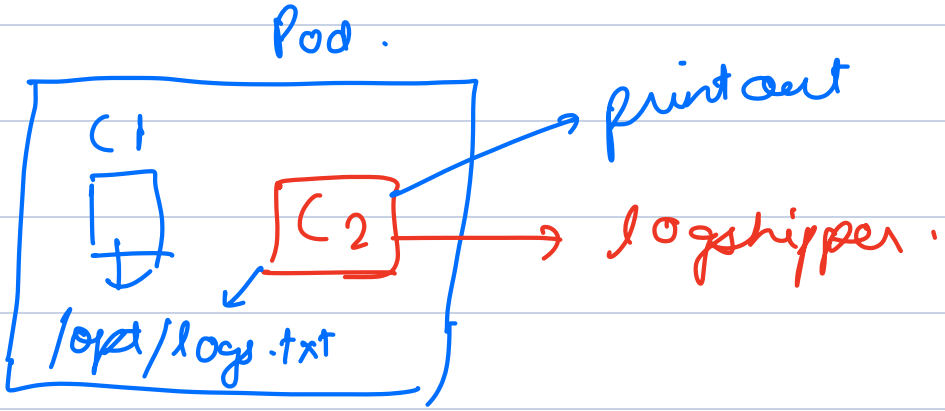
```
kubectl annotate pod pod-name git-commit-hash=abc1234
```

Multi container Pods



→ they share same pod lifecycle

① Sidecar Container.




```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
  labels:
    app: myapp
```

```
spec:
  containers:
    - name: myapp
      image: alpine:latest
      command: ['sh', '-c', 'while true; do echo "logging $(date)" >> /opt/logs.txt; sleep 1; done']
      volumeMounts:
        - name: data
          mountPath: /opt
    - name: logshipper
      image: alpine:latest
      command: ['sh', '-c', 'tail -F /opt/logs.txt']
      volumeMounts:
        - name: data
          mountPath: /opt
  volumes:
    - name: data
      emptyDir: {}
```

```
kubectl logs -f myapp -c logshipper
```

```
kubectl exec -it myapp -c myapp -- sh
```

→ init Containers.

① Initialising Tasks →
→ Downloading

② Preflight checks.
→ verifying DB connection
→ checking for files.

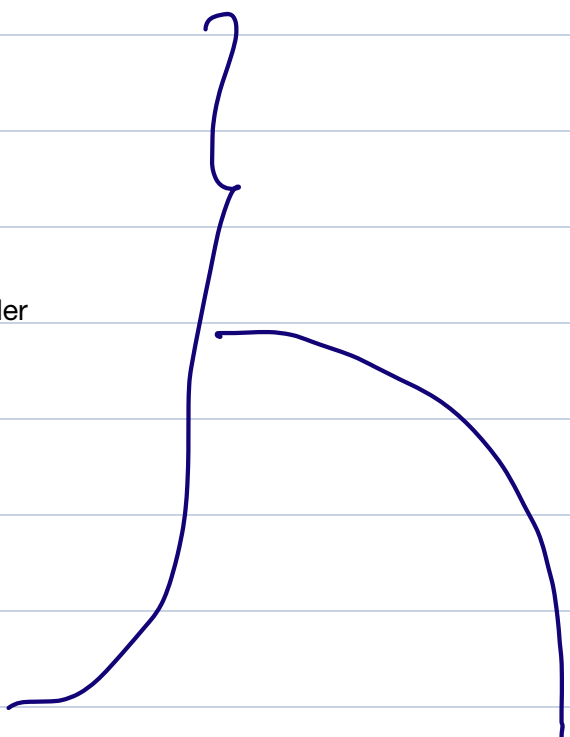
```
apiVersion: v1
kind: Pod
metadata:
 name: init-demo
spec:
 initContainers:
 - name: init-container
  image: alpine:latest
  command: ['sh', '-c', 'echo "Initializing the app..." > /data/init.txt; sleep 5']
  volumeMounts:
   - name: shared-data
    mountPath: /data
 containers:
 - name: main-container
```

```
    image: alpine:latest
    command: ['sh', '-c', 'while true; do echo "Main container is running"; sleep 1; done']
    volumeMounts:
      - name: shared-data
        mountPath: /data
  volumes:
    - name: shared-data
      emptyDir: {}
```

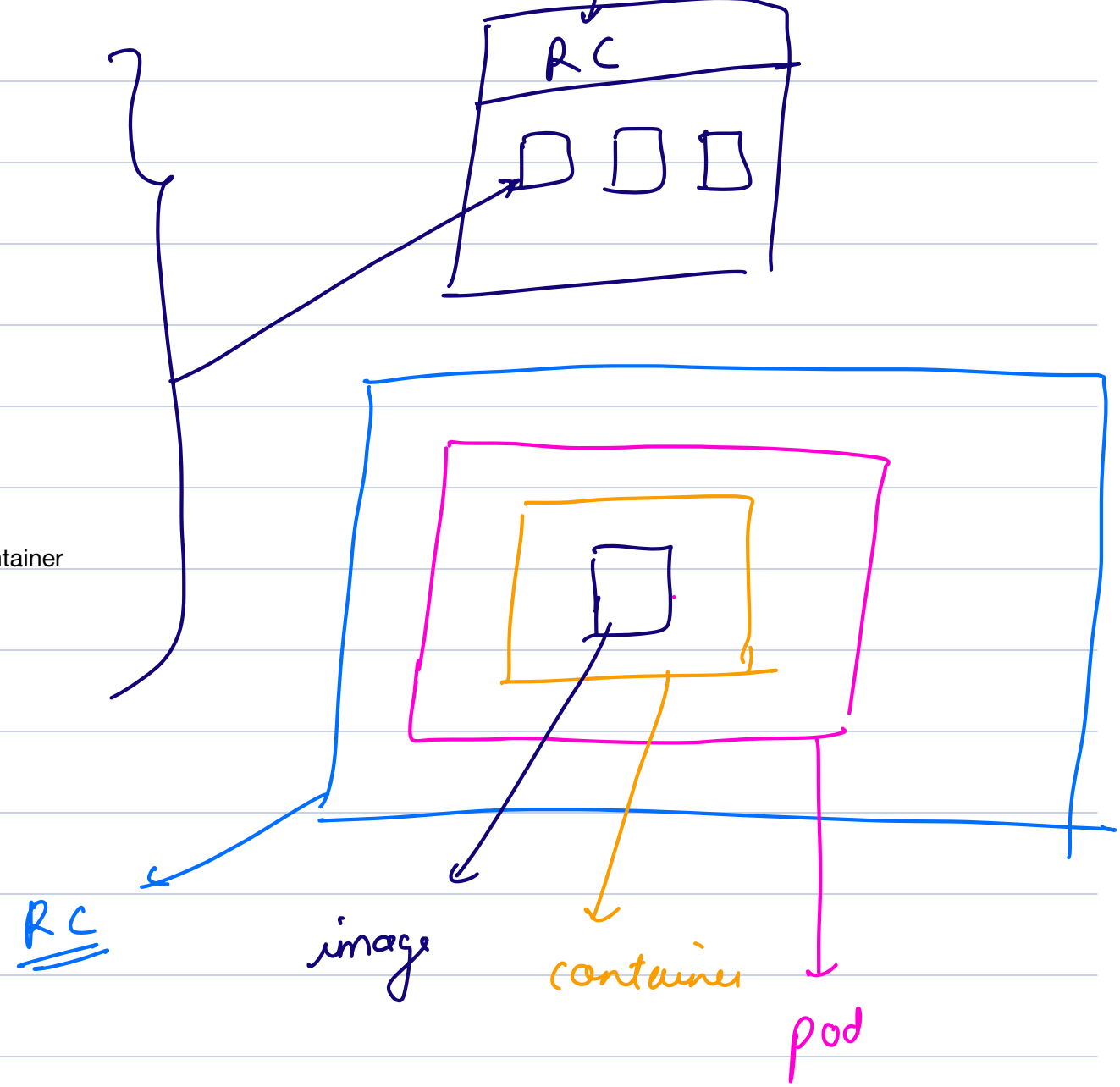
Break → 10:20 pm

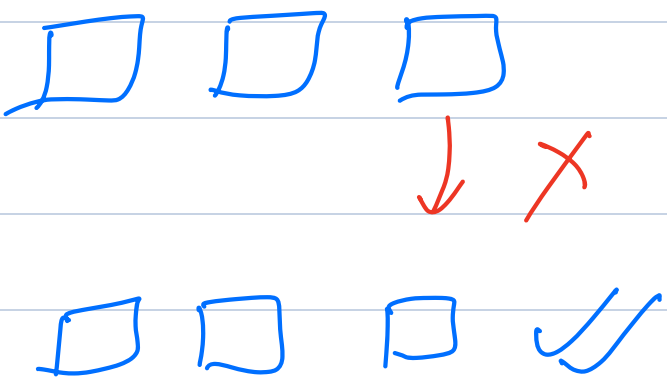
→ Replication Controller & Replicaset

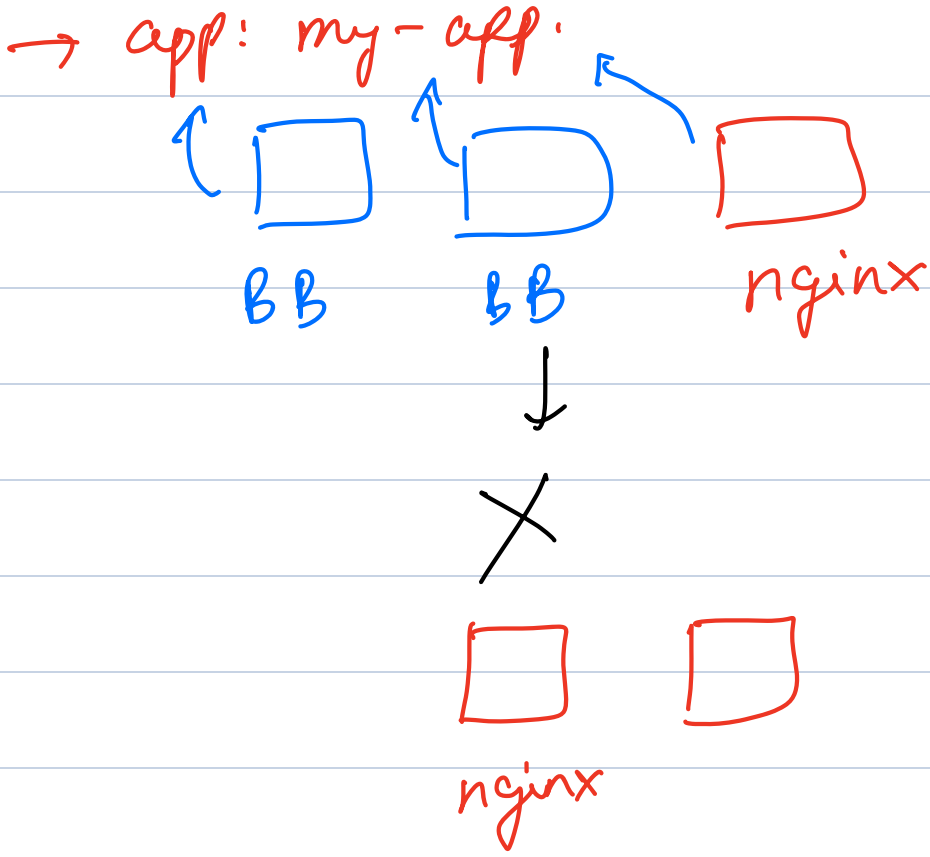
```
apiVersion: v1
kind: ReplicationController
metadata:
  name: my-replication-controller
spec:
  replicas: 3
  selector:
    app: my-app
```



```
template:
  metadata:
    labels:
      app: my-app
  spec:
    containers:
    - name: my-container
      image: nginx
```







Feature	ReplicationController	ReplicaSet
Label Selector	Supports **equality-based selectors** (e.g., `key=value`).	Supports both **equality-based** and **set-based selectors** (e.g., `key in (value1, value2)`).
Flexibility	Less flexible in managing pods.	More flexible due to advanced label selectors.
Integration with Deployments	Not used by Deployments.	Used as the underlying mechanism for Deployments.
Purpose	Basic functionality for managing pod replicas.	Enhanced functionality for modern Kubernetes workflows.
Deprecation	Considered **legacy** and no longer recommended for new setups.	Actively supported and preferred.

selector:

app: nginx

selector:

matchExpressions:

→ set based selector.

```
- key: app
  operator: In
  values:
  - nginx
  - apache
```

app = [nginx, apache]

Pod 1 app = nginx

Pod 2 app = apache.

1. **Ensures Desired State**: It ensures that a specific number of pod replicas are always running.

2. **Supports Set-based Selectors**: It can filter pods using labels with set-based selectors.

3. **Scaling**: You can scale the number of replicas up or down.

4. **Self-healing**: If pods fail, it ensures new ones are created to replace them. Increasing or decreasing the number of replicas).

→ Demo

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-replicaset
spec:
  replicas: 3
```

```
  selector:
    matchExpressions:
    - key: app
      operator: In
      values:
      - my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: nginx-container
        image: nginx
        ports:
        - containerPort: 80
```

```
kubectl apply -f replicaset.yaml
```

```
kubectl get rs
```

kubectl get pods | grep replic

kubectl delete pod (any 1 pod)

Scale Out ReplicaSet

kubectl scale rs my-replicaset --replicas=5

Scale in Replicaset

kubectl scale rs my-replicaset --replicas=1

Limitations

→ No rolling updates

update image in replicaset → ✗

→ No rollbacks.

→ Deployments

→ Rolling update

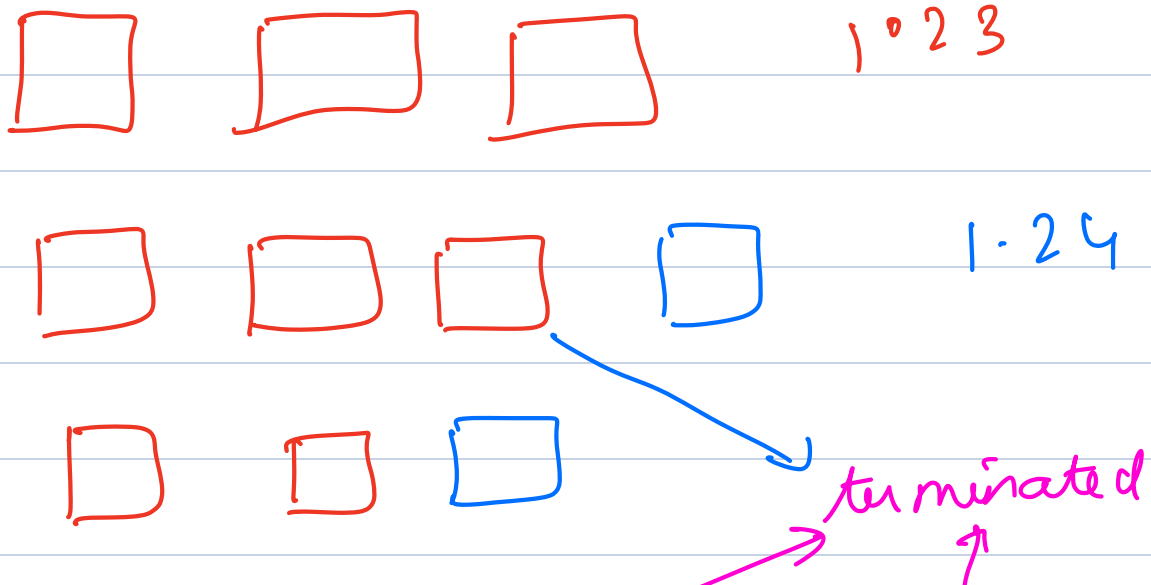
→ Rollbacks.

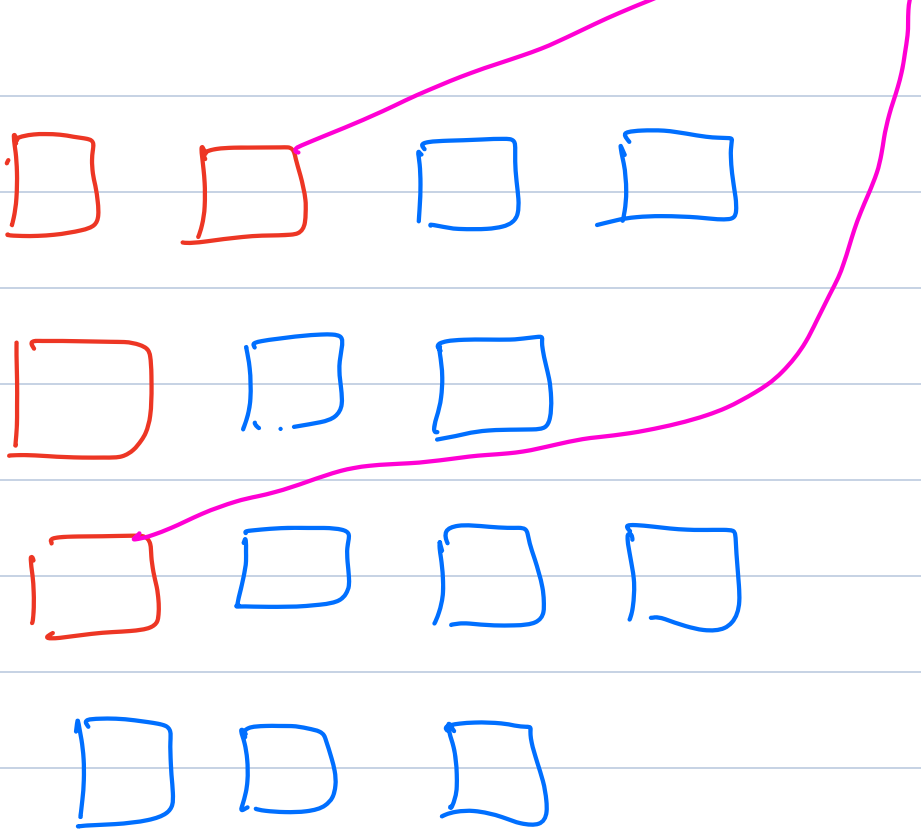
Deployments provide a higher abstraction layer, enabling features like rolling updates, rollbacks, and version management while relying on ReplicaSets under the hood.

A **Deployment** is an abstraction that manages ReplicaSets and ensures the desired state of your applications. It is the most common way to run applications in Kubernetes and adds rolling update functionality and rollback features.

How Deployment Works

1. **Pod Creation**: Kubernetes will create a new ReplicaSet under the hood when you create a Deployment. The ReplicaSet will manage the pods according to the `template`.
2. **Rolling Updates**: When the Deployment is updated (e.g., a new container image is provided), Kubernetes performs a rolling update by gradually replacing old pods with new ones.
3. **Rollbacks**: If something goes wrong with the update, you can rollback to the previous version of the Deployment.
4. **Scaling**: Similar to ReplicaSets, you can scale the number of pods by adjusting the `replicas` field in the Deployment.





```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
```

```
      app: my-app
  spec:
    containers:
    - name: nginx-container
      image: nginx:1.23
      ports:
      - containerPort: 80
```

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

Update the Deployment:

```
kubectl set image deployment/my-deployment nginx-container=nginx:1.24
```

```
kubectl get pods -w | grep my-deployment
```

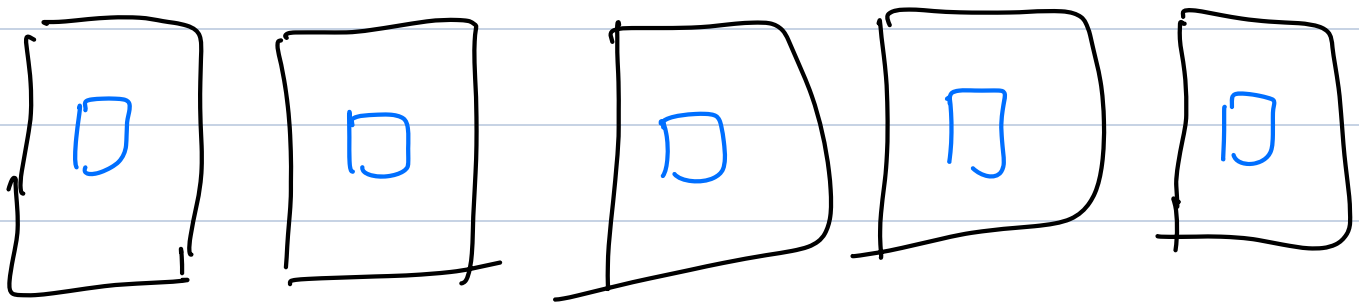
Rollback the Deployment:

```
kubectl rollout undo deployment/my-deployment
```

Rollback history - View the history of rollouts for `my-deployment`

```
kubectl rollout history deployment/my-deployment
```

→ Daemonsets



logging agents
monitoring
networking components

nodeSelector:

role: storage

role = storage.

DS → YAML

```
apiVersion: apps/v1

kind: DaemonSet

metadata:

  name: nginx-daemonset

spec:
```

```
  selector:
    matchLabels:
      name: nginx
  template:
    metadata:
      labels:
        name: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

with nodeselector.

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: custom-nginx-daemonset
spec:
  selector:
    matchLabels:
      name: nginx
```

```
 template:
   metadata:
     labels:
       name: nginx
   spec:
     nodeSelector:
       kubernetes.io/hostname: new-cluster1-worker2
     containers:
     - name: nginx
       image: nginx:1.14.2
       ports:
       - containerPort: 80
```