



Kubernetes (K8s)

# Background.

- **An open-source system for automating the deployment, scaling and management of containerized applications.**
  - **A container orchestration platform.**
- **Originally announced by Google in 2014.**
- **In 2015, Google donated it to the Cloud Native Computing Foundation (CNCF).**
- **Kubernetes is the Greek word for helmsman – a person that steers a ship that carries containers of goods.**

# The problems it solves.

- **Problem 1: An application is comprised of multiple containers running on a cluster of nodes (on-premise server, Cloud VMs like AWS EC2). A container or an entire node crashes.**
  - **K8s provides monitoring and self-healing, to ensure high availability.**
- **Problem 2: A container experiences a spike in traffic resulting in increased latency.**
  - **K8s provides auto-scaling and load balancing. It replicates the container on the node and/or across the cluster of nodes and load balances the requests between them. It scales up and down based on demand.**

# The problems it solves.

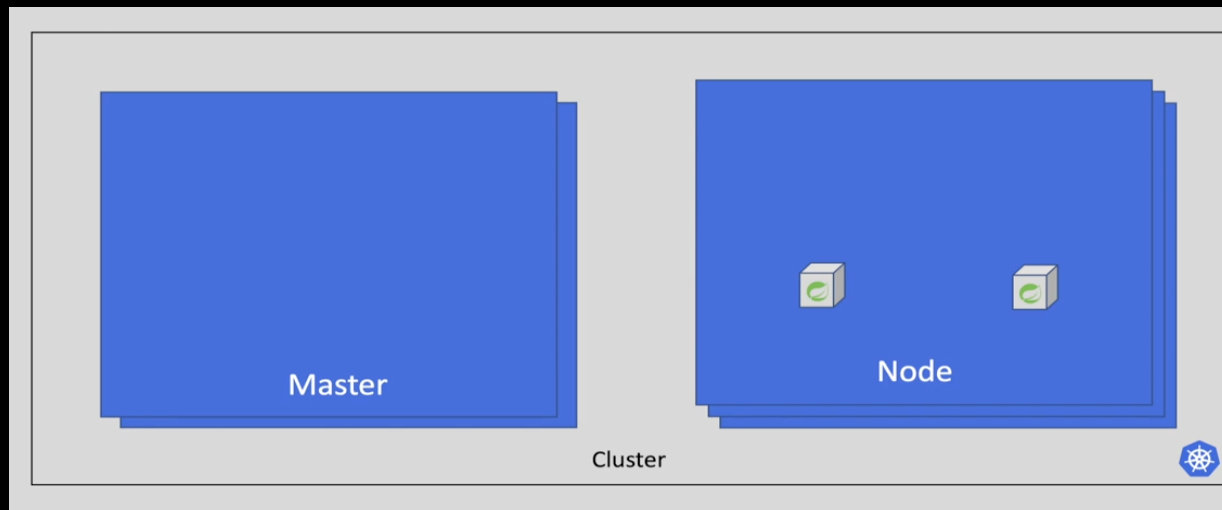
- **Problem 3: You regularly upgrade an image and want to apply the change to a fleet of running containers without effecting downtime.**
  - **K8s provides rolling deployments - delete and restart each container, one by one rather than all at once.**
  - **Also supports canary deployments.**
- **Problem 4: I want to upgrade my entire application from v1 to v2.**
  - **K8s provides automatic rollout and rollback.**

# The problems it solves.

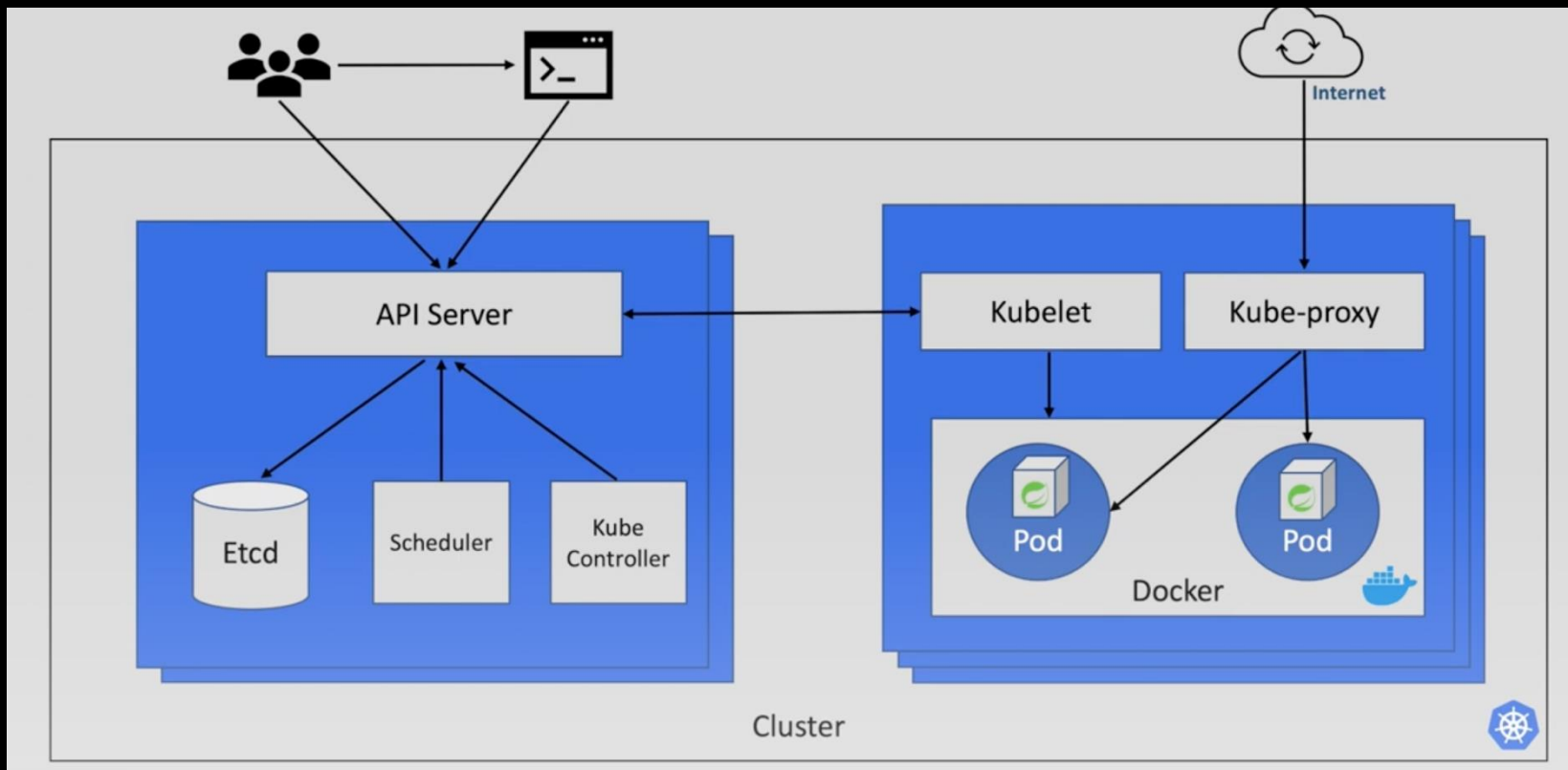
- **Problem 5: I want to reconfigure my application's without rebuilding images.**
- **K8s provides secrets and configuration mappings to safeguard sensitive data and avoid unnecessary rebuilding of images.**

# K8s Architecture

- K8s is installed over a set of nodes (VMs)
- Worker nodes host containers - the Data plane
- Master nodes (Control plane) manages the workers – More than one master for fault tolerance and high availability.
- The set of masters and workers is called a cluster

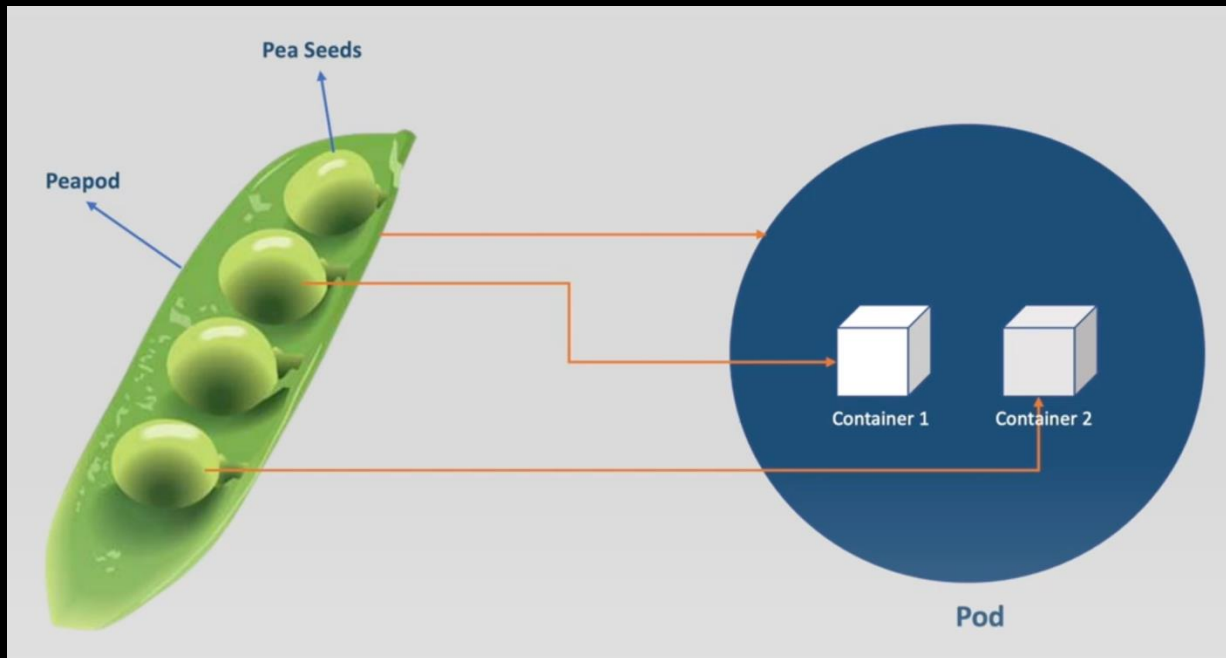


# K8s Architecture



# All about Pods

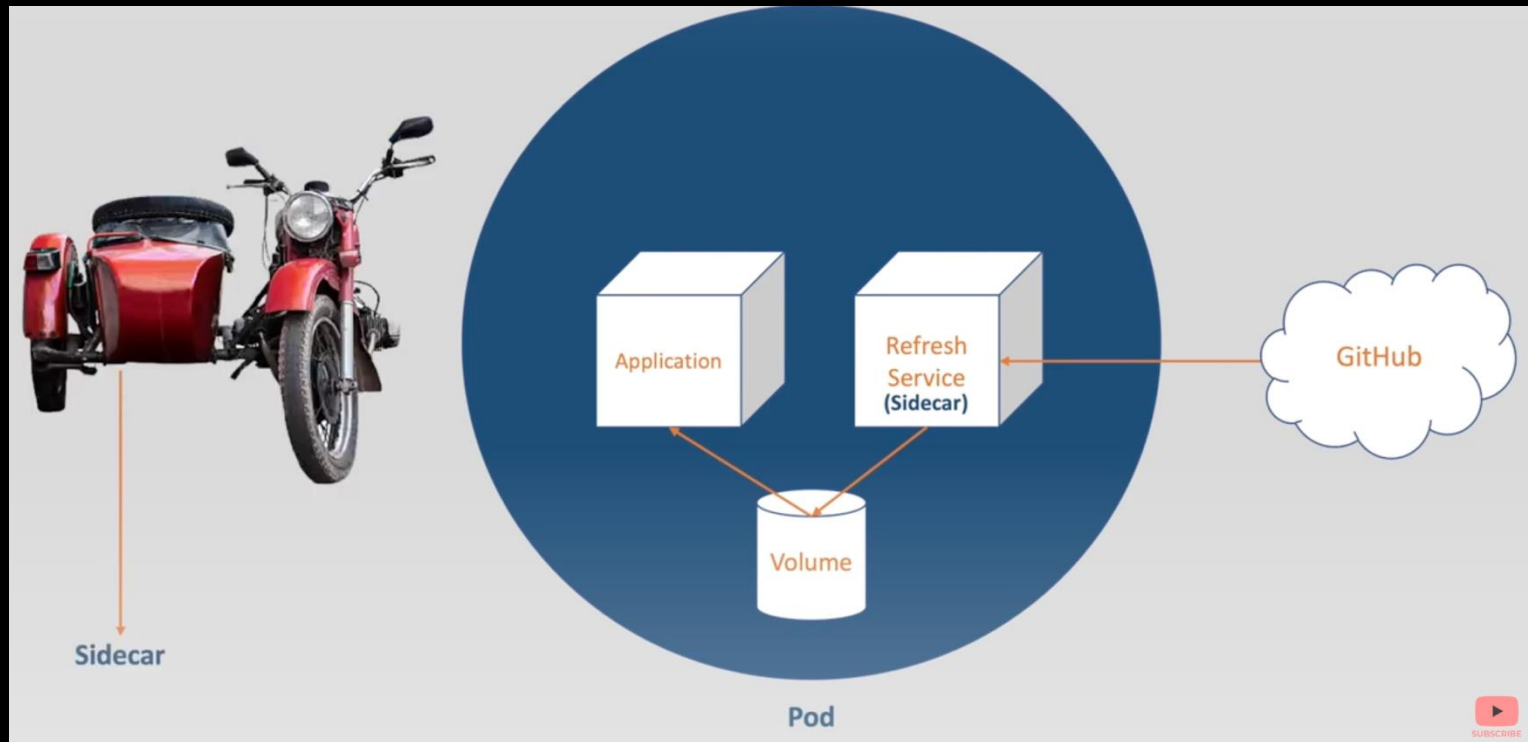
- The smallest unit of deployment is a pod.
- It encapsulates a group of containers, that share the same localhost network and storage.
- A pod is assigned a private IP address, but it's dynamic.





# All about Pods

- When a pair of containers are tightly coupled, they must reside on the same worker node. The pod construct ensures this is satisfied.



# Pod basics

- The containers in a pod share the same network (localhost), but use different ports.
- Deleting a pod will delete all its containers.
- To scale an app, we replicate its pod, where each one has its own group of containers.
- IP addressing:
  - A pod is assigned a private IP address.
  - On scale up, each pod replica is assigned its own IP.
  - Pod IPs are dynamic.

# Pod basics

```
➤ oc run nginx-pod --image=nginx:latest  
pod/nginx-pod created
```

```
➤ oc get pods
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-pod	1/1	Running	0	12s

- Creating Kubernetes resources (e.g. pods) from the command line is cumbersome; instead, we use declarative code files for better maintainability.
  - Files are termed manifests – yaml or json options.

# Pod basics - Manifest files.

```
➤ oc apply -f 01-nginx-pod.yaml  
pod/nginx-pod1 created
```

```
➤ oc get pods
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-pod	1/1	Running	0	3d5h
nginx-pod1	1/1	Running	0	16s

```
➤ oc get pods -l team=integration (Pod labels)
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-pod1	1/1	Running	0	2m7s

# Pod basics - Attributes

- The etcd stores lots of attributes about pods

➤ oc get pod nginx-pod1 -o wide

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE
READINESS GATES								
nginx-pod1	1/1	Running	0	13m	10.130.1.156	worker1	<none>	<none>

- Use describe command to get all etcd info:
  - oc describe pod nginx-pod1
    - Response also shows the ‘events’ that occurred for a pod.

# Pod basics - Debugging

- Three main options:

1. Port-forwarding is possible with cluster access:

➤ `oc port-forward nginx-pod1 3000:80`

Forwarding from 127.0.0.1:3000 -> 80

Forwarding from [::1]:3000 -> 80

2. Check container's logs:

➤ `oc logs nginx-pod1`

3. Open a terminal shell inside the container:

➤ `oc exec -it nginx-pod1 -- /bin/bash`

`root@nginx-pod1:/#`

# ReplicaSets

- For high availability of an app (pod), K8s can create multiple copies in the cluster, called replica sets.

```
➤ oc apply -f 02-nginx-replica.yaml
```

```
replicaset.apps/nginx-replica created
```

```
➤ oc get pods
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-replica-dmfvn	0/1	ContainerCreating	0	7s
nginx-replica-wc7j9	0/1	ContainerCreating	0	7s

```
➤ oc get rs      (rs - replicaset)
```

NAME	DESIRED	CURRENT	READY	AGE
nginx-replica	2	2	2	14s

# Self-healing

```
> oc delete pod nginx-replica-dmfvn
```

```
pod "nginx-replica-dmfvn" deleted
```

```
> oc get pods
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-replica-kkhs2	0/1	ContainerCreating	0	3s
nginx-replica-wc7j9	1/1	Running	0	7m36s

- The new pod's IP address is different to the one it replaced.
- A replica set's spec.selector property determines the pods it controls.
  - A pod's labels match the replica set's selector



# Deployments

- Deployments provide extra features over replica sets:
  - Rollout and rollback;
  - Deployment strategy.
  - Etc
- It automatically creates a replica set resource.
- The resource hierarchy:
  - A deployment manages a replica set.
  - A replica set manages a set of pods, based on matching the replica set's selectors with a pod's labels.
  - A pod manages its containers.

# Deployments

➤ oc apply -f 03-nginx-deployment.yaml

deployment.apps/nginx-depl created

➤ oc get deployments

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx-depl	1/2	2	1	16s

➤ oc get rs

NAME	DESIRED	CURRENT	READY	AGE
nginx-depl-7569b665c5	2	2	2	20s

➤ oc get po

NAME	READY	STATUS	RESTARTS	AGE
nginx-depl-7569b665c5-dvxrj	1/1	Running	0	25s
nginx-depl-7569b665c5-nd77g	1/1	Running	0	25s

# Deployment Updates

- Each Deployment update creates a new replica set

➤ `oc apply -f 03-nginx-deployment.yaml.` (After update)

`deployment.apps/nginx-depl` configured

➤ `oc get rs`

NAME	DESIRED	CURRENT	READY	AGE
<code>nginx-depl-588f77bb97</code>	2	2	1	15s
<code>nginx-depl-7569b665c5</code>	1	1	1	16m

➤ `oc get rs`

NAME	DESIRED	CURRENT	READY	AGE
<code>nginx-depl-588f77bb97</code>	2	2	2	30s
<code>nginx-depl-7569b665c5</code>	0	0	0	16m

# Deployment Rollout

➤ `oc apply -f 03-nginx-deployment.yaml` (1<sup>st</sup> deploy)

`deployment.apps/nginx-depl` **created**

➤ `oc rollout history deployment nginx-depl`

`deployment.apps/nginx-depl`

REVISION CHANGE-CAUSE

1 Initial deployment

- Suppose we edit and apply the YAML twice:

➤ `oc rollout history deployment nginx-depl`

`deployment.apps/nginx-depl`

REVISION CHANGE-CAUSE

1 Initial deployment

2 Change to Nginx 1.21.5

3 Change to port 82

# Deployment Rollback

- Rollback – Undo a Deployment rollout.

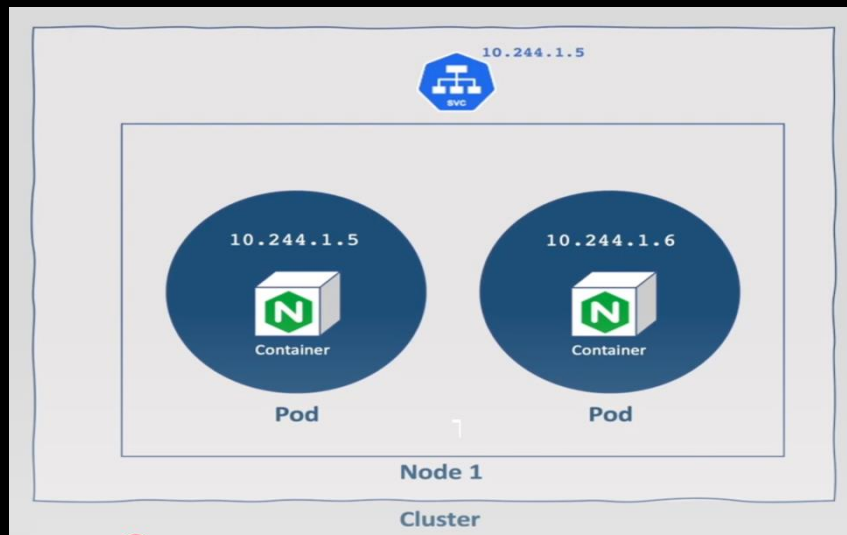
```
> oc rollout undo deployment nginx-depl  
--to-revision=2  
deployment.apps/nginx-depl rolled back
```

```
> oc rollout history deployment nginx-depl  
deployment.apps/nginx-depl  
REVISION  CHANGE-CAUSE
```

```
1      Initial deployment  
3      Change to port 82  
4      Change to Nginx 1.21.5
```

# Services

- How do pods communicate with each other?
  - Pods are assigned IP addresses, BUT these are NOT static and are not accessible outside the cluster.
  - Most rollouts and rollback cause pods to be recreated, resulting in new IP address assignments.
- Solution: Services.



# Services

- Services abstract a set of pods (typically a replica set).
  - They're assigned a static IP address.
  - Unlike pods, a service is not assigned to a node.
  - Services load balance the workload between its pods.
  - Service Types:
    1. ClusterIP (Accessible inside the cluster only)
    2. NodePort
    3. LoadBalancer
    4. Ingress

# ClusterIP Service creation

```
> oc apply -f 05-nginx-service.yaml
```

```
service/nginx-service created
```

```
> oc get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nginx-service	ClusterIP	172.30.250.186	<none>	8081/TCP	40s

```
> oc get endpoints nginx-service
```

NAME	ENDPOINTS	AGE
nginx-service	10.130.0.75:80,10.131.1.91:80	2m10s

- These are the IPs (and ports) of its associated pods



# ClusterIP Service access

- Service IPs are private (only accessible inside the cluster).

```
➤ curl 172.30.250.186:8081
```

```
^C
```

```
➤ oc exec -it nginx-depl-7569b665c5-2mfzg -- sh
```

```
# curl 172.30.250.186:8081
```

```
<!DOCTYPE html><html>...this works ... </html>
```

```
# curl http://nginx-service:8081
```

```
<!DOCTYPE html><html>... </html>
```

- K8s support domain name resolution and service discovery.
- Port-forwarding with services is also allowed.

# Services – Load balancing

- A service load balances the traffic to the pods

➤ `oc exec -it nginx-depl-756965c5-2mfzg -- bin/bash`

```
# i=1
```

```
while [ $i -le 20 ]
```

```
do
```

```
  curl nginx-service:8081;
```

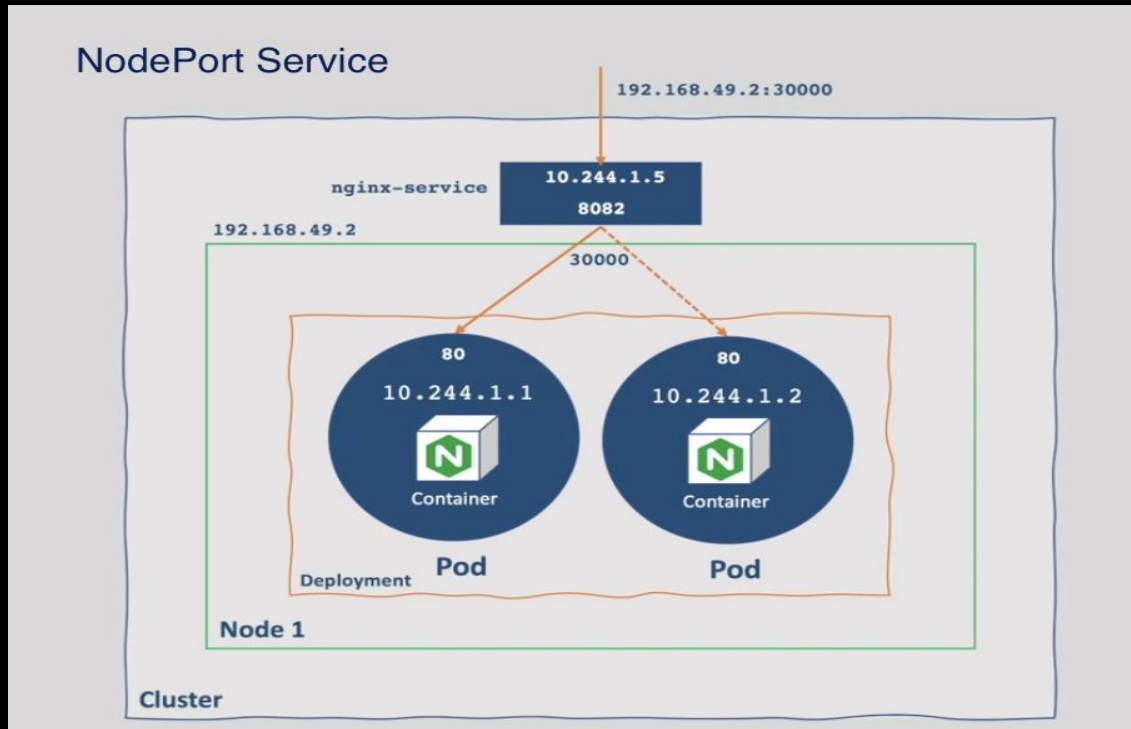
```
  i=$(( $i + 1 ))
```

```
done
```

- Monitor the pods by streaming their logs:

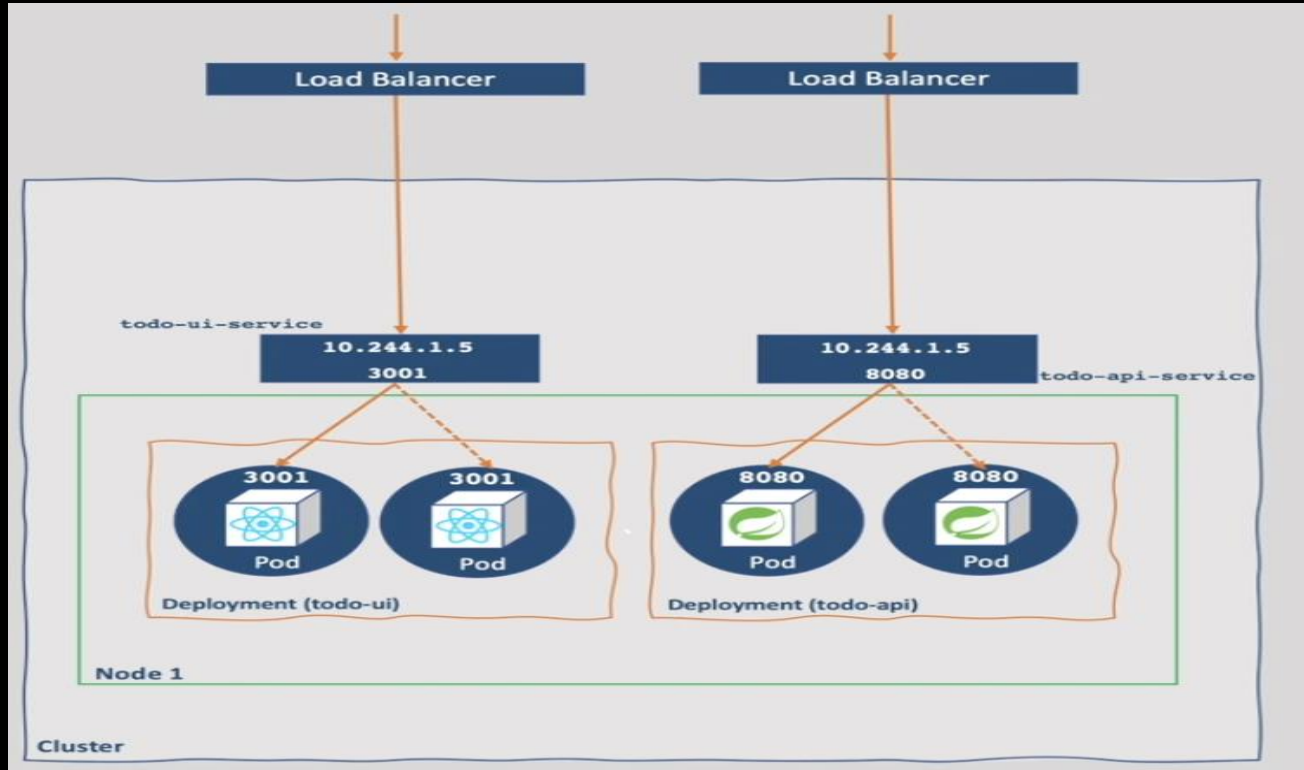
➤ `oc logs nginx-depl-7569b665c5-2mfzg -f`

# NodePort Service



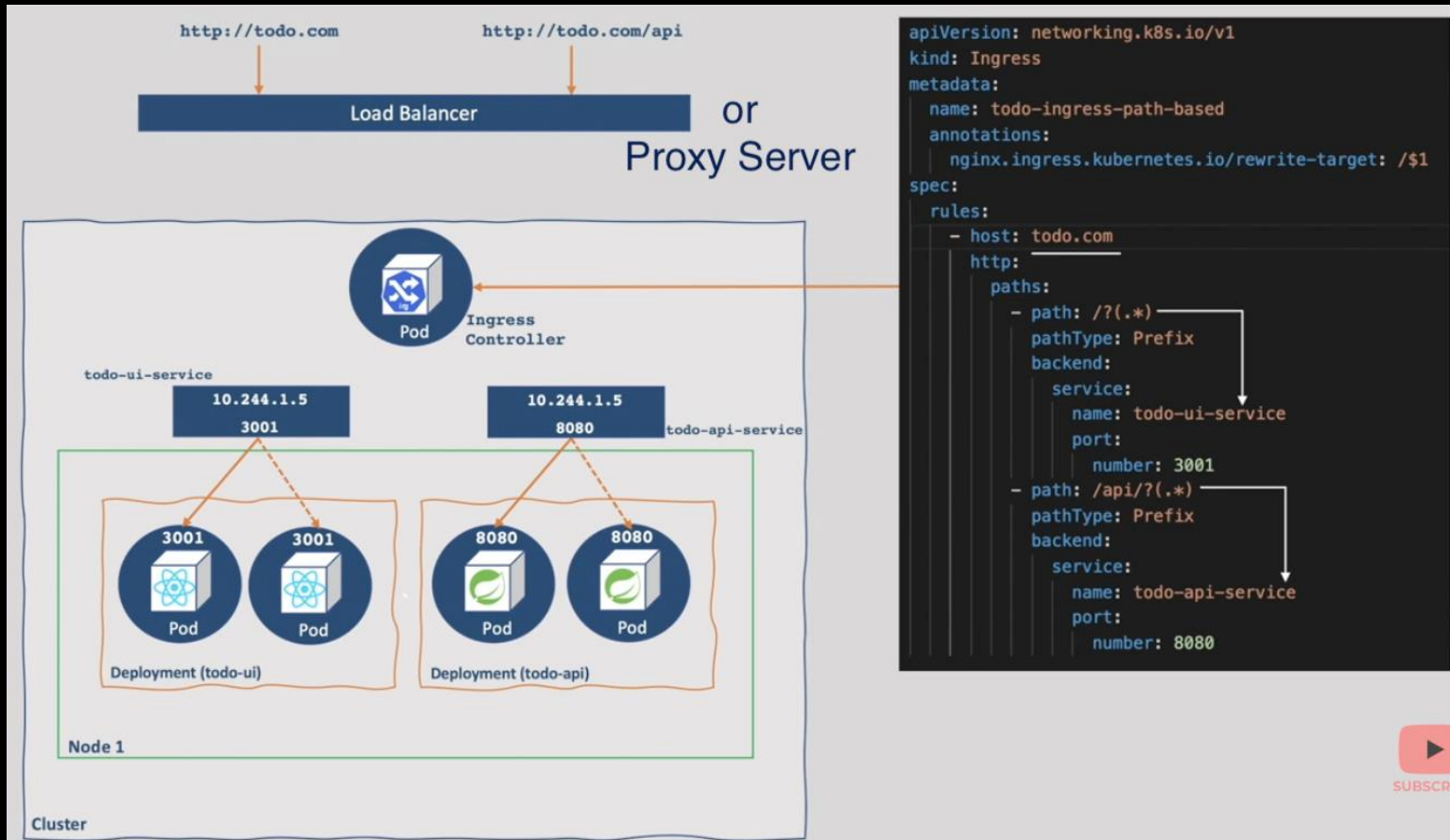
- Disadv: Insecure - Opens a worker's port – 1 port per service. Unstable – Node IPs change on rebooting.

# LoadBalancer Service



- Disadv: Must configure an external Load Balancer, e.g. AWS Load Balancer. Costly, as each NP service needs its own LB.

# Ingress Service



# Ingress Service

› oc get ing

NAME	CLASS	HOSTS	ADDRESS	PORTS	AGE
nginx-ingress	<none>	doc-nginx.apps.ocp-setu.waltoninstitute.ie	router-default.apps.ocp-setu.waltoninstitute.ie	80	33m

› oc describe ing/nginx-ingress

Name: nginx-ingress

Labels: <none>

Address: router-default.apps.ocp-setu.waltoninstitute.ie

Rules:

Host	Path	Backends
------	------	----------

----

-----

doc-nginx.apps.ocp-setu.waltoninstitute.ie		
--	--	--

/	nginx-service:8081 (10.129.3.3:80,10.130.1.247:80)	
---	--	--

# Volumes

- Problem 1: When a pod crashes, its data is lost.
  - The same applies to a node crash.
- Problem 2: The pods in a set cannot share data.
- Use volumes to overcome the above problems.
- A volume is a directory containing data that is accessible.
- Three types of volumes:
  1. emptydir – data outside the container, but private to a pod
  2. hostPath – data outside the pod. but private to a node
  3. Persistent Volumes – data outside the cluster and accessible by a pod set

# Persistent Volumes

- Three components (Kubernetes resources):
  1. Persistent Volumes (PV)
  2. Persistent Volumes claims (PVC)
  3. Storage Classes
  - All are represented as cluster resources.
- A *Persistent Volume* (PV) is a piece of storage that has been provisioned by an administrator or dynamically provisioned.
  - Created with a YAML manifest.
- Storage Classes – types of storage devices, e.g. AWS EBS, NFS server, etc
  - A PV specifies the storage class, size, access mode, etc.



# Persistent Volumes

- A persistent volume claim (PVC) is a declaration by a pod that it will require storage.
  - A PVC is bound to a PV by matching class, size, etc
  - The PV is mounted to a container path.
- Pod(s) → PVC → PV
- PV access modes
  - ReadWriteMany – volume can be mounted by many nodes
  - ReadWriteOnce – volume can be mounted by one node only. All the pods must be running on one worker node.
  - ReadMany / ReadOnce
  - ReadWriteOncePod