Kubernetes

Principles of Cloud Computing

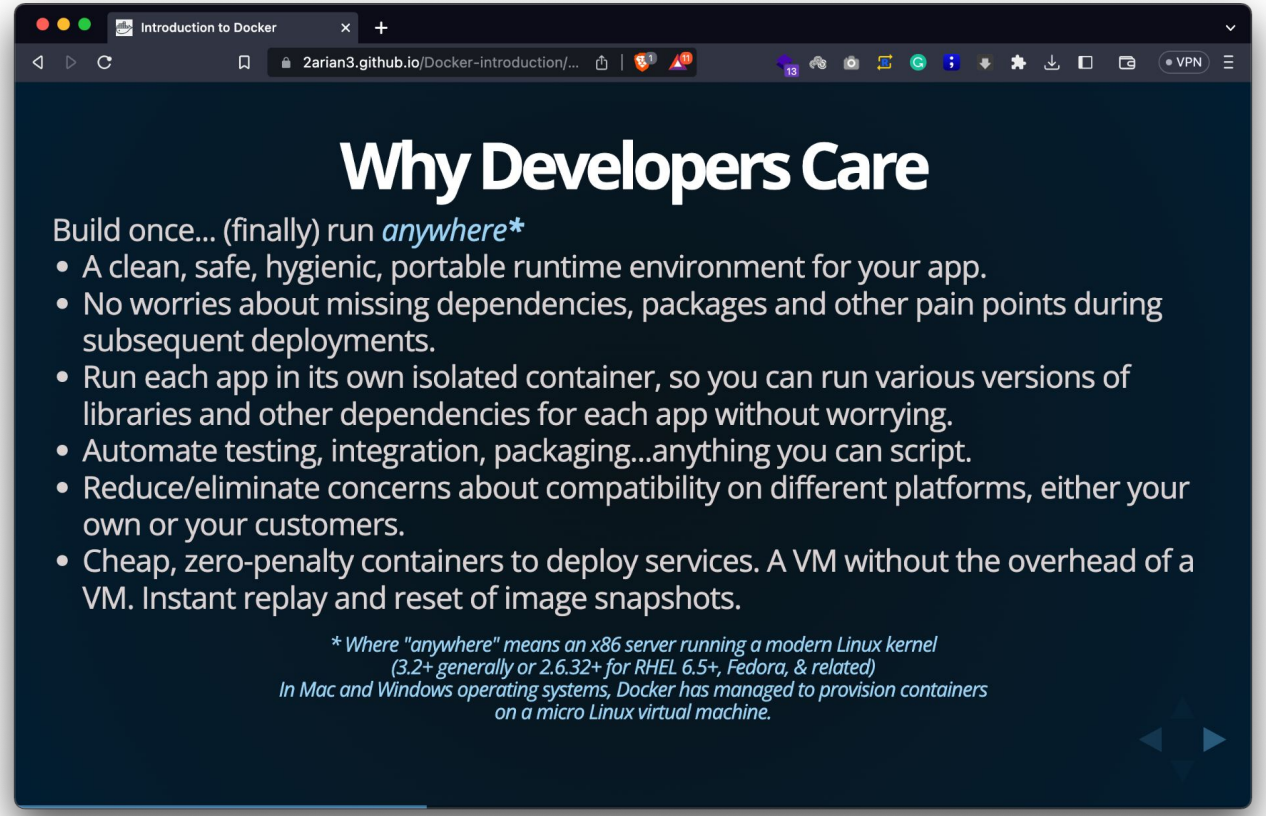
Spring 2023 - Instructed by Dr. A. Javadi

Amirkabir University of Technology
(Tehran Polytechnic)

Outline

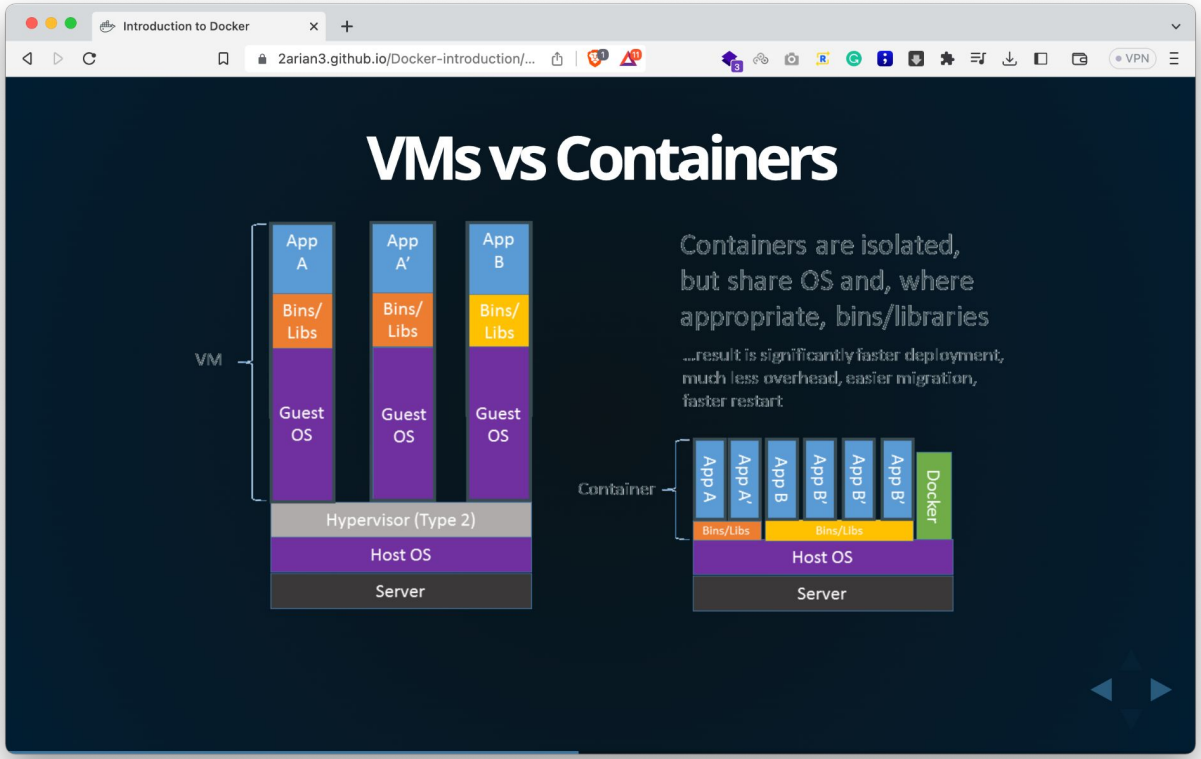
- Review of Containerization
- Monolithic Applications and their Drawbacks
- Microservices Operational Challenges
- Container Orchestrations
- Orchestration Tools: Kubernetes
- Kubernetes from a Developer Perspective
- Kubernetes Resources
- Kubernetes Internal Architecture

Review of Containerization



Review of Containerization

Review of Containerization



Monolithic Applications

- Monolithic
  - A collection of tightly coupled components that have to be developed, deployed and managed as one entity.

- What are the challenges for a monolith application?
  - Let’s take an example.

Authentication

CPU: 2
memory: 2Gi

Post Submission

CPU: 2
memory: 4Gi

Example: A Classified Ads website

Chat

CPU: 4
memory: 8Gi

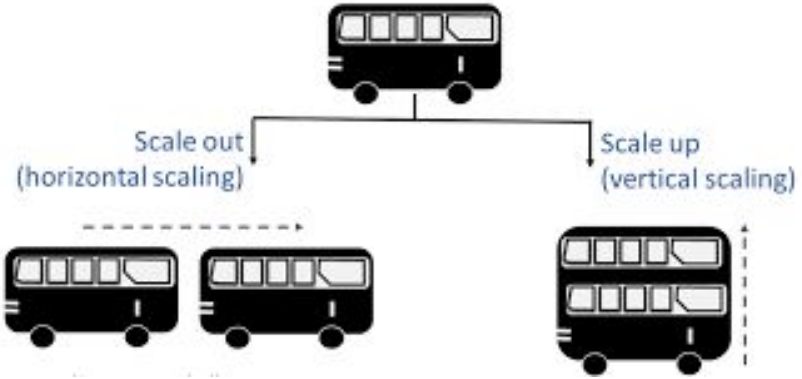
Recommenation

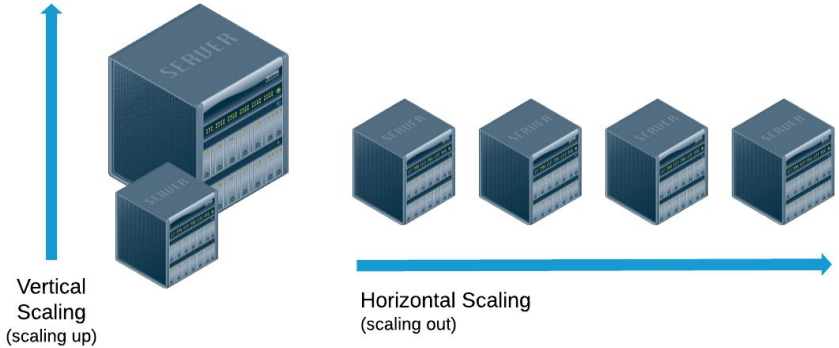
CPU: 2
memory: 4Gi

Post View

CPU: 2
memory: 4Gi

Scaling


Scale out
(horizontal scaling)
Scale up
(vertical scaling)


Vertical
Scaling
(scaling up)
Horizontal Scaling
(scaling out)

Monolithic

- Monolithic
  - A collection of tightly coupled components that have to be developed, deployed and managed as one entity.
- What are the challenges for a monolith application?
  - Components need different
    - Software requirements: Programming Language, Libraries and versions
    - Hardware requirements: CPU, GPU, Memory and Disk
  - Separation of Concerns: Tightly coupled modules, added complexity
  - Operational challenges
    - Maintenance: Having a single point of failure
    - Scaling: Scaling up is easy but scaling out requires major code changes. If a single part of an application is unscalable then the whole application becomes unscalable.
    - Monitoring

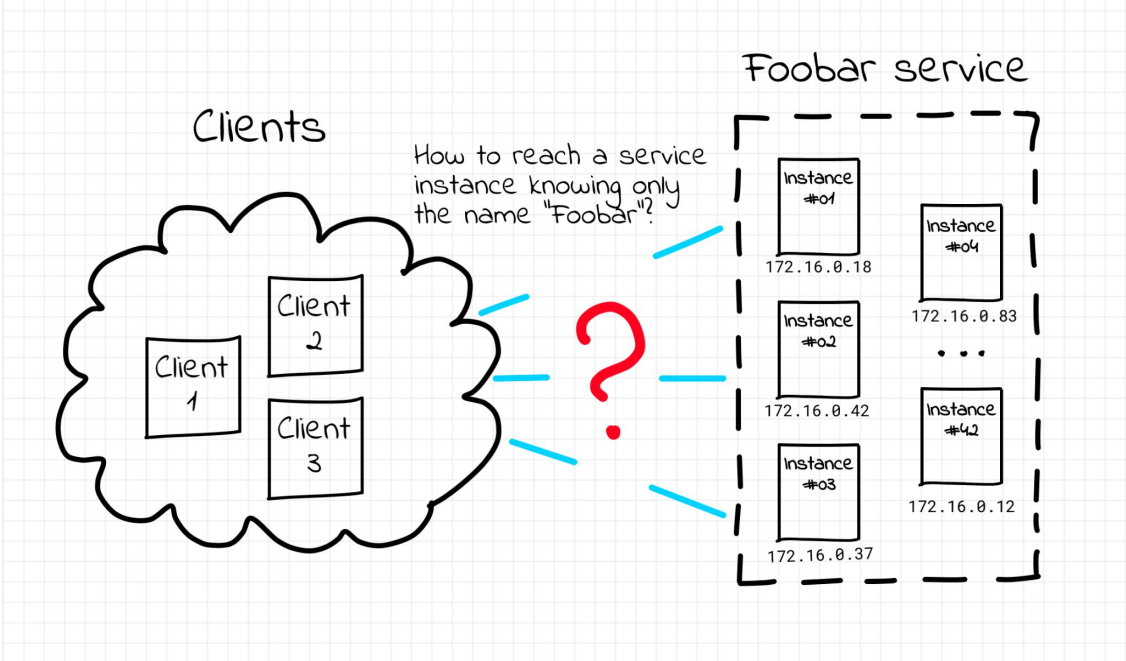
Microservices: From components to services.

- Advantages
  - Different Software and Hardware requirements can be met.
  - Separation of concerns is achieved.
  - **Operational Challenges are *different* now.**

- Challenges
  - Resource allocation is not trivial anymore.
    - Deploy
    - Scale
  - Networking is now needed.
    - Service discovery (How can services find each other?)
    - Security
  - Monitoring
    - and error recovery

Microservices: Service Discovery

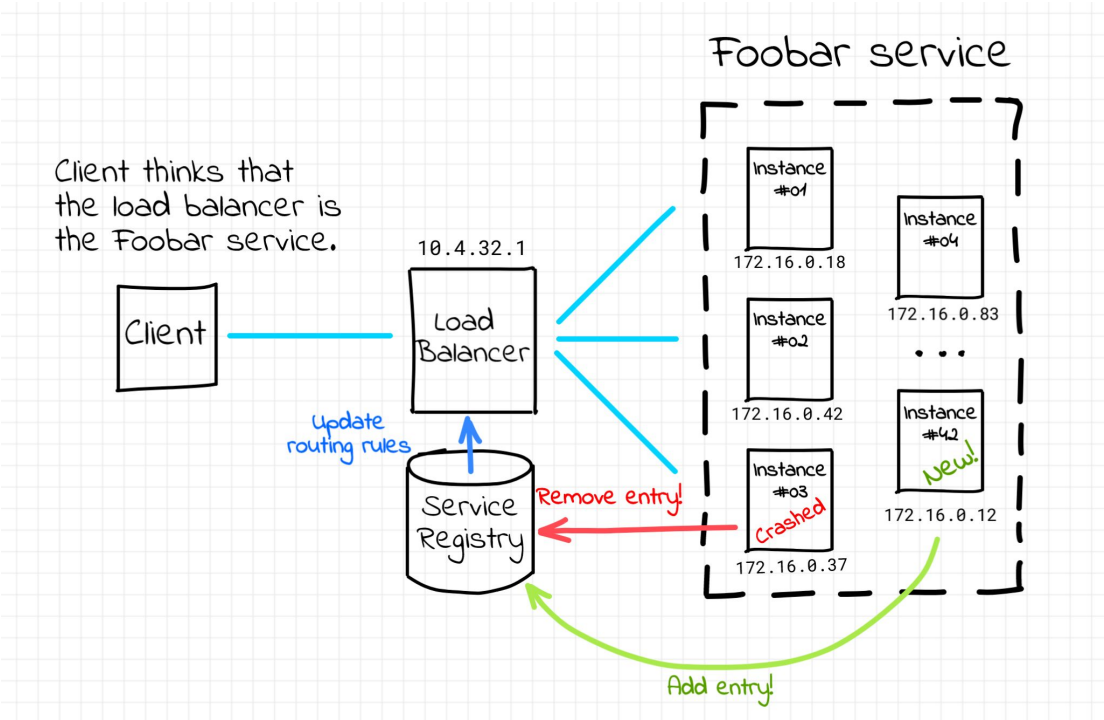
- Microservices communicate through:
  - Synchronous protocols such as REST API or gRPC
  - Asynchronous protocols such as AMQP & MQTT
- Service Discovery: How can services find each other?



Service Discovery Strategies: Server-Side

- A Load Balancer distributes the requests.
- Instances register themselves in service registry.

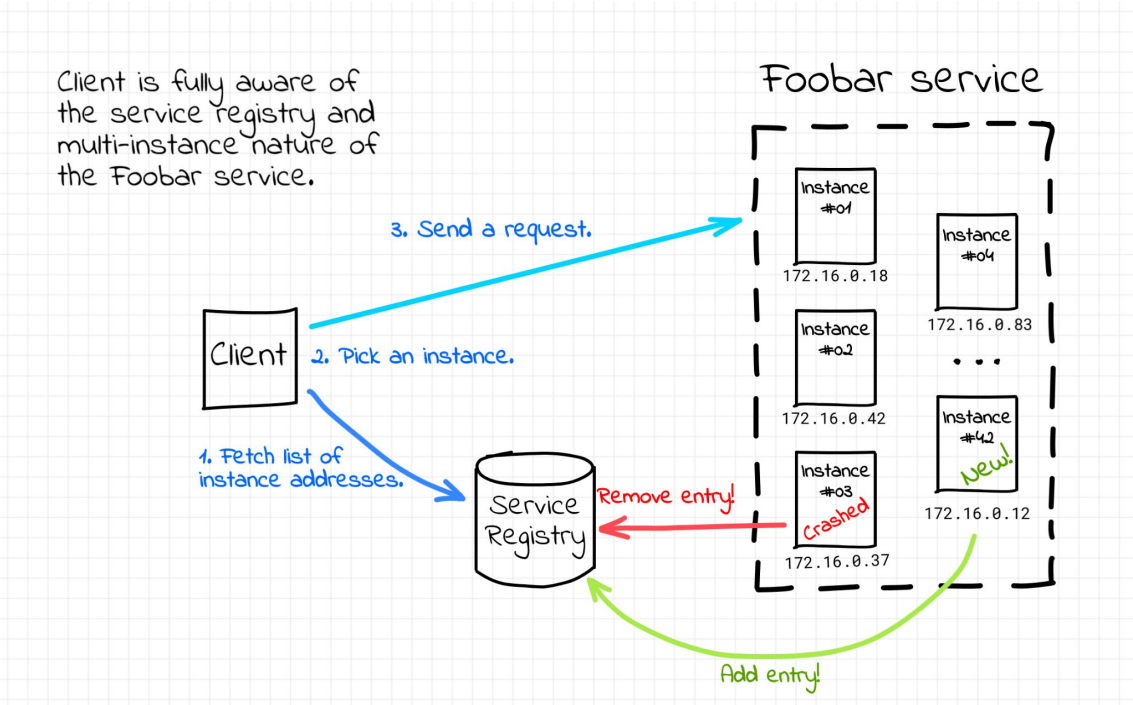
- Challenges of Load balancer:
  - Single point of Failure - bottleneck
  - Should support all communication protocols
  - Extra Network Hop



Service Discovery Strategies: Client-Side

- Advantages:
  - No single point of failure
  - No bottleneck
  - No extra hop

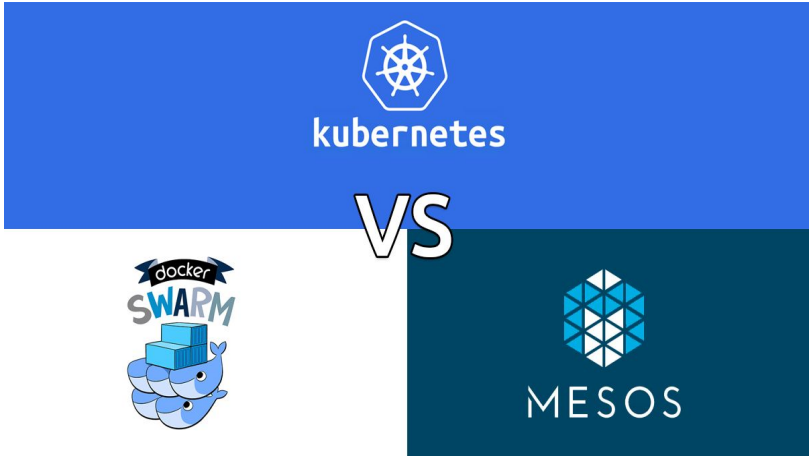
- Challenges:
  - couples clients with the service registry
  - Complicates the client: requires integration code to be written for any programming language used.



Microservices Operational Challenges Solution: Orchestration Tools

- High Availability (No Down Time)
  - Update containers without bringing down the application
  - Replace problematic servers on demand
- Scalability (High Performance)
  - To Scale out and scale up on demand, and even automatic.
  - Smart Resource Allocation
- Disaster Recovery (Backup and restore)
- Easy Networking
  - Advanced network between containers in different hosts
  - Cluster Security

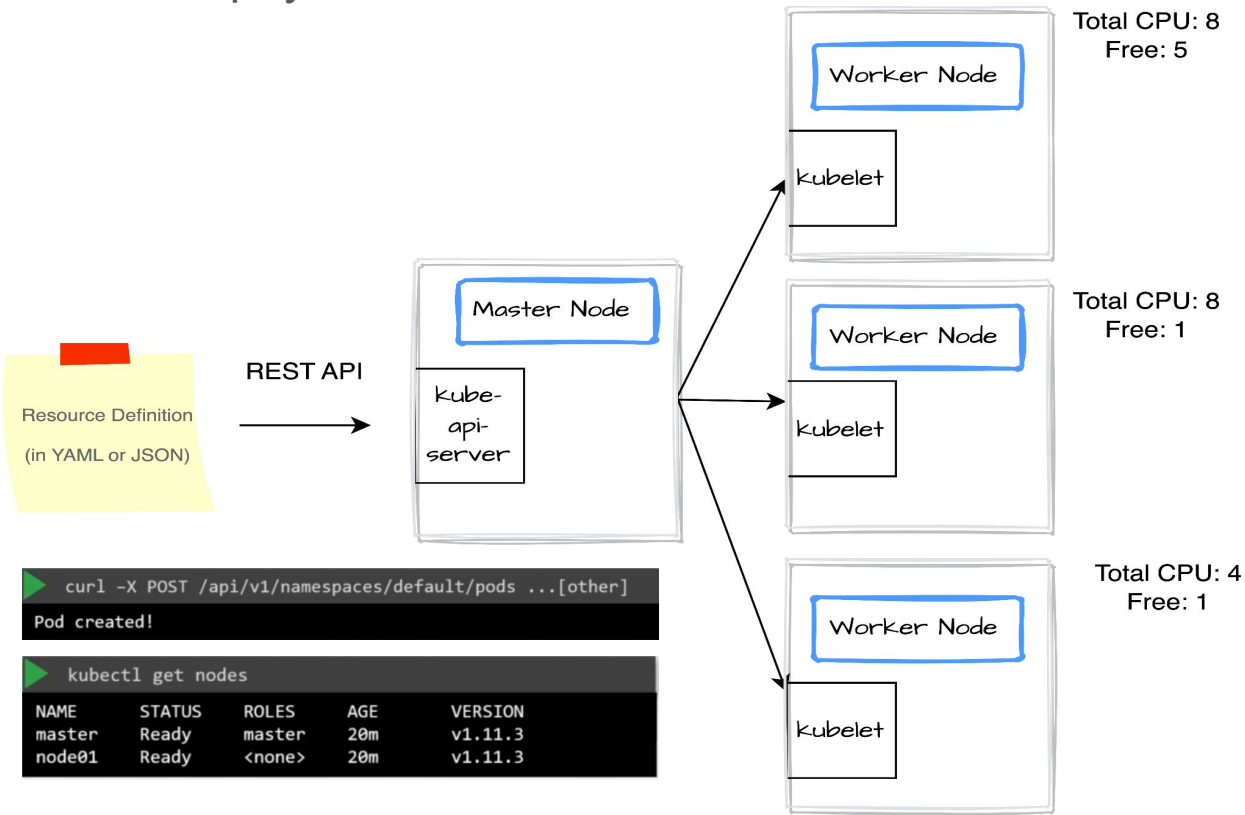
Orchestration Tools: Kubernetes



- Born in Google
  - Donated to CNCF in 2014 (open source)
  - https://github.com/kubernetes/kubernetes
- Written in Go/Golang
- More than 4,000,000 lines!
- Meaning Helmsman
- Often shortened to k8s
- Used in Microsoft Azure, Amazon AWS, Google GCE as a standard.
- There are other orchestration tools out there.
  - Docker Swarm, Apache Mesos, Kubernetes

Kubernetes: Developers Perspective

- Kubernetes is a “Desired State Manager”.
- Node: A virtual or physical machine.



Kubernetes Resources: Pod

- Smallest deployable units of computing
- A group of one or more containers, with shared storage and network resources
  - *Why would we need more than one container? (Hint: Sidecar pattern)*

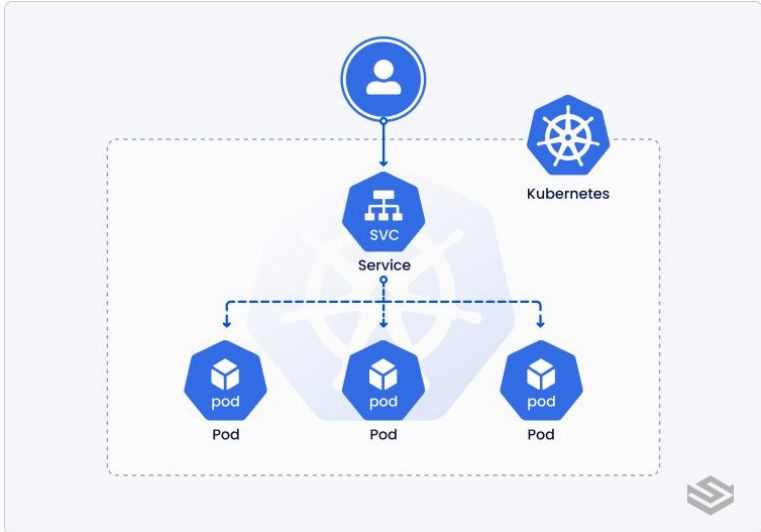
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

Kubernetes Resources: Configmaps and Secrets

- Many applications rely on configuration which is used during either application initialization or runtime, e.g. database credentials.
  - Non-Confidential: Configmap
  - Confidential: Secret
- They will help us avoid rebuilding our images every time we need to change the config.

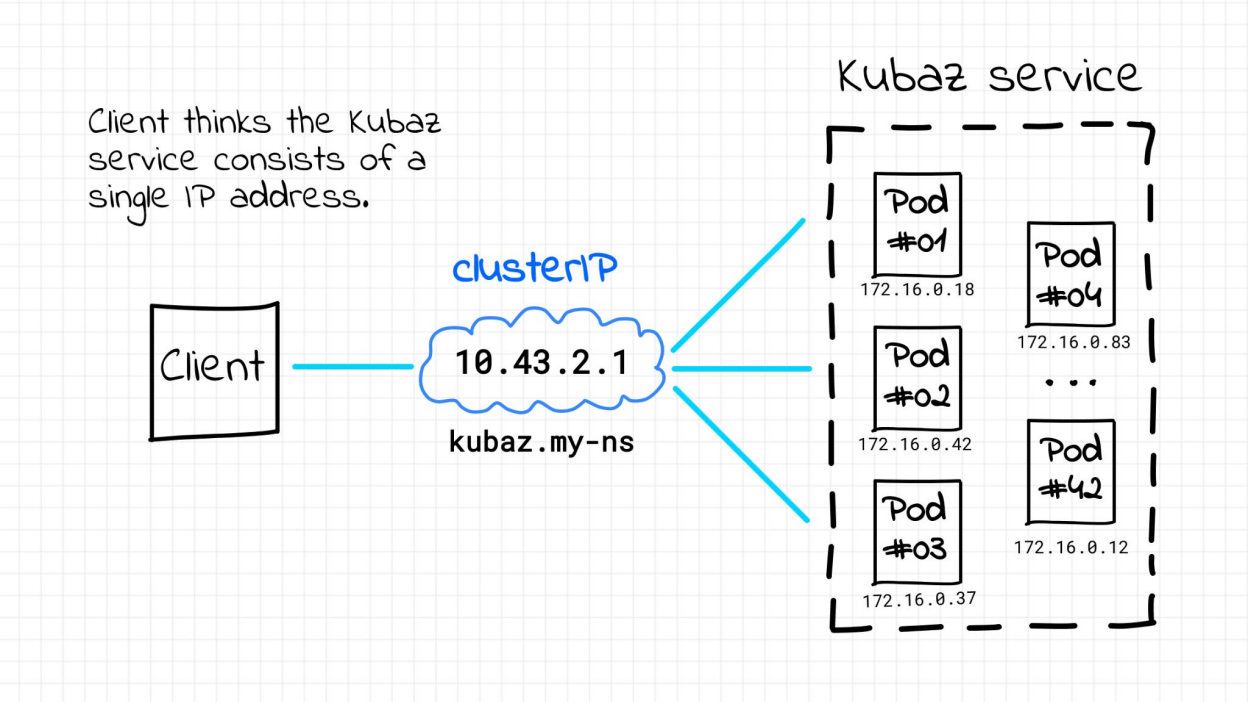
Kubernetes Resources: Service

- An abstraction which defines a logical set of Pods and a policy by which to access them
- Can we connect to the pods directly?




```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app.kubernetes.io/name: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Service Discovery in Kubernetes: Best of both worlds



Kubernetes Resources: Deployments & Replicasets

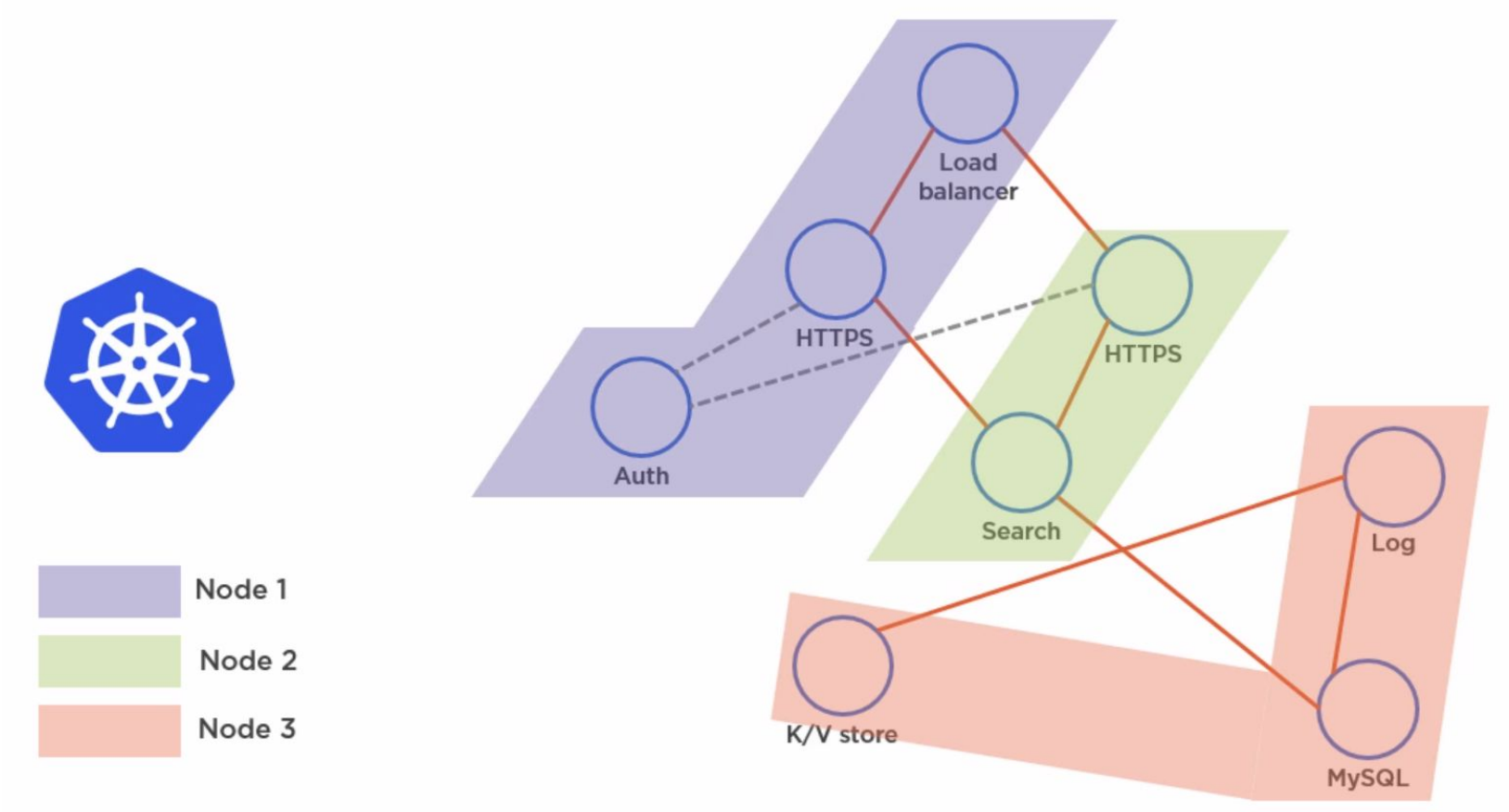
- Kubernetes pods are ephemeral by nature. They are disposable and replaceable.
- Kubernetes deployments are blueprints for our applications.
- They define a group of similar pods.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

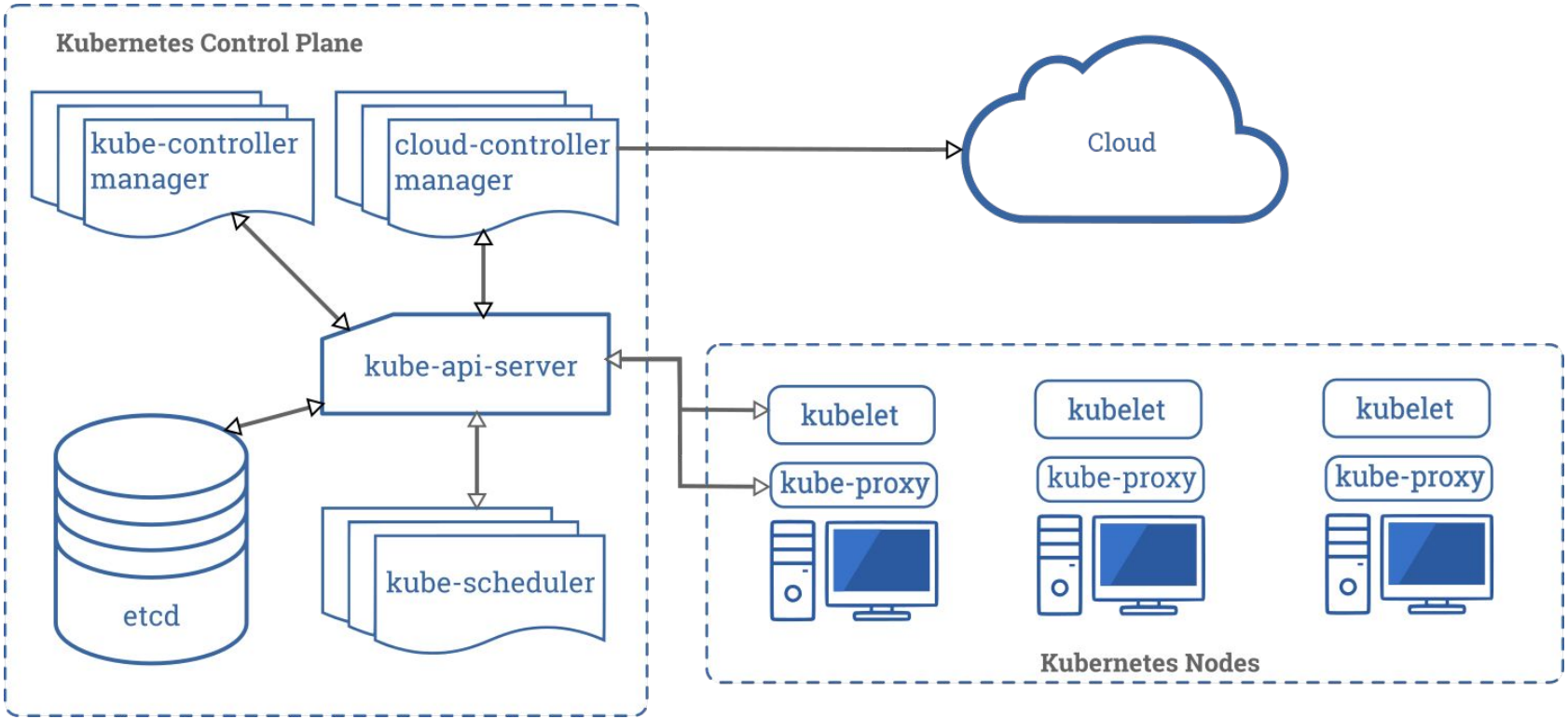
Kubernetes Resources: Others

- There are many other Kubernetes resources out there.
  - Statefulsets
  - Jobs and CronJobs
  - Persistent Volumes and Persistent Volume Claims
  - Custom Resources and Custom Resource Definitions
- You will get to know some of these in your future assignments.

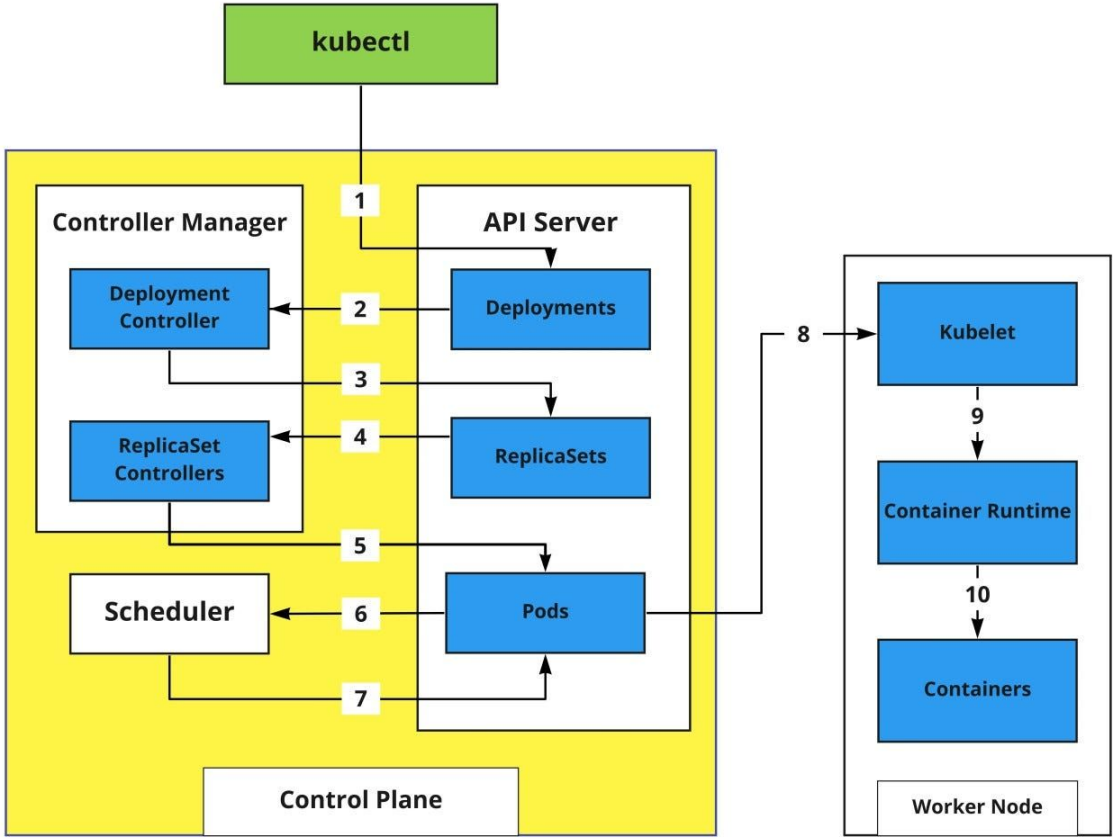
K8s Resources Mapping to Infrastructure



Kubernetes: Administrators Perspective



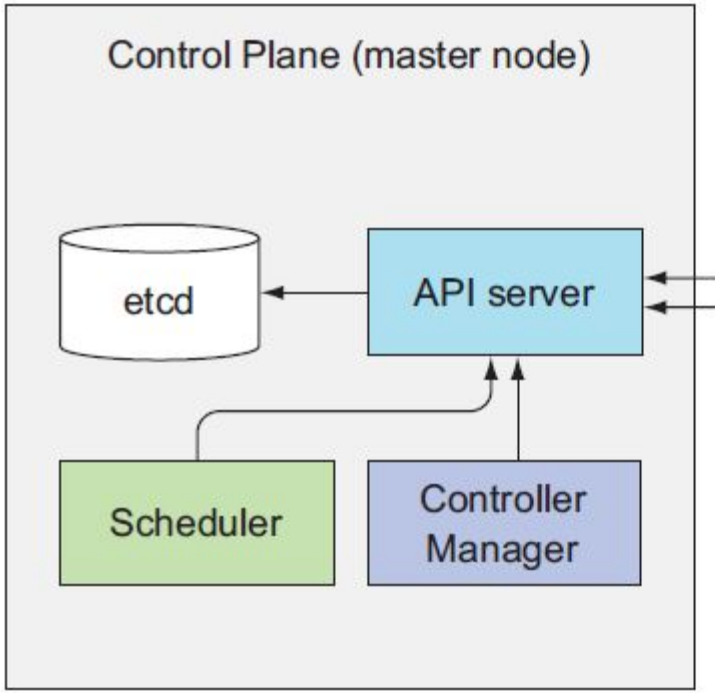
Pod Creation Chain of Events



api-server

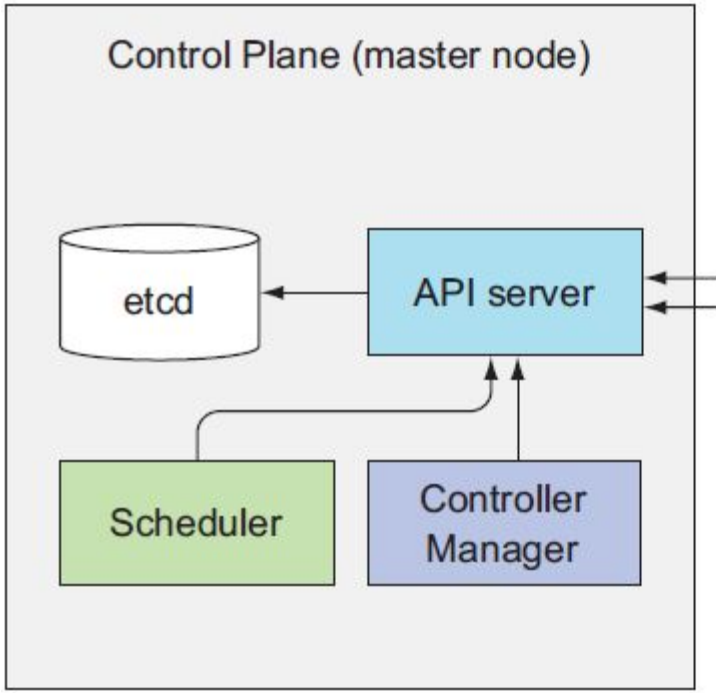
- Front-end to the control plane
- Exposes the API (REST)
- Consumes JSON
- The only component that interacts with etcd

- For each request:
  - Authenticates User
  - Validates Request
  - Retrieves Data
  - Updates ETCD



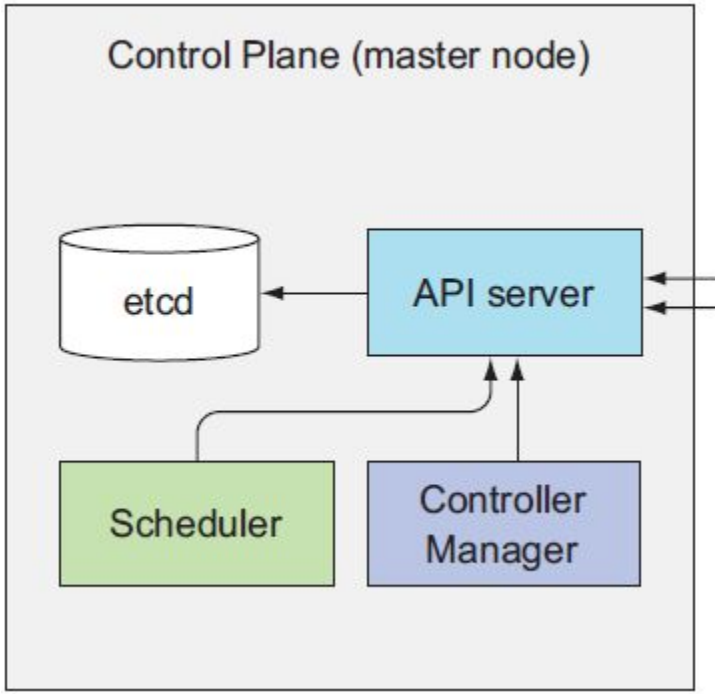
Cluster store

- Persistent storage
- Cluster state and config
- It uses etcd
  - etcd is a distributed, reliable key-value store for the most critical data of a distributed system

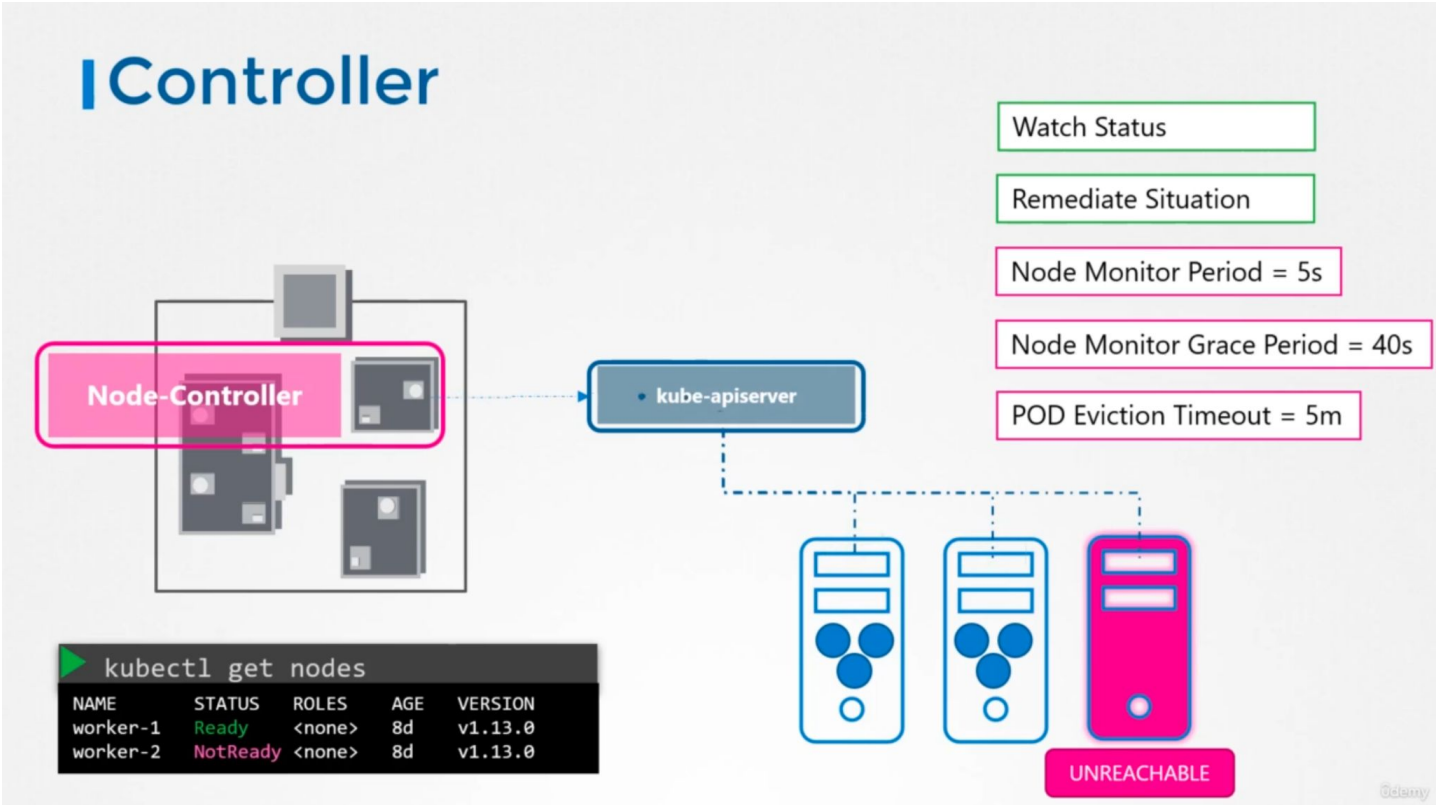


controller-manager

- Each controller
  - Continuously watches the state of components
  - Helps maintain desired state
- Controller-manager is the controller of controllers:
  - Node controller
  - Endpoints controller
  - Namespace controller
  - …
-

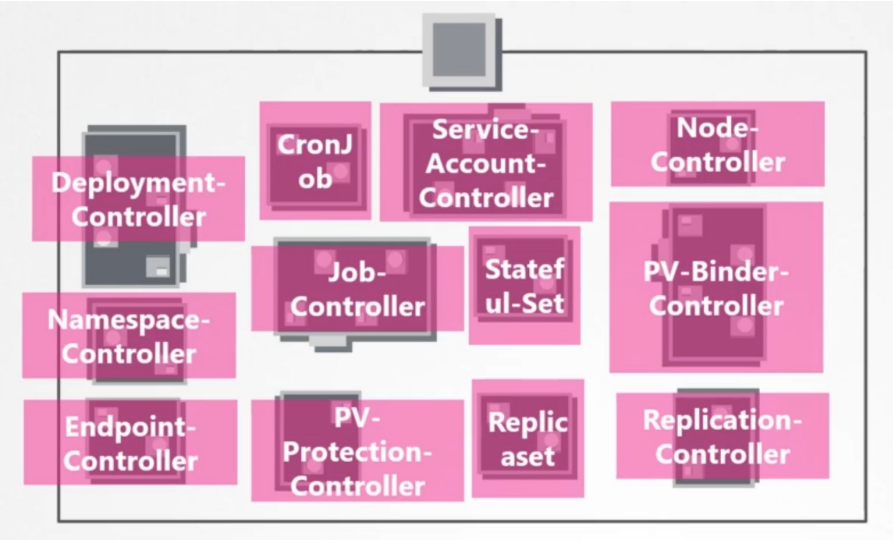


Controller Example: Node Controller



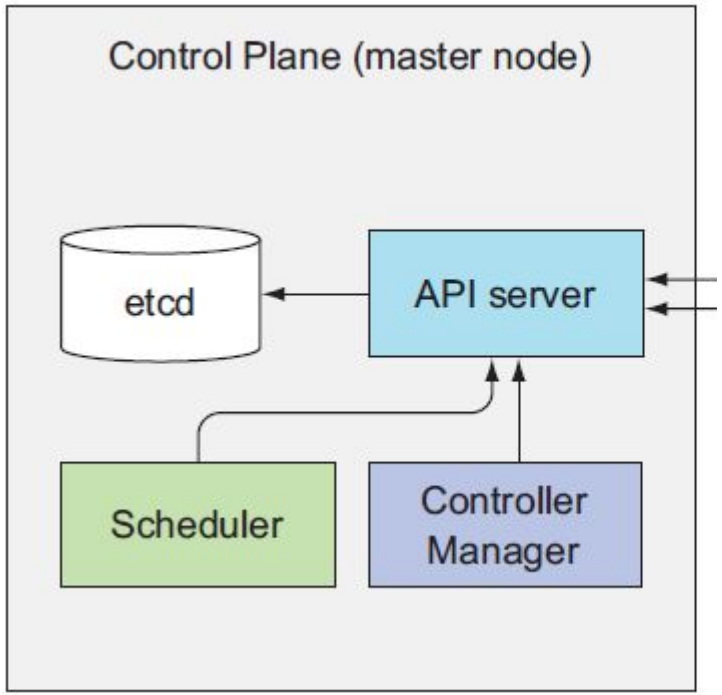
Kubernetes Controllers: controller-manager

All controllers are packaged in a single process named kube-controller-manager.



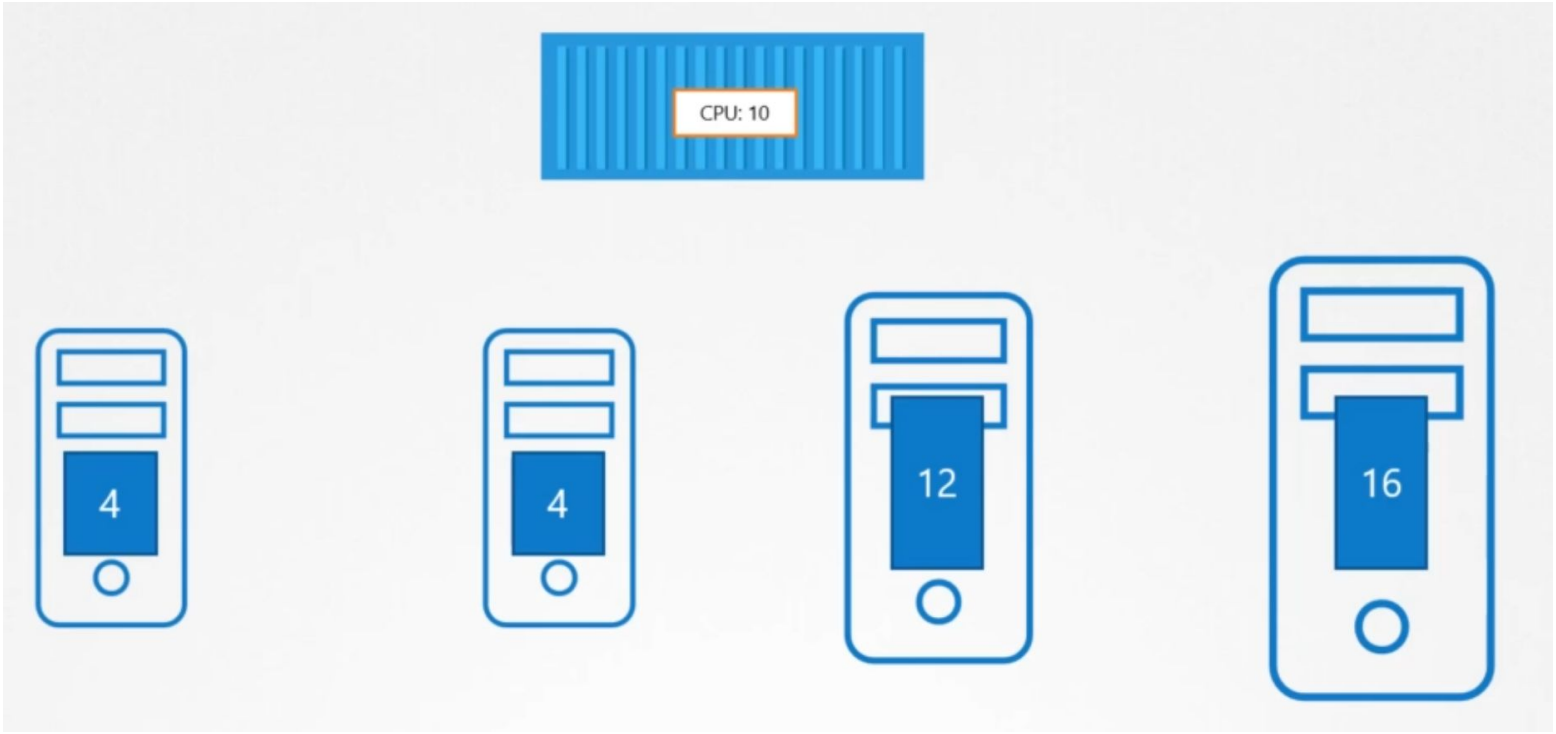
Scheduler

- Watches api-server for new pods
- Assign work to nodes
  - Only responsible for deciding which pod goes into which node - doesn't actually place the pod on the nodes.
- Scheduling consists of
  - Filtering Nodes
  - Ranking Nodes.



- Ship analogy: An office that decides which ship should carry our container.

Scheduler

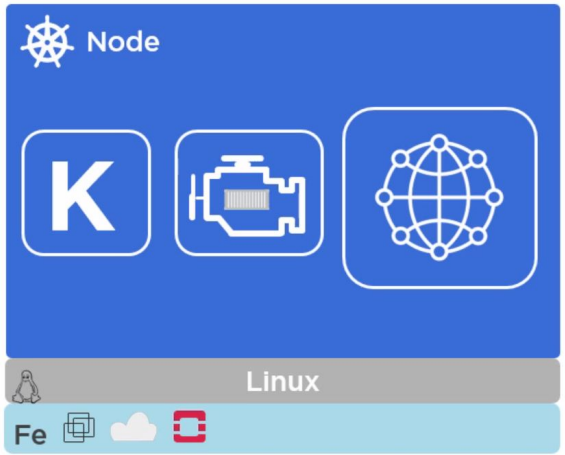


Worker Node

Kubelet
Main Kubernetes agent

Container engine
Docker or rkt

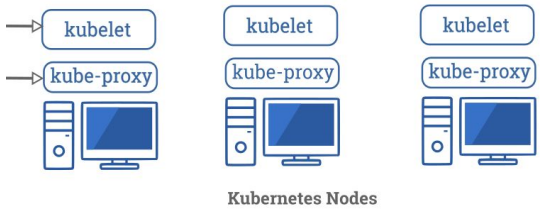
kube-proxy
Kubernetes networking



kubelet

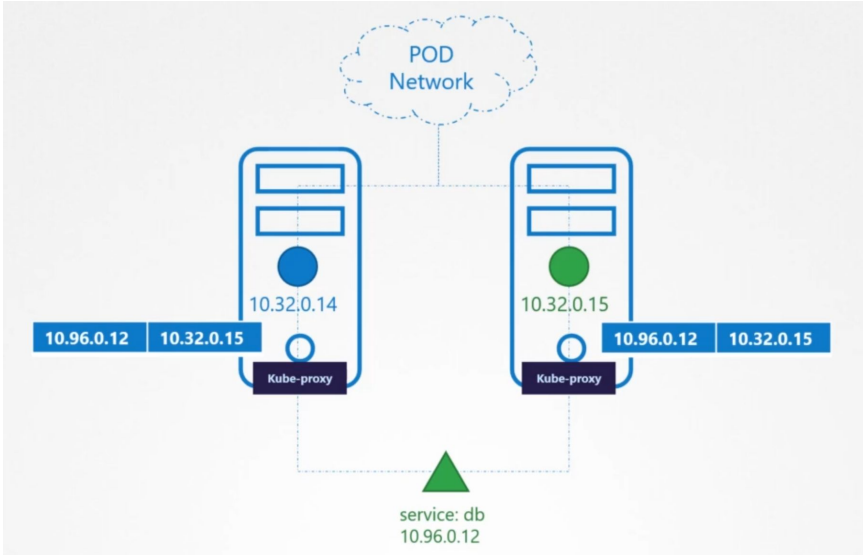
- The main Kubernetes agent
  a. Registers node with cluster
  b. Watches api-server
  c. Instantiates pods
  d. Monitor Nodes & PODS: Reports back to kube-api-server at regular intervals

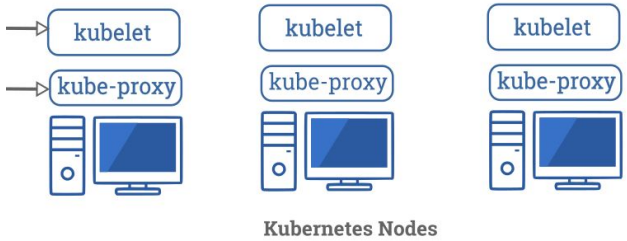
- Ship Analogy: Captain of the Ship - contacts the master ship - loads or unloads containers, sends back reports in at intervals



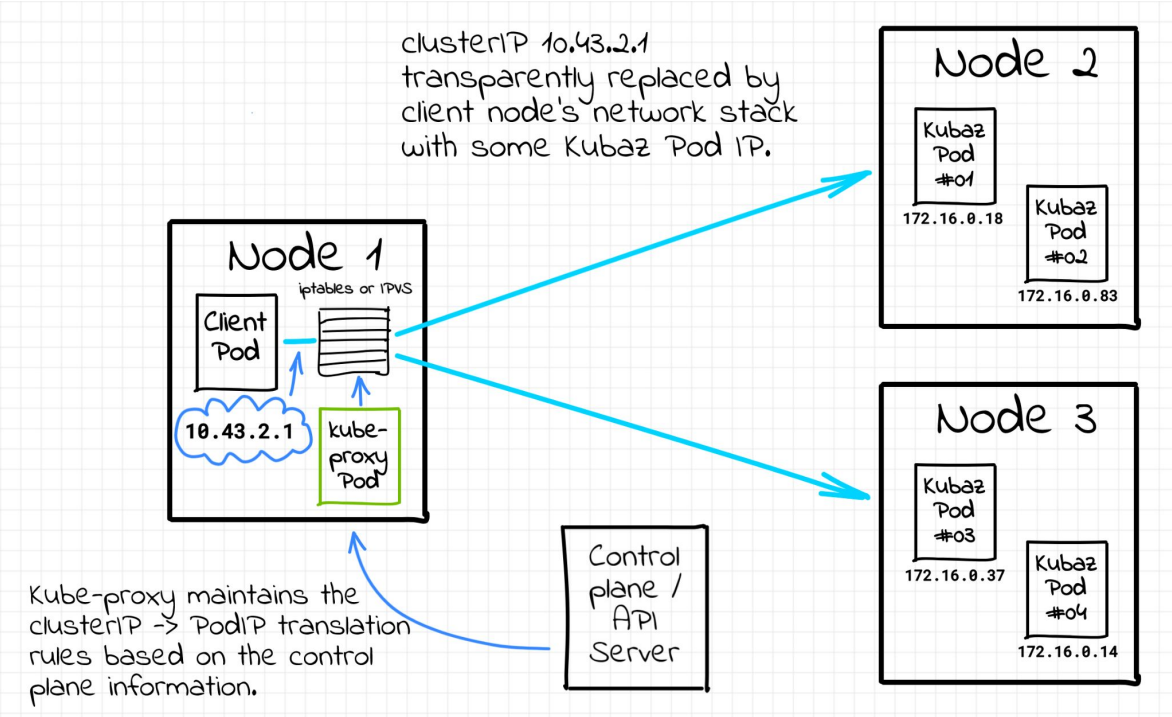
kube-proxy

- Pod Networking
  - In a K8s Cluster, any pod can connect to any pod. An Internal Virtual Network that spans all the nodes.
- Recall that Pod IP address are temporary.
  - Solution: Service
  - Service is not an actual process. It's a virtual component that only lives in K8S memory.
- kube-proxy is a process running on each node. It uses IP tables to forward traffic.
  - Linux kernel module netfilter uses IP tables to route traffic.



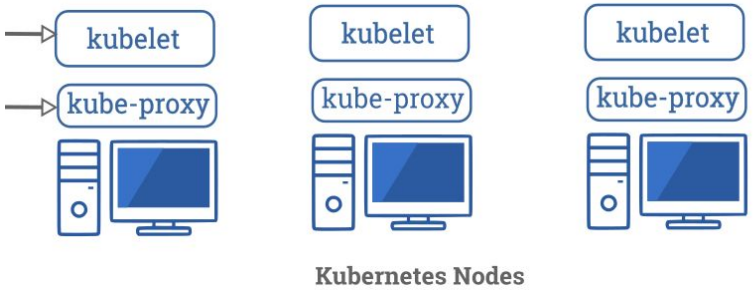


Service Discovery in Kubernetes: A Closer Look

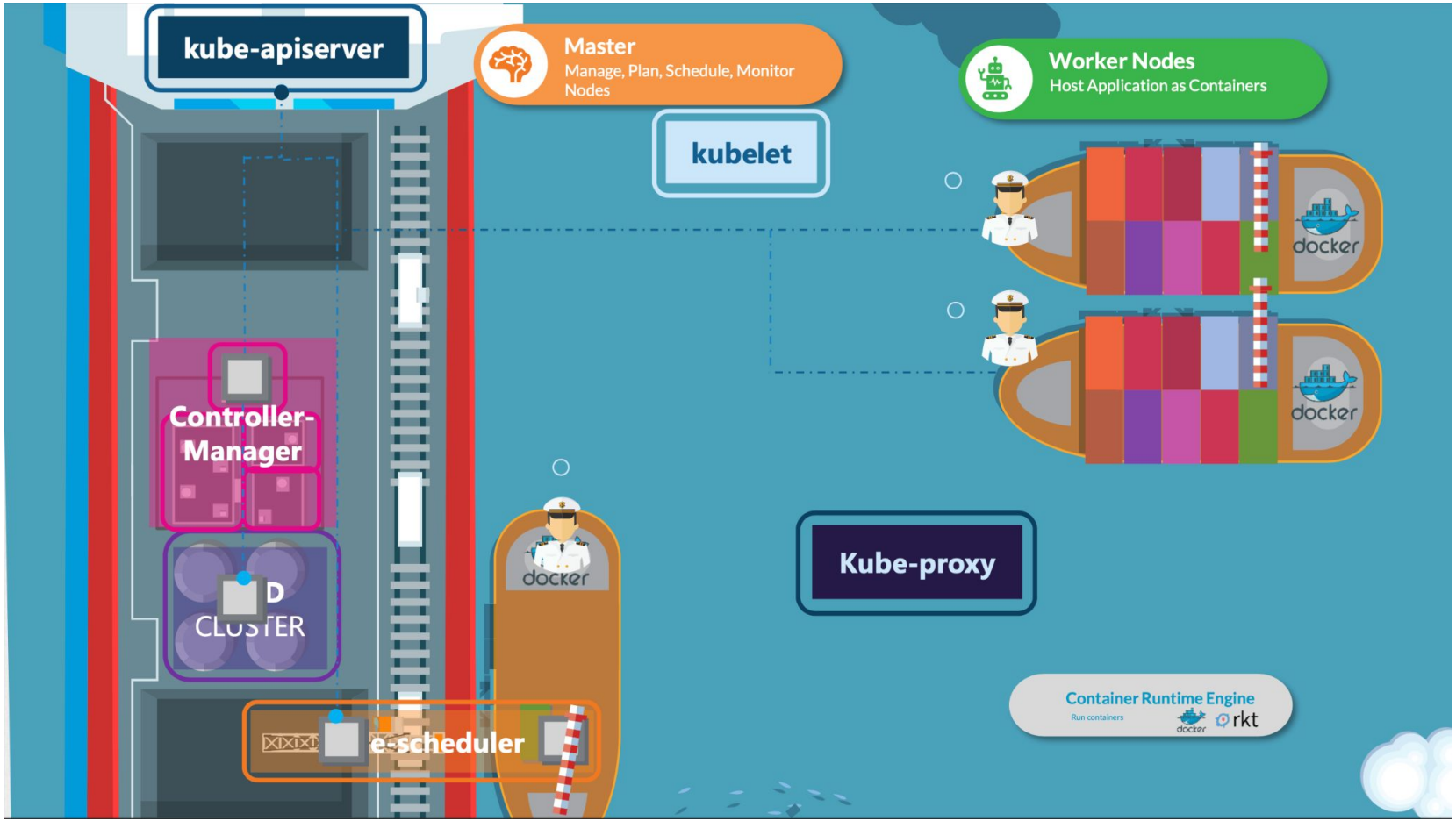


Container Runtime

- Does container management:
  - Pulling images
  - starting/stopping containers
- Pluggable:
  - Usually Docker
  - Can be rkt



Kubernetes Architecture: Ship Analogy



Conclusion

- Kubernetes disadvantages?
- What did we leave out?
  - Lots of K8S resources.
  - K8S access levels, security features.
  - K8S different setups: From KaaS to minikube.
  - *So Much* more.
- A few points on HW2.
  - Hove you completed the Docker section?
  - There is an upcoming video on hands-on Kubernetes.
  - Tell us *any* challenges that are holding you back.

References

- Arian Boukani’s “Introduction to Docker” presentation
- Saman Hosseini’s “Introduction to Kubernetes” presentation
- IBM Kubernetes Essentials
- VMWare Kubernetes Tutorial
- Kubernetes official document
- KodeKloud Certified Kubernetes Administrator Course
- …