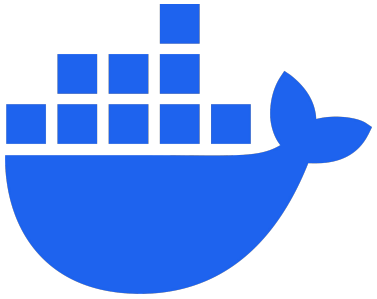
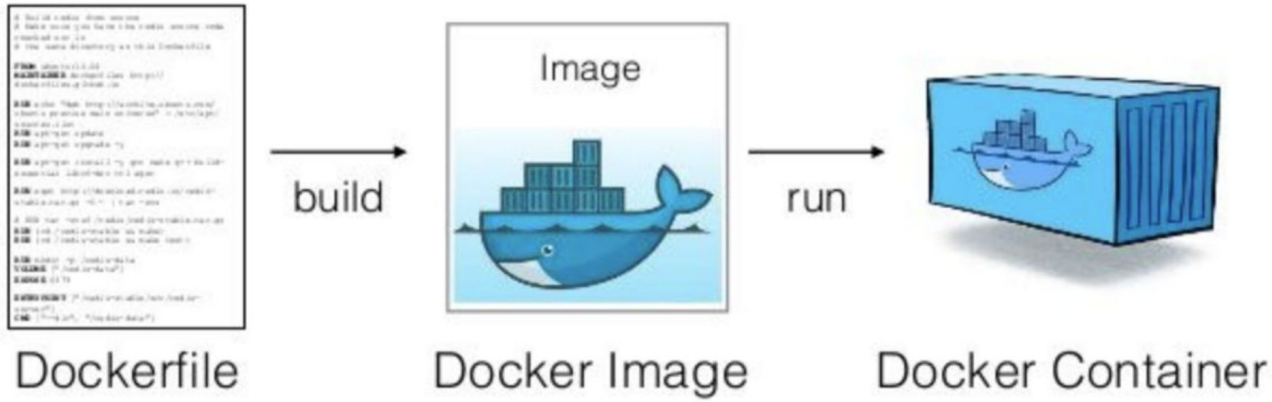
# Deeper dive into containers and its management (Docker, K8s)

CS695 – Topics in Virtualization and Cloud Computing

Debojeet Das



# Inspect Docker Image




Container images can either be built locally or “pulled” from a registry (which was built by someone).

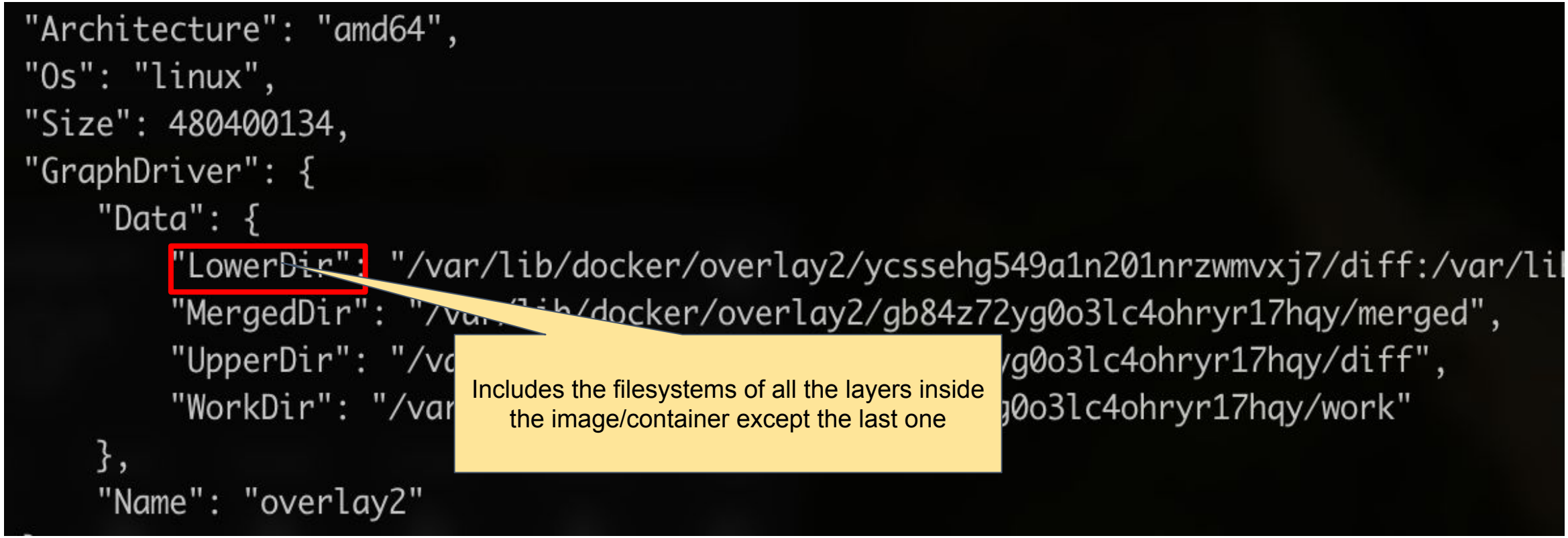
Let’s try to build a docker image and inspect it to better understand it.

In this docker image we will host a flask application.

```
$ docker build -t hellocs695:latest .
$ docker images
$ docker image inspect hellocs695:latest
```

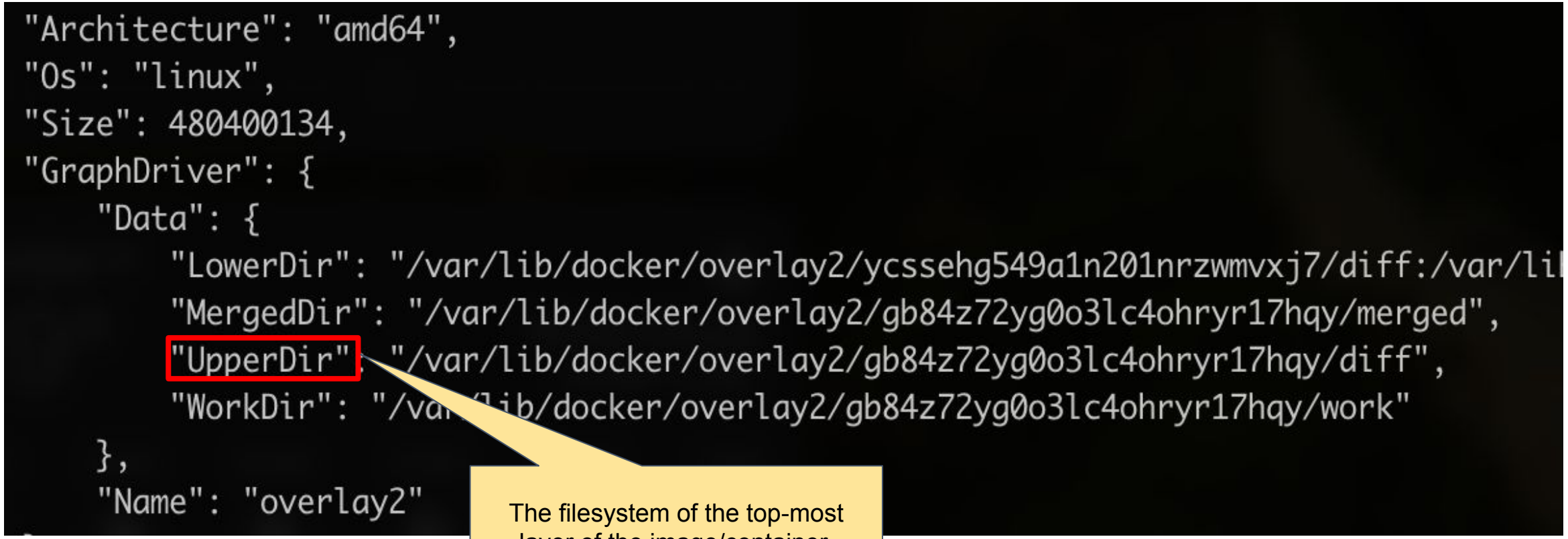
# Inspect Docker Image

```
$ docker image inspect hellocs695:latest
```



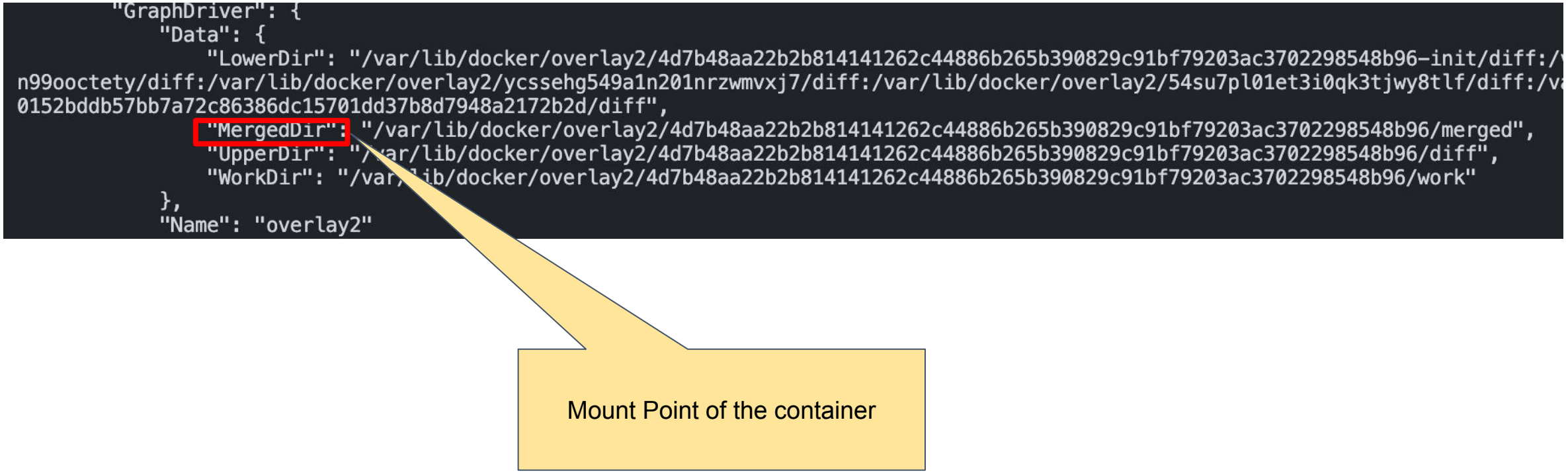
# Inspect Docker Image

```
$ docker image inspect hellocs695:latest
```



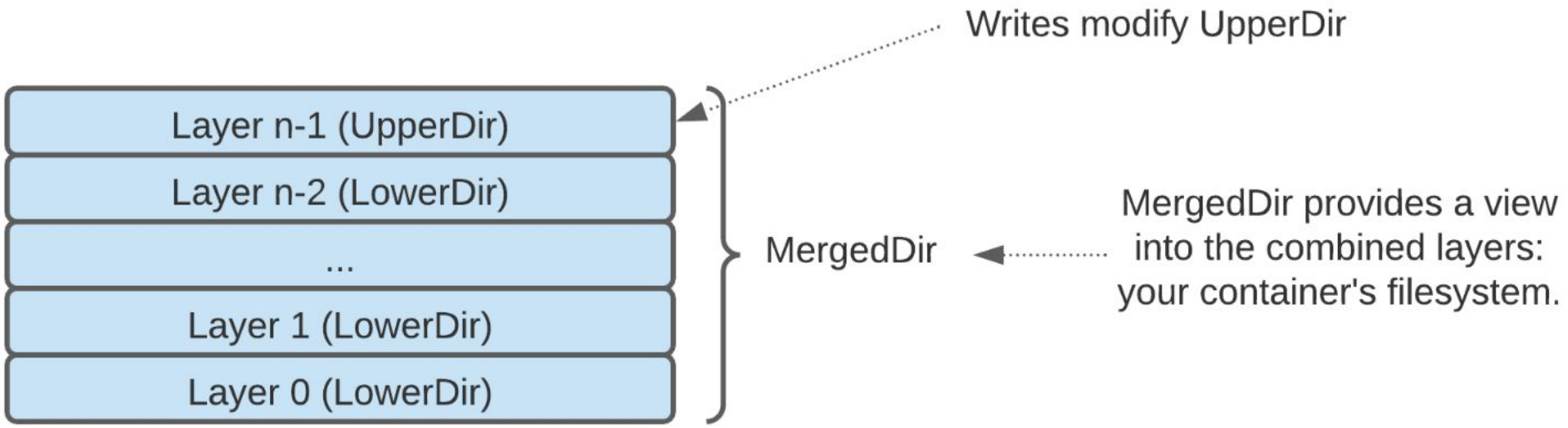
# Inspect Docker Container

```
$ docker run -p 80:8000 --name cont-test hellocs695:latest
$ docker ps
$ docker inspect cont-test
```



# Inspect Docker Container

```
$ docker run -p 80:8000 --name cont-test hellocs695:latest
$ docker ps
$ docker inspect cont-test
```

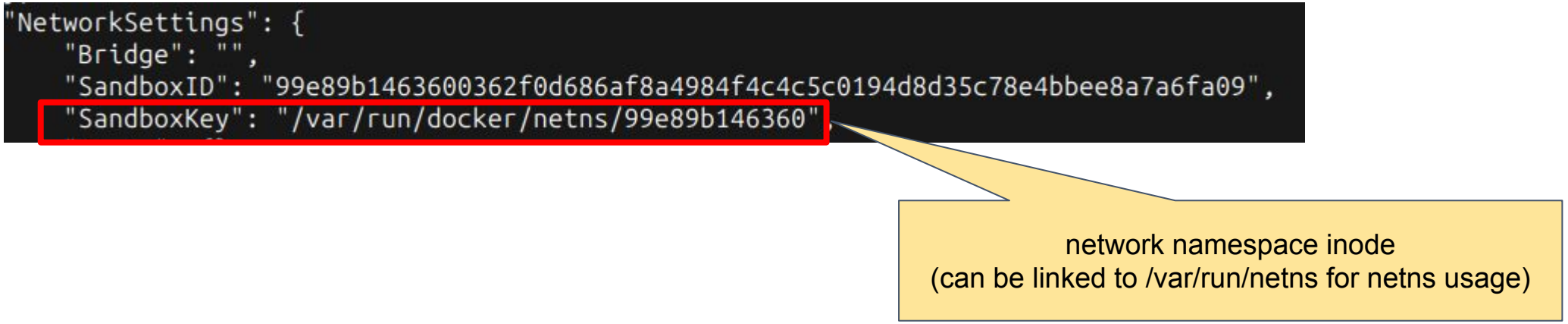





[image credit: https://blog.px.dev/container-filesystems/]

# Inspect Docker Container

```
$ docker run -p 80:8000 --name cont-test hellocs695:latest
$ docker ps
$ docker inspect cont-test
```




```
$ sudo mkdir /var/run/netns
$ sudo ln -s <sandbox-key> /var/run/netns/test
$ ip netns ls
```

# Inspect Docker Container

```
$ docker exec -it cont-test /bin/bash
$ apt install iproute2
$ ip a

- In other terminal type
$ sudo ip link add dev inside-test type veth peer name outside-test
netns test

- In previous terminal type
$ ip a
```

Using the same logic we used iproute2 to manipulate the networking in Assignment 3

# Kubernetes

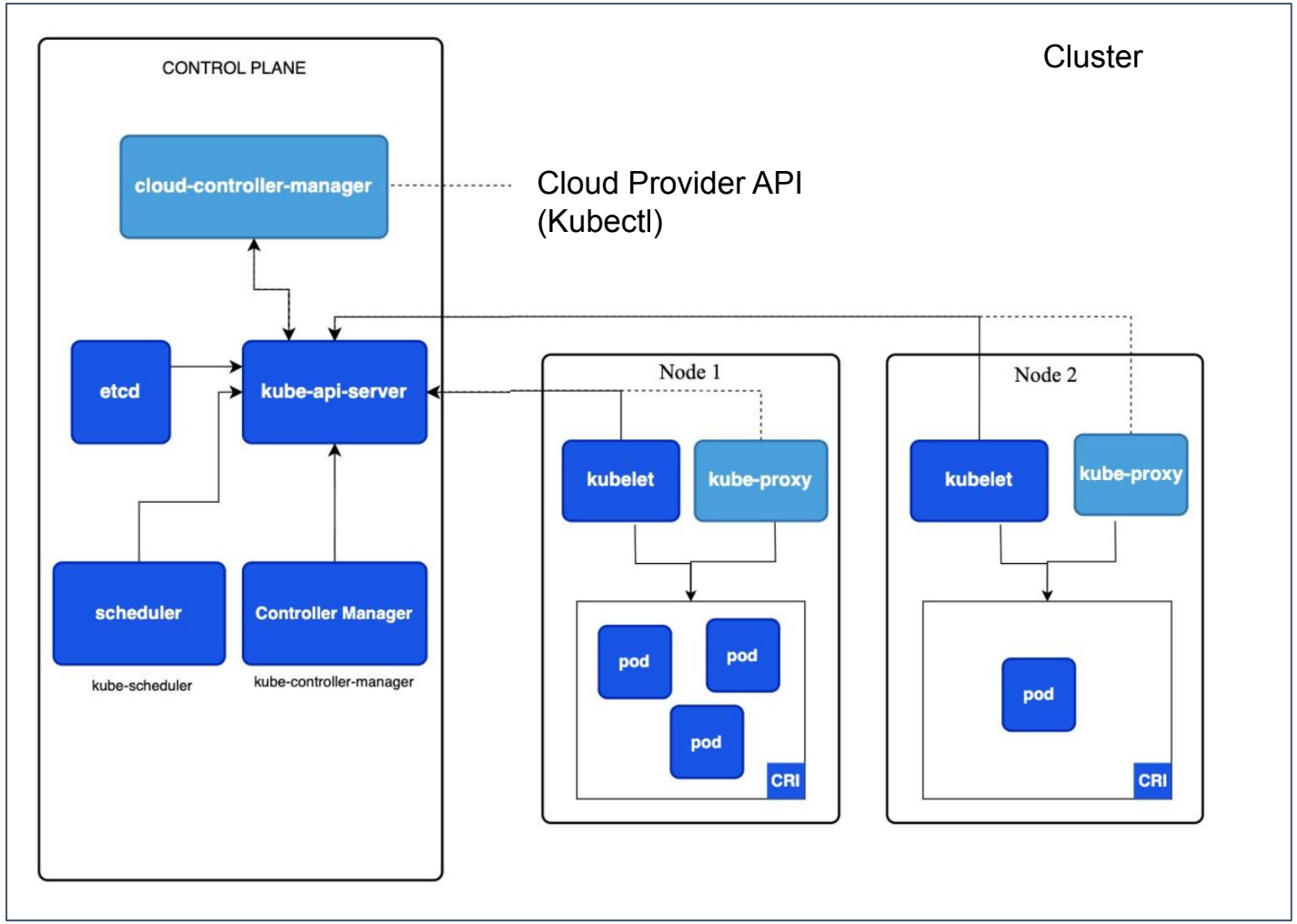
Docker: Single machine container deployment

Kubernetes (k8s): Container Orchestration

- Across a cluster of machines
- Manage automated deployment, scaling

Kubernetes is also a server-client application like docker. The k8s control plane (server) implements the cluster management and exposes HTTP API for communication which is used by applications like kubectl (client).

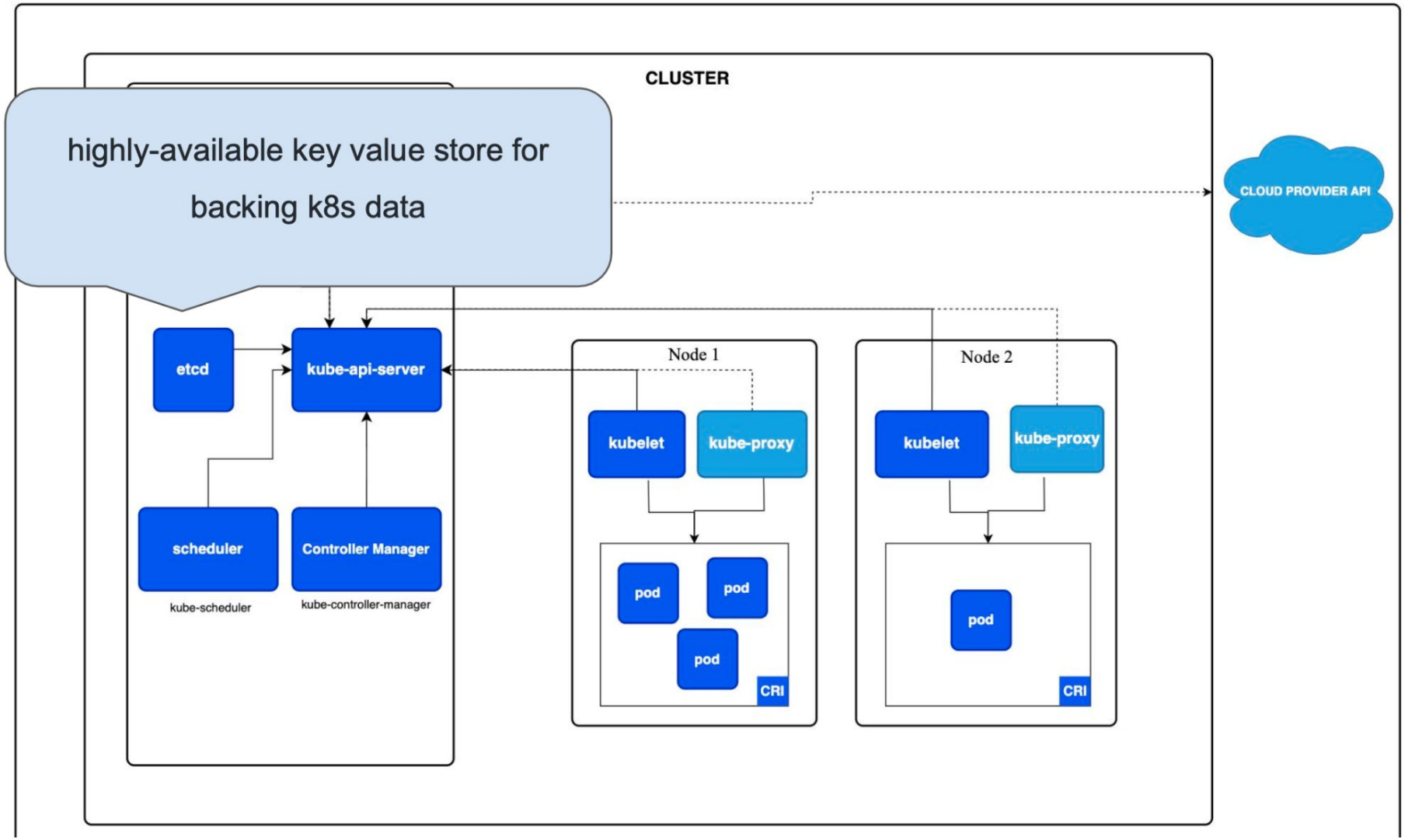
# Kubernetes Architecture





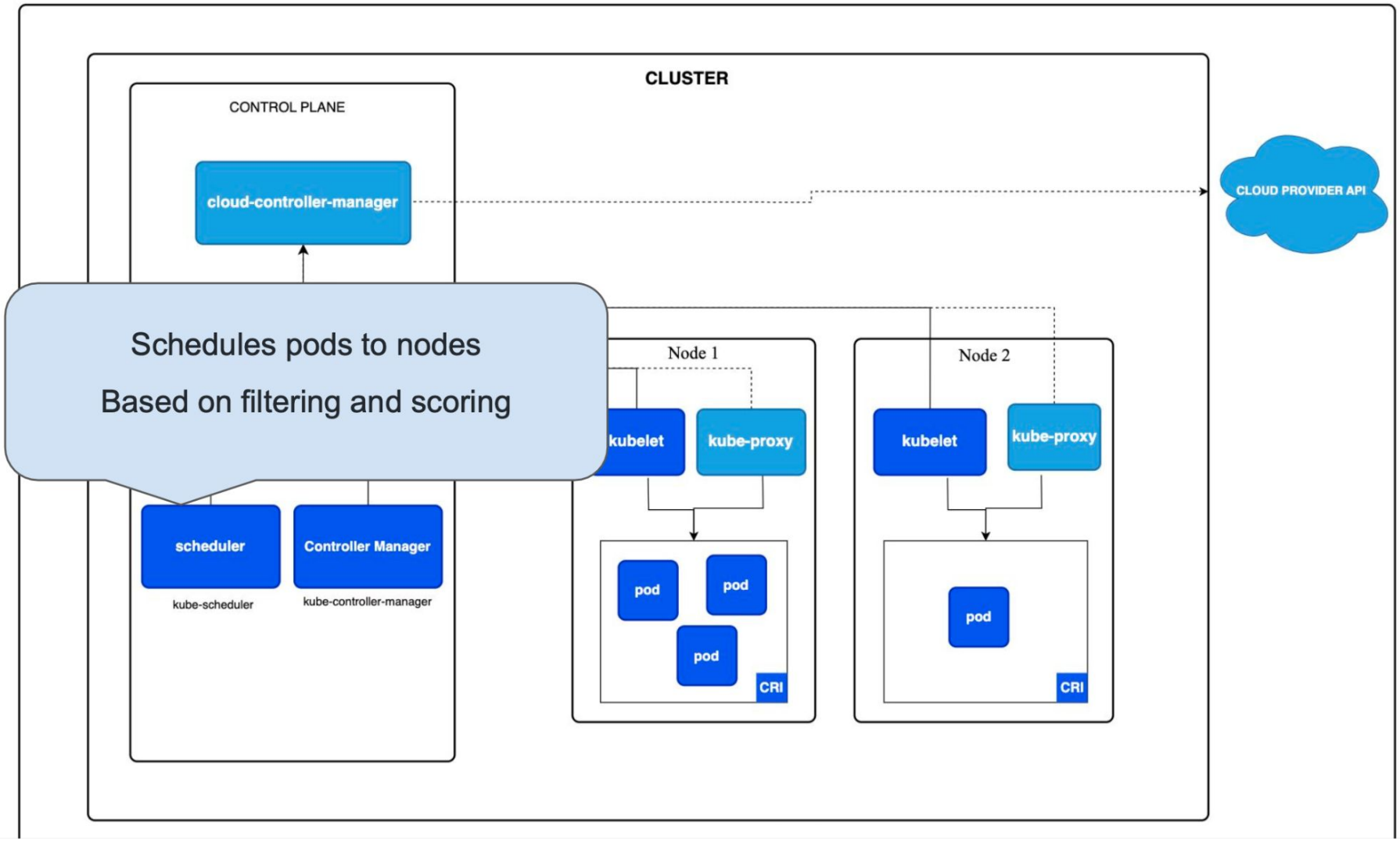

[image taken from ACM India Winter School on "Full-stack Networking (FSN)" ]

# Kubernetes Architecture




[image taken from ACM India Winter School on "Full-stack Networking (FSN)" ]

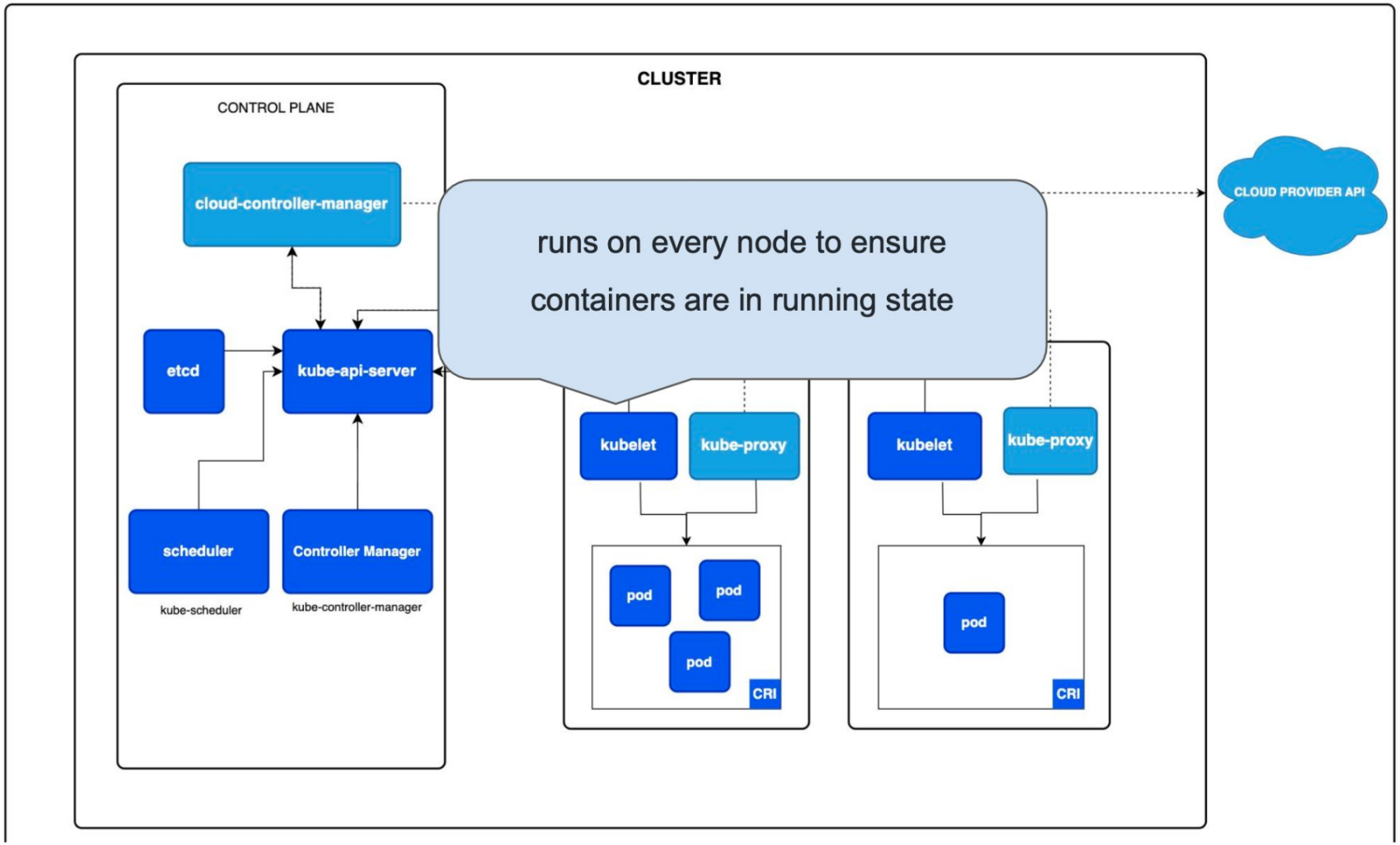
# Kubernetes Architecture





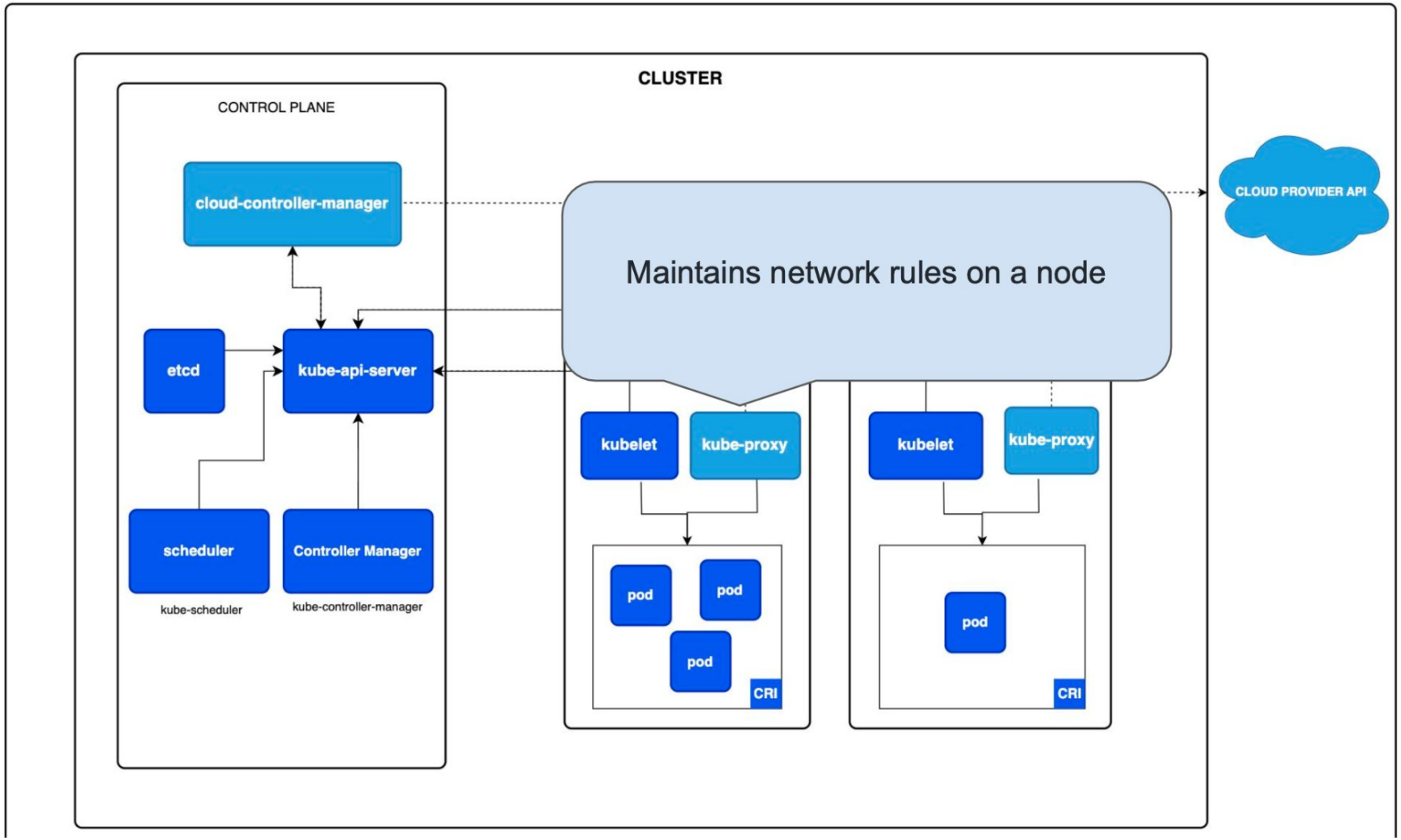

[image taken from ACM India Winter School on "Full-stack Networking (FSN)" ]

# Kubernetes Architecture




[image taken from ACM India Winter School on "Full-stack Networking (FSN)" ]

# Kubernetes Architecture




[image taken from ACM India Winter School on "Full-stack Networking (FSN)" ]

# Kubernetes Overview: Master and Worker Nodes

- Kubernetes clusters consist of workers, each running pods
- Control Plane: Manages workers and pods scheduling, fault-tolerance
  - kube-apiserver: REST based front end for k8s frontend
  - etcd: highly-available key value store for backing k8s data
- Node Components:
  - kubelet: runs on every node to ensure containers are in running state
  - kube-proxy: Maintains network rules on a node.
- Leverages system packet filtering layer if available
- container runtime: software for running containers. e.g. containerd, docker

# Get started with Kubectl

```
$ alias kubectl="kubectl --kubeconfig $PWD/config.yaml"
$ kubectl get pods
$ kubectl describe node
```

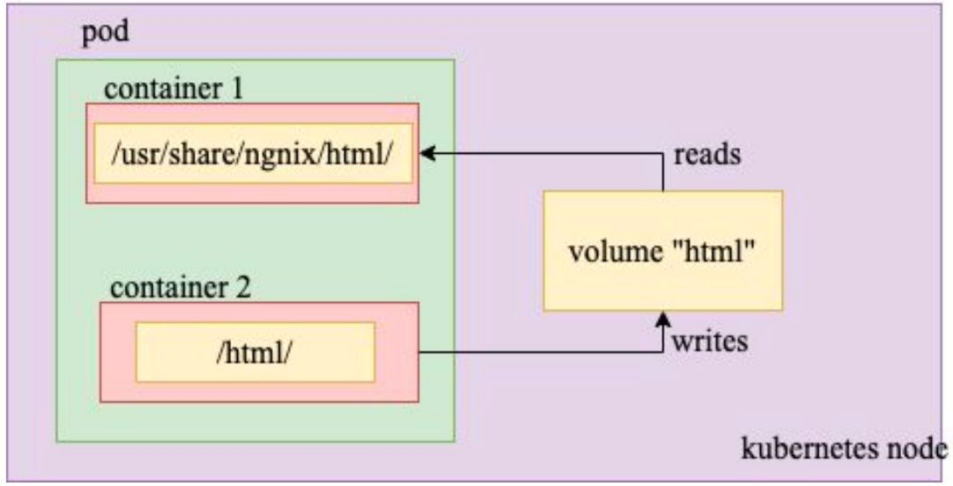
Create your own namespace so that they remain isolated from others

```
$ kubectl create namespace <ur-cseldap>
$ kubectl config set-context --current --namespace=<ur-cseldap>
```

# Pods

Application specific logical host.

- single unit of deployment
- group of containers with shared storage and network resources.






[image taken from ACM India Winter School on "Full-stack Networking (FSN)" ]

# Pods - Why Pod?

- Pod acts like a single server
- Logical wrapper around container
  - K8s container management
  - restart policy, liveness probe

Why multiple container/pod?

- Remember: microservice architecture
- "one process per container"
- Ease of debugging

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  containers:
    - name: test
      image: debojeetdas/hellocs695:latest
      ports:
        - containerPort: 8000
```

```
$ kubectl apply -f pod.yaml
$ kubectl get pods
$ kubectl describe pod test
$ kubectl logs test
$ kubectl get pods test -o jsonpath='{.spec.containers[*].name}'
$ kubectl exec -it test -c test -- /bin/bash
$ kubectl delete pod test
```

# Pods - Multiple container per pod

- Init:
  - Handle task that are needed before the app container starts (disk mounting, other microservice reachable)
  - Run to completion container
- Sidecar
  - Runs along with the application container
  - Performs tasks like:
    - Syncing data from a remote source
    - Log and metric collection
    - Network proxy
    - Encryption/decryption

# Pods - Multiple container per pod

```
$ kubectl apply -f sidecar.yaml
$ kubectl get pods
$ kubectl describe pod testshared
```

Since network namespace in shared across containers in pods. It should be straightforward to access the flask application.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: testshared
spec:
  containers:
  - name: first
    image: debojeetdas/hellocs695:latest
  - name: second
    image: debian
    command: ["/bin/sh", "-c"]
    args:
      - while true; do
          sleep 1;
        done
```

```
$ kubectl get pods testshared -o jsonpath='{.spec.containers[*].name}'
$ kubectl exec -it testshared -c second -- /bin/bash
$ apt update && apt install curl
$ curl localhost:8000
```

# Kubernetes - Running your application

Deployment (and, indirectly, ReplicaSet), the most common way to run an application on your cluster. Deployment is a good fit for managing a stateless application workload on your cluster, where any Pod in the Deployment is interchangeable and can be replaced if needed.


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-deployment
  labels:
    app: hellocs695
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hellocs695
  template:
    metadata:
      labels:
        app: hellocs695
    spec:
      containers:
      - name: hellocs695
        image: debojeetdas/hellocs695:latest
        ports:
        - containerPort: 8000
```



[image taken from ACM India Winter School on "Full-stack Networking (FSN)" ]

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

# Kubernetes - Running your application

```
$ kubectl get pods
$ kubectl delete pod test-deployment-6c5c8d8596-2c5wh
$ kubectl get pods
```




[image credit: https://www.reddit.com/r/kubernetes/comments/oymqvw/trying_to_delete_a_pod_thats_part_of_a_deployment/]

# Kubernetes - Running your application

Statefulset

- Deployment and scaling of stateful pods
- Stateful pod: requires persistent storage

Daemonset

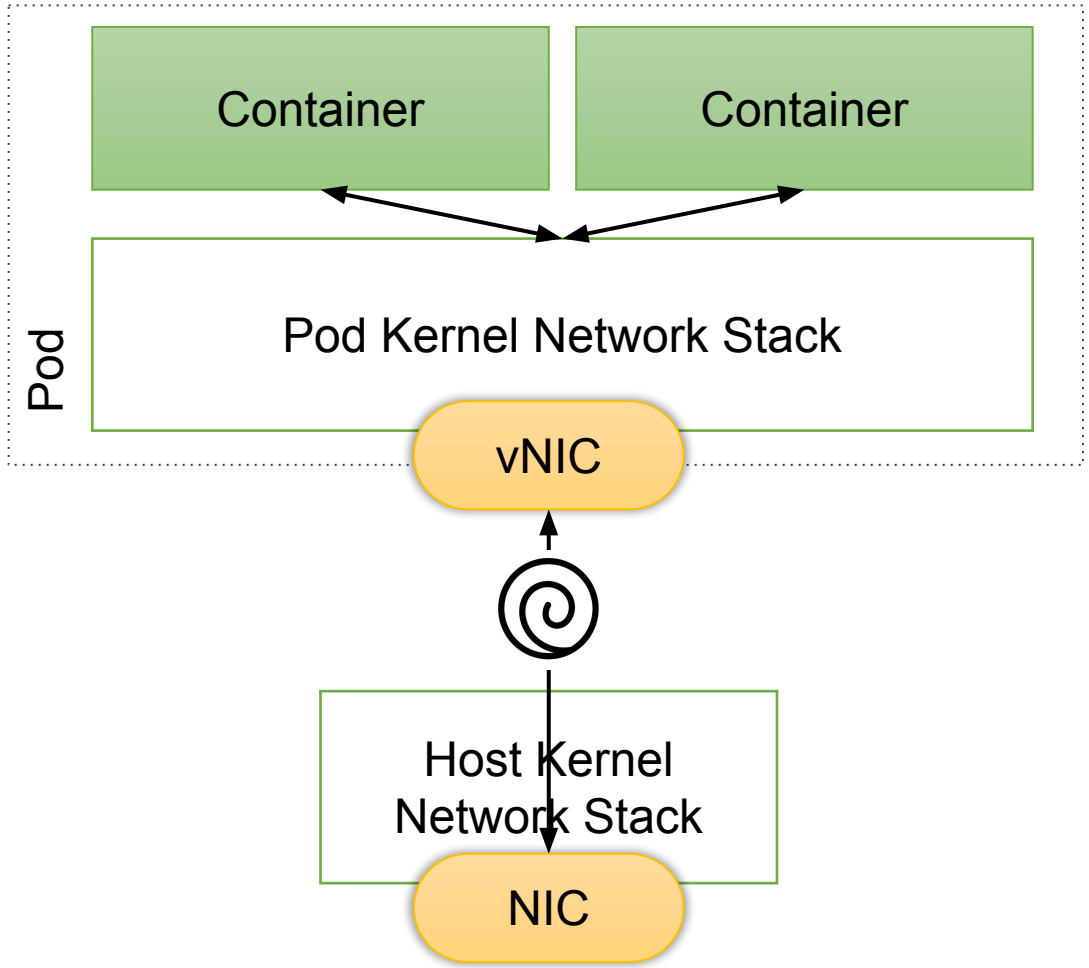
- An instance of the pod runs on each node
- System daemons and background processes

# Kubernetes & CNI

In K8s, pods are deployment unit which is created, destroyed and scaled dynamically.

Therefore to provide networking to this dynamic components K8s enforces the following network model:

- All Pods should be IP addressable and should not require any NAT.
- All agents should be able to communicate with all pods on the host.

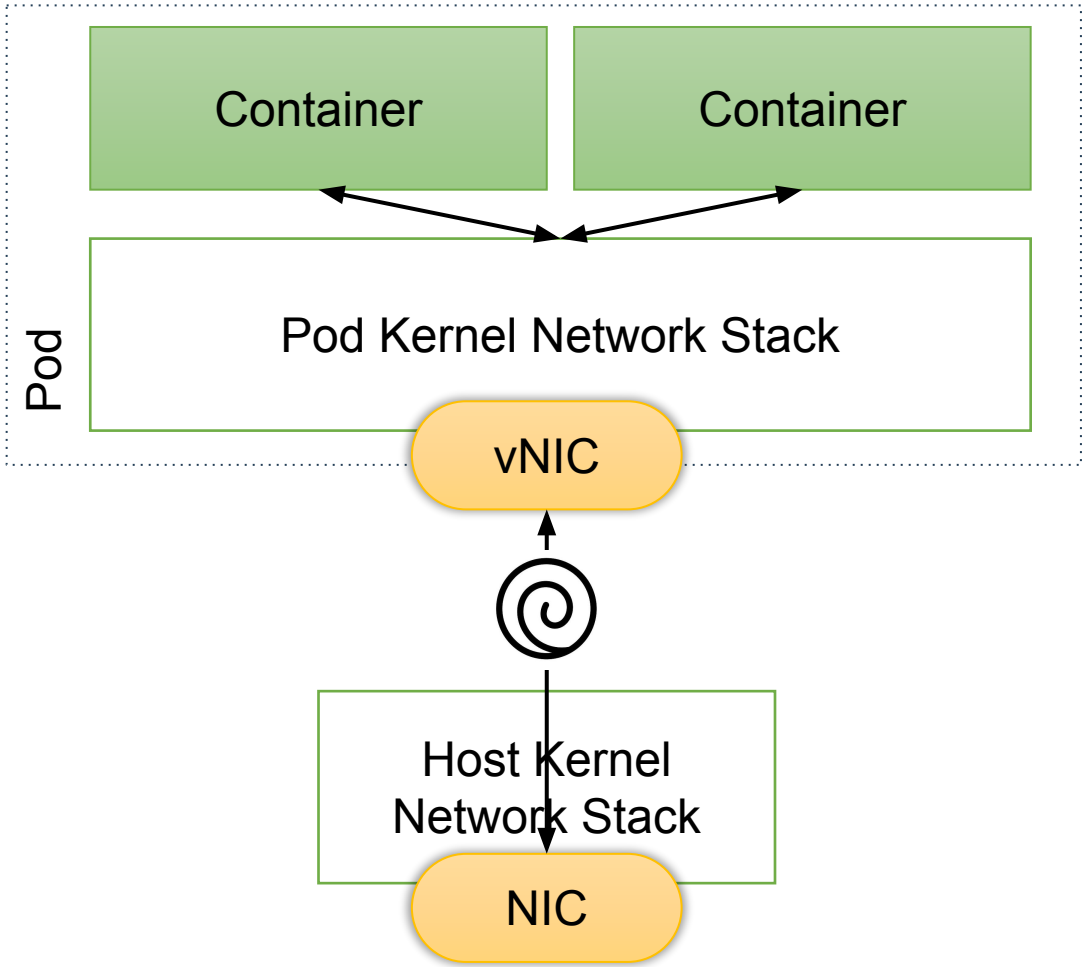


# Kubernetes & CNI

The K8s network model is enforced by CNI.

It uses some combination of inter-host and intra-host communication techniques.

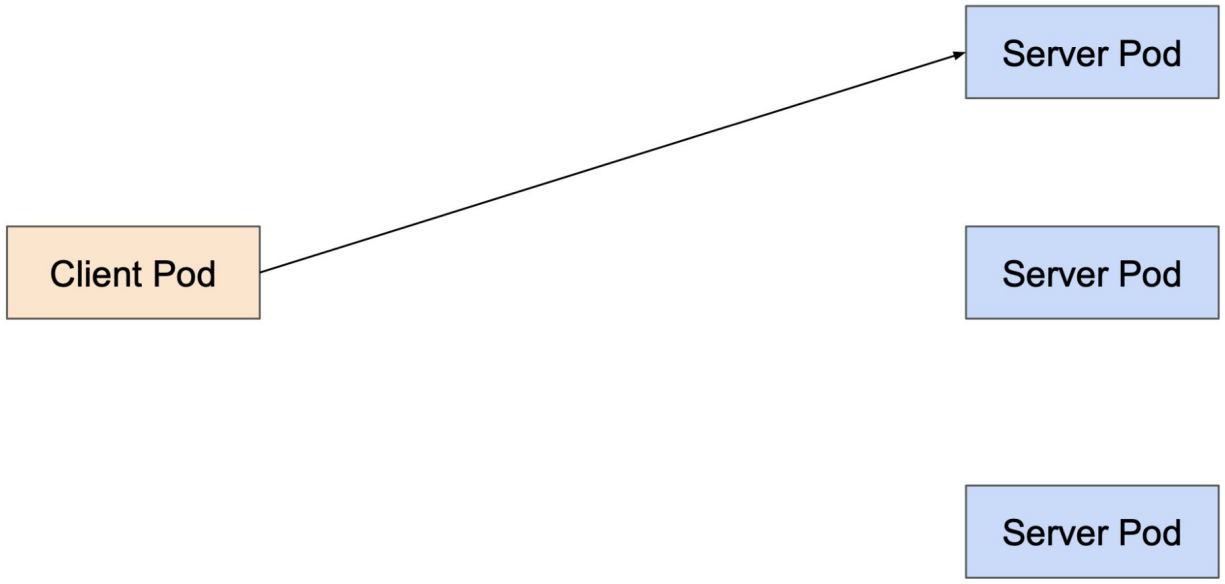
For e.g. Calico uses Layer 3 routing for intra-host communication and Overlay/Underlay for inter-host communication.



# Kubernetes - Services

Service:

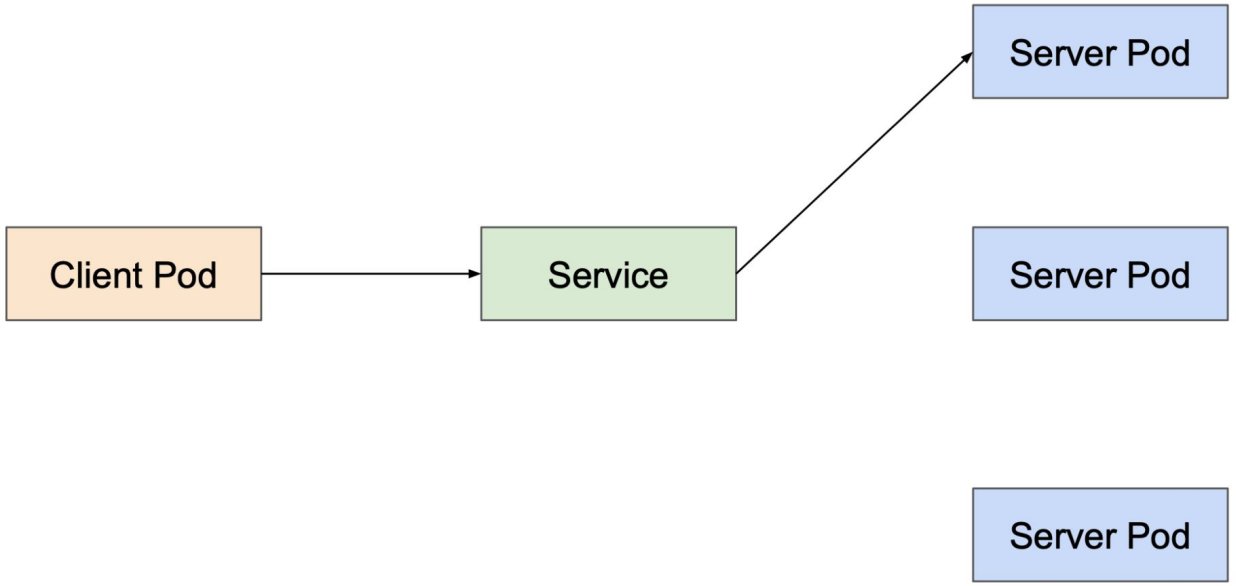
- Network abstraction for a pod - both for within cluster access and outside cluster access.
- Assign a single virtual IP address for a set of pods



# Kubernetes - Services

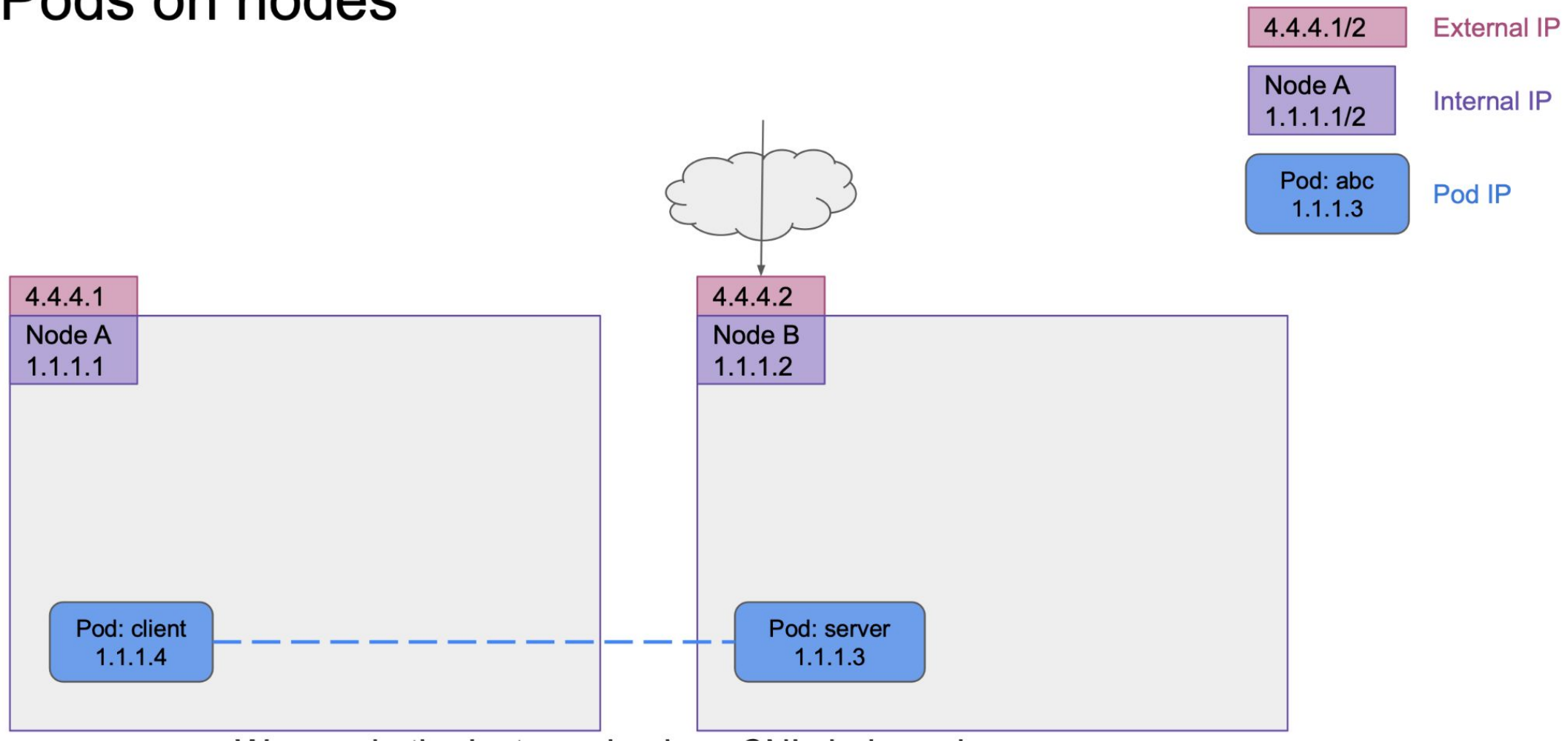
Service:

- Network abstraction for a pod - both for within cluster access and outside cluster access.
- Assign a single virtual IP address for a set of pods



# Kubernetes - Services

## Pods on nodes






[image taken from ACM India Winter School on "Full-stack Networking (FSN)" ]

# Kubernetes - Accessing Services

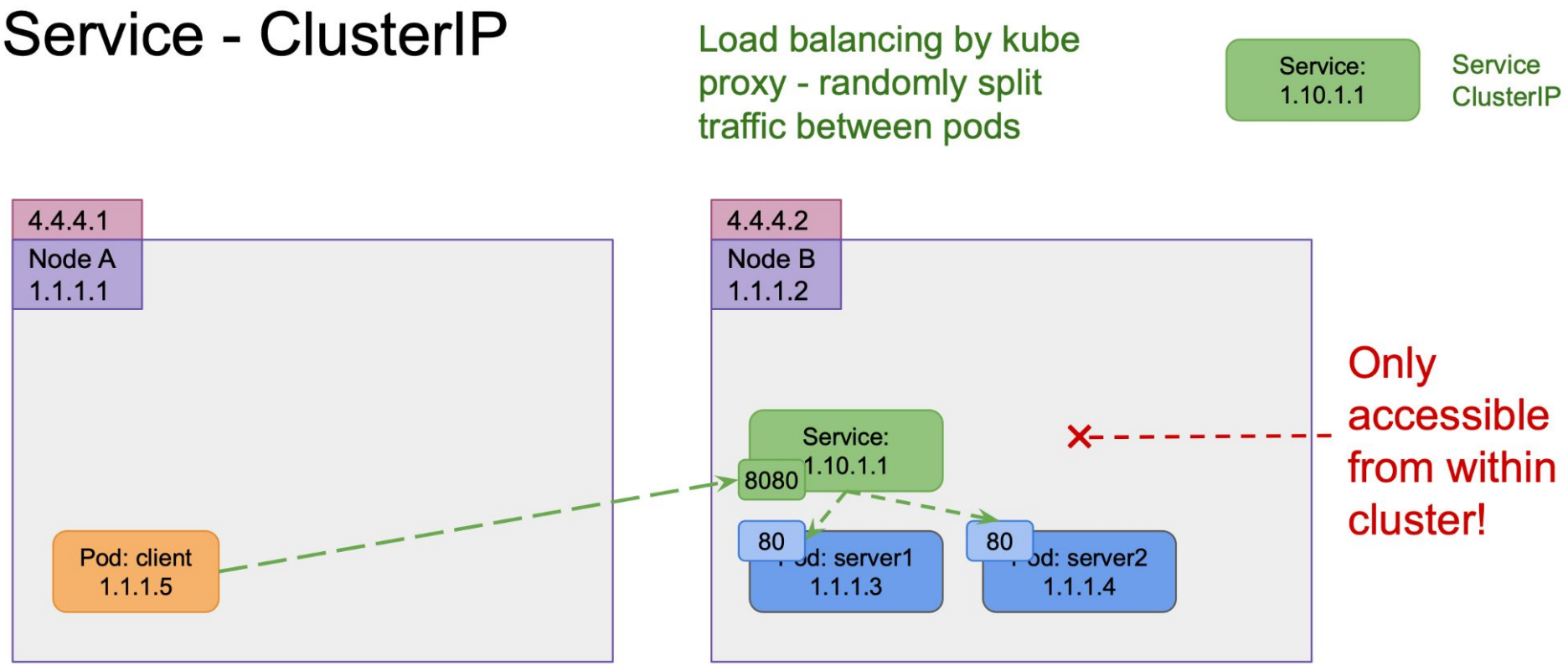
3 ways of doing this:

- ClusterIP:
  - Assign a new IP for this service (which will remain constant, irrespective of the pods)
  - Only accessible from within the cluster
- NodePort:
  - Remember docker's port forwarding?
  - Assign a port on the worker node (so accessible by using the node's IP)
  - Pod for that service can run anywhere (not necessarily on that node)
- Load Balancer:
  - Use external load balancer (accessible from anywhere on the internet)
  - Usually used with cloud provider LBs

# Kubernetes - Accessing Services

## Service - ClusterIP

Load balancing by kube proxy - randomly split traffic between pods



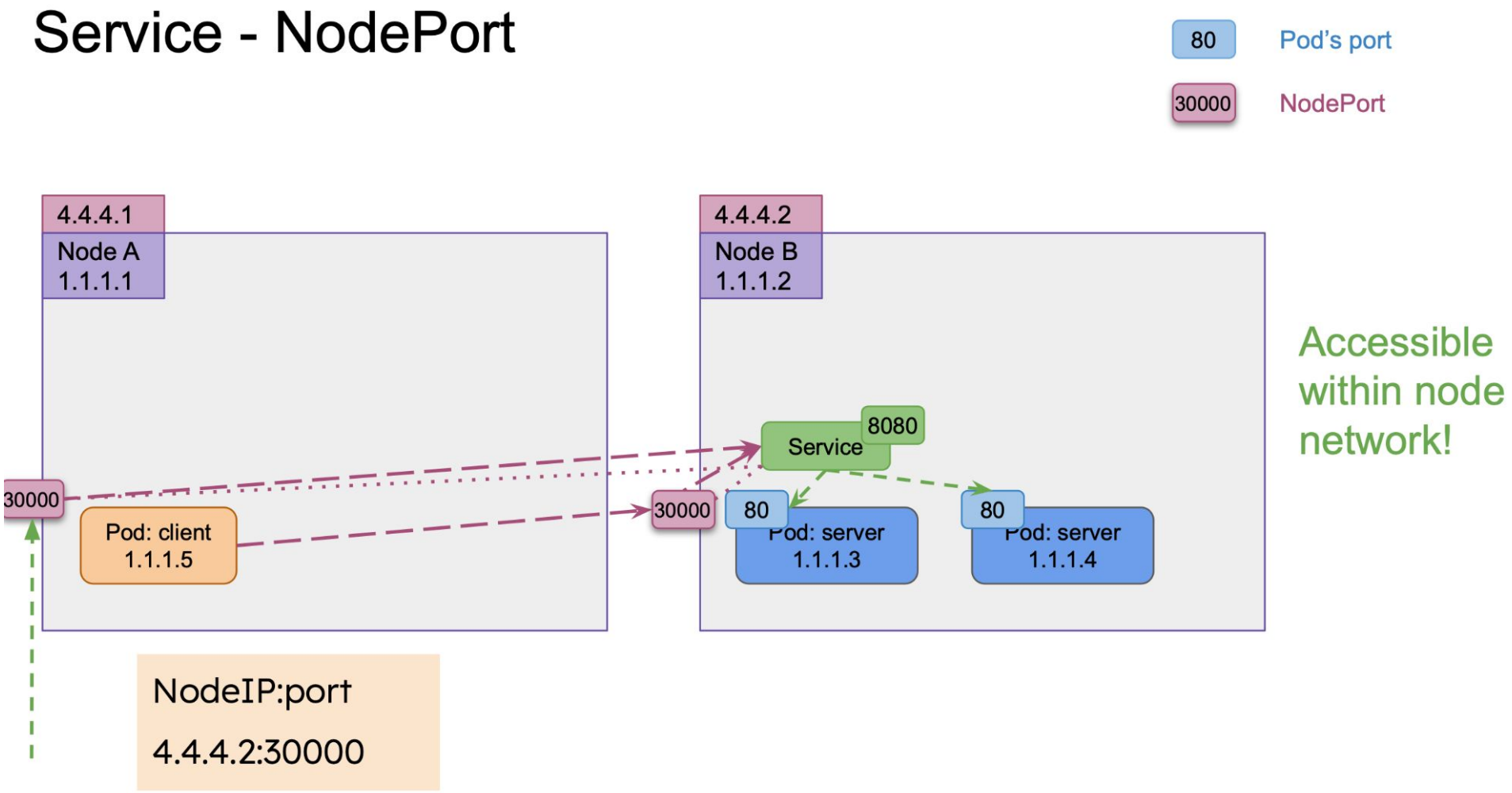

clusterIP:port / servicename:port (using DNS)

1.10.1.1:8080 / myservice:8080



[image taken from ACM India Winter School on "Full-stack Networking (FSN)" ]

# Kubernetes - Accessing Services



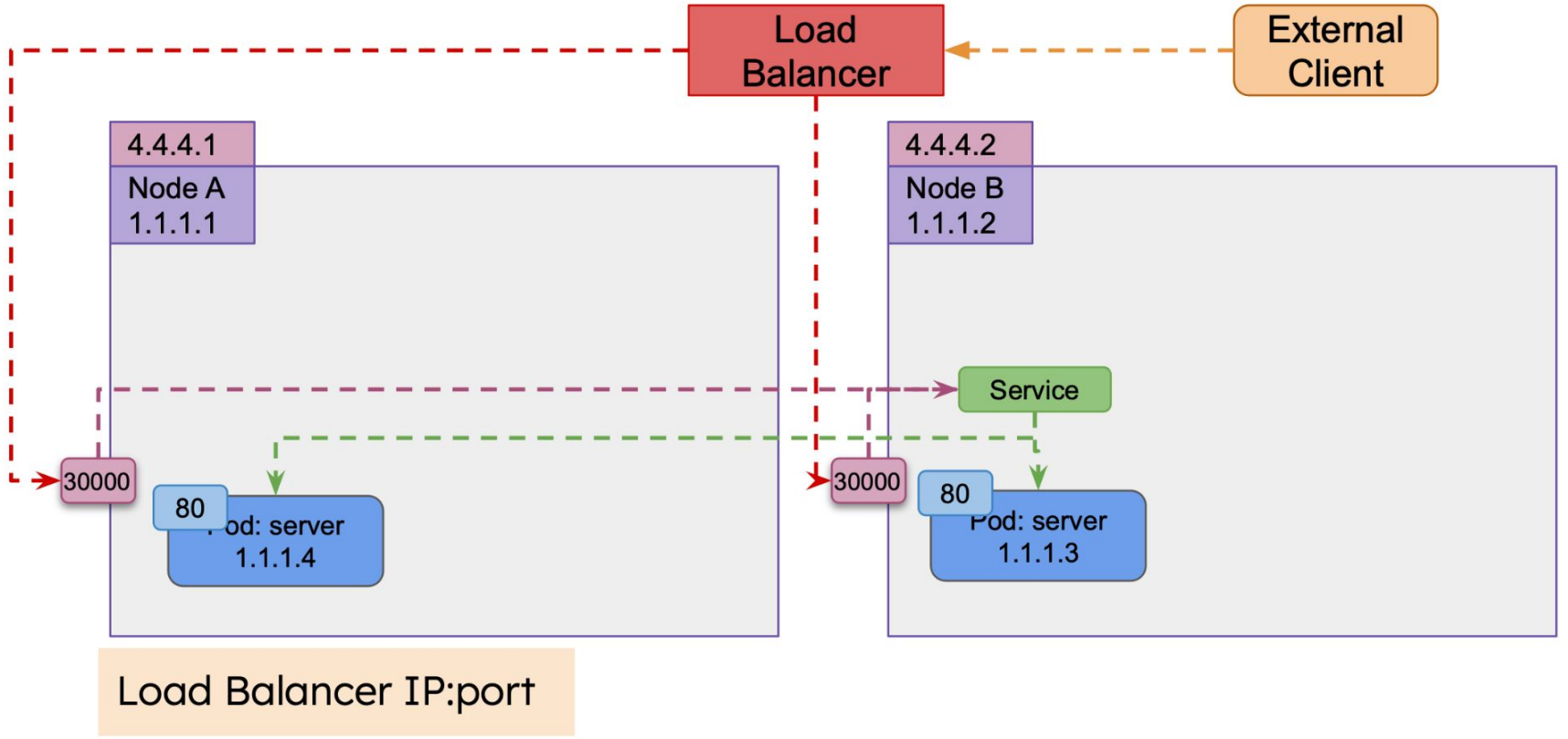



[image taken from ACM India Winter School on "Full-stack Networking (FSN)" ]

# Kubernetes - Accessing Services

## Service - Load balancer

Externally accessible (from the internet) using external load balancer






[image taken from ACM India Winter School on "Full-stack Networking (FSN)" ]

# Kubernetes - Cluster IP hands on

```
$ kubectl apply -f service-deployment.yaml
$ kubectl get pods -o wide
```

Edit service-client.yaml to have a pod-ip

```
$ kubectl apply -f service-client.yaml
$ kubectl logs client-pod
Hello from server-deployment-6cccf9bf66-694gq
Hello from server-deployment-6cccf9bf66-694gq
Hello from server-deployment-6cccf9bf66-694gq

$ kubectl delete -f service-client.yaml
$ kubectl apply -f service-clusterip.yaml
$ kubectl get service
```

Edit service-client.yaml to have a service-ip and port

# Kubernetes - Cluster IP hands on

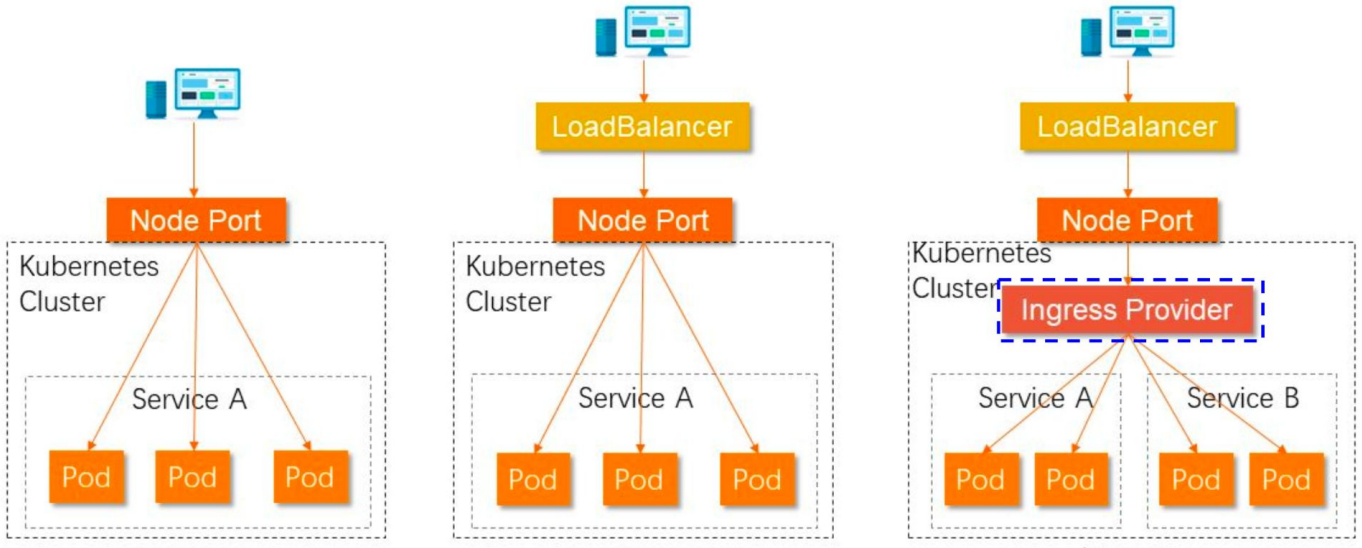
```
$ kubectl apply -f service-client.yaml
Hello from server-deployment-6cccf9bf66-694gq
Hello from server-deployment-6cccf9bf66-rp99m
Hello from server-deployment-6cccf9bf66-694gq
Hello from server-deployment-6cccf9bf66-rp99m

$ kubectl describe service myservice
```

# Kubernetes - Nodeport hands on

Change the value of the port to something different so that it doesn't match your classmates'

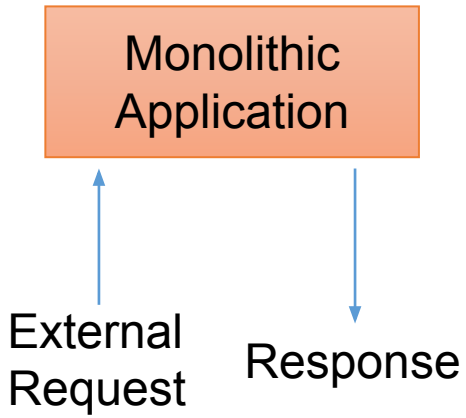
```
$ kubectl apply -f service-nodeport.yaml
$ kubectl get nodes -o wide

curl <Node Internal IP>:<Node Port>/
```

Nodeport is usually used along with load balancer and ingress

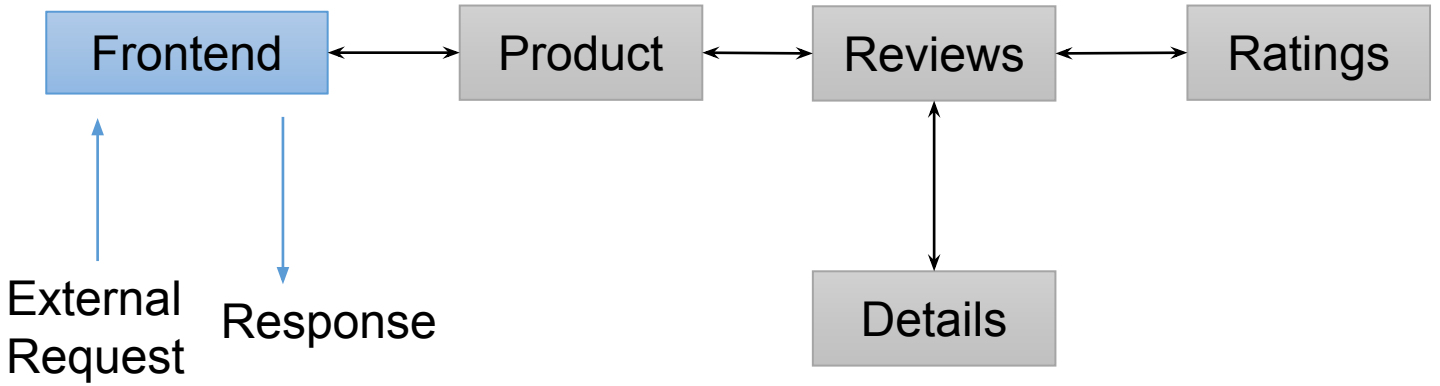



[image credit: https://www.alibabacloud.com/blog/an-implementation-practice-of-kubernetes-ingress-gateway-concept-deployment-and-optimization_598905]

# Use Case: Service Mesh

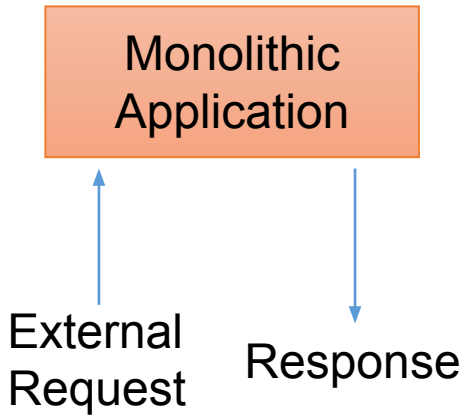


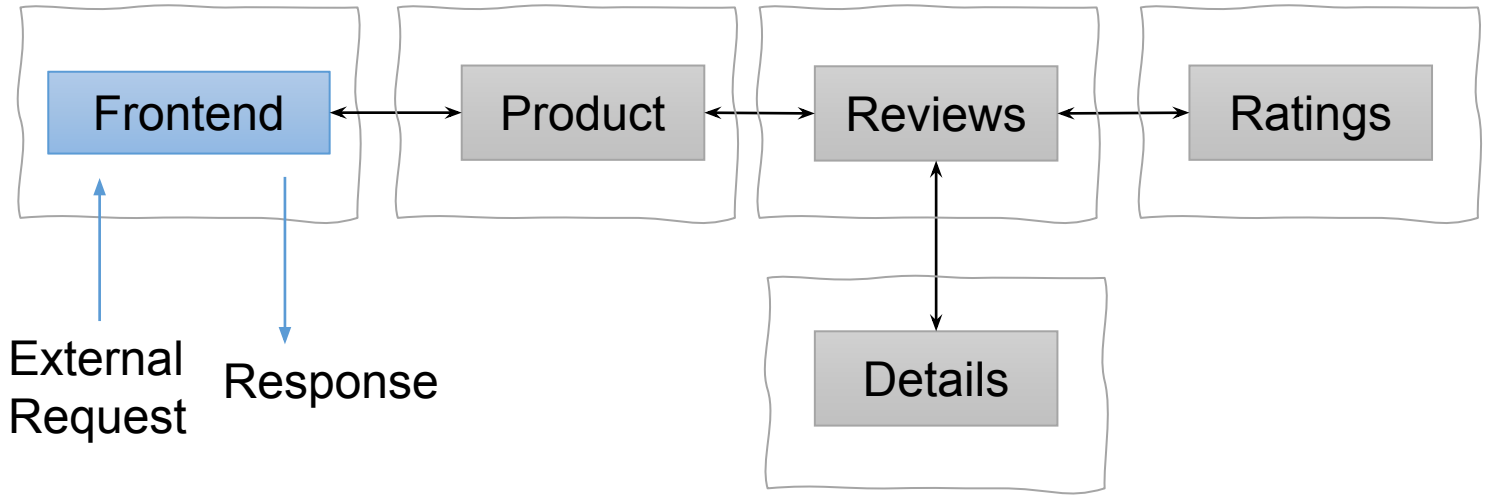

Applications deployed using monolithic architecture




Applications deployed using microservices architecture

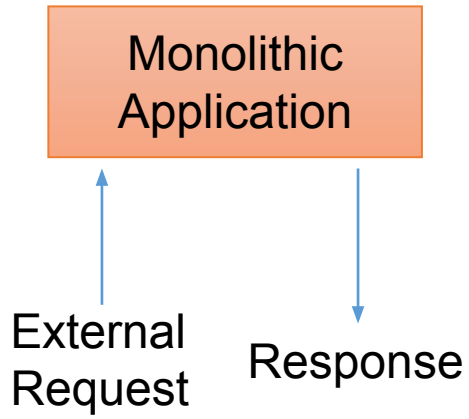
# Use Case: Service Mesh

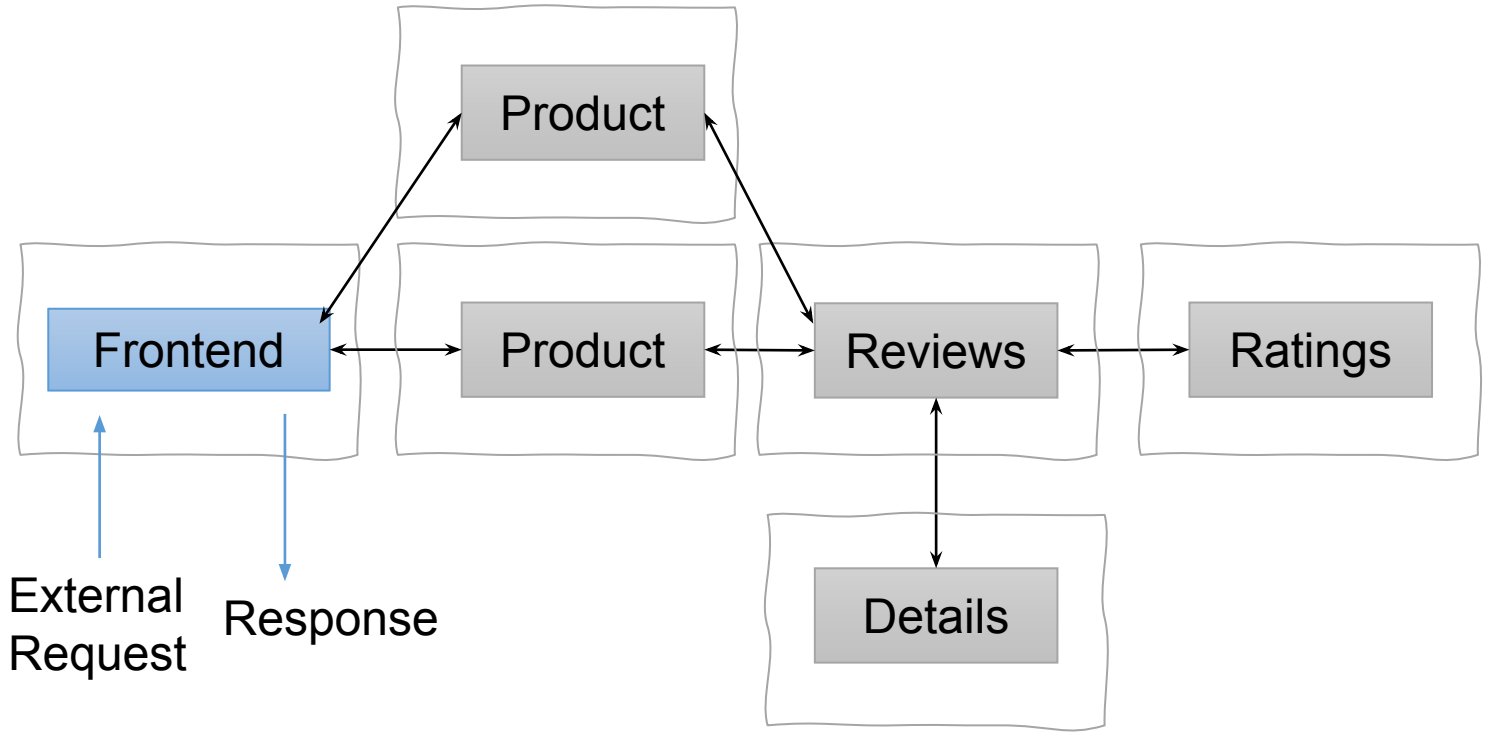



Benefits:
- Easy application management
- Fault-tolerance
- Scalability
- Micro-service reuse, etc.

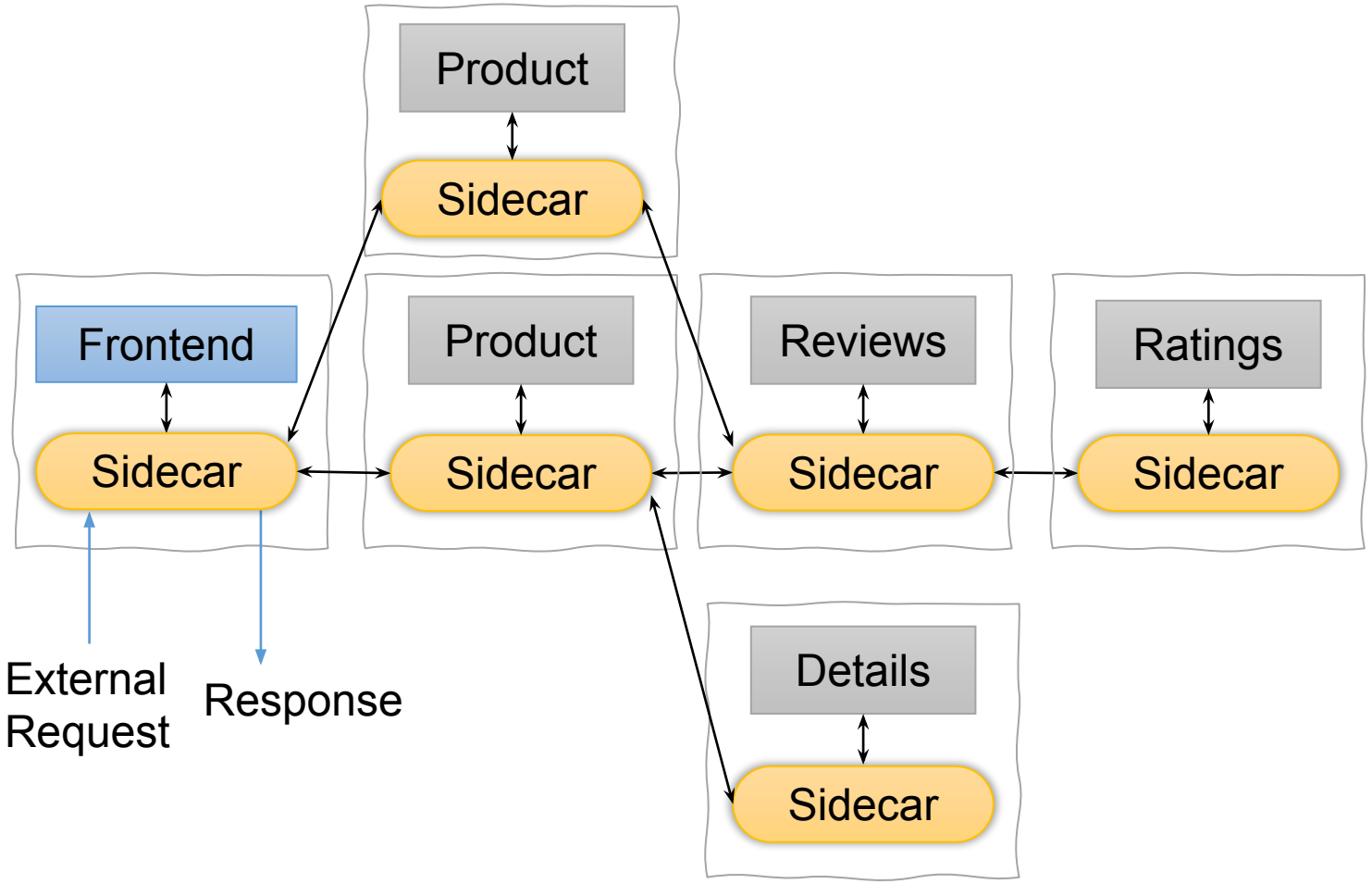
# Use Case: Service Mesh



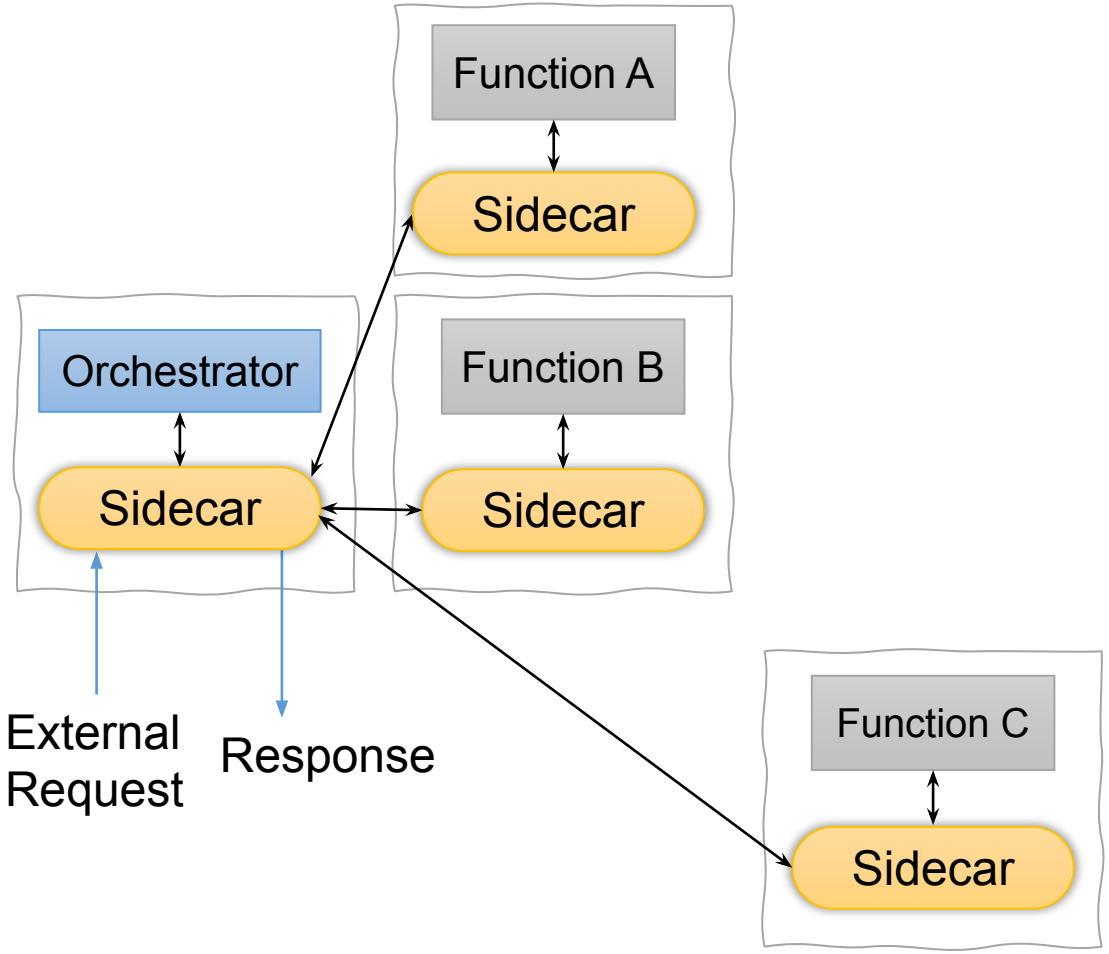

How to:
- discover services?
- setup communication?
- monitor communication and failures?

# Use Case: Service Mesh




Benefits:
- Connect different loosely coupled services.
- Provide other functions like rate limiting, load balancing, etc.

# Use Case: Serverless and FaaS

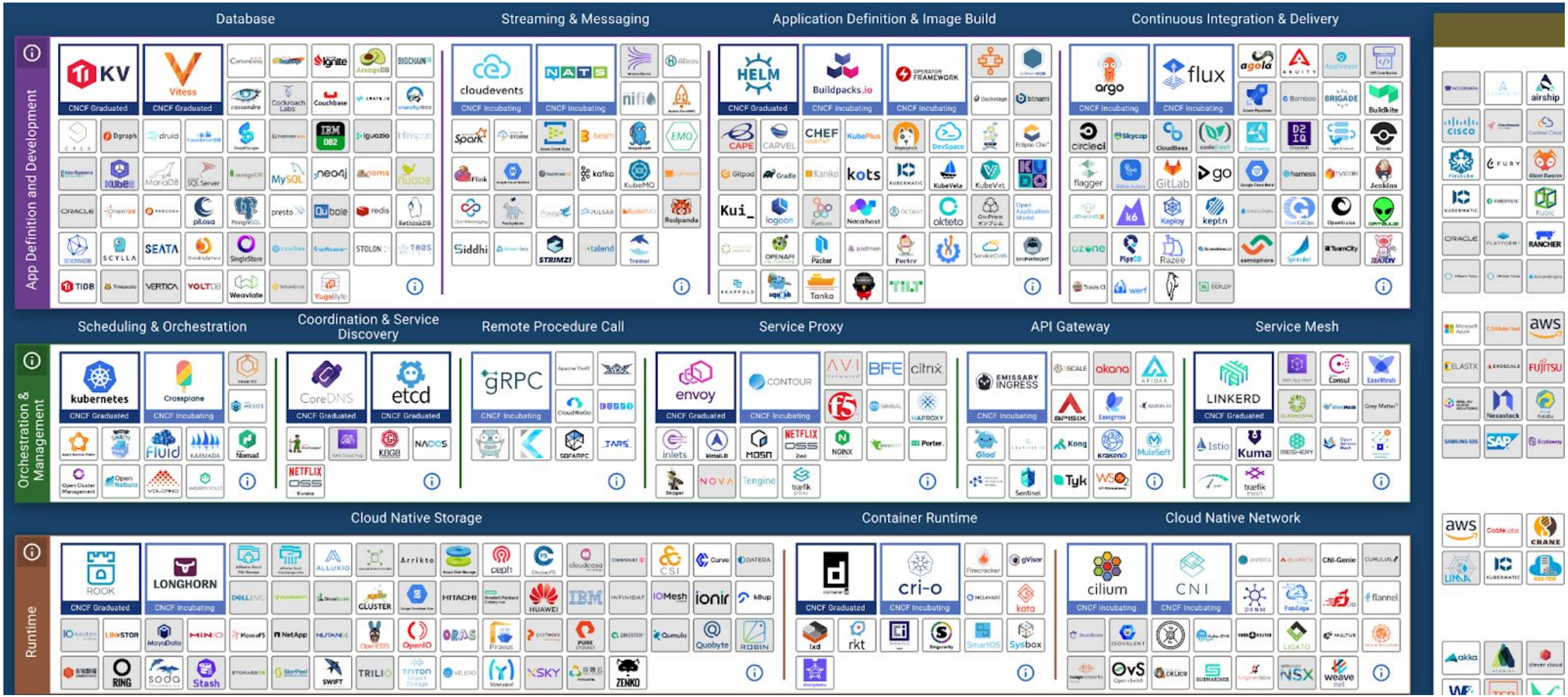



Benefits:
- Event driven.
- You don't need to think about the infrastructure at all.

# Cloud and beyond

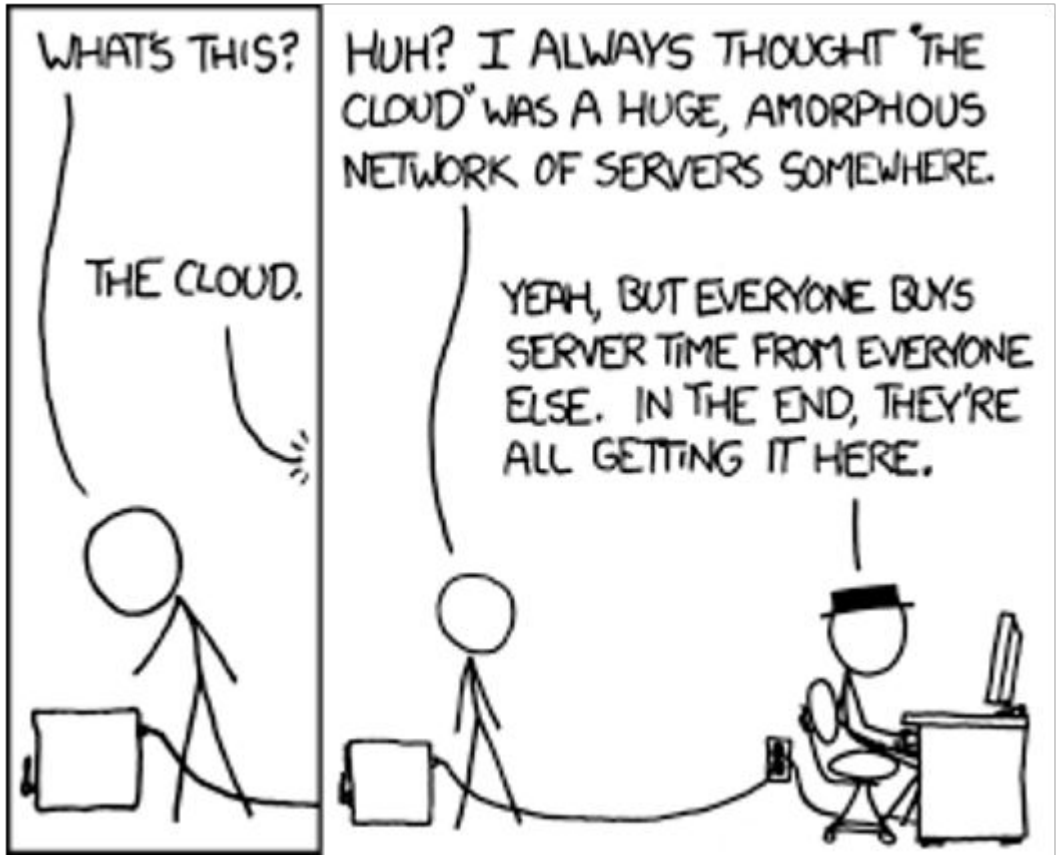

[image credit: https://github.com/cncf/trailmap/blob/master/CNCF_TrailMap_latest.png]

# Cloud and beyond




[image credit:https://landscape.cncf.io/]

# Cloud and beyond




[image credit:https://xkcd.com/908/]

Thank You!!