KUBERNETES WORKSHOP

- Introduction to Kubernetes
- Pods
- Deployments
- Statefulset
- Networking and Services
- RBAC
- Config map & secrets
- Setup -> Local + Kubeadm

INTRO TO K8'S →

- What is Kubernetes?
- Why Kubernetes?
- Kubernetes Architecture
- YAML file
- Kubernetes components
- Imperative vs declarative
- Namespaces
- Labels and Selectors

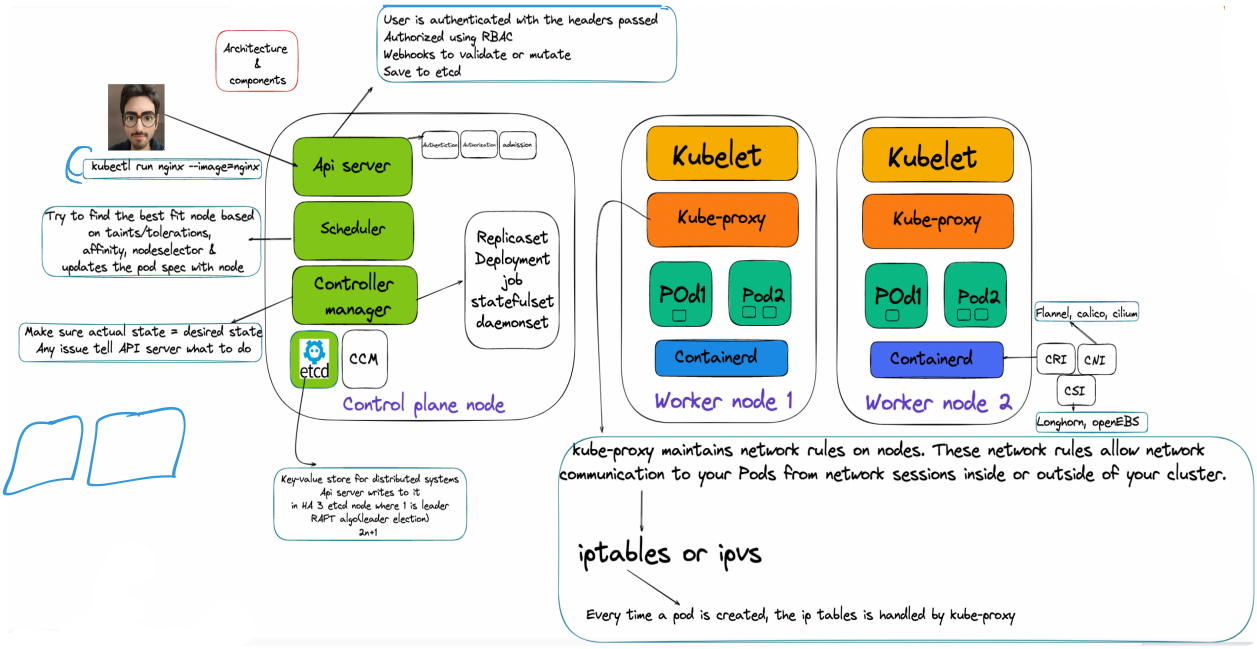
What is Kubernetes ?

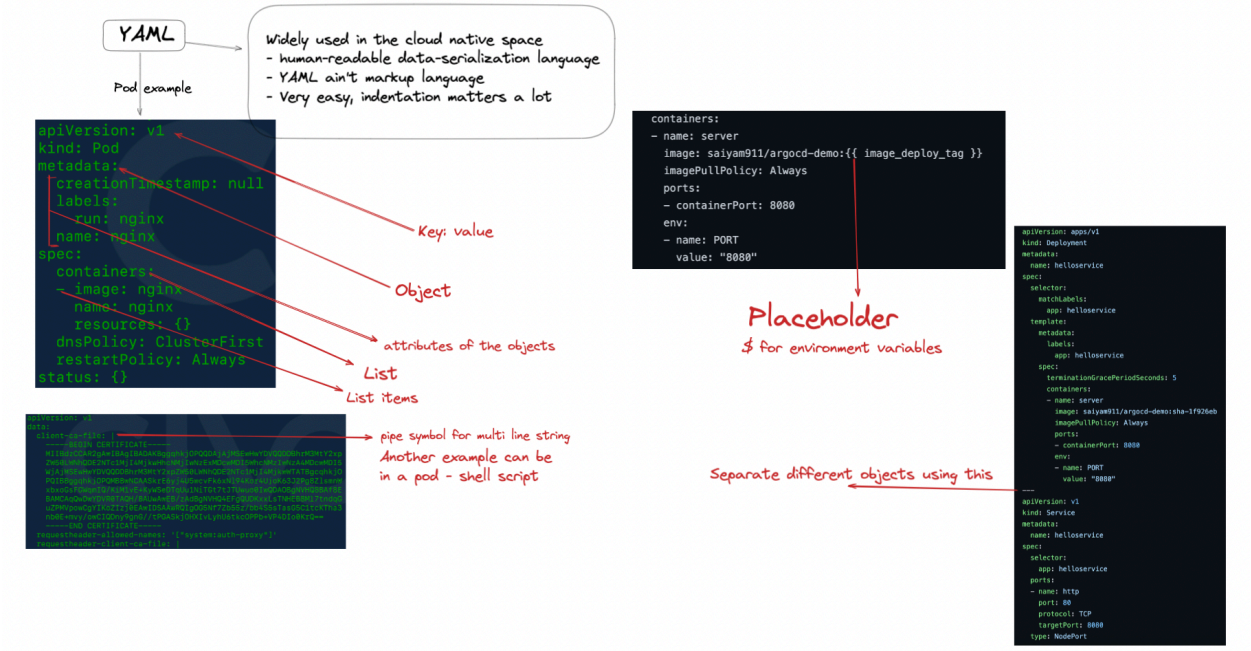
Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.

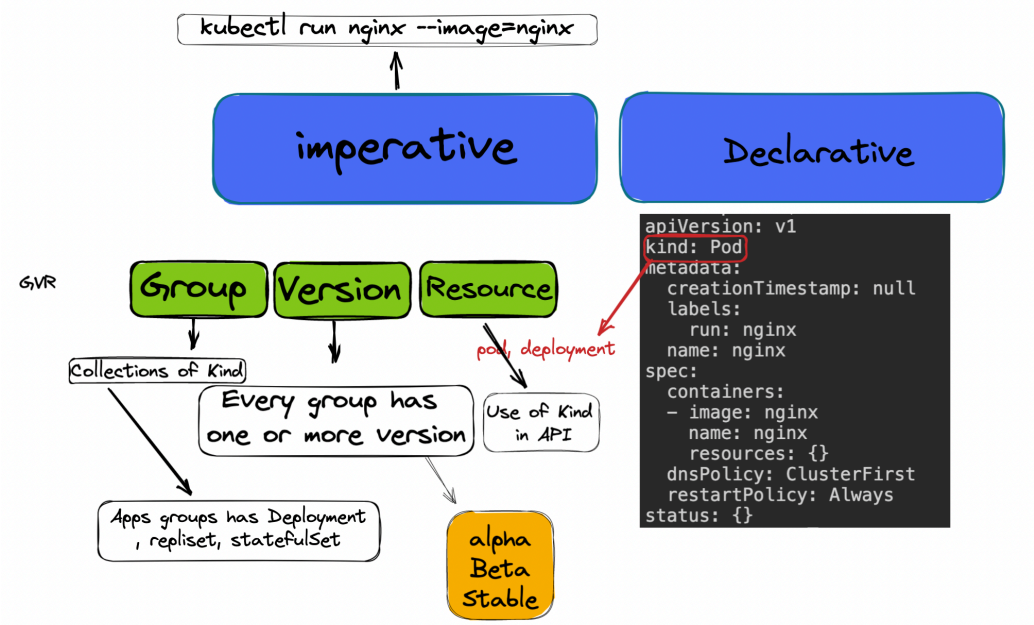
- CNCF Graduated
- Born out of Borg and Omega
- Launched 2014
- Designed for Scale
- Run anywhere

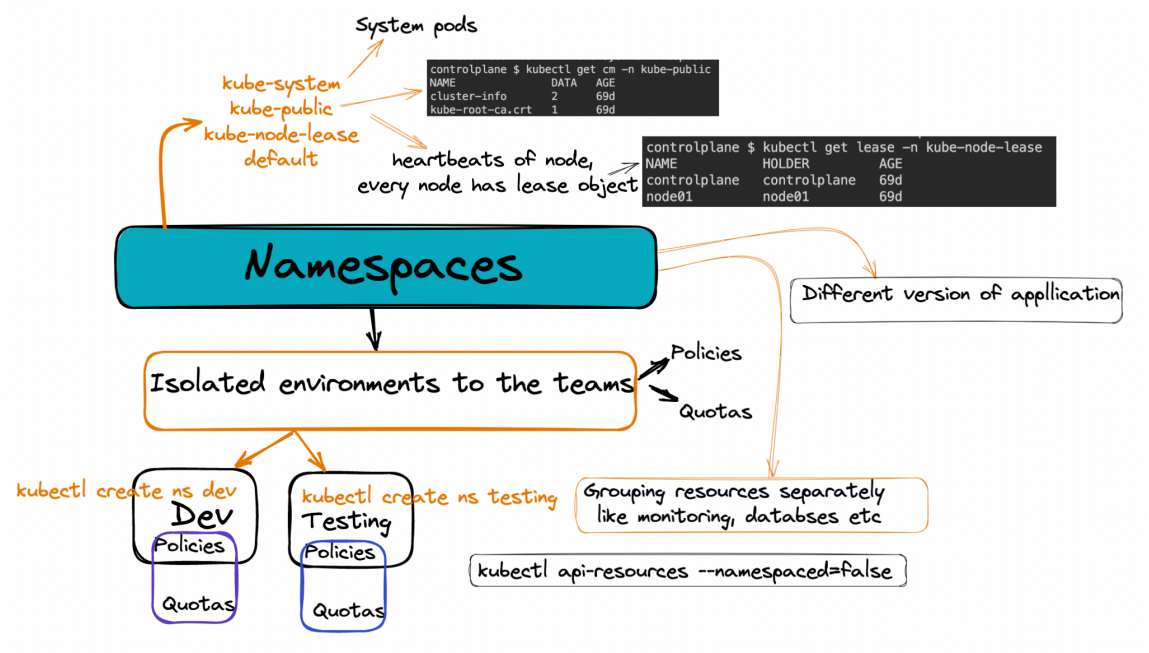
Why Kubernetes ?

- Few containers are fine but what about scale ?
- How do monitor them ?
- What about running pods on multiple nodes?
- What about flexibility ?
- Autoscaling?
- Scheduling
- Self healing capabilities.


Architecture & components
User is authenticated with the headers passed
Authorized using RBAC
Webhooks to validate or mutate
Save to etcd
kubectl run nginx --image=nginx
Api server
Authentication
Authorization
admission
Try to find the best fit node based on taints/tolerations, affinity, nodeselector & updates the pod spec with node
Scheduler
Controller manager
Replicaset
Deployment
job
statefulset
daemonset
Make sure actual state = desired state
Any issue tell API server what to do
etcd
CCM
Control plane node
Kubelet
Kube-proxy
POd1
Pod2
Containerd
Worker node 1
Kubelet
Kube-proxy
POd1
Pod2
Containerd
Worker node 2
Flannel, calico, cilium
CRI
CNI
CSI
Longhorn, openEBS
Key-value store for distributed systems
Api server writes to it
in HA 3 etcd node where 1 is leader
RAFT algo(leader election)
2n+1
kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.
iptables or ipvs
Every time a pod is created, the ip tables is handled by kube-proxy

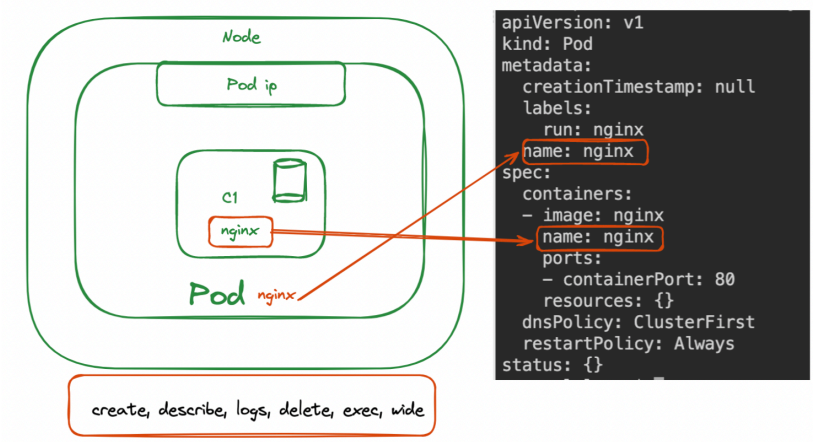

YAML
Widely used in the cloud native space
- human-readable data-serialization language
- YAML ain't markup language
- Very easy, indentation matters a lot
Pod example
apiVersion: v1
kind: Pod
metadata:
creationTimestamp: null
labels:
run: nginx
name: nginx
spec:
containers:
- image: nginx
name: nginx
resources: {}
dnsPolicy: ClusterFirst
restartPolicy: Always
status: {}
Key: value
Object
attributes of the objects
List
List items
pipe symbol for multi line string
Another example can be in a pod - shell script
containers:
- name: server
image: saiyam911/argocd-demo:{{ image_deploy_tag }}
imagePullPolicy: Always
ports:
- containerPort: 8080
env:
- name: PORT
value: "8080"
Placeholder
$ for environment variables
Separate different objects using this
apiVersion: apps/v1
kind: Deployment
metadata:
name: helloservice
spec:
selector:
matchLabels:
app: helloservice
template:
metadata:
labels:
app: helloservice
spec:
terminationGracePeriodSeconds: 5
containers:
- name: server
image: saiyam911/argocd-demo:sha-1f926eb
imagePullPolicy: Always
ports:
- containerPort: 8080
env:
- name: PORT
value: "8080"
---
apiVersion: v1
kind: Service
metadata:
name: helloservice
spec:
selector:
app: helloservice
ports:
- name: http
port: 80
protocol: TCP
targetPort: 8080
type: NodePort

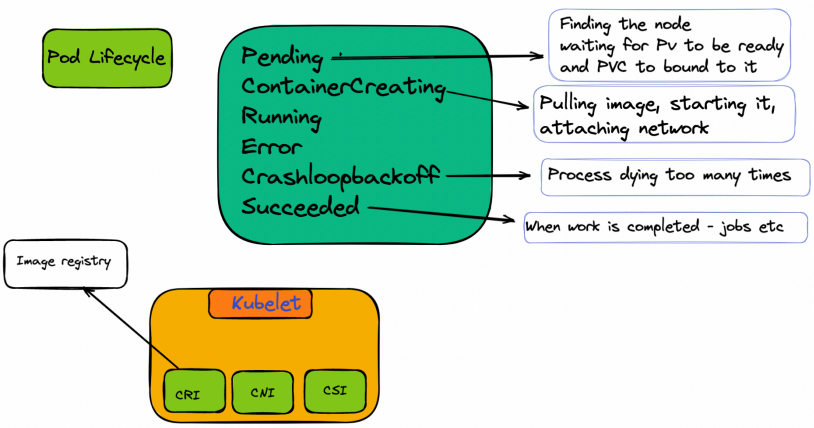

kubectl run nginx --image=nginx
imperative
Declarative
apiVersion: v1
kind: Pod
metadata:
creationTimestamp: null
labels:
run: nginx
name: nginx
spec:
containers:
- image: nginx
name: nginx
resources: {}
dnsPolicy: ClusterFirst
restartPolicy: Always
status: {}
GVR
Group
Version
Resource
pod, deployment
Collections of Kind
Every group has one or more version
Use of Kind in API
Apps groups has Deployment , repliset, statefulSet
alpha
Beta
Stable

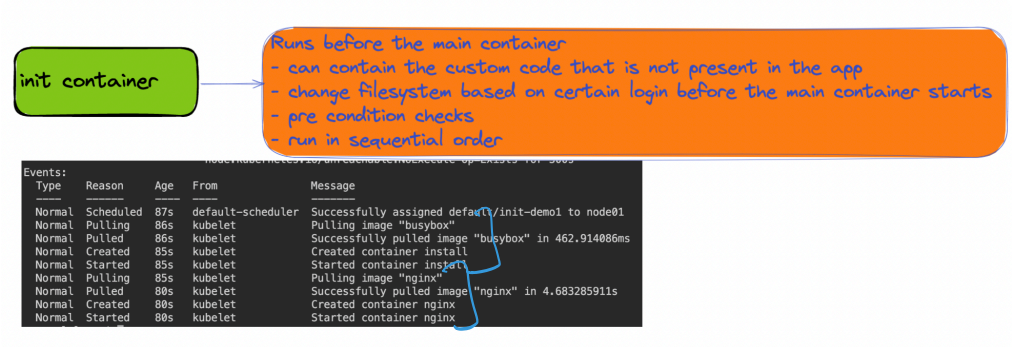

System pods
kube-system
kube-public
kube-node-lease
default
controlplane $ kubectl get cm -n kube-public
NAME DATA AGE
cluster-info 2 69d
kube-root-ca.crt 1 69d
heartbeats of node, every node has lease object
controlplane $ kubectl get lease -n kube-node-lease
NAME HOLDER AGE
controlplane controlplane 69d
node01 node01 69d
Namespaces
Different version of appllication
Isolated environments to the teams
Policies
Quotas
kubectl create ns dev
Dev
Policies
Quotas
kubectl create ns testing
Testing
Policies
Quotas
Grouping resources separately like monitoring, databses etc
kubectl api-resources --namespaced=false

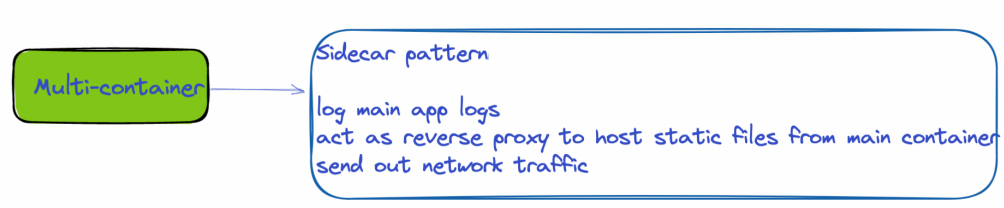

Labels and selectors
key-vlaue pairs
defined in metadata section
add meaning to the kubernetes object
label selectors can be
eg nodeSelector
equity based label
set based labels
in
notin
exists
selecting on the based of labels
metadata:
annotations:
cni.projectcalico.org/containerI
cni.projectcalico.org/podIP: 192
cni.projectcalico.org/podIPs: 19
creationTimestamp: "2022-07-17T07:
labels:
live: demo
run: nginx
kind: Deployment
metadata:
creationTimestamp: null
labels:
app: nginx
name: nginx
spec:
replicas: 1
selector:
matchLabels:
app: nginx
strategy: {}
template:
metadata:
creationTimestamp: null
labels:
app: nginx
spec:
containers:
- image: nginx
name: nginx
resources: {}
status: {}
controlplane $ kubectl get pods --show-labels
No resources found in default namespace.
controlplane $ kubectl run nginx --image=nginx
pod/nginx created
controlplane $ kubectl get pods --show-labels
NAME READY STATUS RESTARTS AGE LABELS
nginx 0/1 ContainerCreating 0 3s run=nginx
controlplane $ kubectl label pod nginx live=demo
pod/nginx labeled
controlplane $ kubectl get pods --show-labels
NAME READY STATUS RESTARTS AGE LABELS
nginx 1/1 Running 0 22s live=demo,run=nginx
controlplane $
selector:
matchLabels:
component: redis
matchExpressions:
- {key: tier, operator: In, values: [cache]}
- {key: environment, operator: NotIn, values: [dev]}

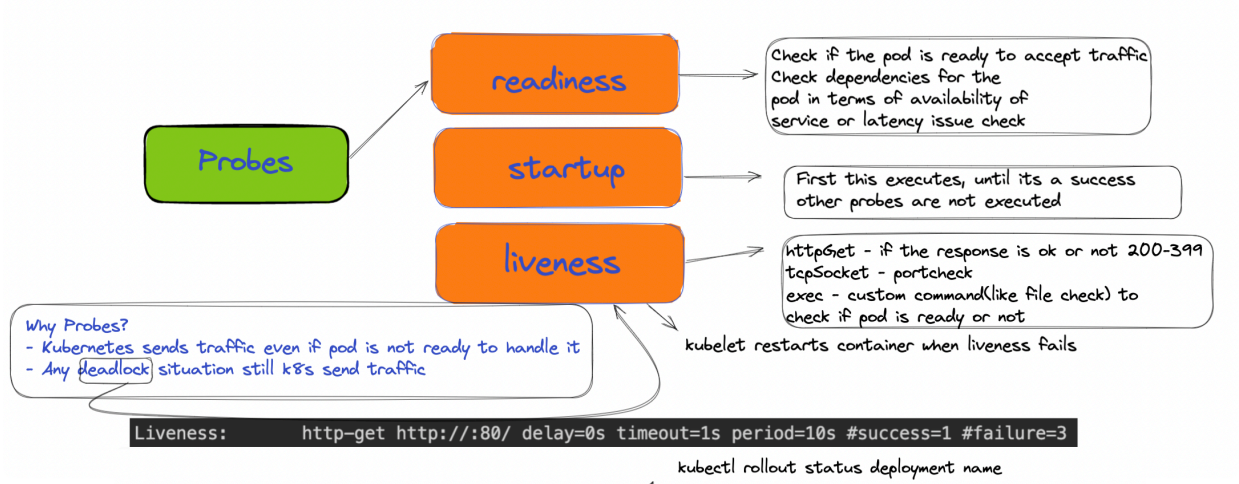
PODS

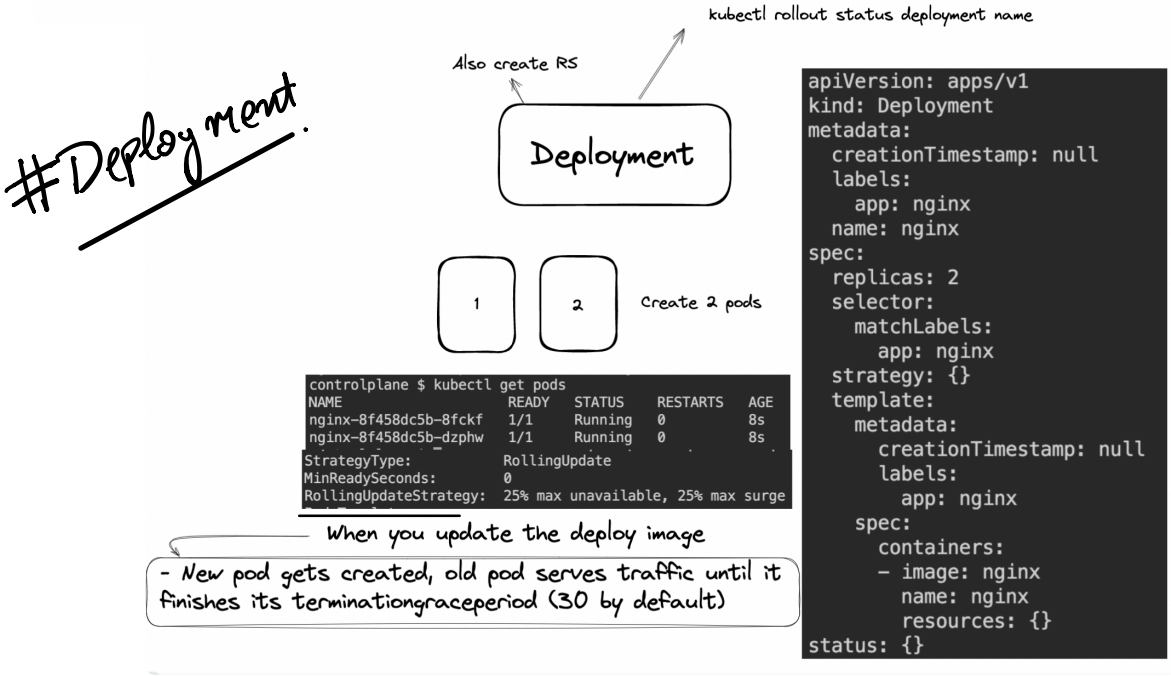

Node
Pod ip
C1
nginx
Pod
nginx
create, describe, logs, delete, exec, wide
apiVersion: v1
kind: Pod
metadata:
creationTimestamp: null
labels:
run: nginx
name: nginx
spec:
containers:
- image: nginx
name: nginx
ports:
- containerPort: 80
resources: {}
dnsPolicy: ClusterFirst
restartPolicy: Always
status: {}


Pod Lifecycle
Pending
ContainerCreating
Running
Error
Crashloopbackoff
Succeeded
Finding the node
waiting for Pv to be ready
and PVC to bound to it
Pulling image, starting it,
attaching network
Process dying too many times
When work is completed - jobs etc
Image registry
Kubelet
CRI
CNI
CSI

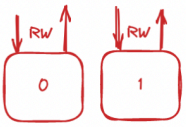

init container
Runs before the main container
- can contain the custom code that is not present in the app
- change filesystem based on certain login before the main container starts
- pre condition checks
- run in sequential order
Events:
Type Reason Age From Message
Normal Scheduled 87s default-scheduler Successfully assigned default/init-demo1 to node01
Normal Pulling 86s kubelet Pulling image "busybox"
Normal Pulled 86s kubelet Successfully pulled image "busybox" in 462.914086ms
Normal Created 85s kubelet Created container install
Normal Started 85s kubelet Started container install
Normal Pulling 85s kubelet Pulling image "nginx"
Normal Pulled 80s kubelet Successfully pulled image "nginx" in 4.683285911s
Normal Created 80s kubelet Created container nginx
Normal Started 80s kubelet Started container nginx


Multi-container
Sidecar pattern
log main app logs
act as reverse proxy to host static files from main container
send out network traffic


Probes
readiness
startup
liveness
Check if the pod is ready to accept traffic
Check dependencies for the
pod in terms of availability of
service or latency issue check
First this executes, until its a success
other probes are not executed
httpGet - if the response is ok or not 200-399
tcpSocket - portcheck
exec - custom command(like file check) to
check if pod is ready or not
Why Probes?
- Kubernetes sends traffic even if pod is not ready to handle it
- Any deadlock situation still k8s send traffic
kubelet restarts container when liveness fails
Liveness: http-get http://:80/ delay=0s timeout=1s period=10s #success=1 #failure=3


#Deployment.
kubectl rollout status deployment name
Also create RS
Deployment
1
2
Create 2 pods
controlplane $ kubectl get pods
NAME READY STATUS RESTARTS AGE
nginx-8f458dc5b-8fckf 1/1 Running 0 8s
nginx-8f458dc5b-dzphw 1/1 Running 0 8s
StrategyType: RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
When you update the deploy image
- New pod gets created, old pod serves traffic until it finishes its terminationgraceperiod (30 by default)
apiVersion: apps/v1
kind: Deployment
metadata:
creationTimestamp: null
labels:
app: nginx
name: nginx
spec:
replicas: 2
selector:
matchLabels:
app: nginx
strategy: {}
template:
metadata:
creationTimestamp: null
labels:
app: nginx
spec:
containers:
- image: nginx
name: nginx
resources: {}
status: {}

StatefulSet → Stateful Applications → Databases



Sticky identity → Creates pod with predictable names even if one dies it will spin up with same name

Replication you need to take care

```
controlplane $ kubectl get pods
NAME                     READY   STATUS              RESTARTS   AGE
nginx-8f458dc5b-6brdg    1/1     Running             0          2m39s
nginx-8f458dc5b-zwnnn    1/1     Running             0          2m39s
web-0                    1/1     Running             0          21s
web-1                    0/1     ContainerCreating   0          8s
```

→ Headless service

You can a pod or your app to directly use DNS names for writing and then have a replication mechanism in place.

Deployment → creates pod with random hash and each time it dies and recreates it will be a random hash → Any service

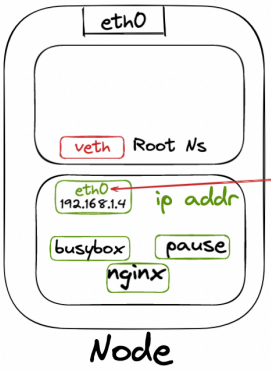
You cannot write to any replica in case of stateful apps as this will lead to data inconsistency

#NETWORKING

What happens when you run the pod?

```
controlplane $ kubectl get nodes
NAME           STATUS   ROLES           AGE   VERSION
controlplane   Ready    control-plane   68d   v1.24.0
node01         Ready    <none>          68d   v1.24.0
controlplane $
controlplane $ cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: shared-namespace
spec:
  containers:
    - name: p1
      image: busybox
      command: ['/bin/sh', '-c', 'sleep 10000']
    - name: p2
      image: nginx
controlplane $ kubectl apply -f pod.yaml
pod/shared-namespace created
```

After pod creation CNI attach Ip address and attach it to network

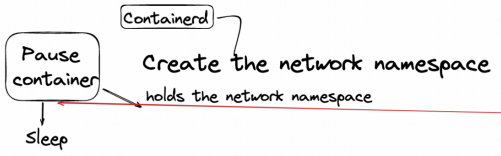



```
controlplane $ kubectl exec -it shared-namespace -- sh
Defaulted container "p1" out of: p1, p2
/ # ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: eth0@if9: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1460 qdisc noqueue
    link/ether be:26:a7:95:24:17 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.4/32 brd 192.168.1.4 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::bc26:a7ff:fe95:2417/64 scope link
       valid_lft forever preferred_lft forever
/ # route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         169.254.1.1     0.0.0.0         UG    0      0        0 eth0
169.254.1.1     *               255.255.255.255 UH    0      0        0 eth0
```

ip netns list → List of network namespaces

```
node01 $ ip netns list
cni-e56b3f61-826e-66eb-d587-e917ee245752 (id: 2)
cni-60337942-7f48-e61d-c65f-984f76de5fe6 (id: 1)
cni-ae432c80-c285-9338-a047-d3ddbf4a9fbe (id: 0)
```

There are veth pairs created or depending on CNI maybe different

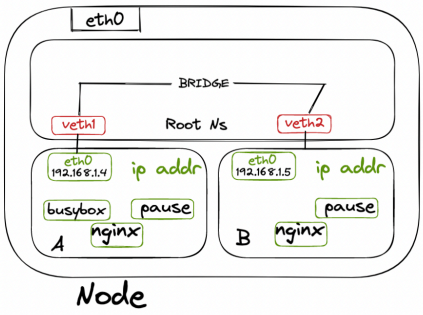


To see that network namespace is held by pause container

```
node01 $ lsns | grep nginx
4026532587 mnt         2 32059 root            nginx: master process nginx -g daemon off;
4026532588 pid         2 32059 root            nginx: master process nginx -g daemon off;
node01 $ lsns -p 32059
        NS TYPE   NPROCS   PID USER  COMMAND
4026531835 cgroup    157     1 root  /sbin/init
4026531837 user      157     1 root  /sbin/init
4026532518 net         3 31973 65535 /pause
4026532584 uts         3 31973 65535 /pause
4026532585 ipc         3 31973 65535 /pause
4026532587 mnt         2 32059 root  nginx: master process nginx -g daemon off;
4026532588 pid         2 32059 root  nginx: master process nginx -g daemon off;
```

ls -lt /var/run/netns → List of namespaces created

```
node01 $ ls -lt /var/run/netns
total 0
-r--r--r-- 1 root root 0 Jul 16 14:25 cni-3cf655f5-9131-e23d-0756-b7ab6556ef8f
-r--r--r-- 1 root root 0 May  8 19:46 cni-06382f28-d5db-63f3-00ee-654e40ee904b
-r--r--r-- 1 root root 0 May  8 19:46 cni-592cc23b-1d2b-6054-bf32-437fd23a04df
node01 $ ip netns exec cni-3cf655f5-9131-e23d-0756-b7ab6556ef8f ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default ql
en 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
3: eth0@if9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc noqueue state UP mode DEFAULT group
 default
    link/ether 9e:bd:c7:d4:e5:99 brd ff:ff:ff:ff:ff:ff link-netnsid 0
node01 $ ip link | grep -A1 ^9
9: calic440f455693@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc noqueue state UP mode DE
FAULT group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netns cni-3cf655f5-9131-e23d-0756-b7a
b6556ef8f
node01 $
```



Inter node pod communication

FLOW

A -> B -> Traffic(packet) goes to eth0

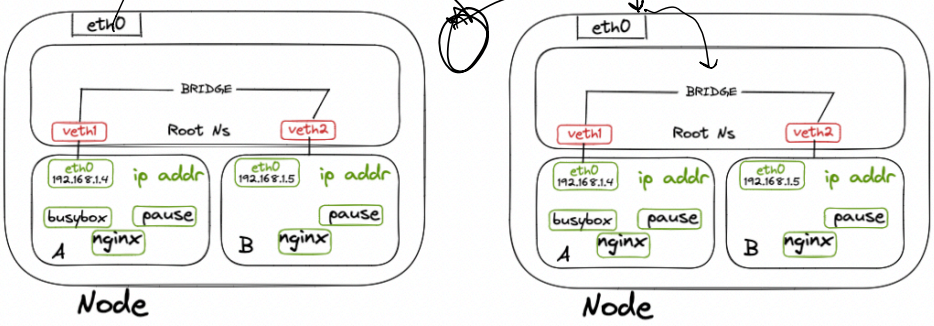
veth1 acts as tunnel and traffic goes to root ns

Bridge resolves the destination address using the ARP table

veth1 send traffic(packet to pod B)

Node to node communication

Cannot use ARP as destination is not on same node



FLOW

To check of the destination is not on the same network bitwise or Adding operation is performed

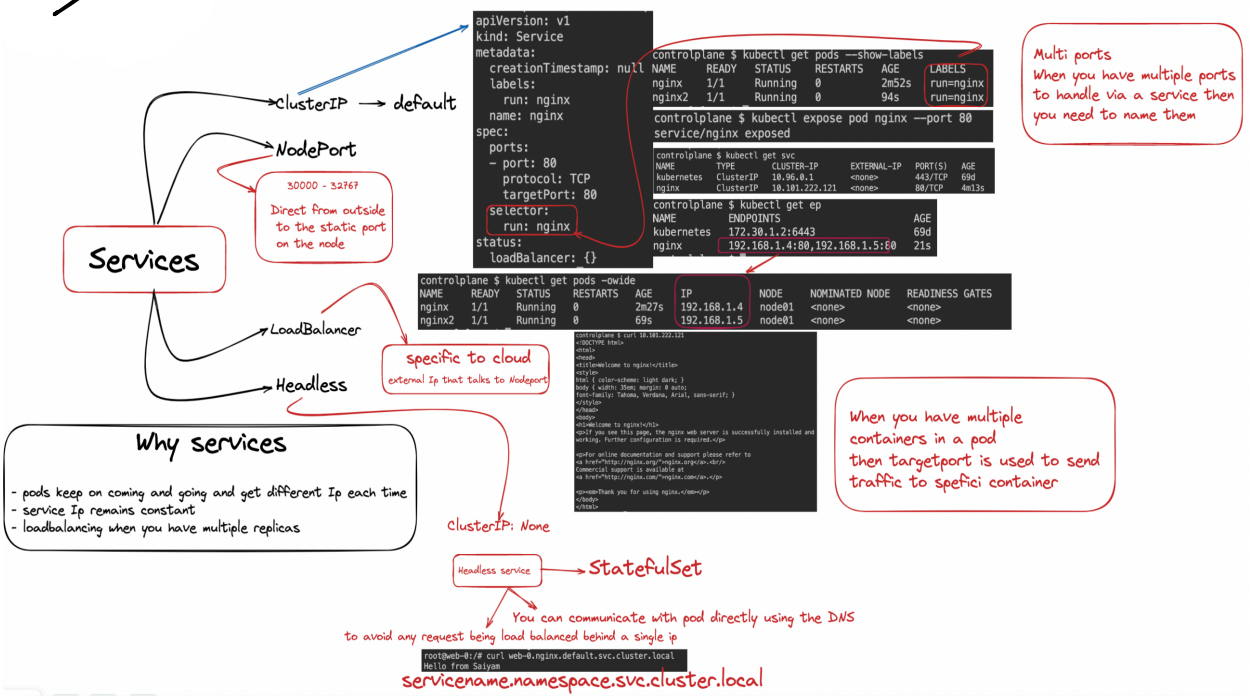
Find the default gateway form ARP

Send to default gateway

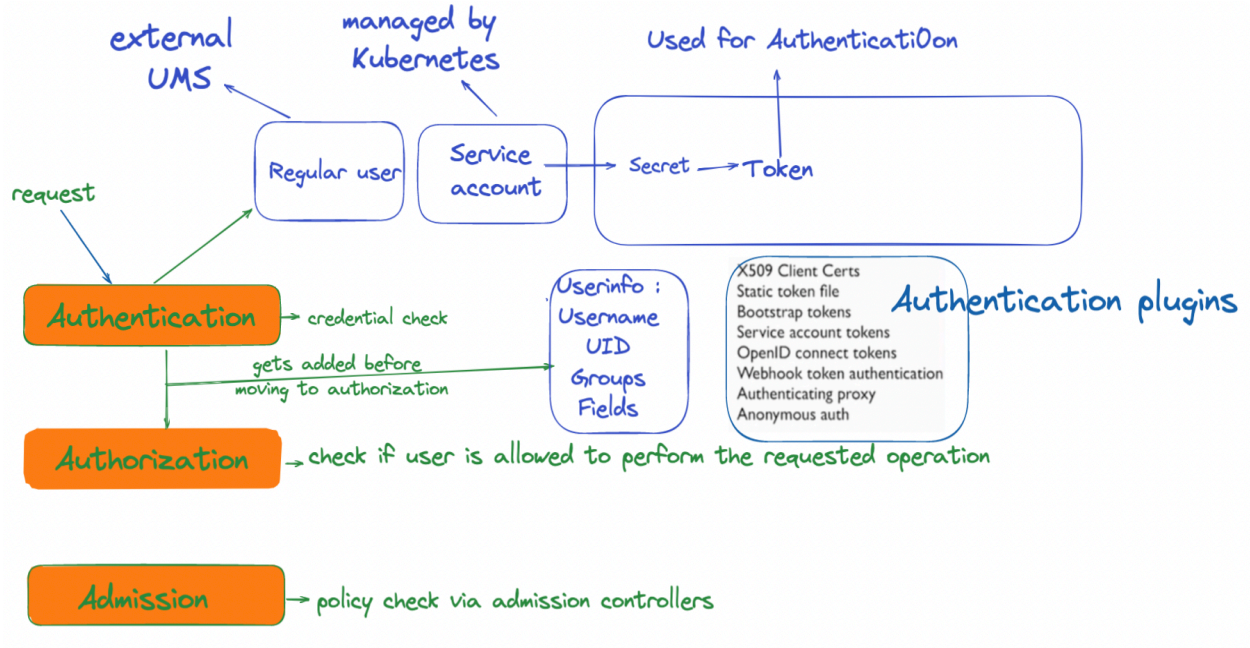
DEfault interface of other node

Services → iptables and netfilter

SERVICES



#AAA



Configmap + secrets