

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

The screenshot shows a web browser window with a dark theme. The title bar reads "Introduction to Docker". The address bar shows the URL "2arian3.github.io/Docker-introduction/...". The main content area features a large white heading "Why Developers Care". Below it, a sub-heading "Build once... (finally) run *anywhere**" is followed by a bulleted list of ten reasons why developers care about Docker:

- A clean, safe, hygienic, portable runtime environment for your app.
- No worries about missing dependencies, packages and other pain points during subsequent deployments.
- Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for each app without worrying.
- Automate testing, integration, packaging...anything you can script.
- Reduce/eliminate concerns about compatibility on different platforms, either your own or your customers.
- Cheap, zero-penalty containers to deploy services. A VM without the overhead of a VM. Instant replay and reset of image snapshots.

* Where "anywhere" means an x86 server running a modern Linux kernel (3.2+ generally or 2.6.32+ for RHEL 6.5+, Fedora, & related)

In Mac and Windows operating systems, Docker has managed to provision containers on a micro Linux virtual machine.

Review of Containerization

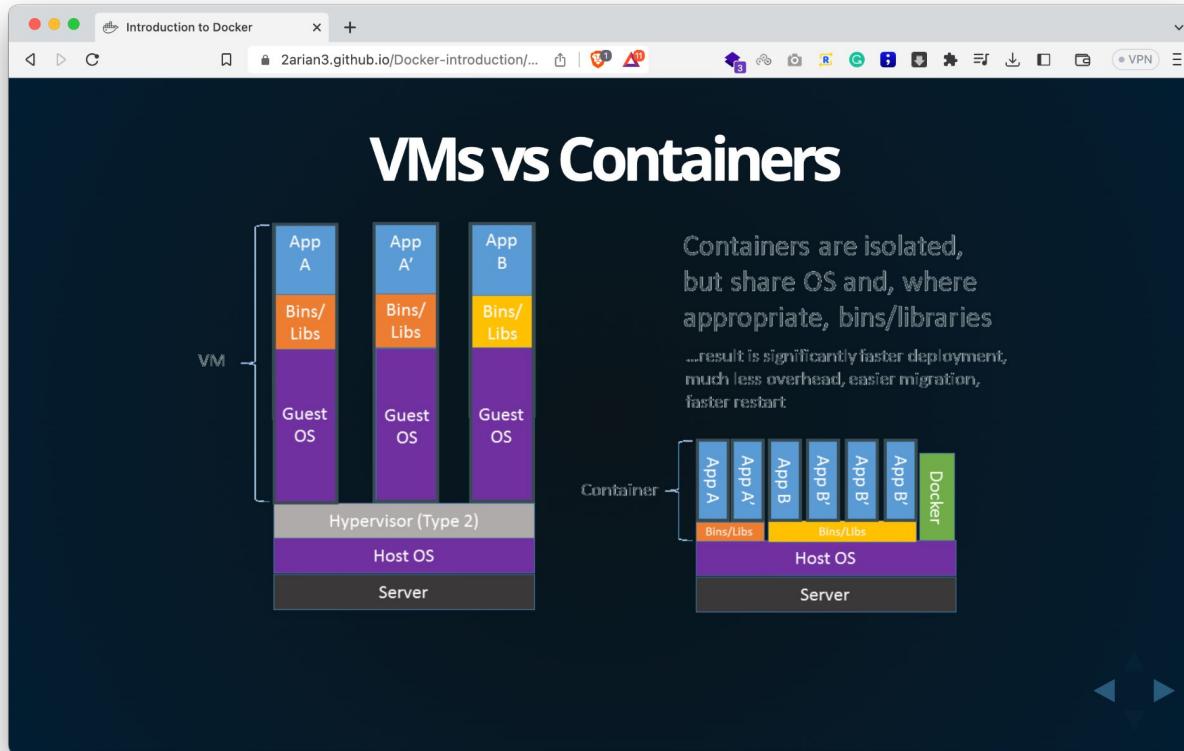
The screenshot shows a presentation slide titled "Introduction to Docker" on a dark-themed browser window. The slide has a large title "Why Administrators Care" and a subtitle "Configure once... run anything". Below the subtitle is a bulleted list of eight reasons why administrators care about Docker. The browser's address bar shows the URL <https://2arian3.github.io/Docker-introduction/>. The browser interface includes standard Mac OS X window controls and a toolbar with various icons.

Why Administrators Care

Configure once... run anything

- Make the entire lifecycle more efficient, consistent, and repeatable
- Increase the quality of code produced by developers.
- Eliminate inconsistencies between development, test, production, and customer environments.
- Support segregation of duties.
- Significantly improves the speed and reliability of continuous deployment and continuous integration systems.
- Because the containers are so lightweight, address significant performance, costs, deployment, and portability issues normally associated with VMs.

Review of Containerization



The diagram illustrates the architectural differences between Virtual Machines (VMs) and Docker containers.

VM (Virtual Machine): This model shows multiple virtual machines running on a single host. Each VM consists of a guest operating system (Guest OS), application software (App A, App A', App B), and their respective binary/library files (Bins/Libs). These VMs run on top of a hypervisor (Type 2) which sits on a host operating system (Host OS) running on a physical server.

Container: This model shows a single host operating system (Host OS) running on a physical server. Multiple applications (App A, App A', App B, App B') are run within separate containers. Each container contains its own application, binary/library files (Bins/Libs), and a Docker runtime environment. The containers share the host's kernel and resources.

Text on the right:

Containers are isolated, but share OS and, where appropriate, bins/libraries
...result in significantly faster deployment, much less overhead, easier migration, faster restart

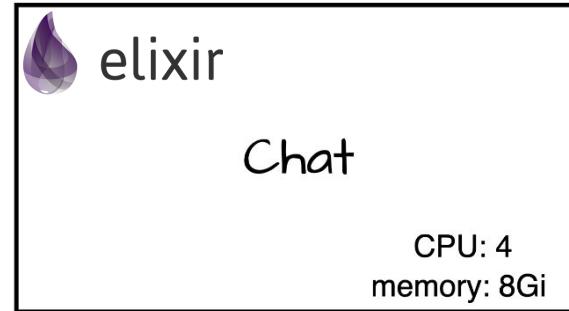
Monolithic Applications

- 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

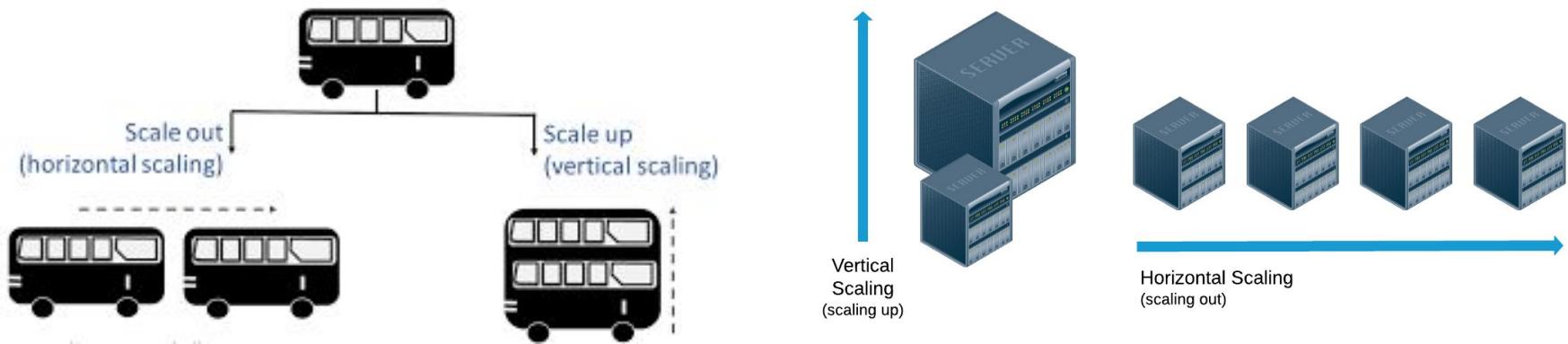
CPU: 2
memory: 2Gi



Example: A Classified Ads website



Scaling



Monolithic

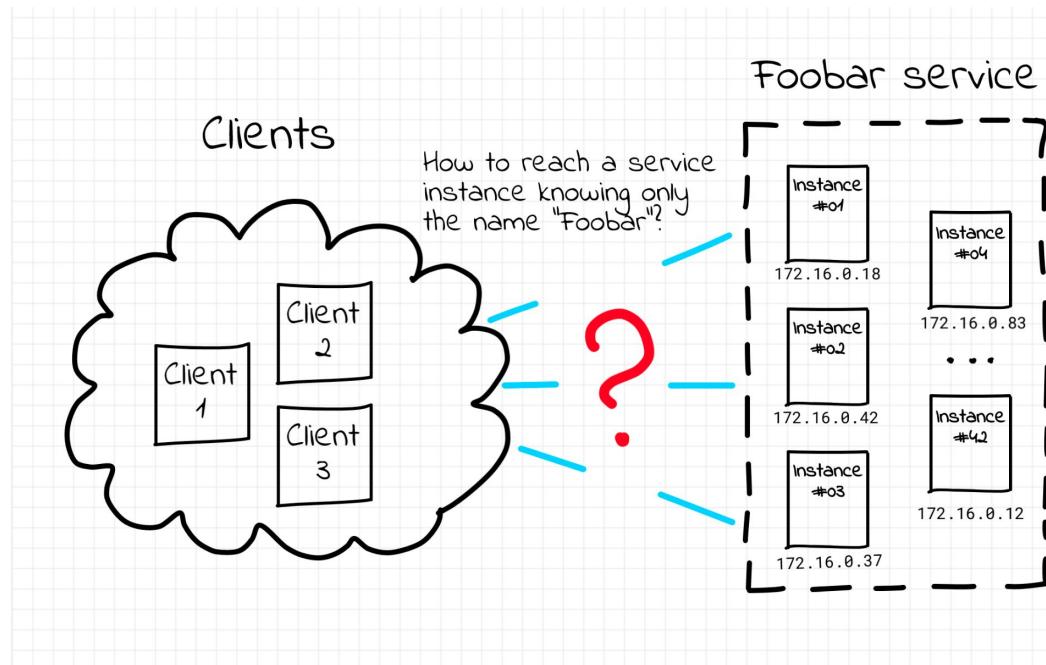
- Monolithic
 - A collection of tightly coupled components that have to be developed, deployed and managed as one entity.
- What are the challenges for a monolith application?
 - Components need different
 - Software requirements: Programming Language, Libraries and versions
 - Hardware requirements: 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

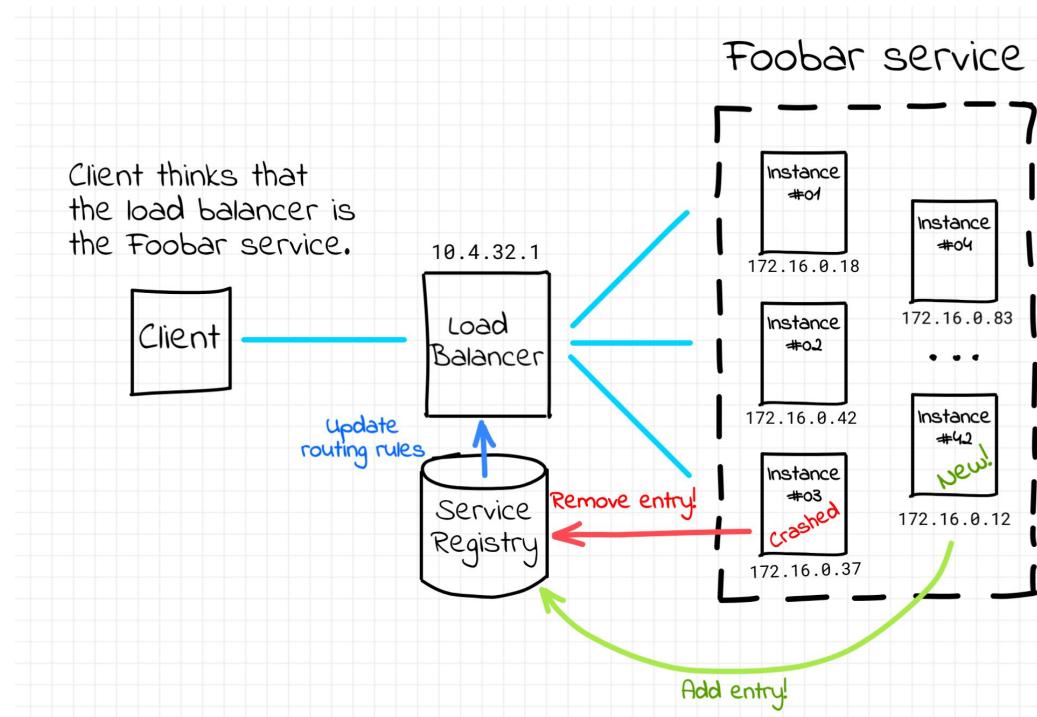
Microservices: Service Discovery

- 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

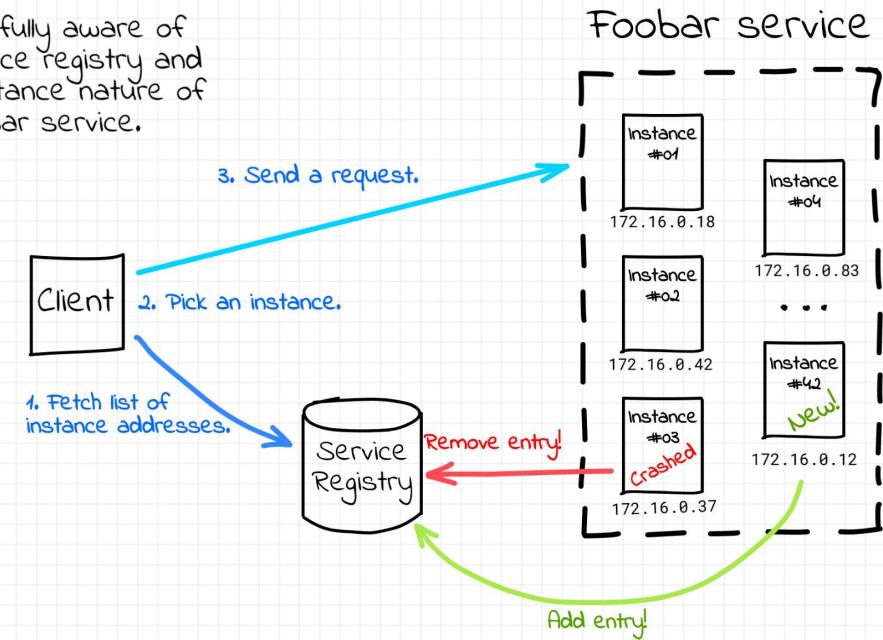
- A Load Balancer distributes the requests.
- Instances register themselves in service registry.
- Challenges of Load balancer:
 - Single point of Failure - bottleneck
 - Should support all communication protocols
 - Extra Network Hop



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.

Client is fully aware of the service registry and multi-instance nature of the Foobar service.

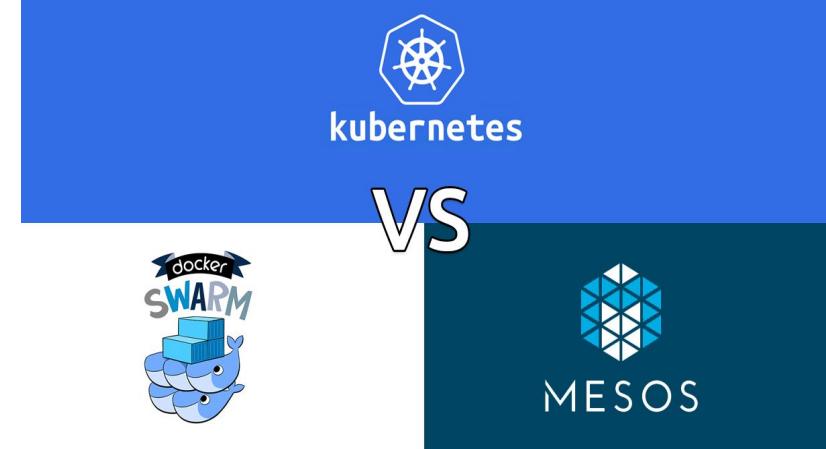


Microservices Operational Challenges Solution: Orchestration Tools

- 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

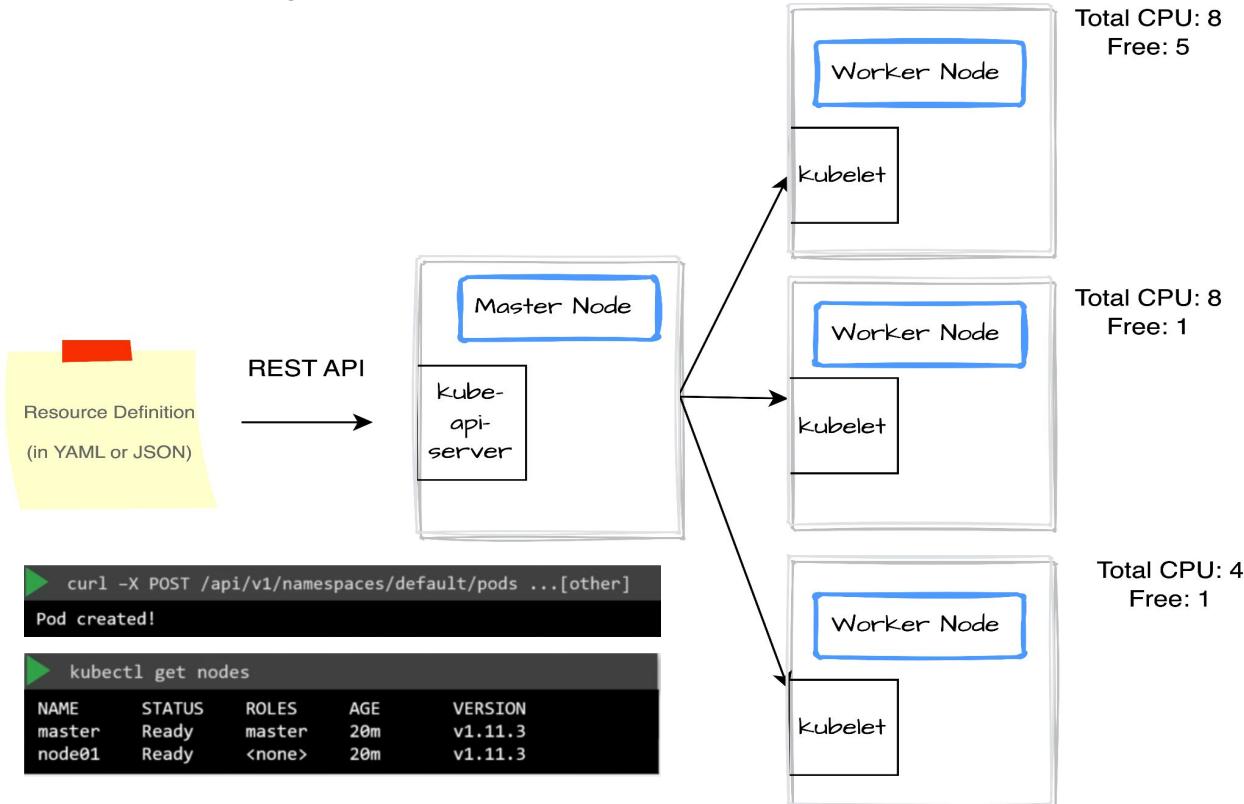
Orchestration Tools: Kubernetes

- 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)



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

Kubernetes Resources: Configmaps and Secrets

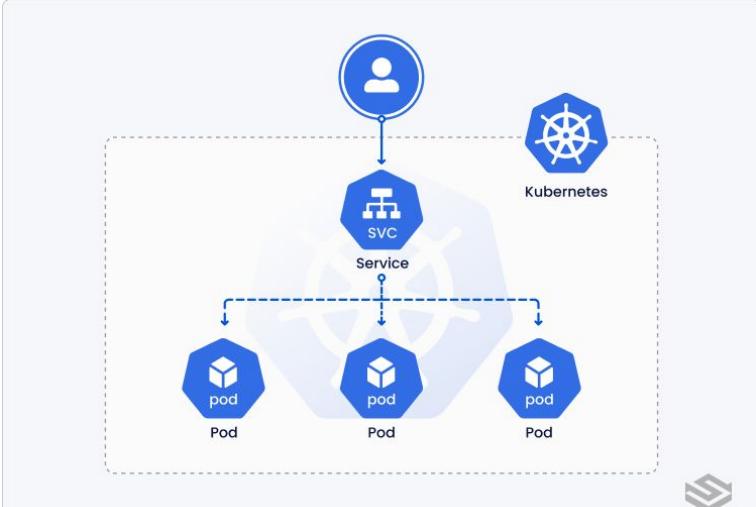
- Many applications rely on configuration which is used during either application initialization or runtime, e.g. database credentials.
 - Non-Confidential: Configmap
 - Confidential: Secret
- They will help us avoid rebuilding our images every time we need to change the config.



```
apiVersion: v1
kind: ConfigMap
metadata:
  creationTimestamp: 2019-12-27T18:36:28Z
  name: game-config-env-file
  namespace: default
  resourceVersion: "809965"
  uid: d9d1ca5b-eb34-11e7-887b-42010a8002b8
data:
  allowed: '"true"'
  enemies: aliens
  lives: "3"
```

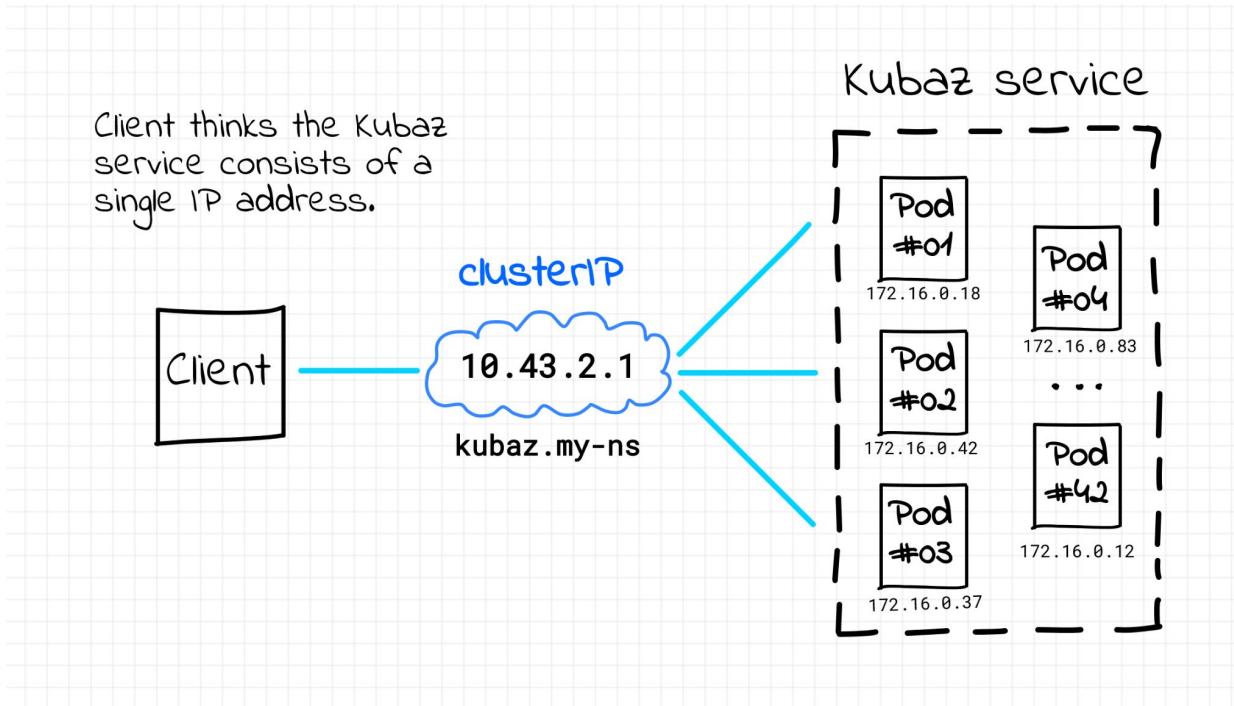
Kubernetes Resources: Service

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



```
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 & Replicasesets

- 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.

```
● ● ●  
  
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
```

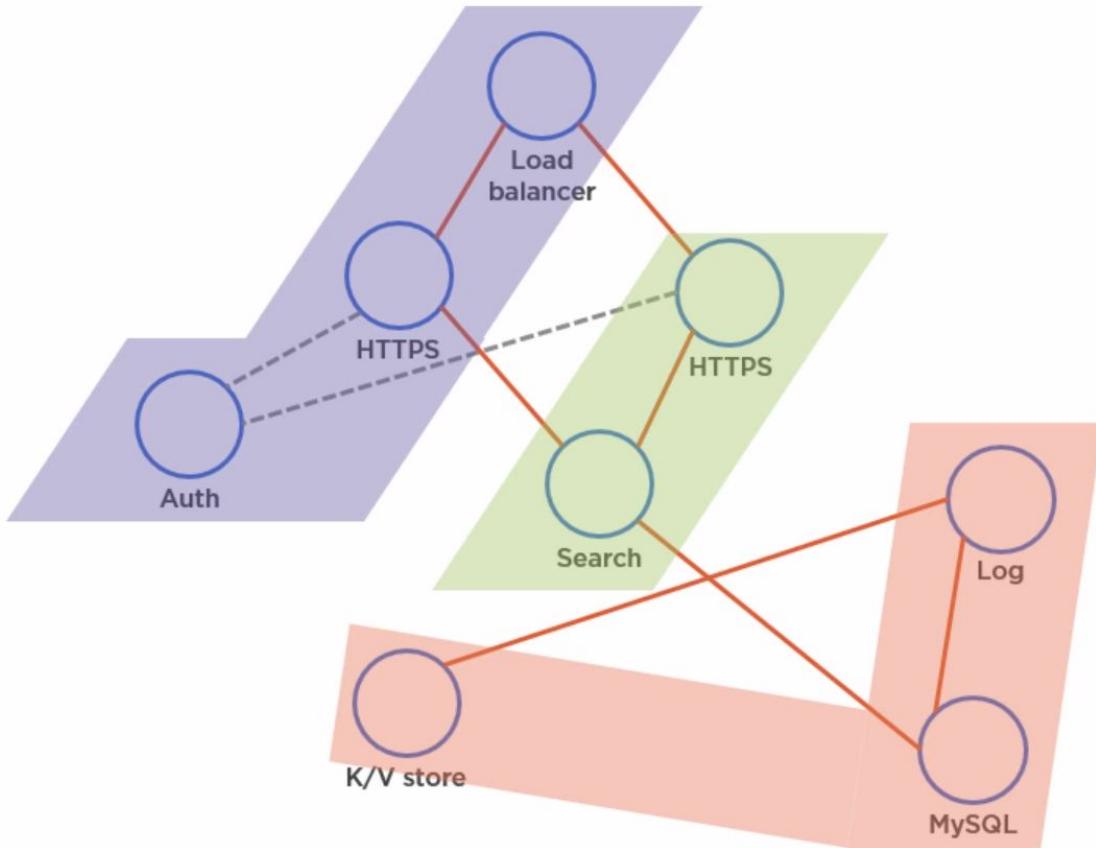
Kubernetes Resources: Others

- 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.

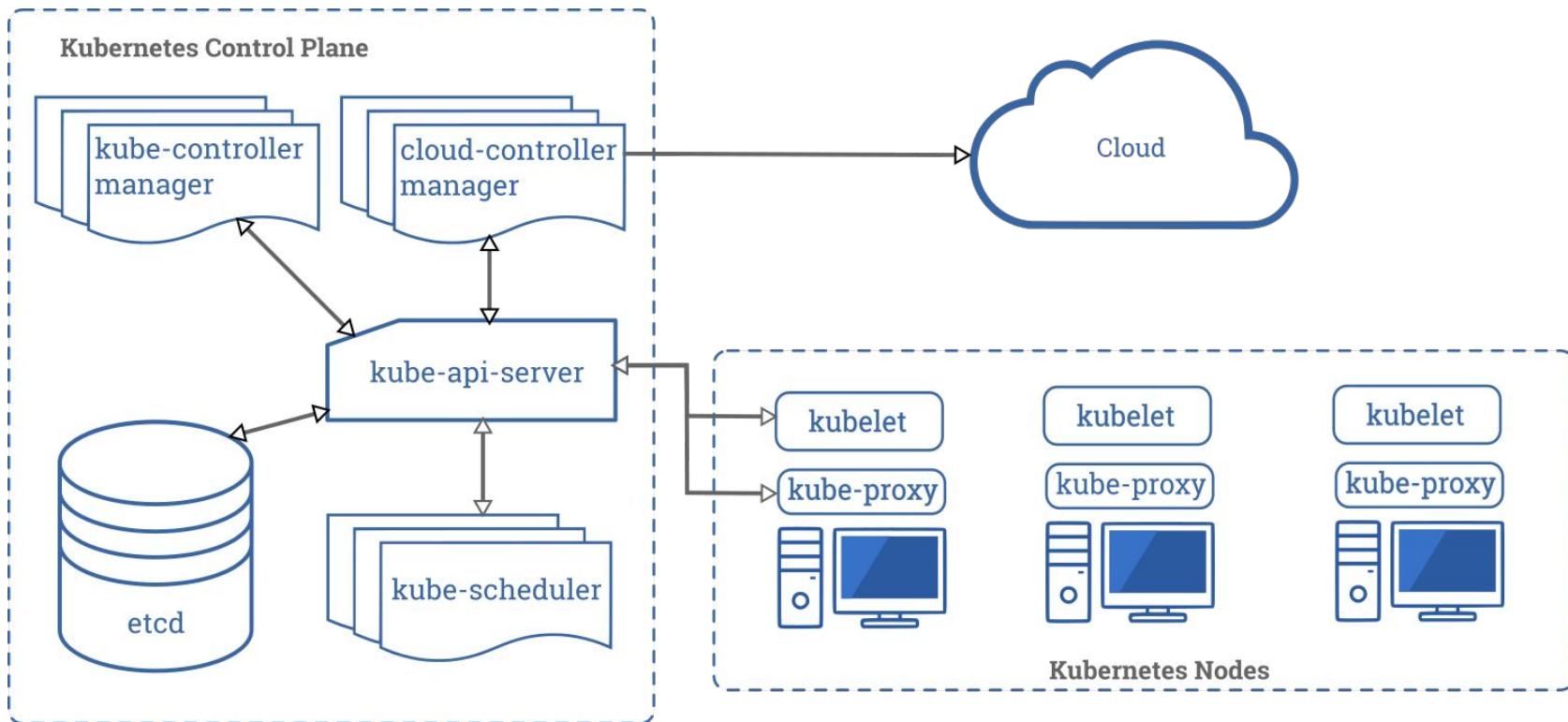
K8s Resources Mapping to Infrastructure



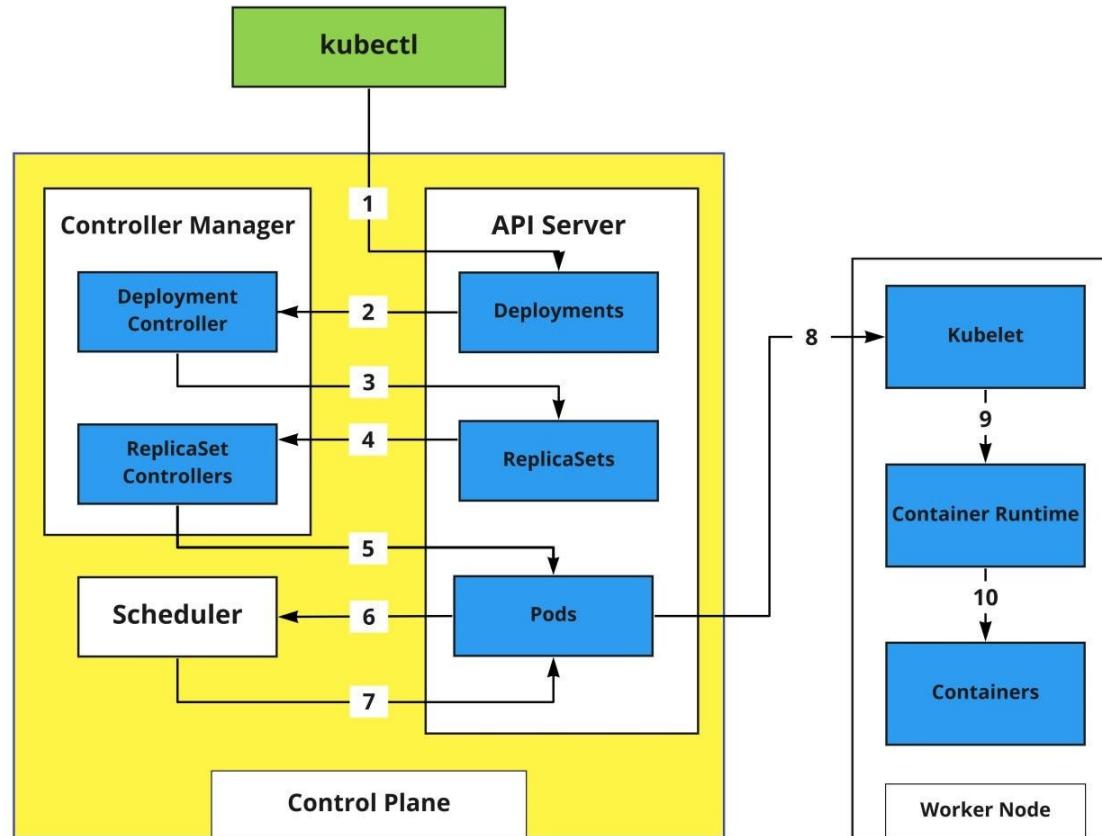
- Node 1
- Node 2
- Node 3



Kubernetes: Administrators Perspective

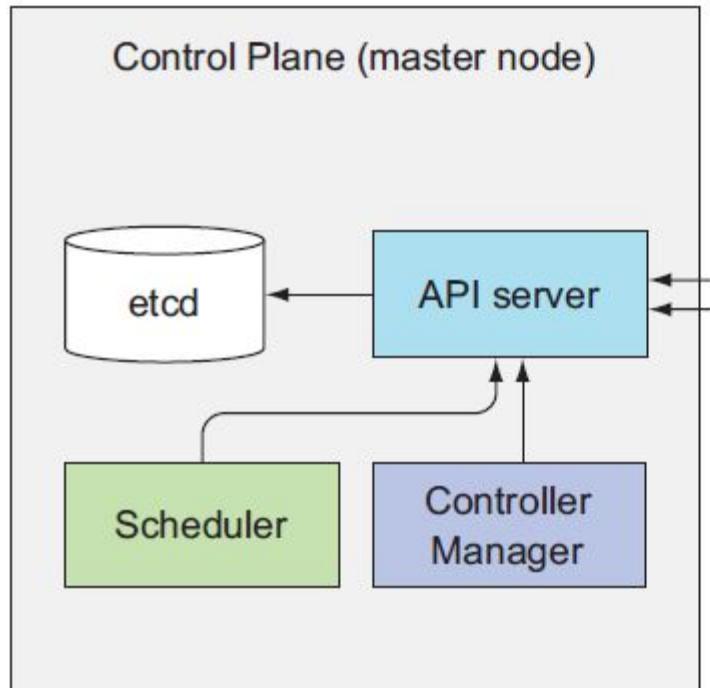


Pod Creation Chain of Events



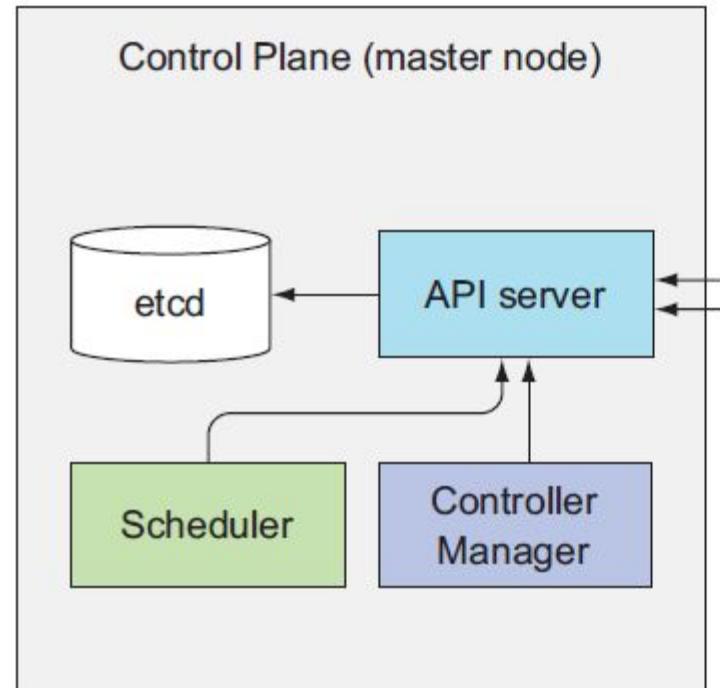
api-server

- 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



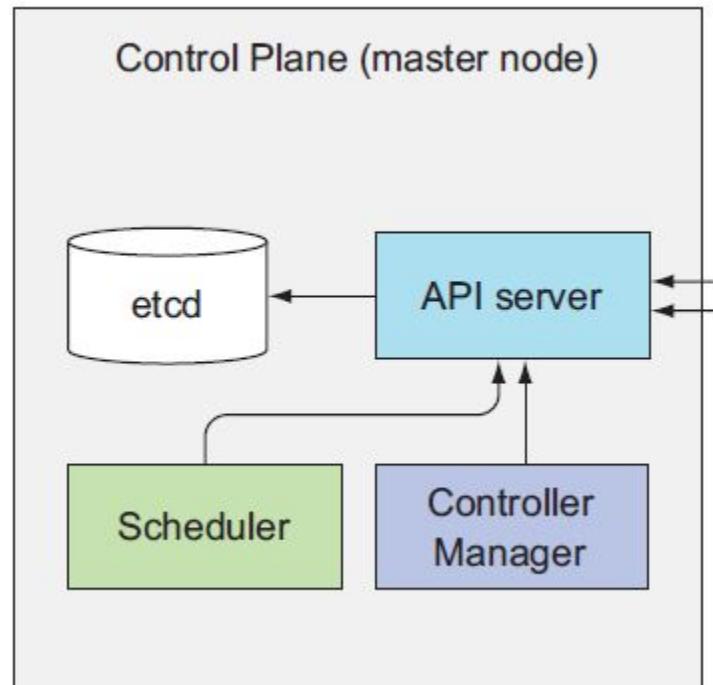
Cluster store

- 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

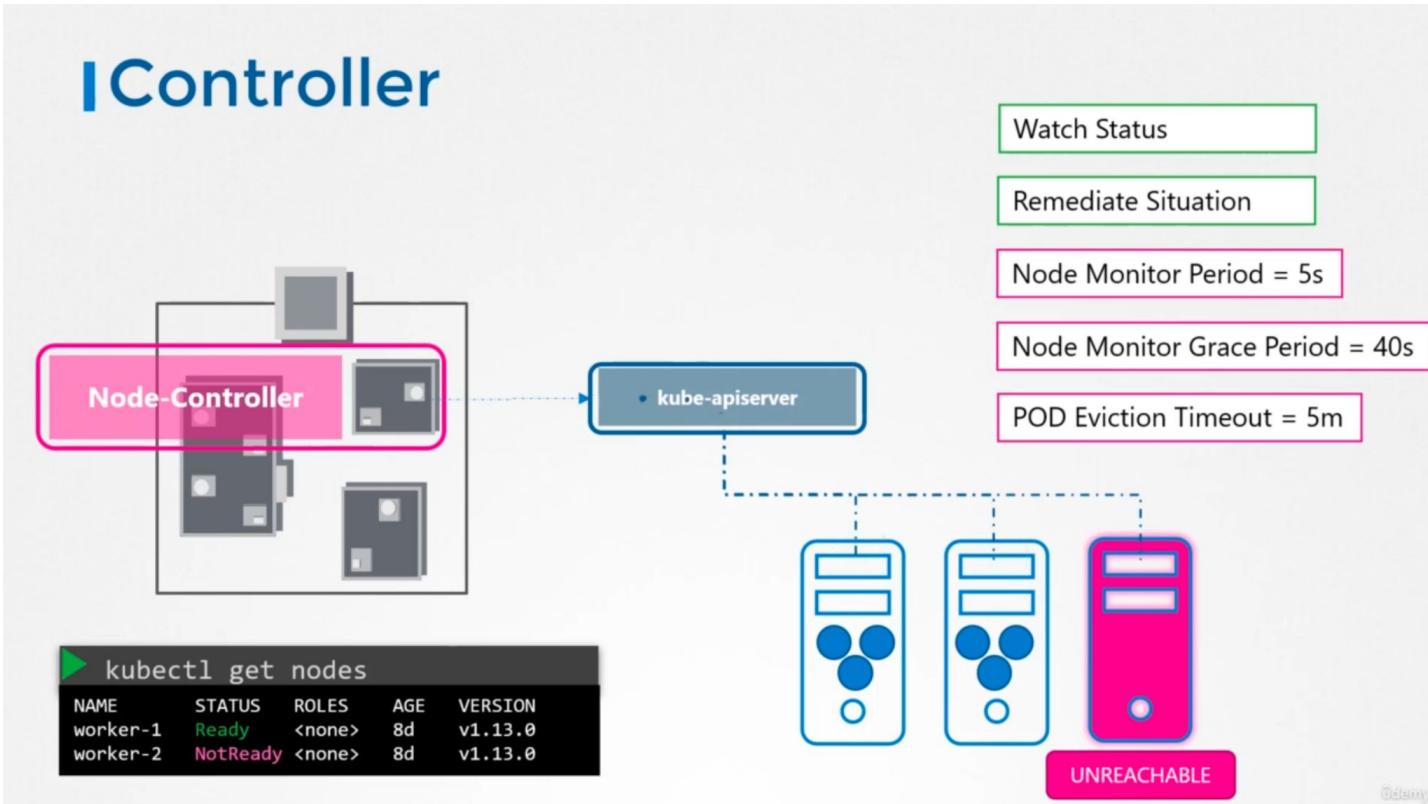


controller-manager

- 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
 - ...
-

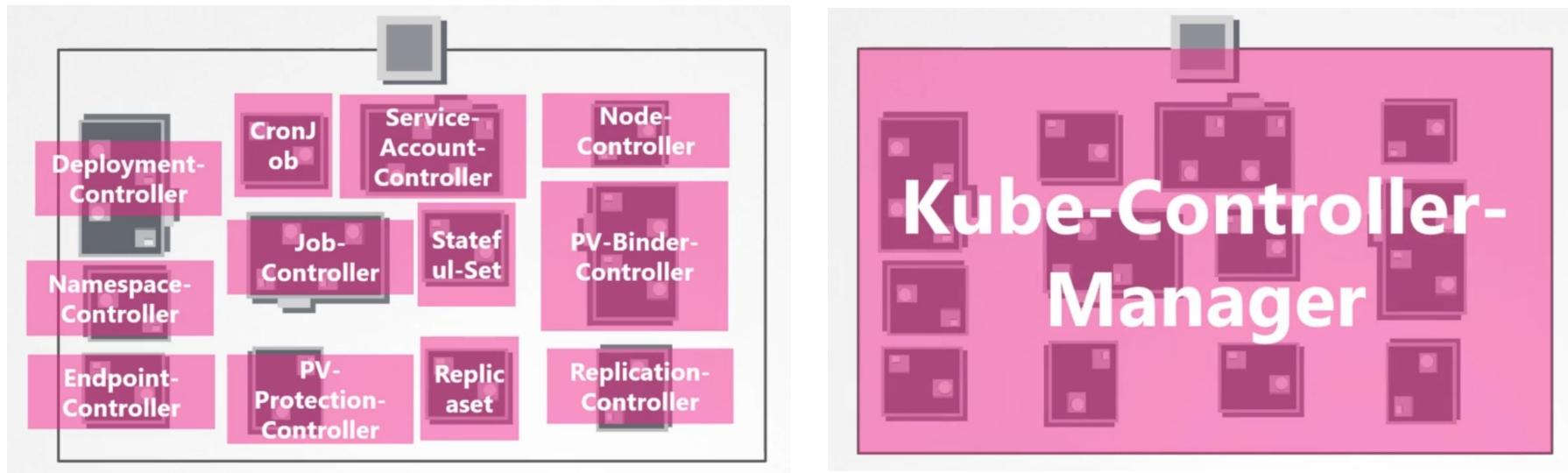


Controller Example: Node Controller



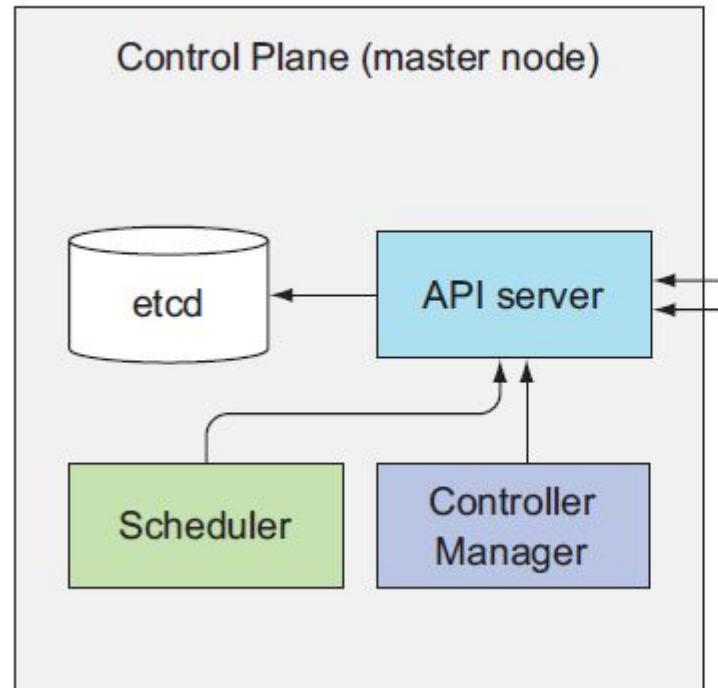
Kubernetes Controllers: controller-manager

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

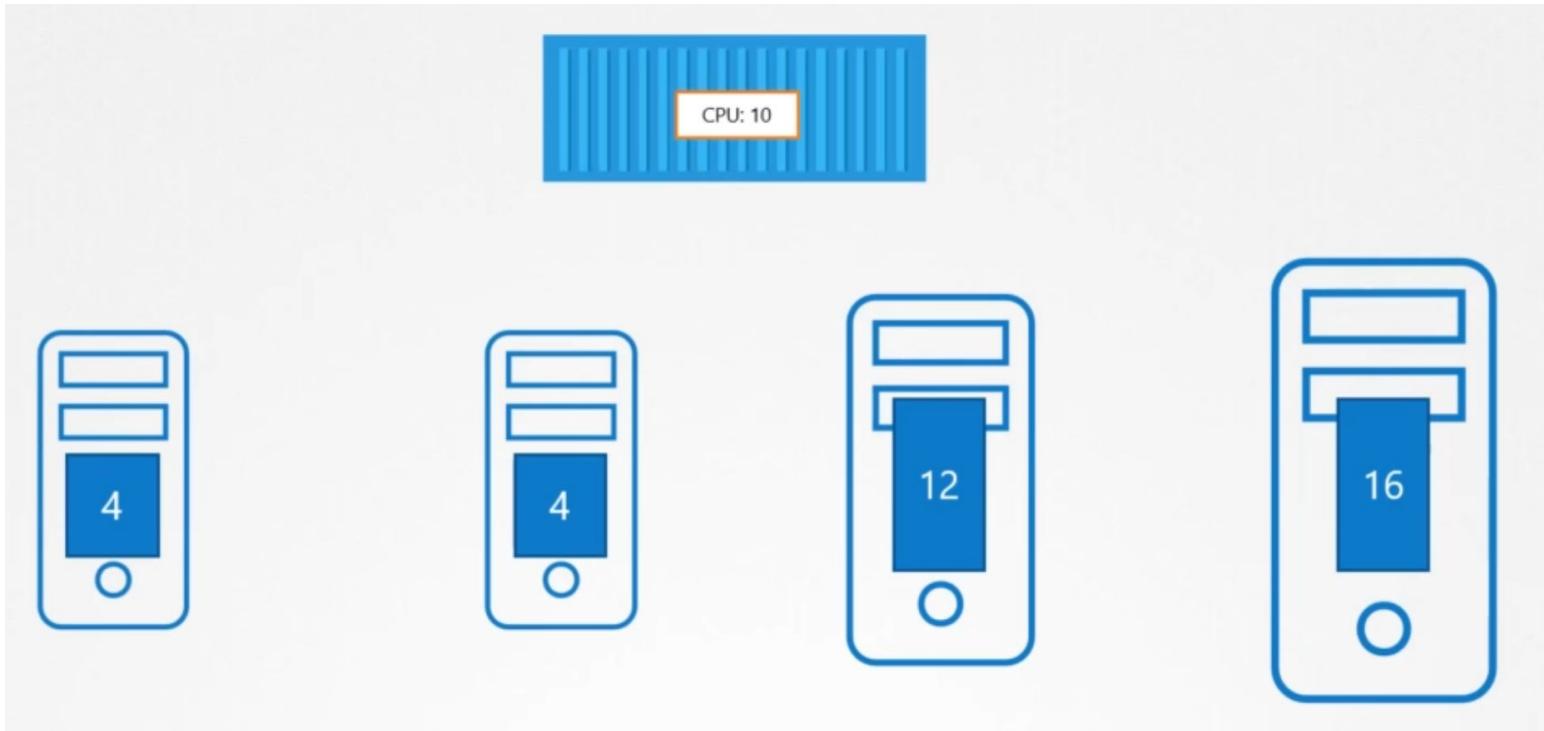


Scheduler

- 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



Node

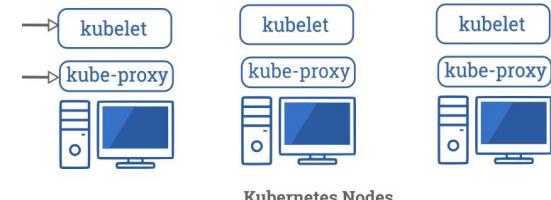


Linux



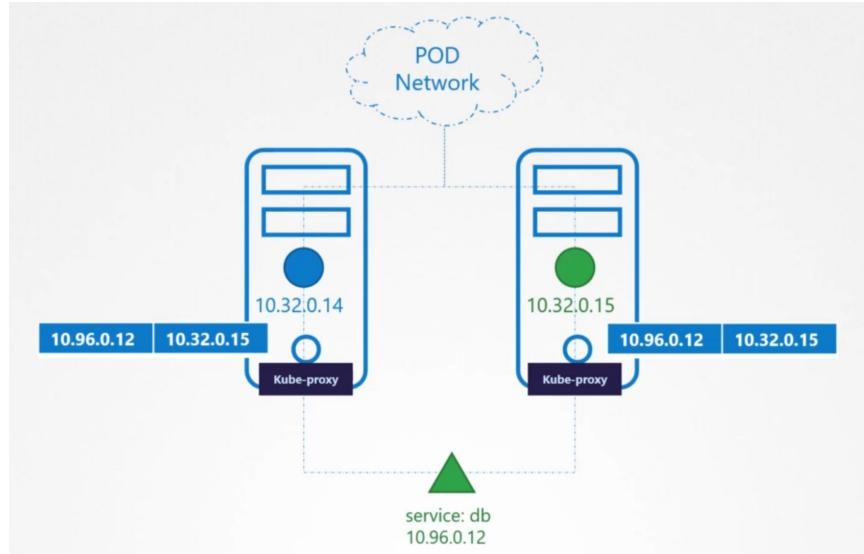
kubelet

- 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



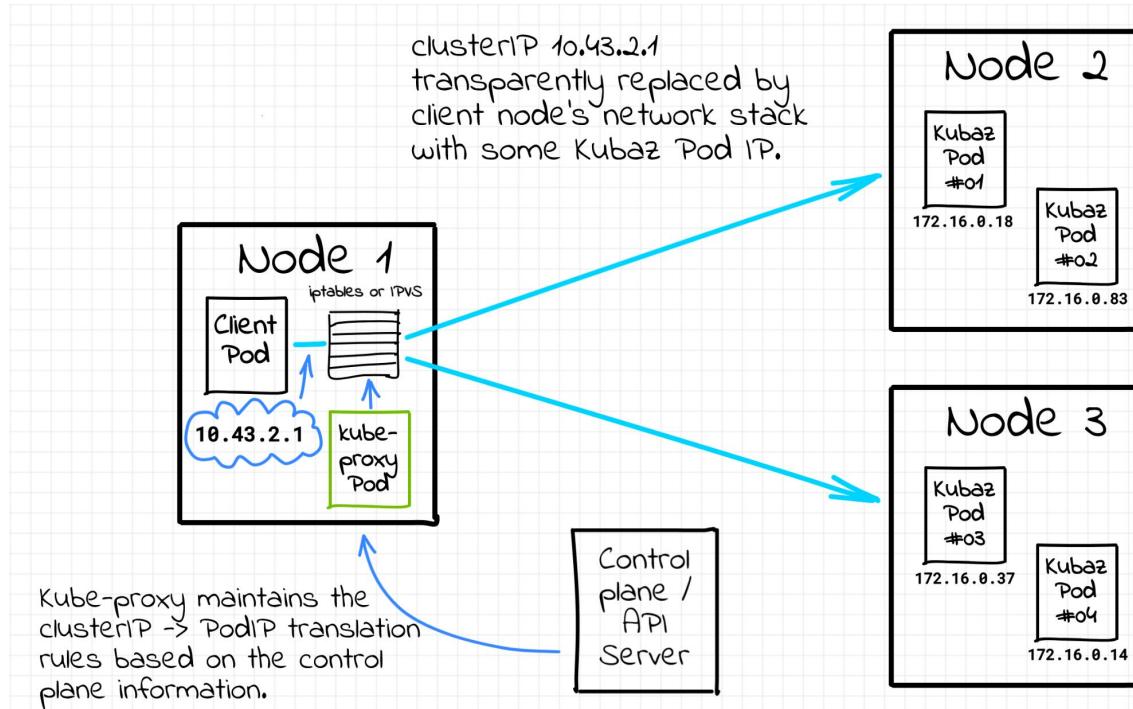
kube-proxy

- 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.



Kubernetes Nodes

Service Discovery in Kubernetes: A Closer Look

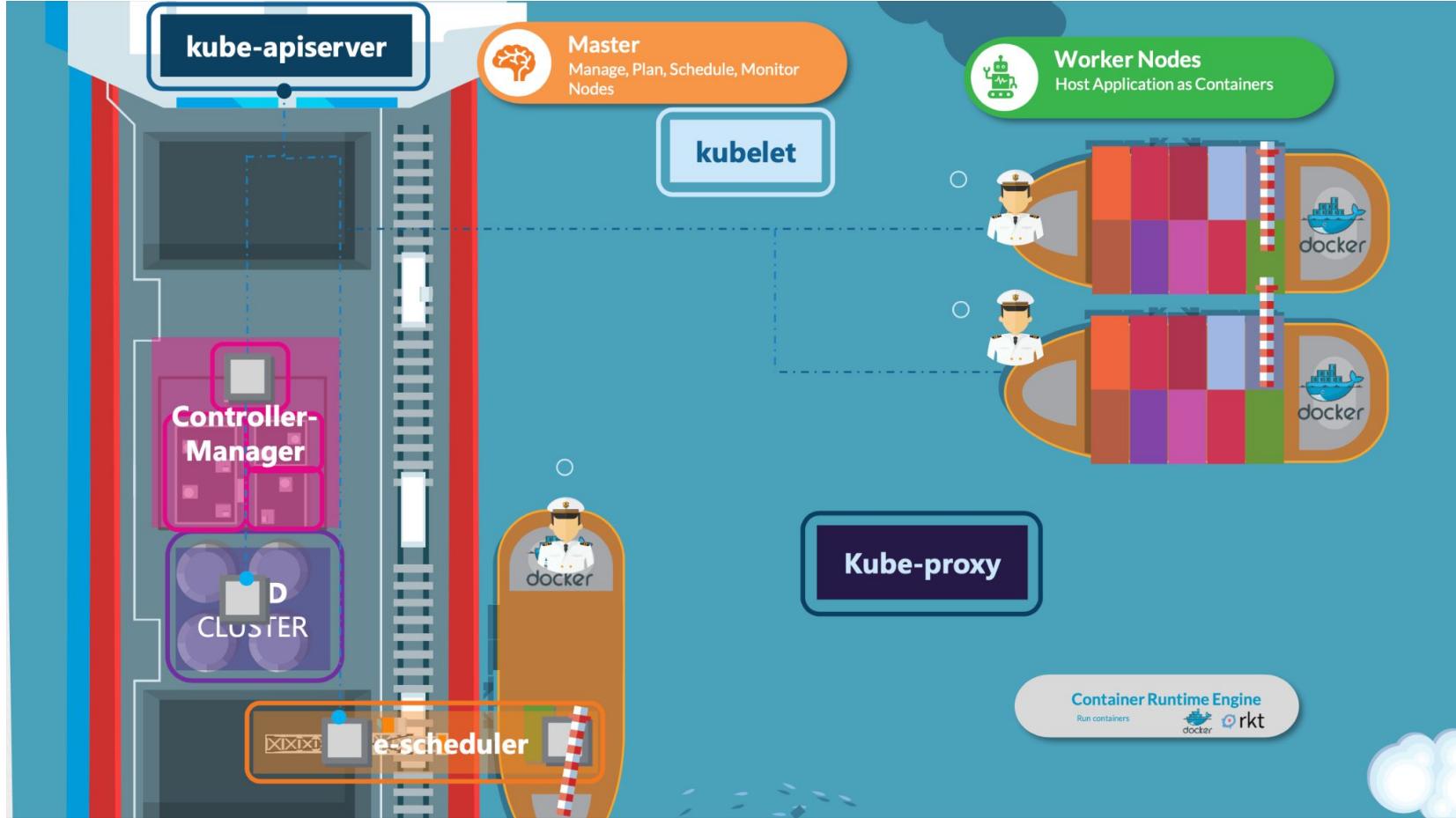


Container Runtime

- 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.
 - Have 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
- ...