

Demystifying Kubernetes with 100 slides



kubernetes



docker

Who am I?

- Developer Relations Manager at Redis Labs
- Worked in Dell EMC, VMware, CGI
- Docker Bangalore Community Leader
- DockerLabs Incubator
- Founder of Docker Labs(4700+ Slack members)



<http://www.collabnix.com>

Let's start with an analogy..



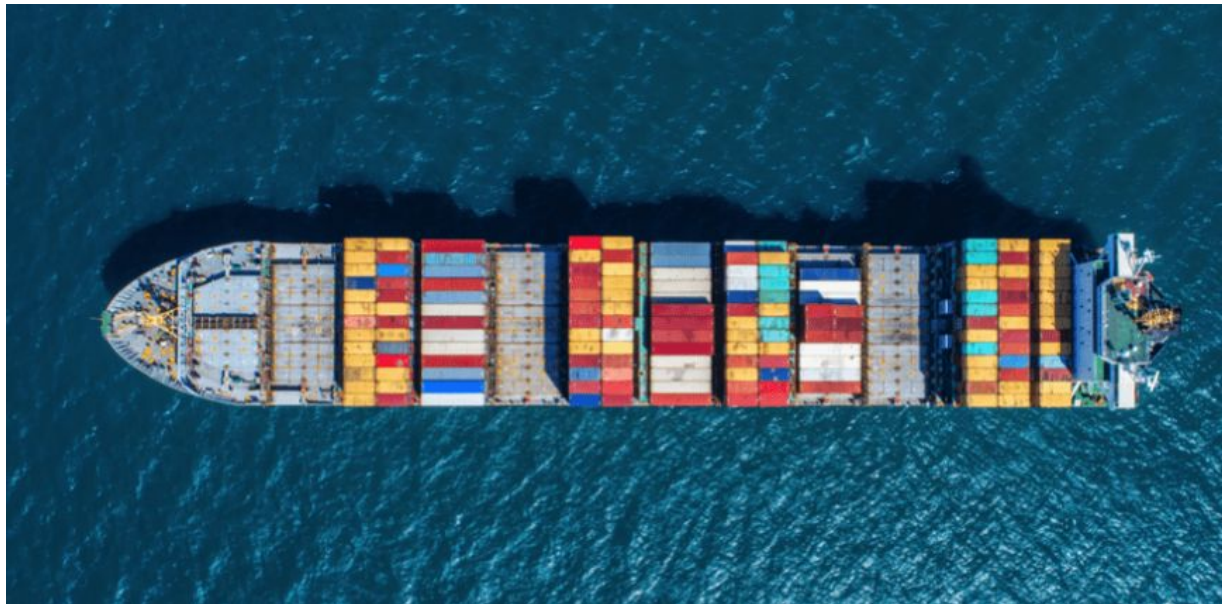
A Cargo Ship...

Carries containers across the sea

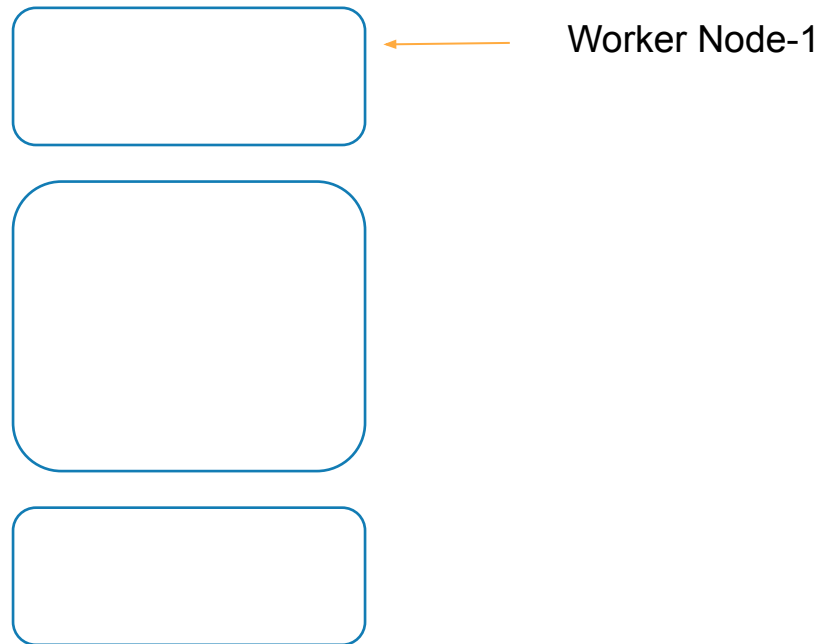


A Cargo Ship...

Host Application as Containers ~ Worker Nodes



Overview



Control Ships..

Managing & Monitoring of the cargo ships

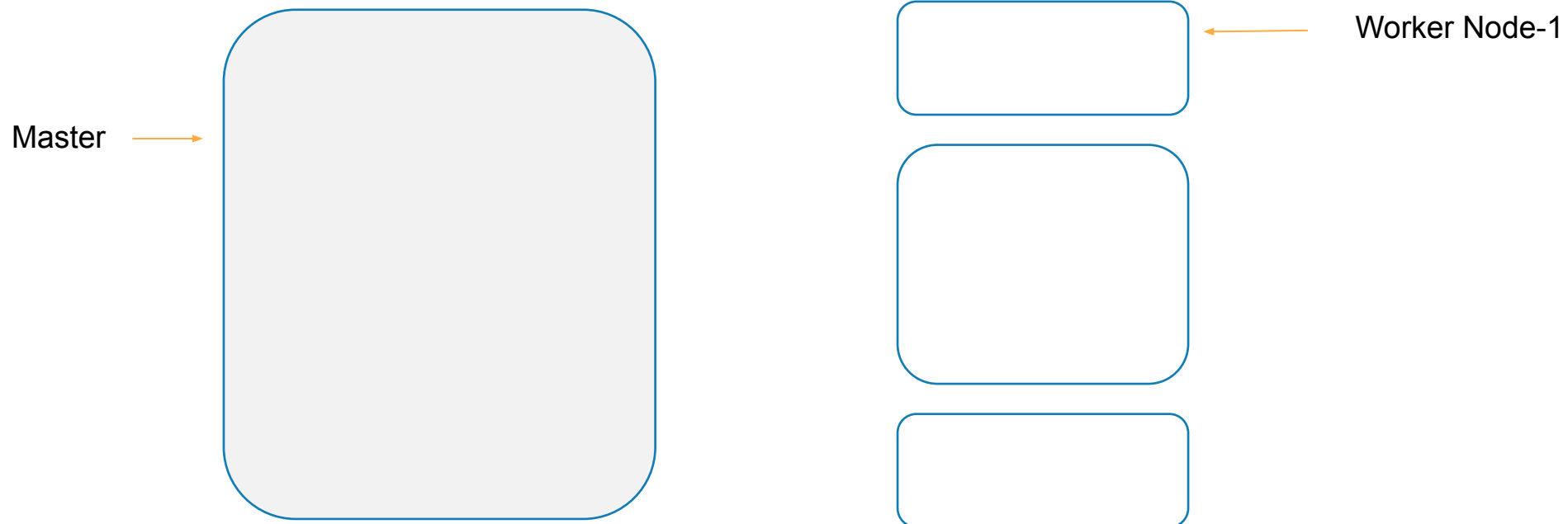


Control Ships..

Manage, Plan, Schedule, Monitor ~ Master



Overview



Let's talk about Master Components..



Ship Cranes

Identifies the placement of containers

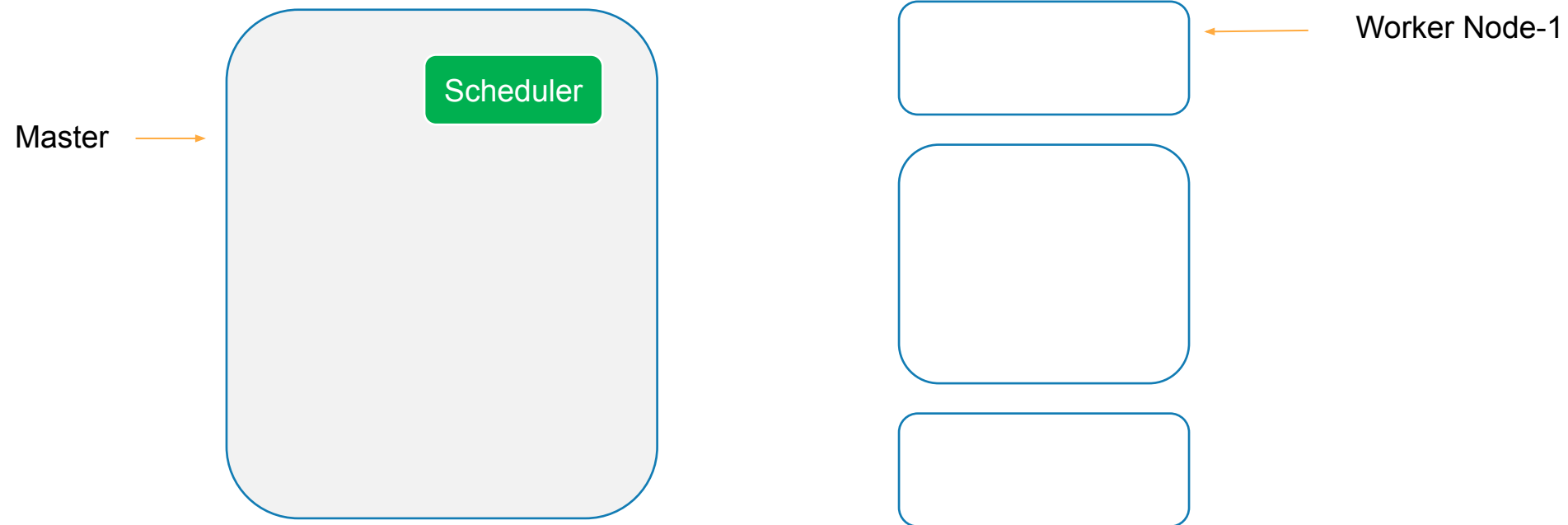


Ship Cranes

Identifies the right node to place a containers ~ Kube-Scheduler



Overview



Cargo Ship Profiles

HA database ~ Which containers on which ships? When was it loaded?

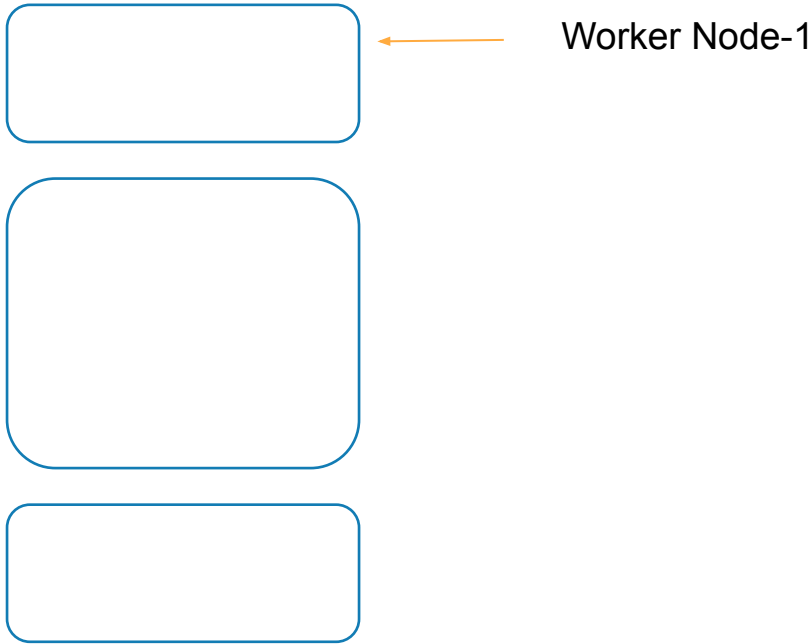
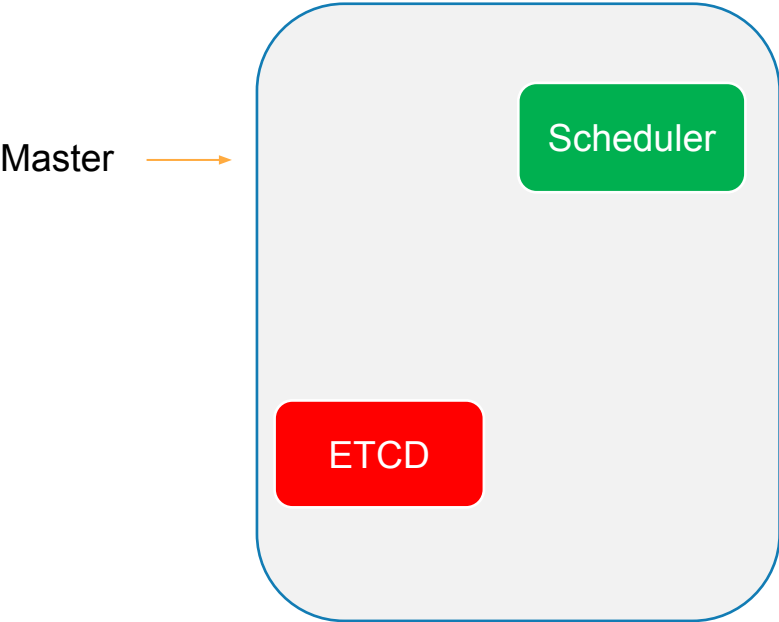


Cargo Ship Profiles

HA database ~ Which containers on which ships? When was it loaded? ~ **The ETCD Cluster**

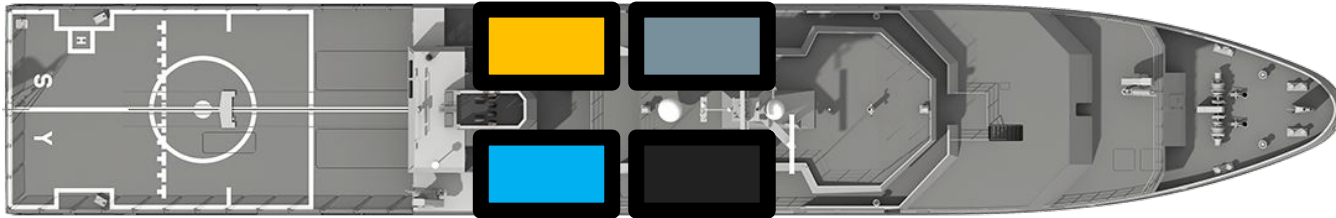


Overview



Offices in Dock

- Operation Team Office ~ Ship Handling, Control
- Cargo Team Office ~ verify if containers are damaged, ensure that new containers are rebuilt
- IT & Communication Office – Communication in between various ships

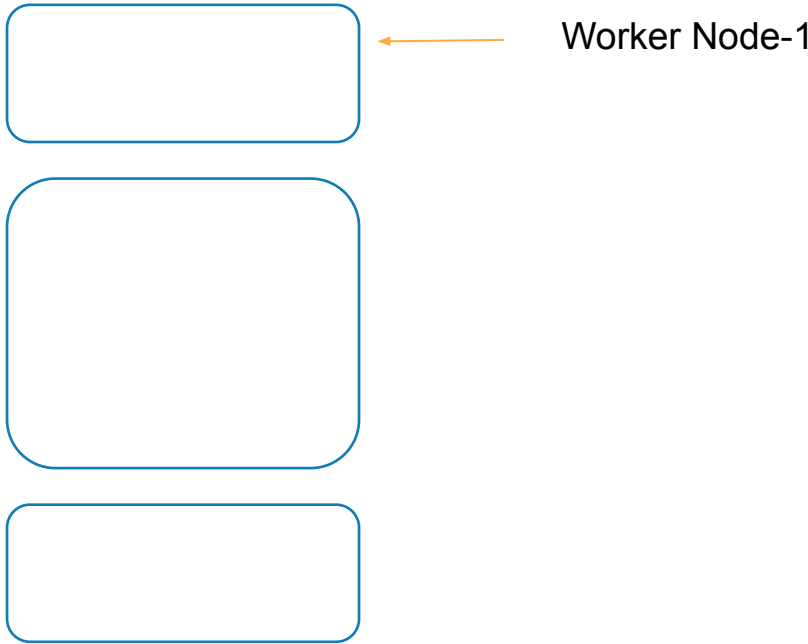
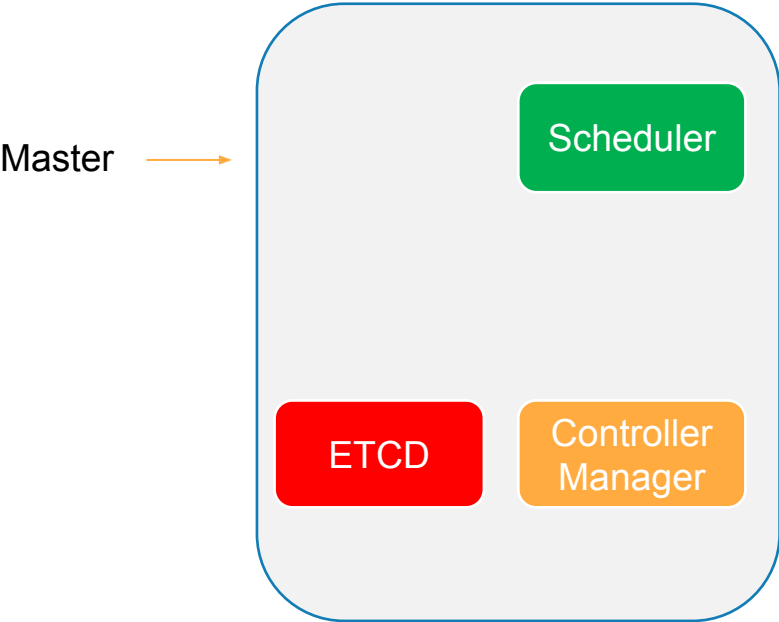


Controllers

- **Node Controllers** – Takes care of Nodes | Responsible for onboarding new nodes in a cluster | Availability of Nodes
- **Replicas Controller** – Ensures that desired number of containers are running at all times
- **Controller Manager** - Manages all these controllers in place



Overview



How does each of these services communicate with each other?

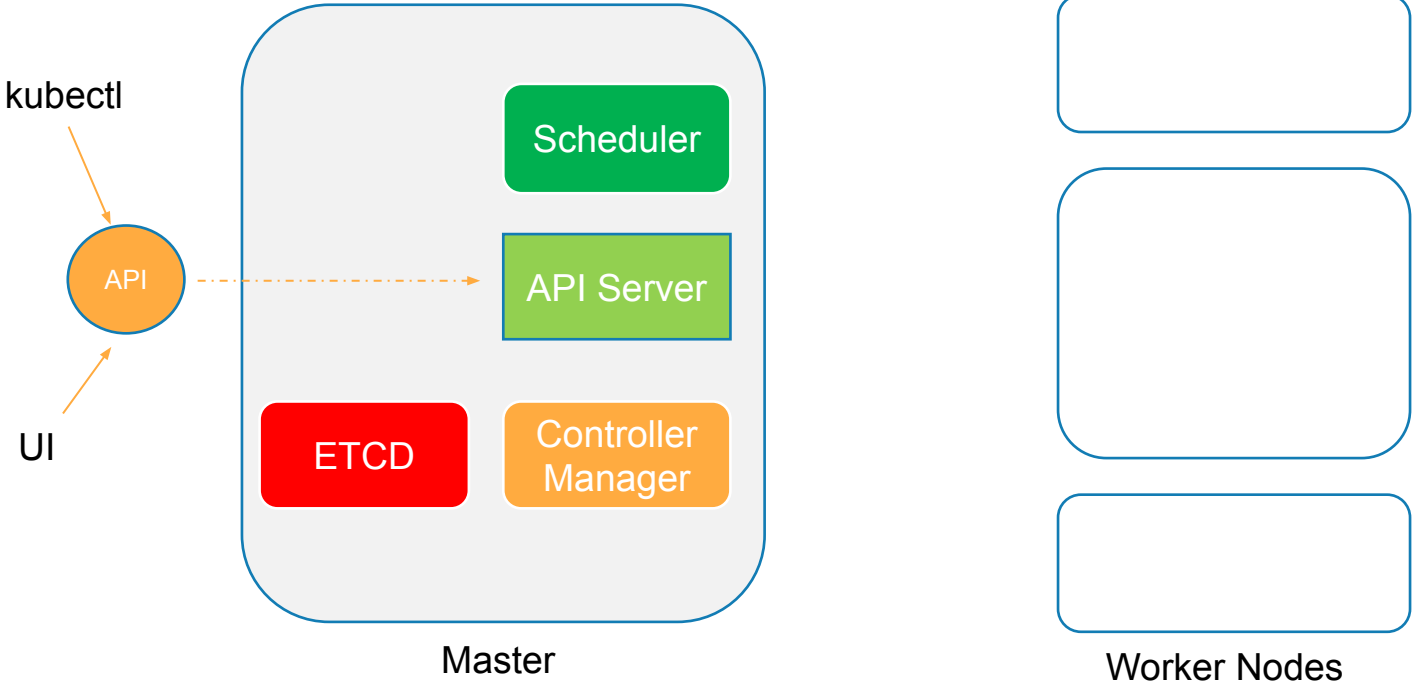


Kube API Server

- A primary management component of k8s
- Responsible for orchestrating all operations within a cluster
- Exposes K8s API, used by external users to perform management operation in the cluster and number of controller to monitor the state of the cluster



Overview



In nutshell...

\$kubectl get componentstatus

```
[node1 install]$ kubectl get nodes -o wide
```

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	CONTAINER-RUNTIME
node1	Ready	master	92s	v1.14.2	192.168.0.18	<none>	CentOS Linux 7 (Core)	4.4.0-141-generic	docker://18.9.6
node2	Ready	<none>	57s	v1.14.2	192.168.0.17	<none>	CentOS Linux 7 (Core)	4.4.0-141-generic	docker://18.9.6
node3	NotReady	<none>	39s	v1.14.2	192.168.0.16	<none>	CentOS Linux 7 (Core)	4.4.0-141-generic	docker://18.9.6
node4	NotReady	<none>	32s	v1.14.2	192.168.0.15	<none>	CentOS Linux 7 (Core)	4.4.0-141-generic	docker://18.9.6

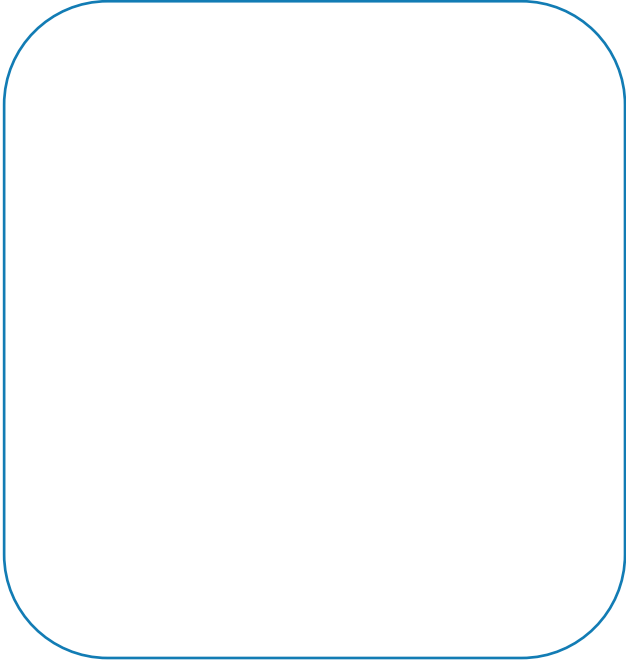
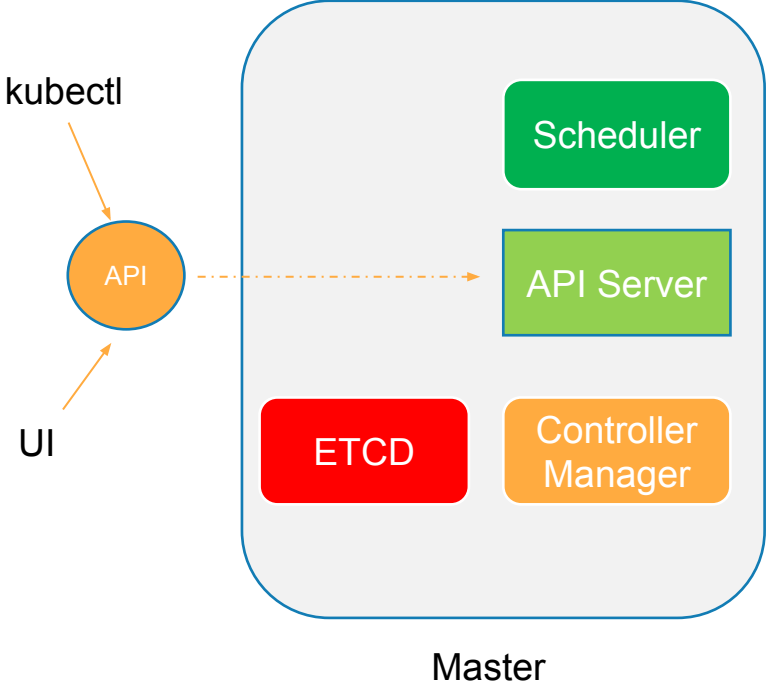
```
[node1 install]$ kubectl get componentstatus
```

NAME	STATUS	MESSAGE	ERROR
scheduler	Healthy	ok	
controller-manager	Healthy	ok	
etcd-0	Healthy	{"health": "true"}	

Let's talk about Worker Components..



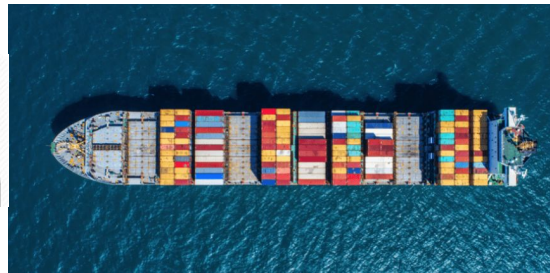
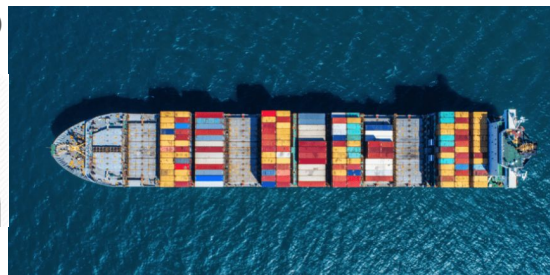
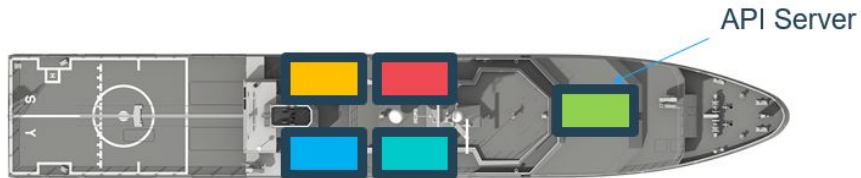
Overview



A Worker Node

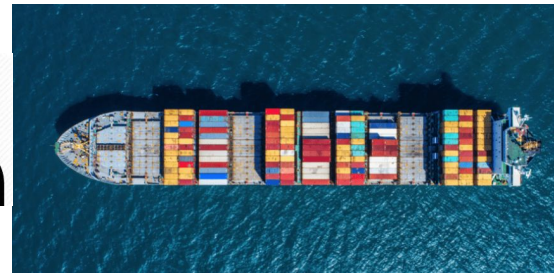
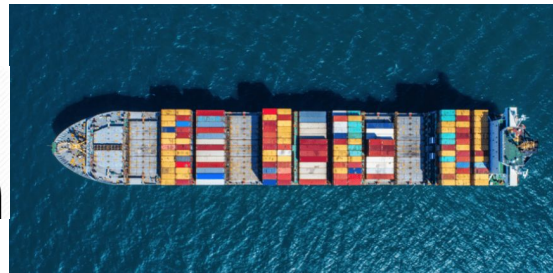
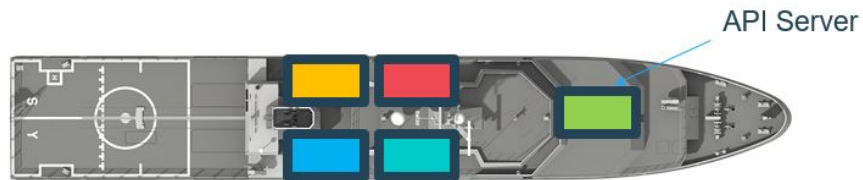
Captain of the Ship

- Manages all sort of activity on the ship
- Let master ship knows they are interested to join
- Sending reports back to master about the status of the ship
- Sending reports about the status of the containers

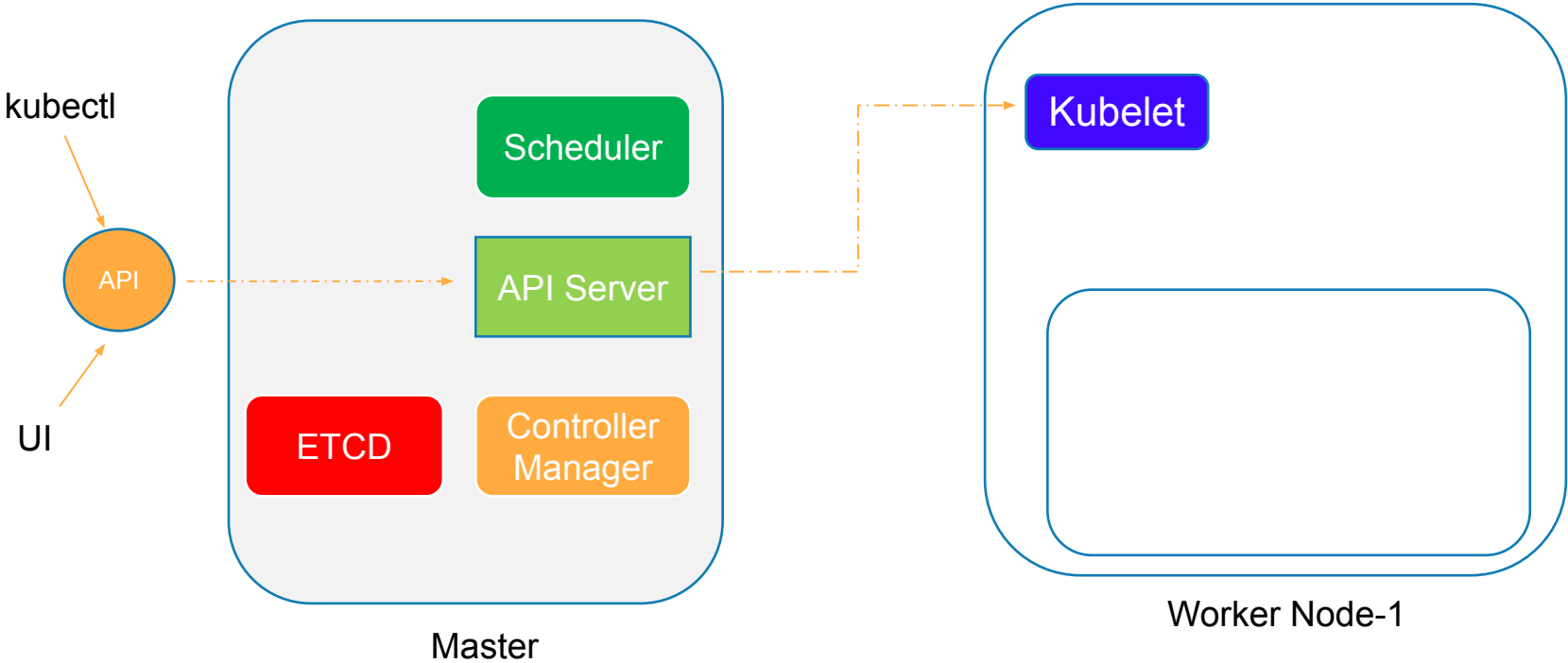


Captain of the Ship ~ Kubelet

Agent which runs on each nodes of the container

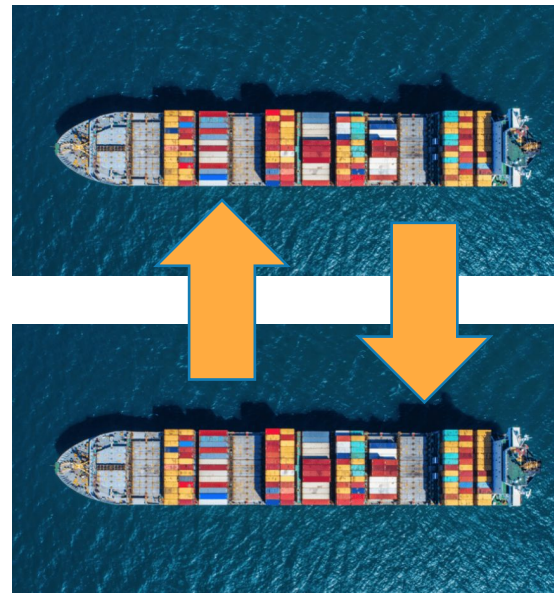


Overview



Communication between Cargo Ships

How does two cargo ships communicate with each other?

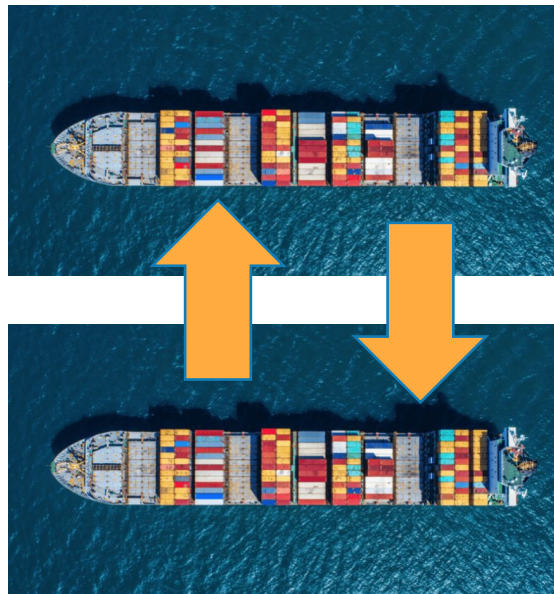


Kube-proxy Service

How will web server running on one worker node reach out to DB server on another worker node?

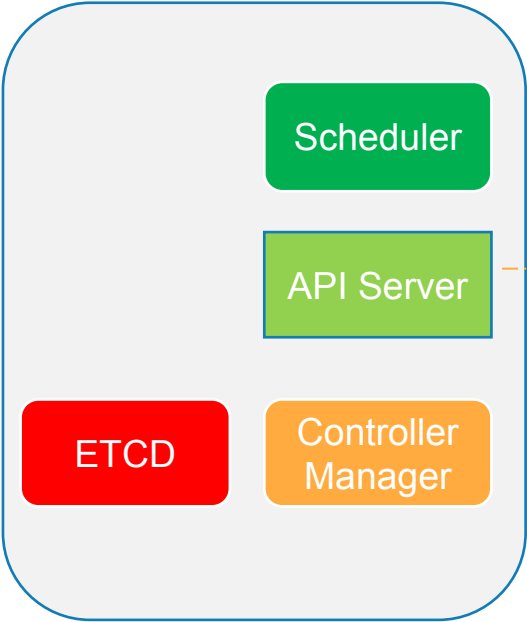
Communication between worker nodes

Kube-proxy

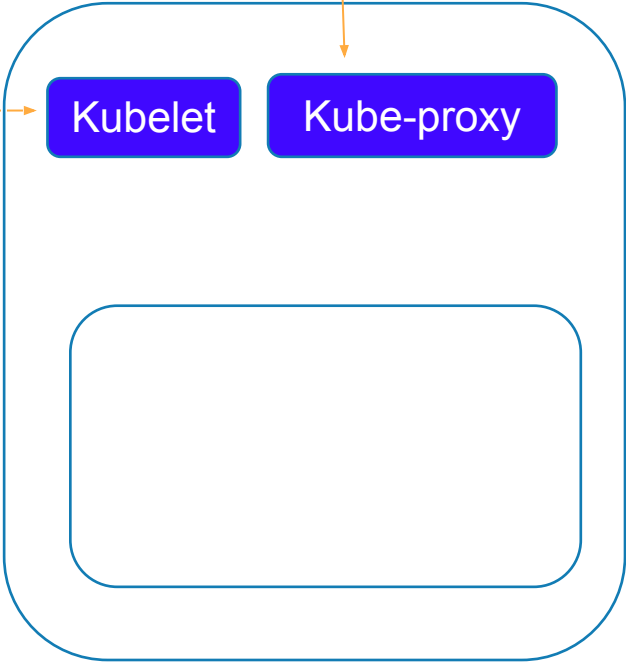


Overview

Master →



Worker Node-1 →

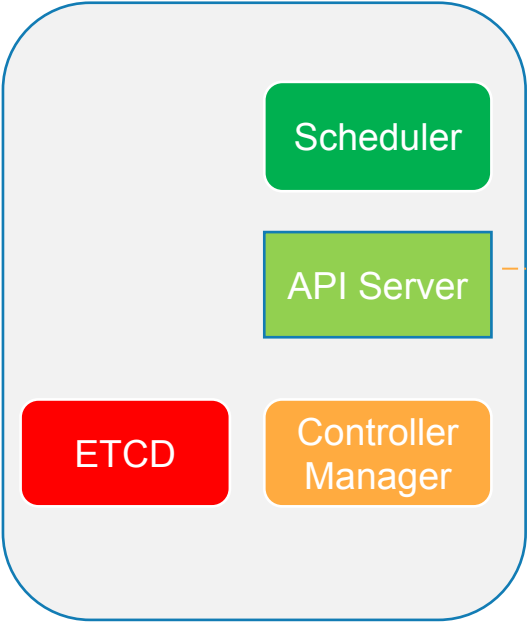


Let's talk about Pods..

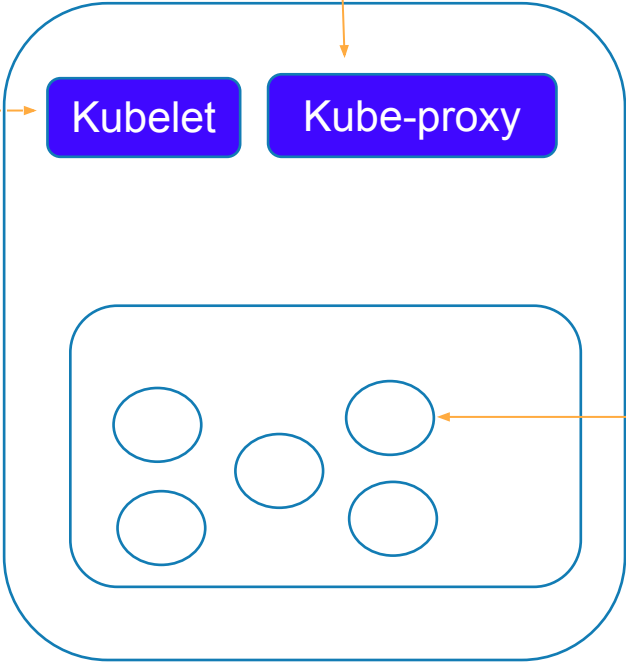


Overview

Master →



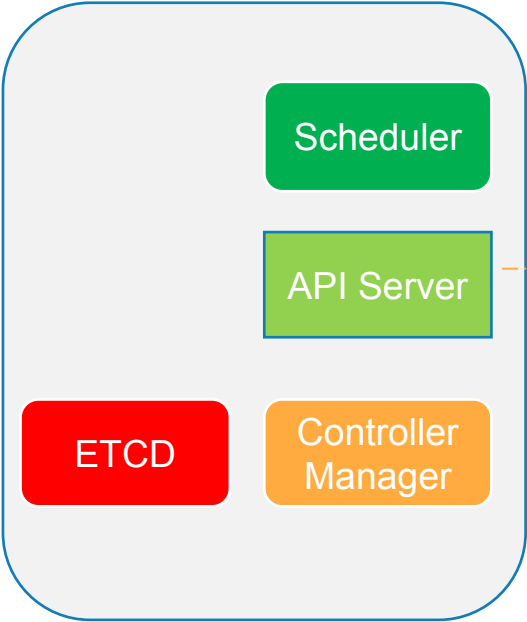
Worker Node-1



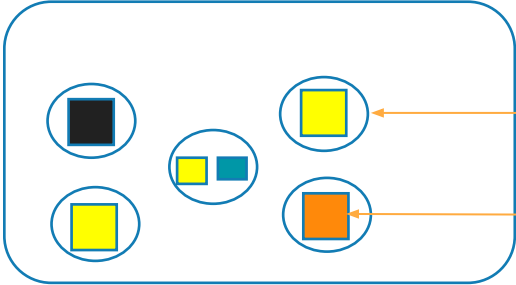
Pod

Overview

Master →



Worker Node-1



Pod

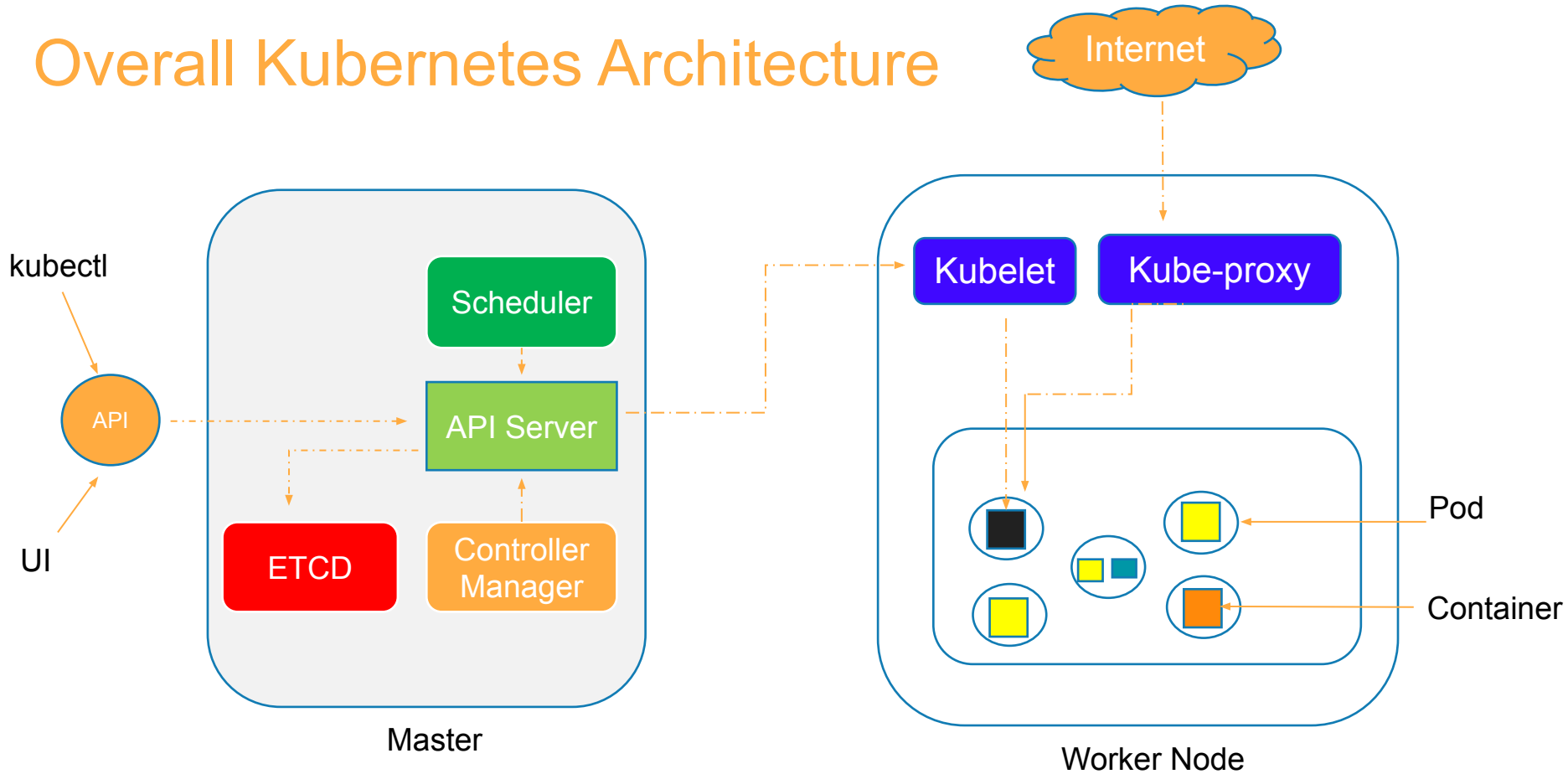
Container

Docker Containers

A popular Container Runtime



Overall Kubernetes Architecture



Demo

- Setting up a single Node K8s cluster on Docker Desktop for Mac / Windows
- Setting up 5 Node Kubernetes Cluster on PWK
- Setting up 3 Nodes K8s Cluster on Bare Metal or VM

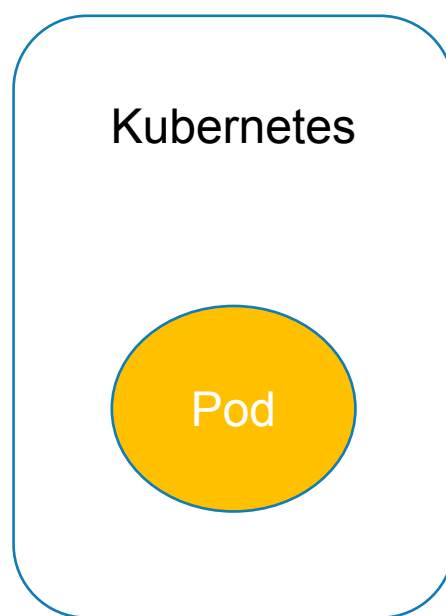
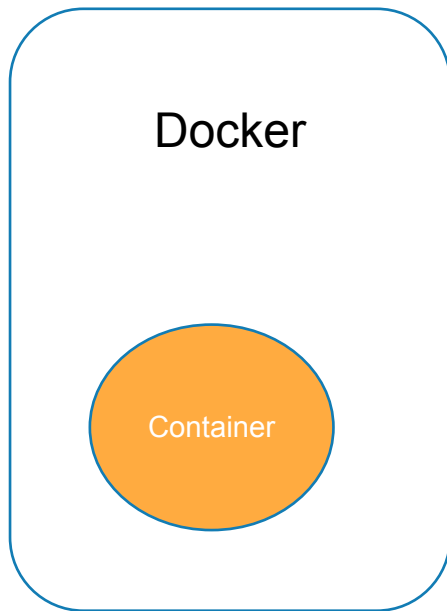
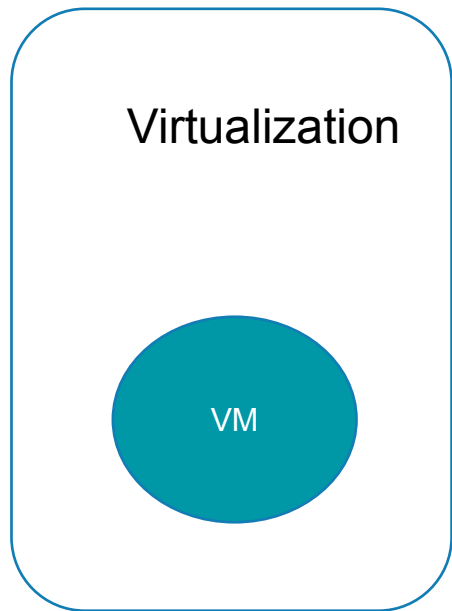
Let's Deep Dive into Pods...



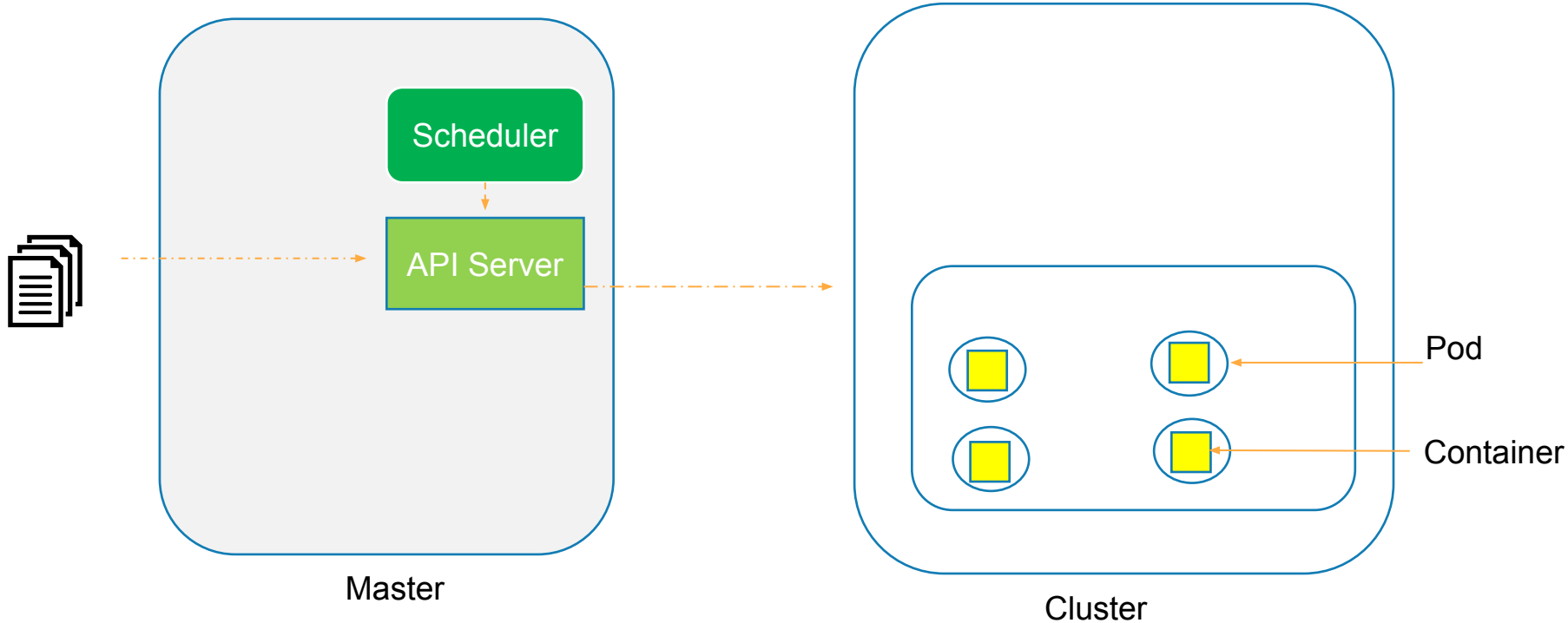
Pod - Concepts

- What is Pod?
- Pod Deployment
- Multi-Container
- Pod Networking
- Inter-Pod & Intra-Pod Networking
- Pod Lifecycle
- Pod Manifest File

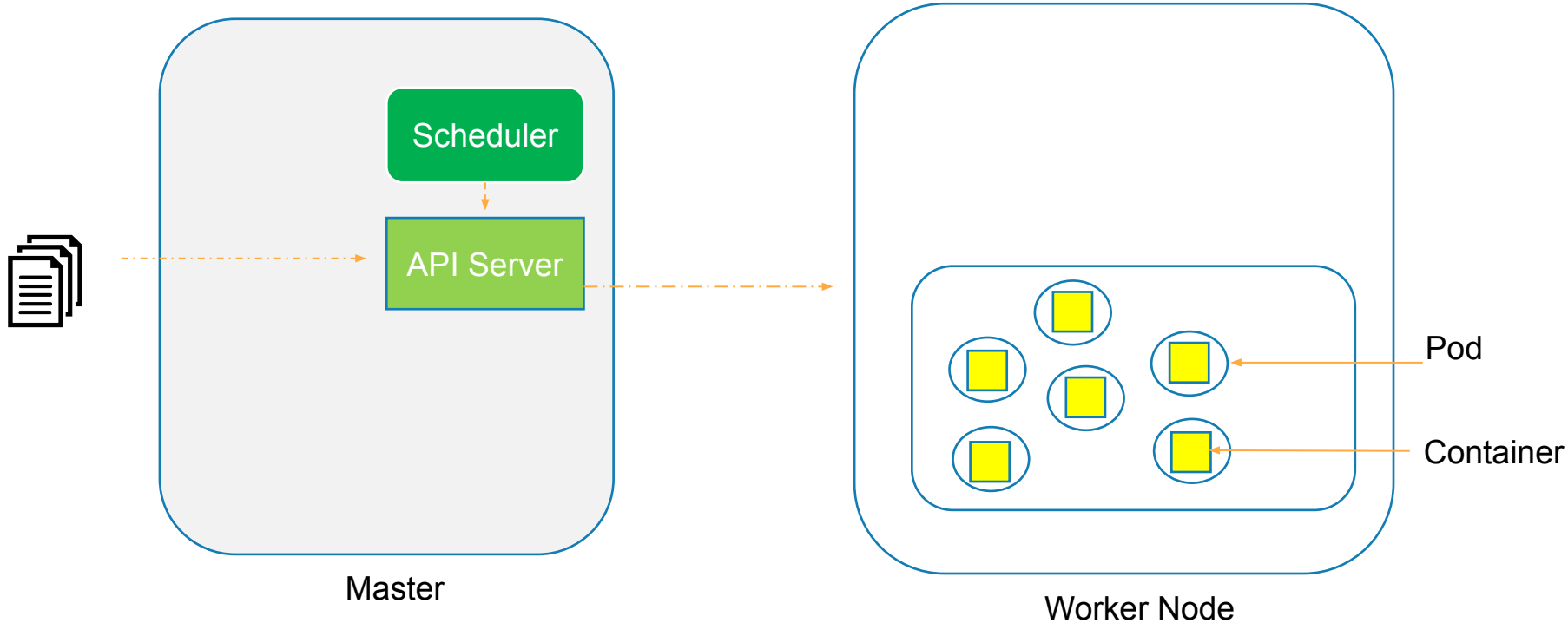
Atomic Unit of Scheduling



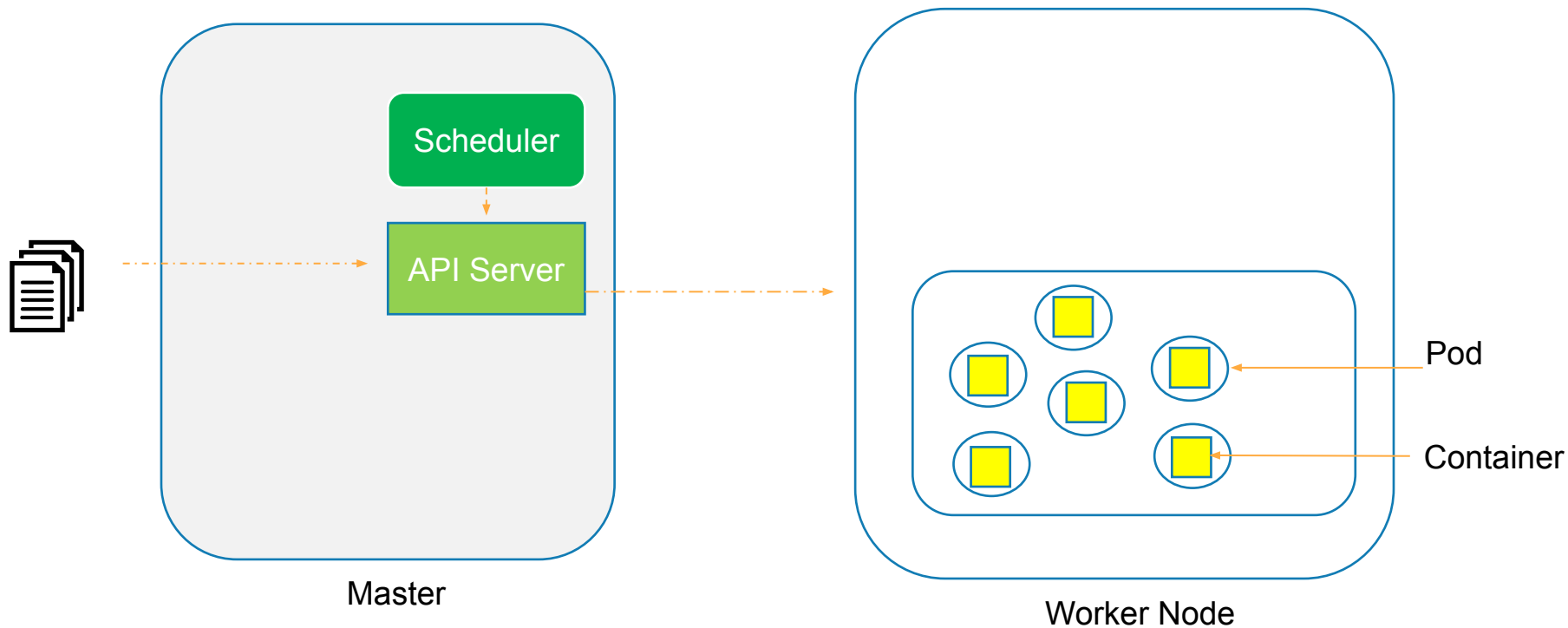
How Pods are deployed?



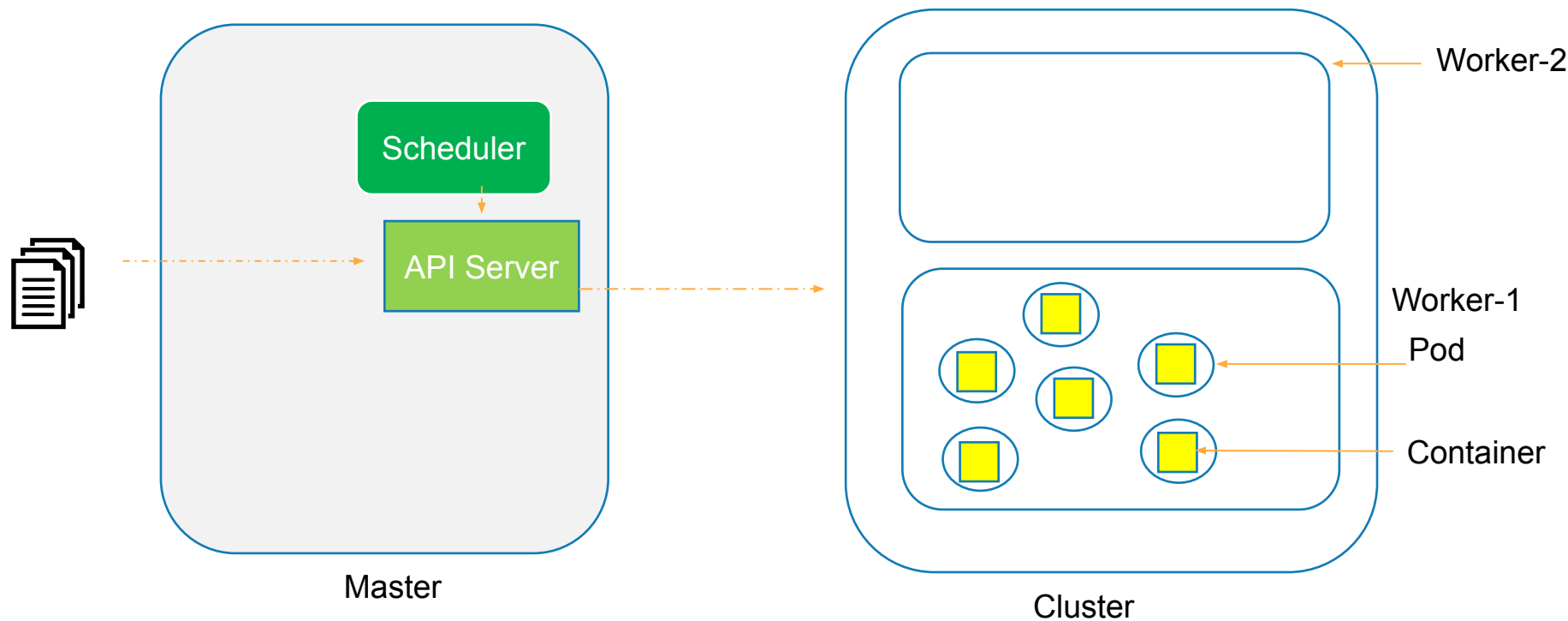
Scaling the Pods to accommodate increasing traffic



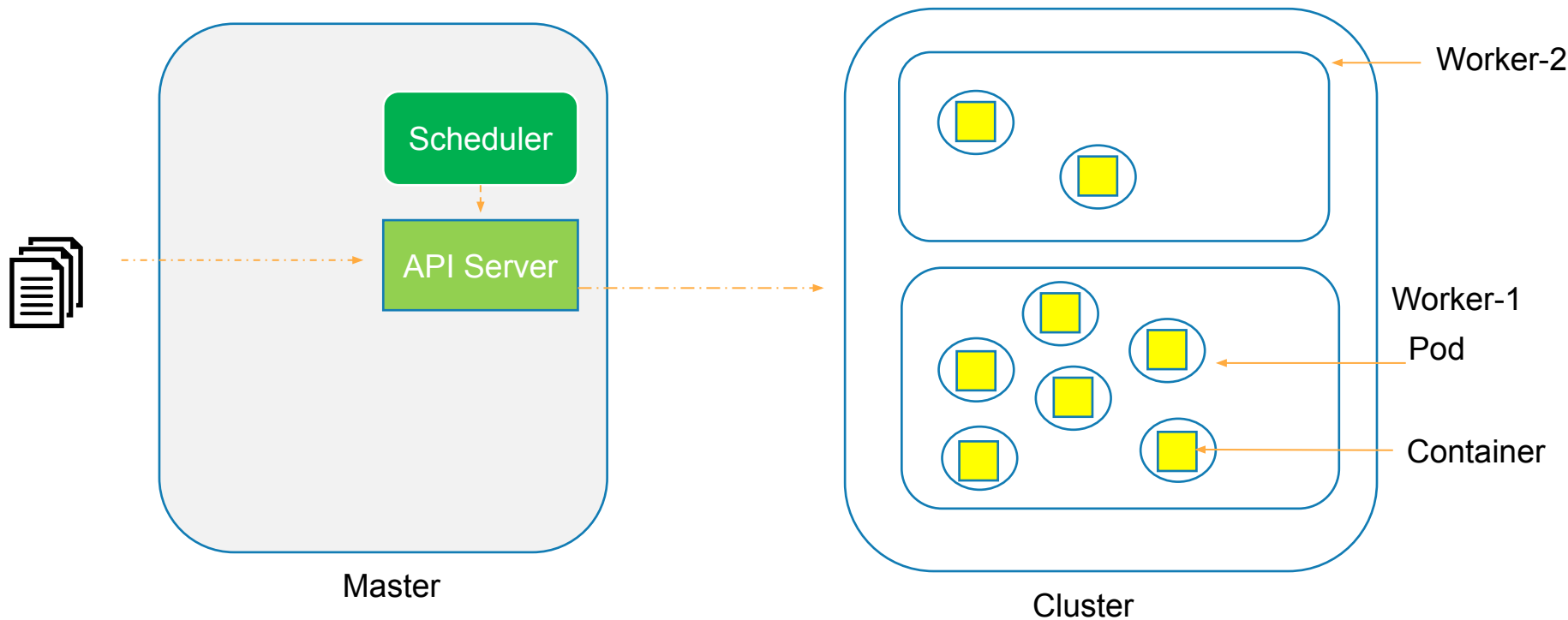
What if node resources is getting insufficient?



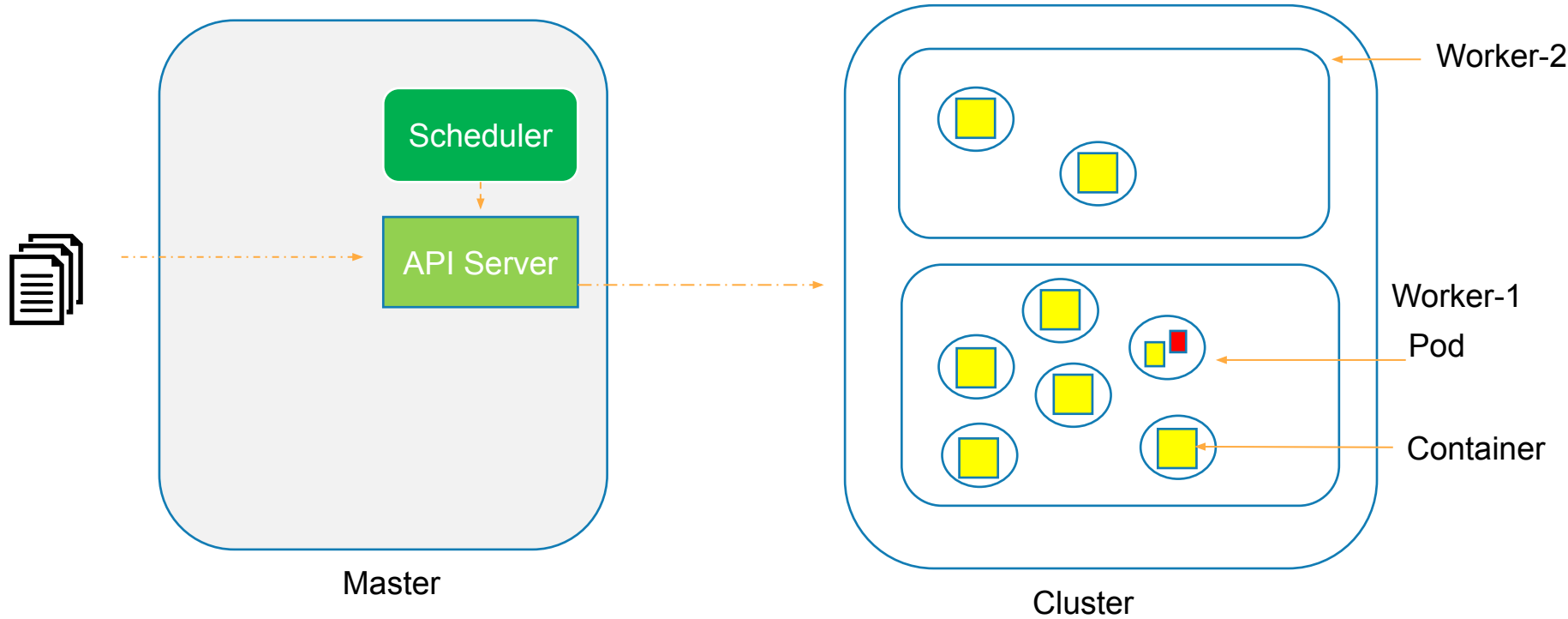
What if node resources is getting insufficient?



What if node resources is getting insufficient?

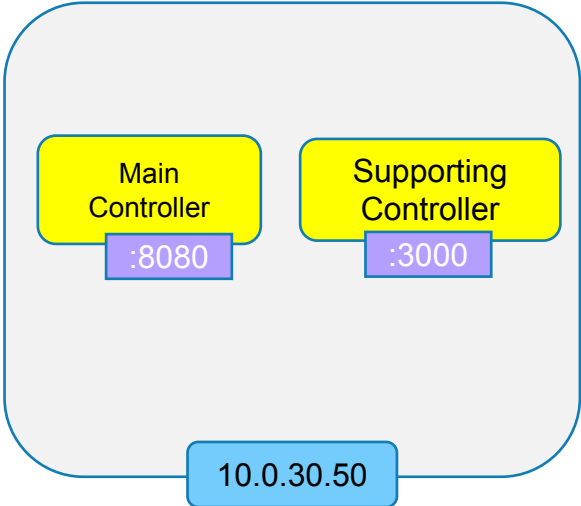


2 Containers in a same Pod

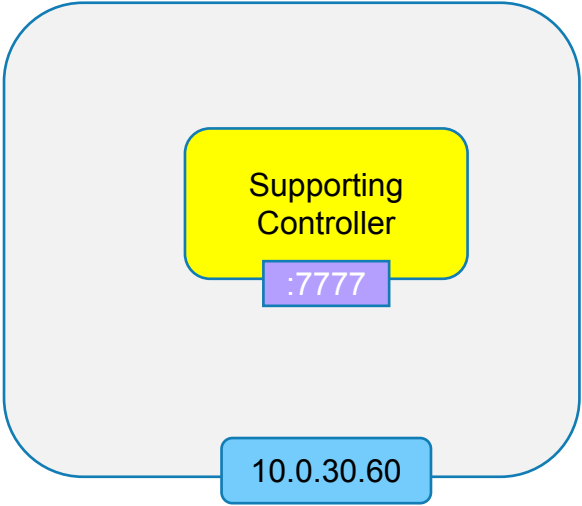


Pod Networking

Pod 1



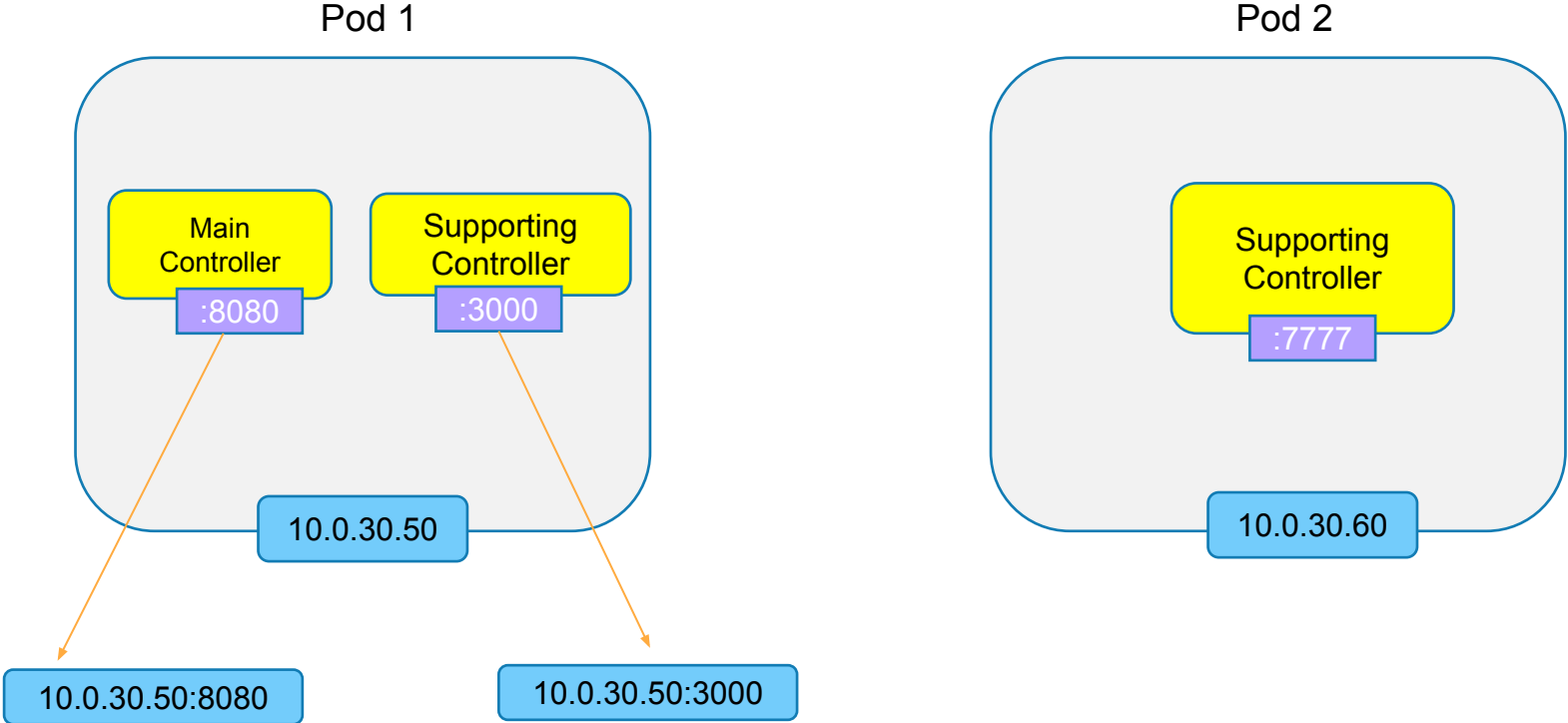
Pod 2



How does these containers
inside Pods communicate with
External World?



Network Namespace

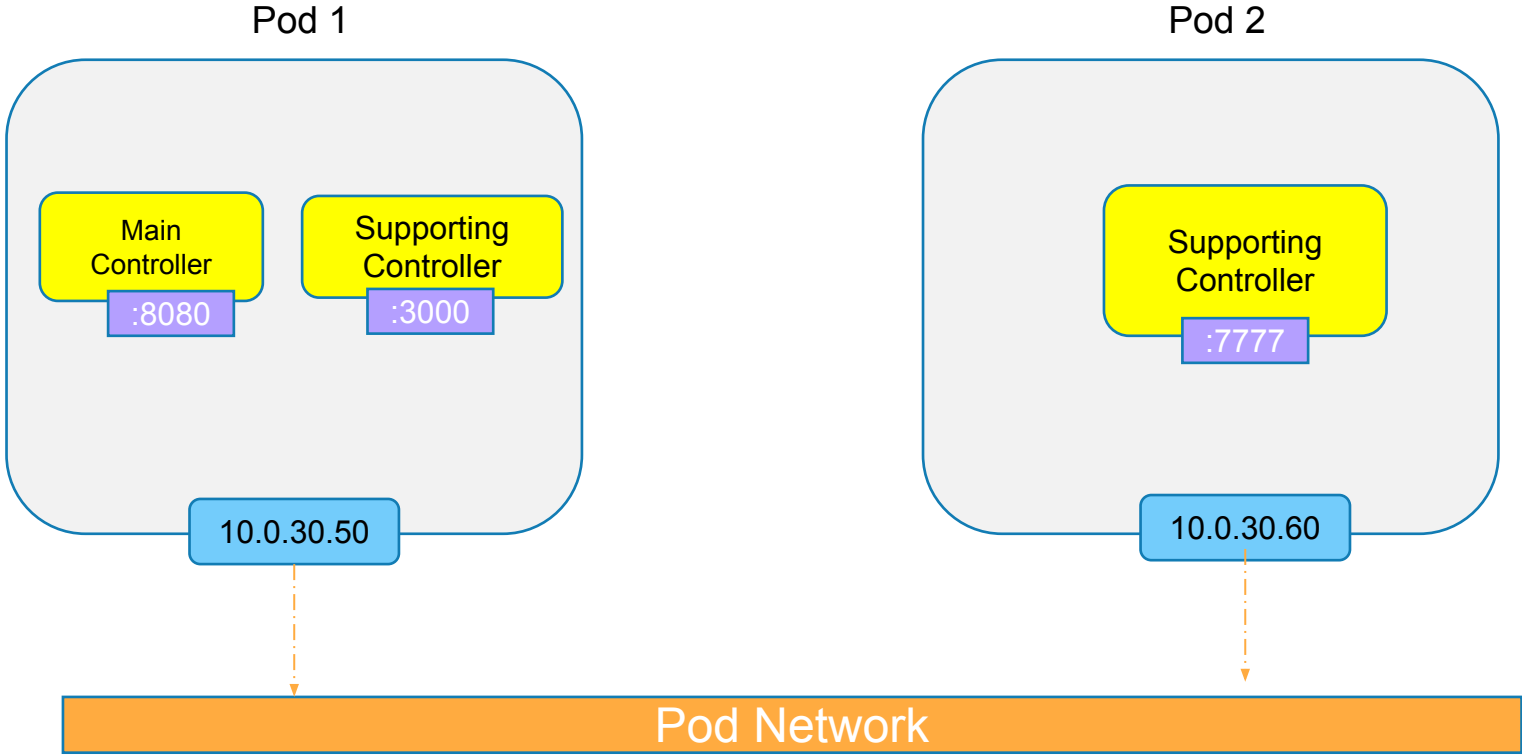


How does one Pod talk to another Pod?

Welcome to Inter-Pod Communication..



Pod Networking

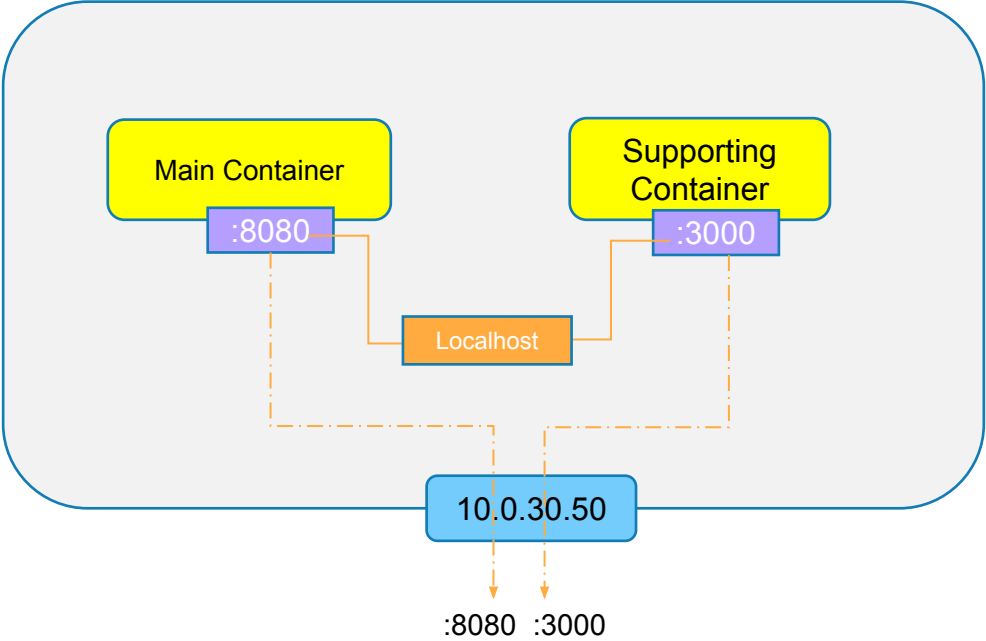


How does Intra-Pod
communication take place?



Intra-Pod Communication

Pod 1



A Look at Pod Manifest

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    name: nginx-pod
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```

Create the pod as shown below:

```
$ kubectl create -f templates/pod.yaml
pod "nginx-pod" created
```

Get the list of pod:

```
$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
nginx-pod    1/1     Running   0           22s
```

Get a shell to a running Container

```
[node1 lab01-creating-nginx-pod]$ kubectl get po
NAME          READY   STATUS    RESTARTS   AGE
nginx-pod     1/1     Running   0           3m22s
[node1 lab01-creating-nginx-pod]$ kubectl exec -it nginx-pod -- /bin/bash
```

Verifying the Operating System

```
root@nginx-pod:/# ls
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr
root@nginx-pod:/# cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 9 (stretch)"
NAME="Debian GNU/Linux"
VERSION_ID="9"
VERSION="9 (stretch)"
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
```


Get a shell to a running Container

```
root@nginx-pod:/# echo Hello shell demo > /usr/share/nginx/html/index.html
```

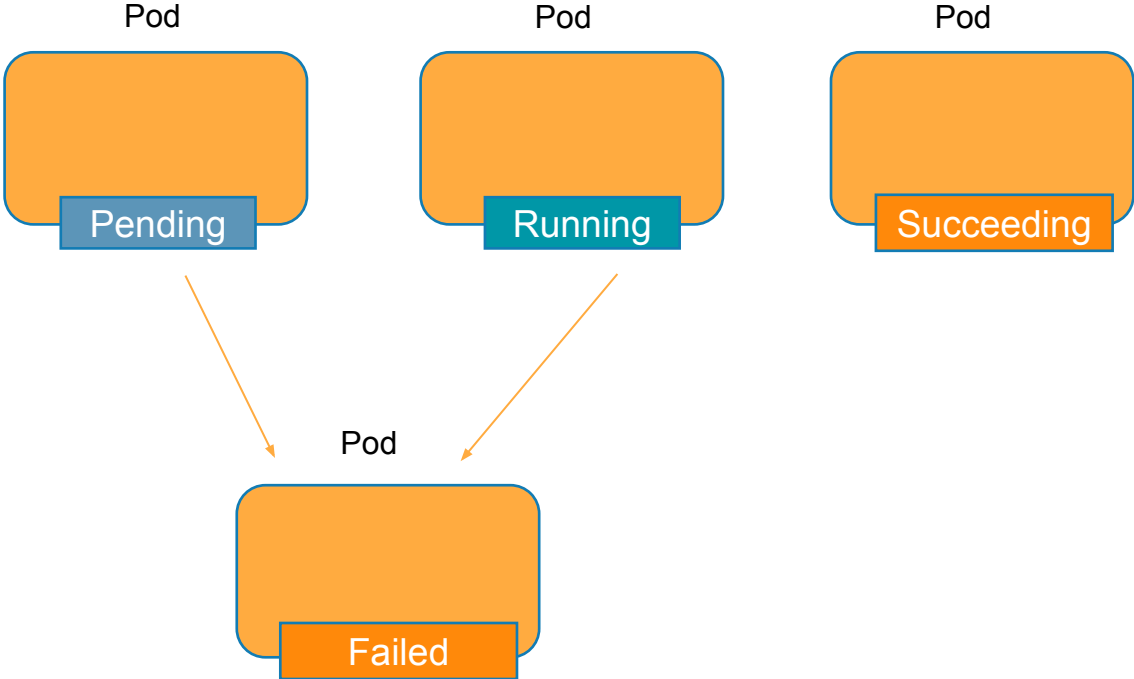
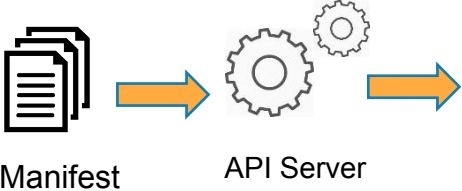
Verifying the index page

```
[node1 lab01-creating-nginx-pod]$ kubectl get po
NAME          READY   STATUS    RESTARTS   AGE
nginx-pod     1/1     Running   0           13m
[node1 lab01-creating-nginx-pod]$ kubectl get po -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP           NODE     NOMINATED NODE   READINESS GATES
nginx-pod     1/1     Running   0           13m   10.44.0.1    node2    <none>            <none>
[node1 lab01-creating-nginx-pod]$ curl 10.44.0.1:80
Hello shell demo
[node1 lab01-creating-nginx-pod]$
```

Stages of Life Cycle of Pod



Lifecycle of a Pod



How can you ensure that there are 3 Pods instances which are always available and running at point in time?

ReplicaSet



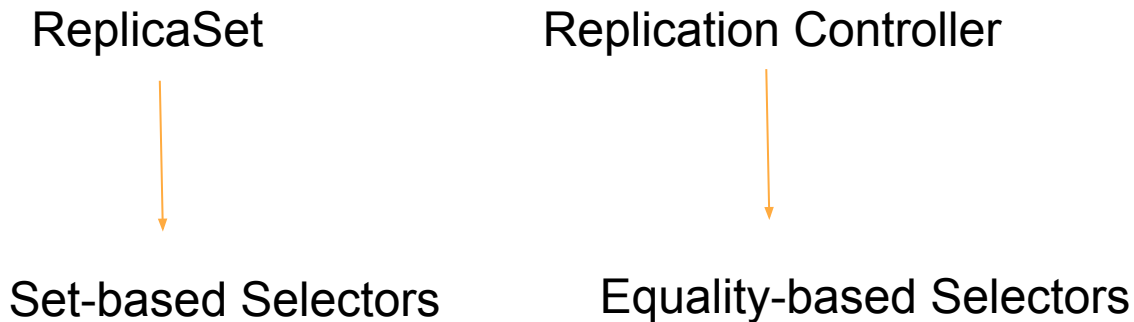
What is ReplicaSet all about?

Maintain a stable set of replica Pods running at any given time

- Ensures that a specified number of Pods are running at any time
 - a. If there are excess Pods, they get killed and vice versa
 - b. New Pods are launched when they get failed, get deleted and terminated
- ReplicaSet & Pods are associated with “labels”

Replication Controller Vs ReplicaSets

- ReplicaSet is the next generation of Replication Controller
- Both serve the same purpose



Labels & Selectors

When Pods are scaled, how are these Pods Managed at such large scale?

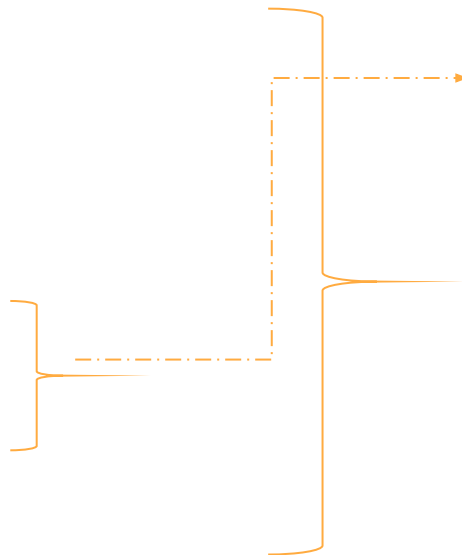
Pods

Controllers & Services

Labels

```
#Pod-Spec
apiVersion: v1
kind: pod
metadata:
  name: nginx-Pod
  labels:
    app: guestbook
    tier: frontend
    env: dev
spec:
  replicas: 5..
```

Selectors



Equality-based Selectors

Operators:

= and ==

Examples:

```
environment = production
tier! = frontend
```

Commandline:

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

In Manifest:

```
..
selector:
  environment: production
  tier: frontend
..
```

Supports: Services, Replication Controller

Set-based Selectors

Operators:

in notin exists

Examples:

```
environment in (production, qa)
tier notin(frontend, backend)
```

Commandline:

```
$kubectl get pods -l `environment in(production)
```

In Manifest:

```
..
selector:
  matchExpressions:
    - {key:environment,operator:in,values:[prod,qa]}
    - {key:tier,operator:Notin,values:[frontend,backend]}
..
```

Supports: Job, Deployment, ReplicaSet, DaemonSet


```
...
selector:
  app: nginx
  tier: frontend
...
```

=

```
...
selector:
  matchLabels:
    app: nginx
    tier: frontend
...
```

Supports on Older Resources such as:

- ReplicationControllers,
- Services

Supports on newer resources such as:

- ReplicaSets
- Deployments
- Jobs
- DaemonSet

Demo - ReplicaSet

- Manifest file
- Deploy app using RS
Display and validate RS
- Test – Node Fails
- Test – Scale Up
- Test – Scale Down

ReplicaSet Manifest File

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-rs
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx-app
  template:
    metadata:
      name: nginx-pod
      labels:
        app: nginx-app
        tier: frontend
    spec:
      containers:
        - name: nginx
          image: nginx
          ports:
            - containerPort: 80
```

Creating Nginx-rs Pods

```
$kubectl create -f nginx-rs.yaml
```

```
[node1 lab02-creating-replicaset]$ kubectl get po
NAME                READY   STATUS    RESTARTS   AGE
nginx-pod           1/1     Running   0           36m
nginx-rs-jl266      1/1     Running   0           62s
nginx-rs-jq74j      1/1     Running   0           62s
```

```
[node1 lab02-creating-replicaset]$ kubectl get po -l tier=frontend
NAME                READY   STATUS    RESTARTS   AGE
nginx-rs-jl266      1/1     Running   0           2m52s
nginx-rs-jq74j      1/1     Running   0           2m52s
```

```
[node1 lab02-creating-replicaset]$ kubectl get rs
NAME    DESIRED   CURRENT   READY   AGE
nginx-rs  2         2         1       12m
[node1 lab02-creating-replicaset]$ kubectl get rs -o wide
NAME    DESIRED   CURRENT   READY   AGE    CONTAINERS   IMAGES   SELECTOR
nginx-rs  2         2         1       12m    nginx        nginx    app=nginx-app
```

```
[node1 lab02-creating-replicaset]$ kubectl describe rs
Name:          nginx-rs
Namespace:     default
Selector:      app=nginx-app
Labels:        <none>
Annotations:   <none>
Replicas:      2 current / 2 desired
Pods Status:   2 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=nginx-app
           tier=frontend
  Containers:
    nginx:
      Image:      nginx
      Port:       80/TCP
      Host Port:  0/TCP
      Environment: <none>
      Mounts:     <none>
      Volumes:    <none>
Events:
  Type     Reason          Age    From          Message
  ----     -
  Normal   SuccessfulCreate 14m    replicaset-controller Created pod: nginx-rs-jq74j
  Normal   SuccessfulCreate 14m    replicaset-controller Created pod: nginx-rs-jl266
```

Scaling the Nginx Service

```
[node1 lab02-creating-replicaset]$ kubectl scale rs nginx-rs --replicas=5  
replicaset.extensions/nginx-rs scaled
```

Deployment



Deployment

A Deployment controller provides declarative updates for Pods and ReplicaSets.

You describe a desired state in a Deployment, and the Deployment controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

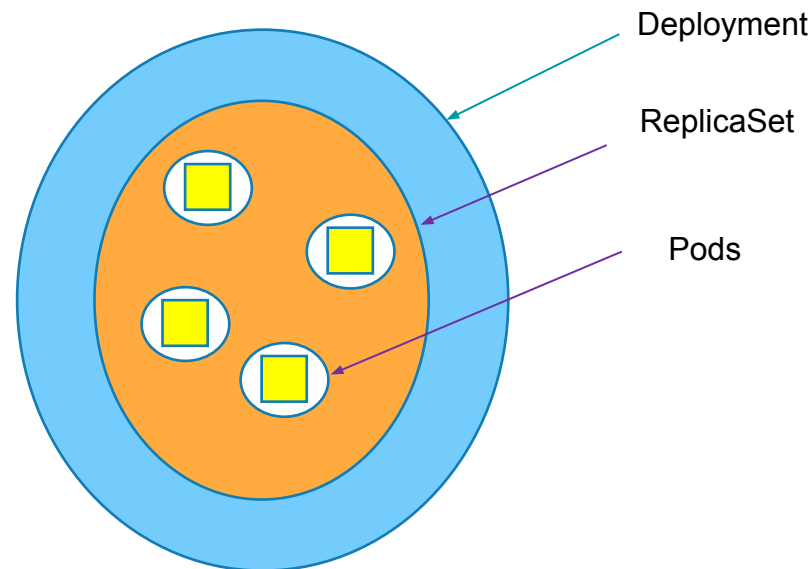
How is it different from Replicaset?

ReplicaSet doesn't provide features like updates & roll backs.

A Single Deployment Manifest File

Do we need to create 3 different manifest files for each on these?

Answer is “No”. We can create all 3 different objects using a single Deployment manifest file



Features of Deployment

- Multiple Replicas
- Upgrade
- Rollback
- Scale Up or Down
- Pause & Resume

Deployment Types - Recreate

- Recreate

How it works?

Shutting down version A and then making sure, version A is turned off... then bringing up version B.

Demerits:

During this, there will be a downtime of the service.

Easy to setup.

- Blue/Green

Deployment Type – Rolling Updates

- RollingUpdate(Ramped or Incremental)
 - Default updating strategy in Kubernetes.
 - It can take sometime for a complete update process

How it works?

Slowly rollout a version of app by replacing instances one after the other until all the instances are successfully rolled out.

Assume that there are 10 instances of version A which is running behind the LB. Then update strategy starts with one instance of version B is deployed When version B is ready to accept traffic, one instance of version A is removed from the pool

Deployment Type - Canary

- Canary
 - Ideal deployment method for someone who want to test newer version before it is deployed 100%.

How it works?

This method is all about gradually shifting production traffic from version A to version B.

Lets imagine that there are about 10 instances of app version A running inside a cluster. You use Canary deployment when you dont want to upgrade all of your instances. Let's say you upgraded your 2 instances of ver A to version B then do some testing. If test results are good, then you upgrade remaining 8 instances to version B. Say, your version B is ready, then you completely shut down version A.

Deployment Type – Blue Green

- Blue Green
 - Instance roll out and roll back.

How it works?

Using this method, version B(which is GREEN) is deployed along side version A(which is BLUE) with exactly same amount of instances.

After testing new version with all the requirement, the traffic is switched from version A to version B at the LB level.

Demo - Deployment

- Manifest file
- Deploy app using RS
- Display and validate RS
- Test – Node Fails
- Test – Scale Up
- Test – Scale Down

Deployment Manifest File

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy
  labels:
    app: nginx-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx-app
  template:
    metadata:
      name: nginx-pod
      labels:
        app: nginx-app
    spec:
      containers:
        - name: nginx
          image: nginx
          ports:
            - containerPort: 80
```

ReplicaSet

Pods

Deployment

```
[node1 lab03-creating-deployment-3replicas-nginx]$ ls
README.md  nginx-deploy.yaml
[node1 lab03-creating-deployment-3replicas-nginx]$ kubectl create -f nginx-deploy.yaml
deployment.apps/nginx-deploy created
[node1 lab03-creating-deployment-3replicas-nginx]$ kubectl get deploy
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx-deploy	0/3	3	0	6s

```
[node1 lab03-creating-deployment-3replicas-nginx]$ kubectl get deploy -o wide
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE	CONTAINERS	IMAGES	SELECTOR
nginx-deploy	0/3	3	0	16s	nginx	nginx	app=nginx-app

```
[node1 lab03-creating-deployment-3replicas-nginx]$ kubectl get deploy -o wide
```

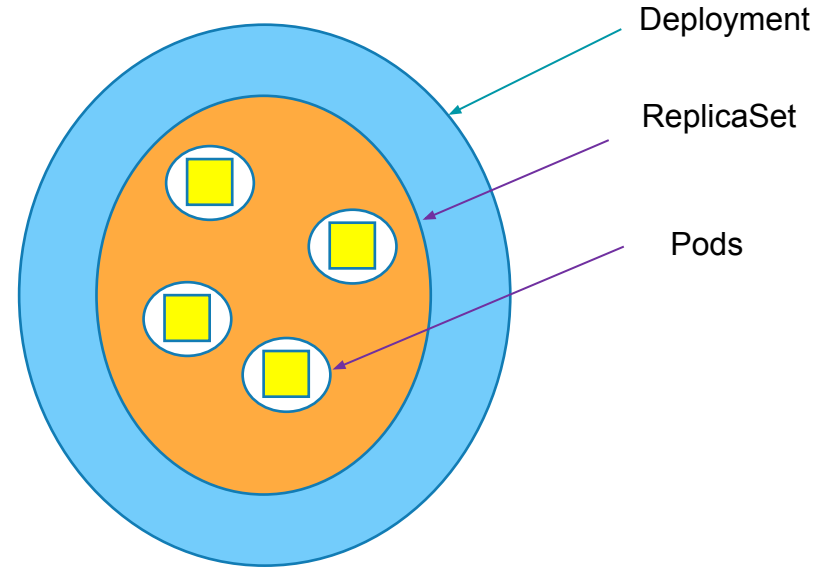
NAME	READY	UP-TO-DATE	AVAILABLE	AGE	CONTAINERS	IMAGES	SELECTOR
nginx-deploy	3/3	3	3	57s	nginx	nginx	app=nginx-app

Deployment => Pods + ReplicaSet

```
[node1 lab03-creating-deployment-3replicas-nginx]$ kubectl get po,rs,deploy
NAME                                READY   STATUS    RESTARTS   AGE
pod/nginx-deploy-c9d474fc-lhz9p     1/1     Running   0           2m25s
pod/nginx-deploy-c9d474fc-v8xwg     1/1     Running   0           2m25s
pod/nginx-deploy-c9d474fc-vx4cm     1/1     Running   0           2m25s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.extensions/nginx-deploy-c9d474fc  3         3         3       2m25s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/nginx-deploy     3/3     3             3           2m25s
```



3 Instances of same Nginx Apps running in the form of Pods

```
[node1 lab03-creating-deployment-3replicas-nginx]$ kubectl get po,rs,deploy -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE	RE
ADINESS GATES									
pod/nginx-deploy-c9d474fc-lhz9p	1/1	Running	0	4m21s	10.47.0.1	node3	<none>		<n
one>									
pod/nginx-deploy-c9d474fc-v8xwg	1/1	Running	0	4m21s	10.44.0.1	node2	<none>		<n
one>									
pod/nginx-deploy-c9d474fc-vx4cm	1/1	Running	0	4m21s	10.36.0.1	node5	<none>		<n
one>									


```
NAME
```

NAME	DESIRED	CURRENT	READY	AGE	CONTAINERS	IMAGES	SELECT
replicaset.extensions/nginx-deploy-c9d474fc	3	3	3	4m21s	nginx	nginx	app=ng
inx-app,pod-template-hash=c9d474fc							

```
[node1 lab03-creating-deployment-3replicas-nginx]$ kubectl get deploy -l app=nginx-app
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx-deploy	3/3	3	3	7m46s

```
[node1 lab03-creating-deployment-3replicas-nginx]$
```

3 Instances of same Nginx Apps running in the form of Pods

```
[node1 lab03-creating-deployment-3replicas-nginx]$ kubectl get rs -l app=nginx-app
```

NAME	DESIRED	CURRENT	READY	AGE
nginx-deploy-c9d474fc	3	3	3	8m33s

Update Deployment

```
[node1 lab03-creating-deployment-3replicas-nginx]$  
[node1 lab03-creating-deployment-3replicas-nginx]$ kubectl set image deploy nginx-deploy nginx=nginx:1.9.1  
deployment.extensions/nginx-deploy image updated
```

```
CreationTimestamp: Sat, 13 Jul 2019 18:50:48 +0000  
Labels: app=nginx-app  
Annotations: deployment.kubernetes.io/revision: 2  
Selector: app=nginx-app  
Replicas: 3 desired | 3 updated | 3 total | 3 available | 0 unavailable  
StrategyType: RollingUpdate  
MinReadySeconds: 0  
RollingUpdateStrategy: 25% max unavailable, 25% max surge  
Pod Template:  
  Labels: app=nginx-app  
  Containers:  
    nginx:  
      Image: nginx:1.9.1  
      Port: 80/TCP  
      Host Port: 0/TCP
```

3 Instances of same Nginx Apps running in the form of Pods

```
CreationTimestamp:    Sat, 13 Jul 2019 18:50:48 +0000
Labels:               app=nginx-app
Annotations:          deployment.kubernetes.io/revision: 2
Selector:             app=nginx-app
Replicas:             3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:        RollingUpdate
MinReadySeconds:     0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx-app
  Containers:
    nginx:
      Image:   nginx:1.9.1
      Port:    80/TCP
      Host Port: 0/TCP
      Environment: <none>
      Mounts:    <none>
      Volumes:   <none>
Conditions:
```

```
[node1 lab03-creating-deployment-3replicas-nginx]$ kubectl rollout status deployment/nginx-deploy
deployment "nginx-deploy" successfully rolled out
[node1 lab03-creating-deployment-3replicas-nginx]$
```

Scaling up

```
[node1 lab03-creating-deployment-3replicas-nginx]$ kubectl scale deployment nginx-deploy --replicas=6
deployment.extensions/nginx-deploy scaled
[node1 lab03-creating-deployment-3replicas-nginx]$ kubectl get deploy
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deploy  5/6     6            5           22m
[node1 lab03-creating-deployment-3replicas-nginx]$
```

```
[node1 lab03-creating-deployment-3replicas-nginx]$ kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deploy-5985c6547d-g8nf4       1/1     Running   0          7m38s
nginx-deploy-5985c6547d-jmfc5       1/1     Running   0          8m16s
nginx-deploy-5985c6547d-jnzhh       1/1     Running   0          96s
nginx-deploy-5985c6547d-nbfd8       1/1     Running   0          96s
nginx-deploy-5985c6547d-qr8r6       1/1     Running   0          96s
nginx-deploy-5985c6547d-rvkn6       1/1     Running   0          8m54s
[node1 lab03-creating-deployment-3replicas-nginx]$
```

Listing Pods by Labels

```
[node1 lab03-creating-deployment-3replicas-nginx]$ kubectl get po -l app=nginx-app
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deploy-5985c6547d-g8nf4       1/1     Running   0           8m25s
nginx-deploy-5985c6547d-jmfc5       1/1     Running   0           9m3s
nginx-deploy-5985c6547d-jnzhh       1/1     Running   0           2m23s
nginx-deploy-5985c6547d-nbfd8       1/1     Running   0           2m23s
nginx-deploy-5985c6547d-qr8r6       1/1     Running   0           2m23s
nginx-deploy-5985c6547d-rvkn6       1/1     Running   0           9m41s
[node1 lab03-creating-deployment-3replicas-nginx]$
[node1 lab03-creating-deployment-3replicas-nginx]$
[node1 lab03-creating-deployment-3replicas-nginx]$
```

Services



Services

- Imagine that, you have been asked to deploy web app
- How does this frontend web app exposed to outside world?
- How do front end app connected to backend database?
- How do we resolve Pod IP changes, when they die?

Agenda

- Why do we need services?
- What is Service?
- Type of Services

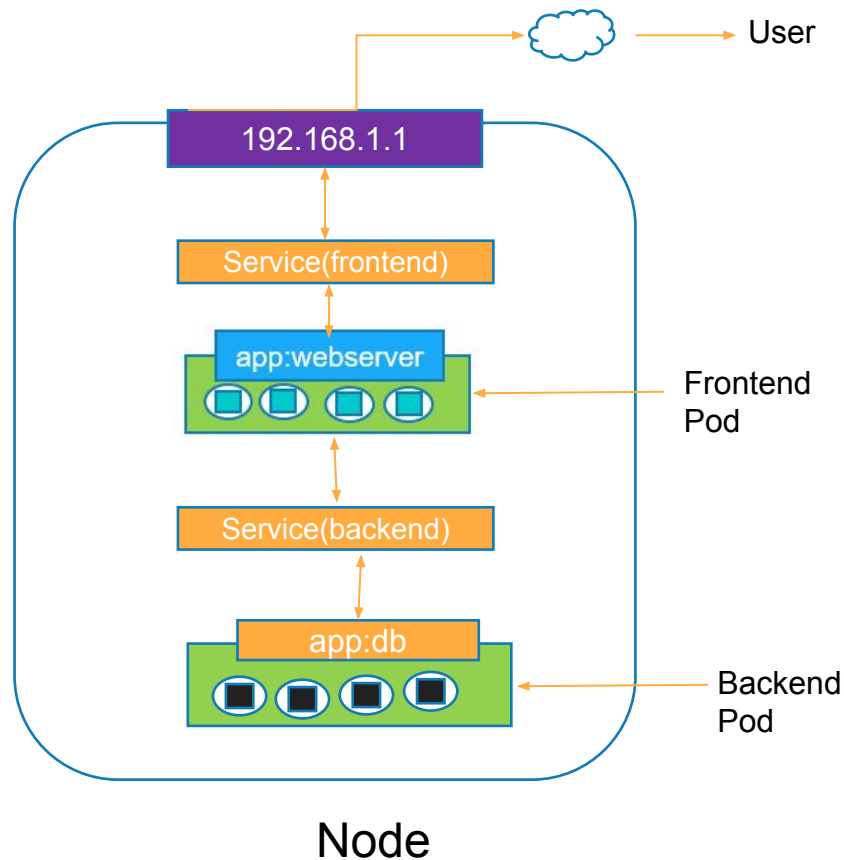
Services

Frontend Service:

A Service which stays between user and frontend pod

Backend Service:

A Service which communicate between frontend Pod and backend end



Types of Services

ClusterIP

Node-1

- Reachable within the cluster.
- Connects Frontend Pods to Backend Pods

NodePort

Node-1

- Exposing Frontend app to external world

LoadBalancer

Node-1

Node-1

Node-1

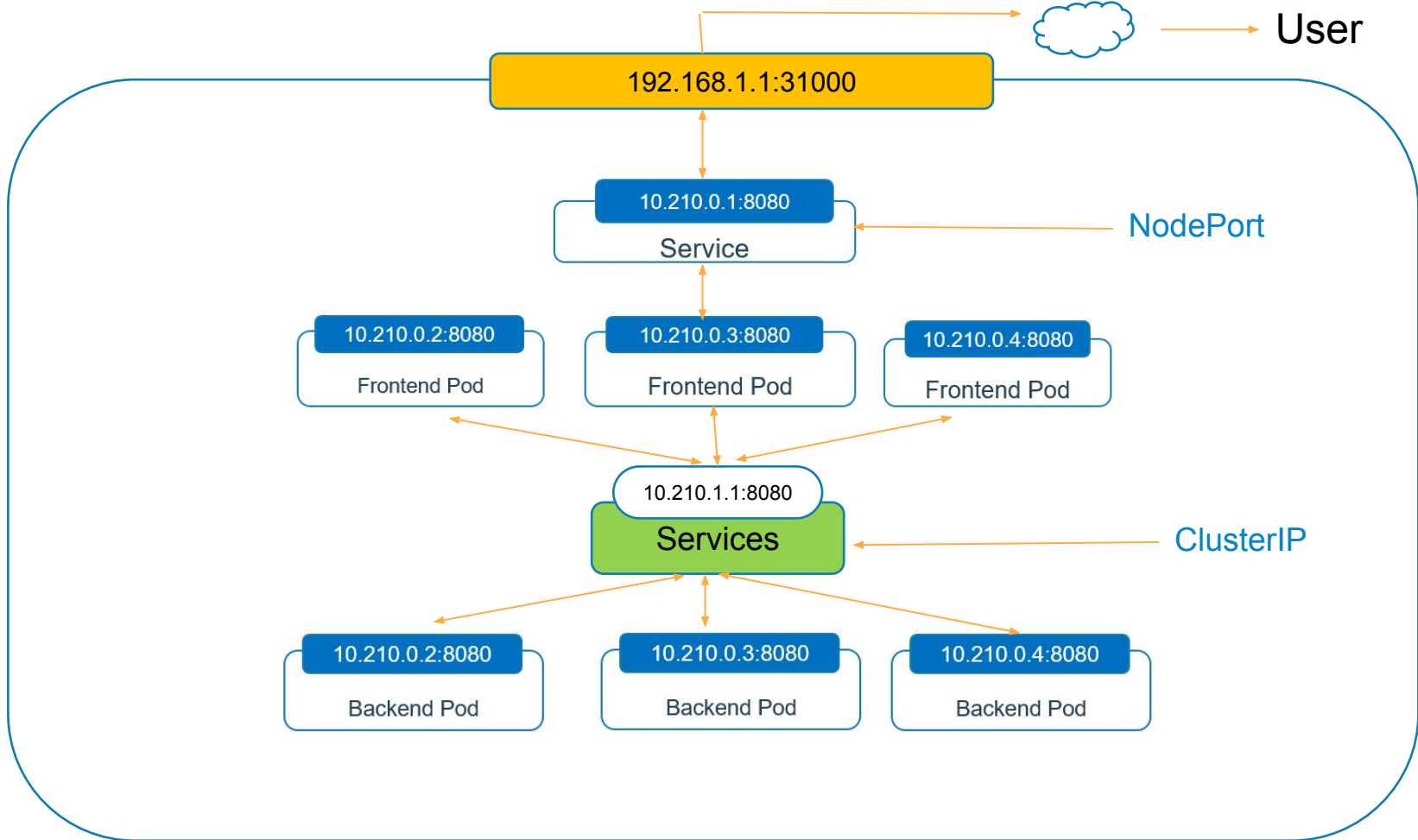
- Equally distribute the loads

Services: ClusterIP



Services

- Imagine you need to deploy one full fledged app which consists of frontend app & backend app
- How can we restrict access of backend database to only within the kubernetes cluster?



Guestbook Demo

- Frontend Web app
- Backend DB - Redis

Thank You

