Cloud Computing Technologies

DAT515 - Fall 2024

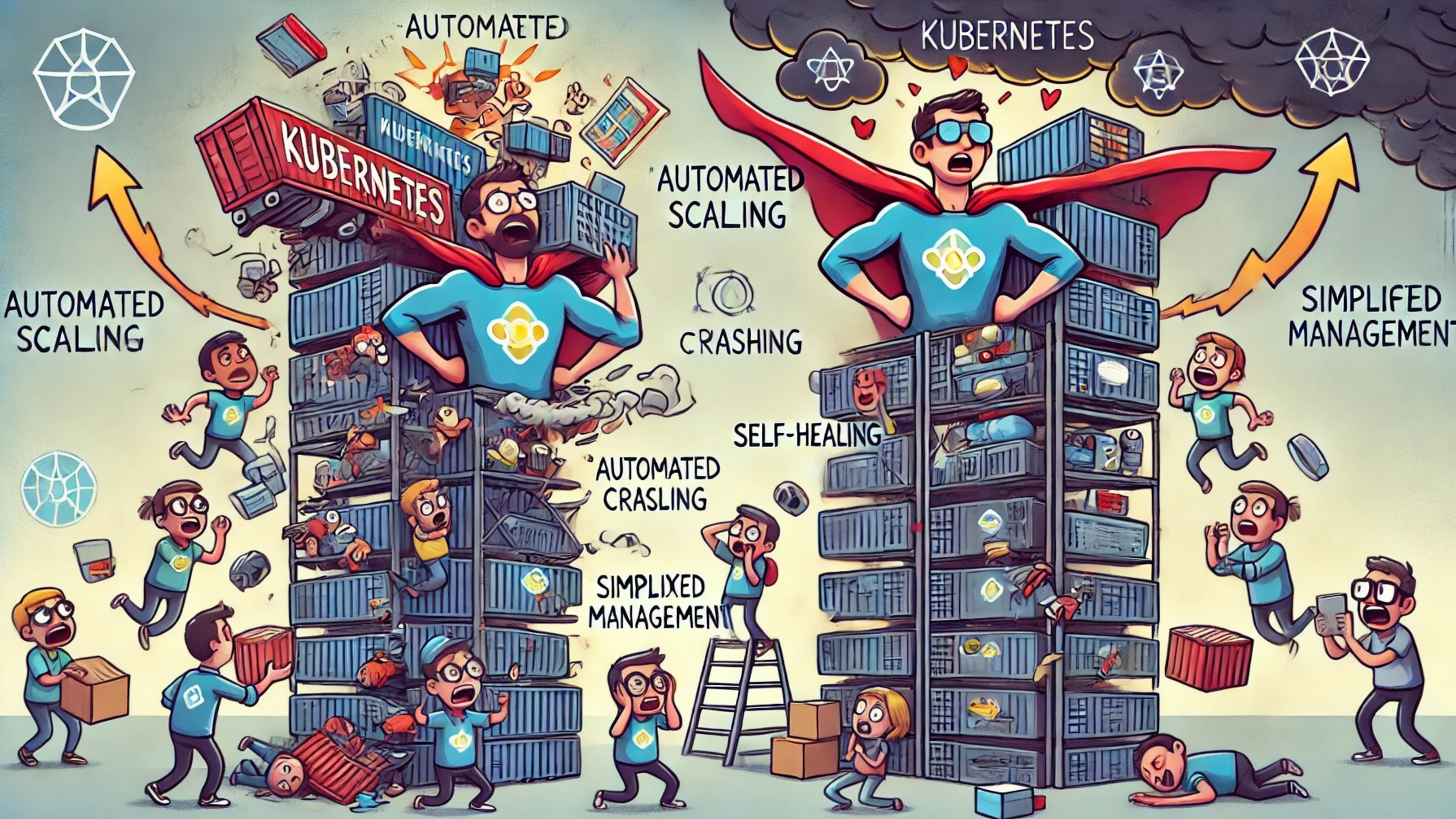
Kubernetes

Prof. Hein Meling

University
of Stavanger


KUBERNETES
KUBERNETES
KUBERNETES
KUBERNETES
KUBERNETES
KUBERNETES
KUBERNETES ORCHESTRE

Motivation for Kubernetes


AUTOMATED
KUBERNETES
AUTOMATED SCALING
AUTOMATED SCALING
CRASHING
SIMPLIFED MANAGEMENT
SELF-HEALING
AUTOMATED CRASLING
SIMPLIXED MANAGEMENT

Mutual Infrastructure

Challenges with Traditional Mutable Systems

- Incremental Updates
  - Mutable systems apply changes gradually
  - Leads to unpredictable states due to accumulated modifications over time
- Operator Modifications
  - Multiple admin users can modify the system
  - Resulting in undocumented changes and configuration drift

Challenges with Traditional Mutable Systems

- Upgrade Complexity
  - Tools like `apt-get` overwrite old binaries with new ones
  - Directly edit files in `/etc/` without version control
  - Difficult to track changes or roll back effectively
- Unclear System State
  - Current state is not represented as a single, unified artifact
  - Mix of updates and changes, increasing risk of errors and inconsistencies

QuickFeed Infrastructure

Tidbits from QuickFeed's Virtual Machine Config

```
sudo setcap CAP_NET_BIND_SERVICE=+eip /path/to/binary/quickfeed

sudo sysctl net.ipv4.ip_forward=1
sudo sysctl -p
sudo service docker restart

./backup.sh

./update.sh
```

QuickFeed update.sh Script

```
#!/bin/bash

if sysctl -q -n net.ipv4.ip_forward | grep -q '0'; then
        echo "IP packet forwarding disabled on host machine; run these commands
        echo ""
        echo "sudo sysctl net.ipv4.ip_forward=1"
        echo "sudo sysctl -p"
        echo "sudo service docker restart"
        exit 1
fi

QUICKFEED=$HOME/quickfeed
GOBIN=$HOME/go/bin
DATABASE=qf.db
LOGFILE=quickfeed.log
BACKUP=$QUICKFEED/backups

cd $QUICKFEED

echo "Backing up executables"
cp $GOBIN/quickfeed $BACKUP/quickfeed.$(date +"%m-%d-%y").bak

echo "Backing up the database file"
cp $QUICKFEED/$DATABASE $BACKUP/$DATABASE.$(date +"%m-%d-%y").bak

echo "Backing up the $LOGFILE file"
cp $HOME/logs/$LOGFILE $BACKUP/$LOGFILE.$(date +"%m-%d-%y").bak
```

```
echo "Running webpack"
if ! make ui-update; then
        echo "Failed to compile the client"
        exit 1
fi

echo "Running go install"
if ! make install; then
        echo "Failed to compile the server"
        exit 1
fi

if pgrep quickfeed &> /dev/null; then
        echo "Server running; shutting down server..."
        if ! killall quickfeed; then
                echo "Failed to stop the server; trying to restart..."
        fi
fi

echo "Starting the QuickFeed server"
quickfeed &> $HOME/logs/$LOGFILE &

if ! pgrep quickfeed &> /dev/null; then
    echo "Failed to start the server"
    exit 1
fi

echo "All done. QuickFeed server restarted and running"
exit 0
```

Immutable Infrastructure

Once an artifact is created in the system, it does not change

- Promotes **consistency** and **reliability**
- Immutable Systems
  - New, complete images replace old ones in a single operation
  - **Avoiding incremental changes**
- Example in Containers
  - Instead of updating a running container
  - Build **new container** image enabling **rollback** capability

Immutable Infrastructure

Once an artifact is created in the system, it does not change

- Reduced Risk of Configuration Drift
  - Immutable containers prevent unrecorded modifications
  - Transparency to changes; easier to troubleshoot
- Antipattern Warning: Imperative **changes** to running containers are **discouraged**, except in extreme, **mission-critical cases**

Mutable vs Immutable

	Mutable	Immutable
Pros	- Does not need to build servers from scratch every time a change is required	- Declarative - Predictable state - Auditable - Easy to rollback - Modular
Cons	- Incremental changes can fail - Indiscrete versioning	- Unable to modify in-place - Misaligned with traditional IT practices

Imperative Configuration

Challenges with Imperative Configuration

- Sequential Execution
  - Imperative commands define actions (e.g., "run A, then B"),
  - Must be executed in order, increasing complexity
- Difficult to Understand Current System State
  - Making debugging harder

Imperative Configuration

Challenges with Imperative Configuration

- Imperative commands are **error-prone**
  - Changes not easily reversible
    - E.g., how do you reverse the steps performed by a shell script
  - Impact may be unclear until after execution
- Imperative instructions more difficult to review in source control
  - Limiting collaboration and rollback options

Declarative Configuration

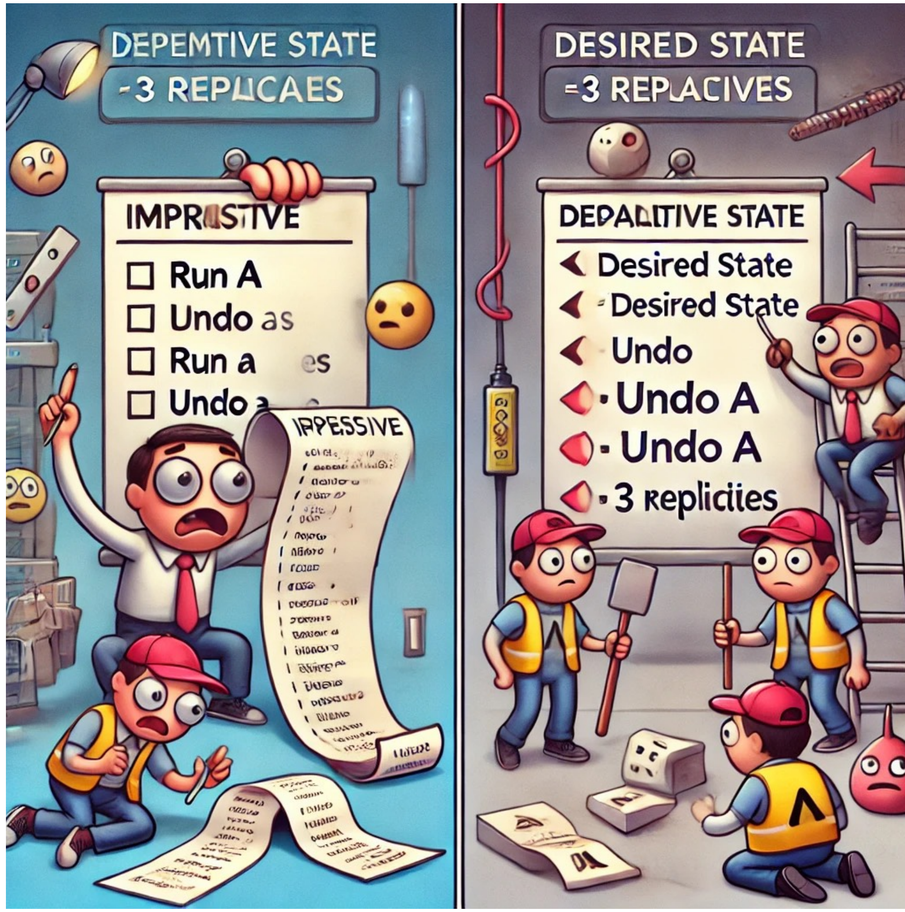
Specify the Desired State of the System

- State-Based Definition
  - Declarative configurations define the desired state of the system
  - E.g., `replicas = 3`, is clear and easy to understand without execution
  - Less room for error
- Declarative configuration integrated with version control (GitOps)
  - Single source of truth, code review, and testing

Declarative Configuration

Specify the Desired State of the System

- **Easy Rollback**
  - Restating previous declarative state to roll back
  - No need to provide reverse instructions
- Kubernetes **automatically matches** the actual state to the desired state



Self-Healing Systems

Continuously take action to retain desired state

- Continuous monitoring and recovery — **guard against failures**
- Reduced Operator Burden
  - Automates repairs, eliminating the need for manual intervention
- Developer Productivity
  - Less maintenance, more time developing new features
- Impact: **Improved system reliability**

What is Kubernetes?

Open-Source Container Orchestration Platform

kubernetes

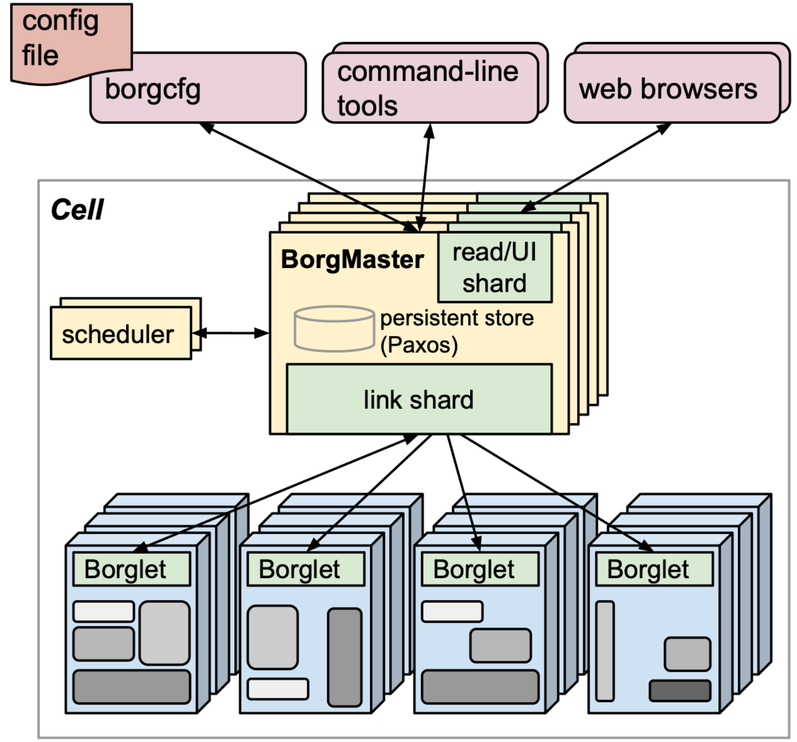
- Automatically distributes traffic (**load balance**) and exposes containers (**service discovery**)
- Automated (staged) **rollouts** and **rollbacks** (on failures)
- Perform **health checks** and **restart** failed containers
- Automatically **scale** applications up or down based on demand
- Automatically **mounts storage systems** (local or cloud-based)

Kubernetes (k8s)

Open-Source Container Orchestration Platform

kubernetes

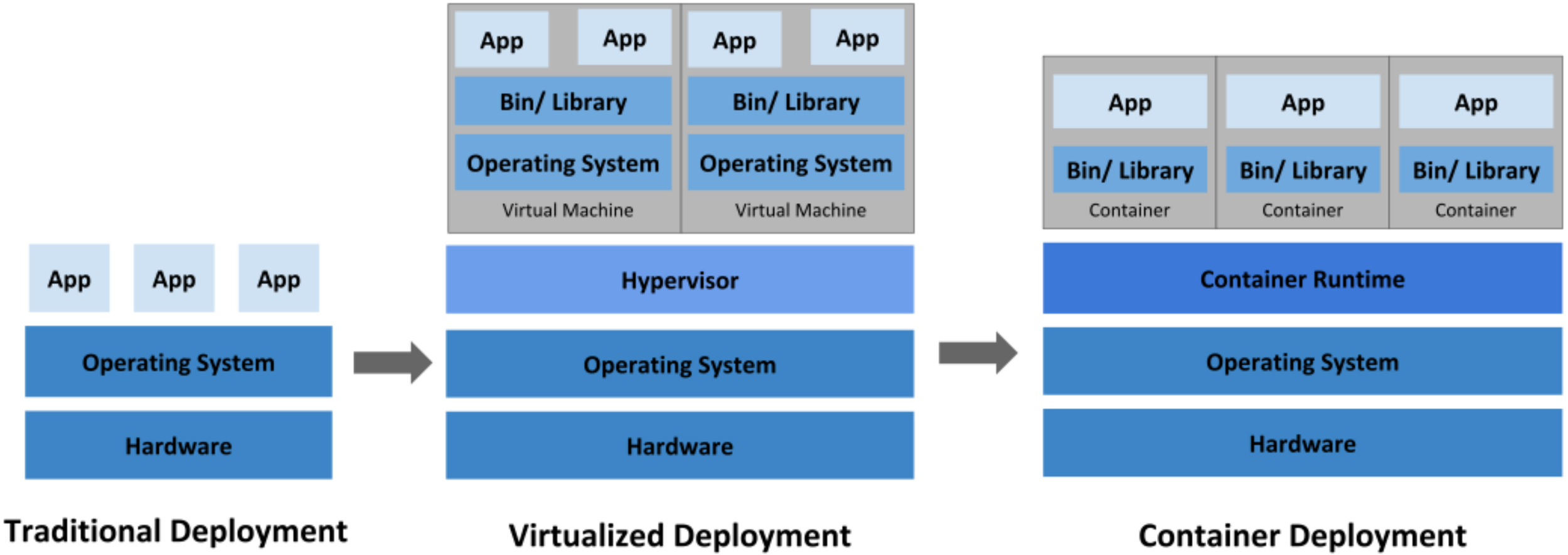
- “Kubernetes” is Greek, meaning **helmsman** or **pilot**
- Often abbreviated **k8s**
  - Eight letters between the **k** and the **s**
- Open sourced by Google
- Based on many years of experience with Google’s proprietary Borg systems
  - (Omega)



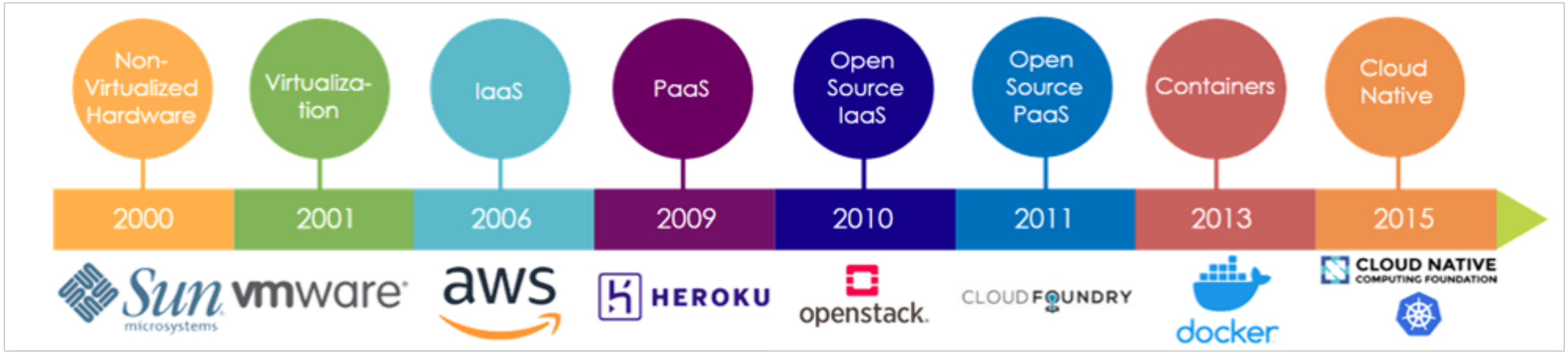
Once Configured, Kubernetes Operates Autonomously, Requiring Minimal Supervision

Historical Context for Kubernetes

Recap about Containers

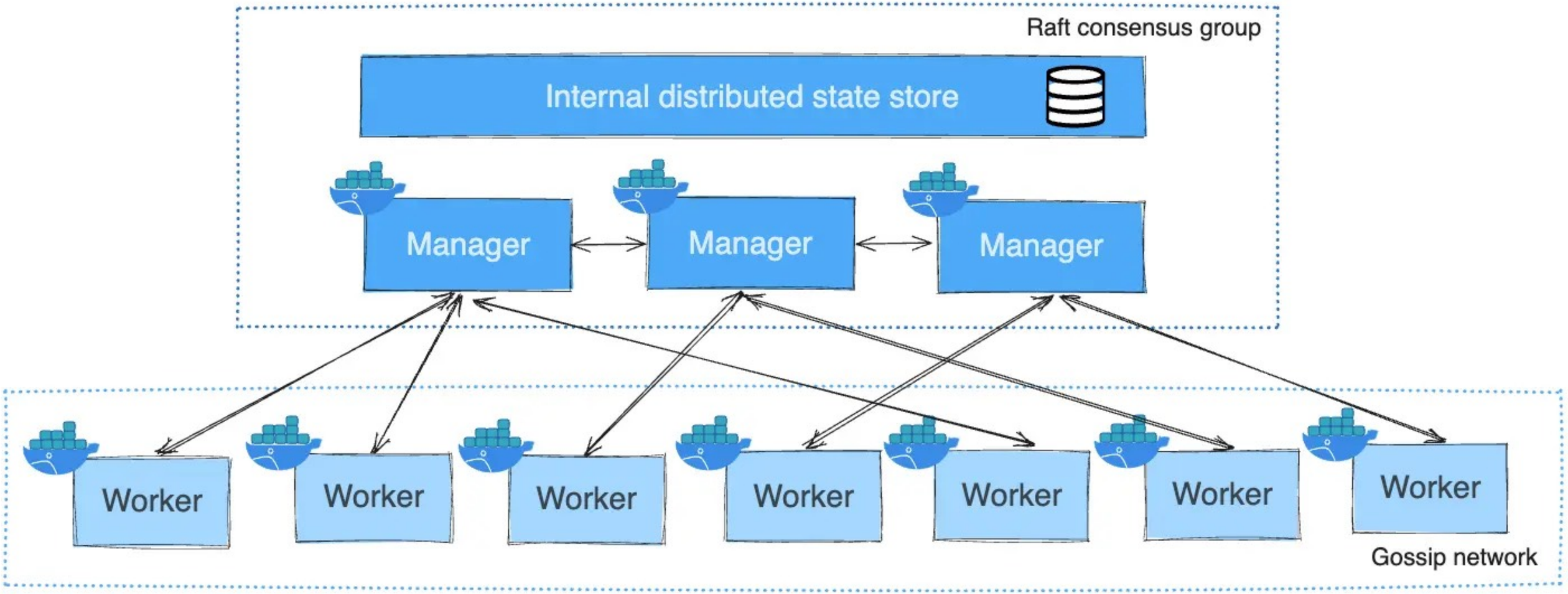


Another View of History

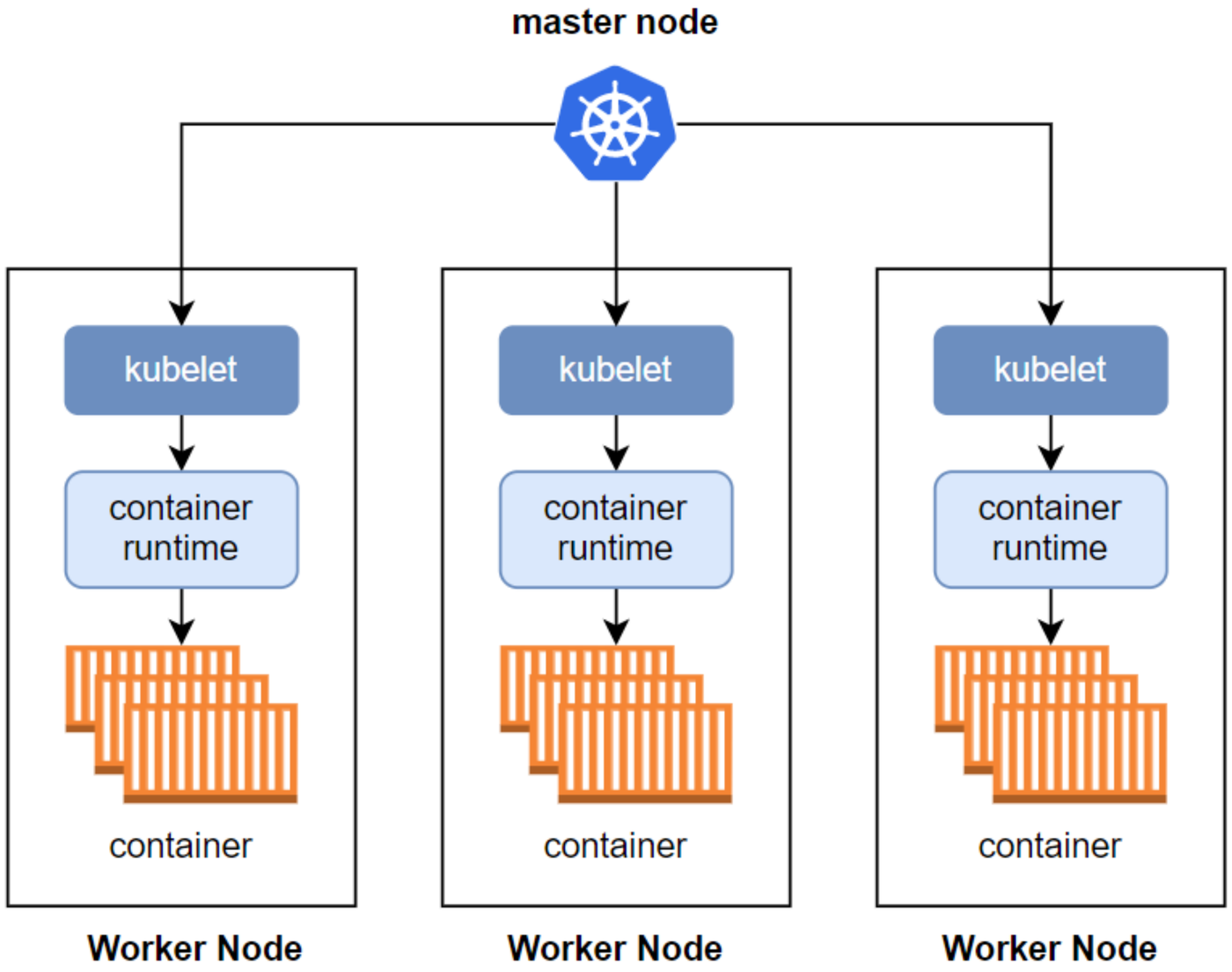


Kubernetes Architecture

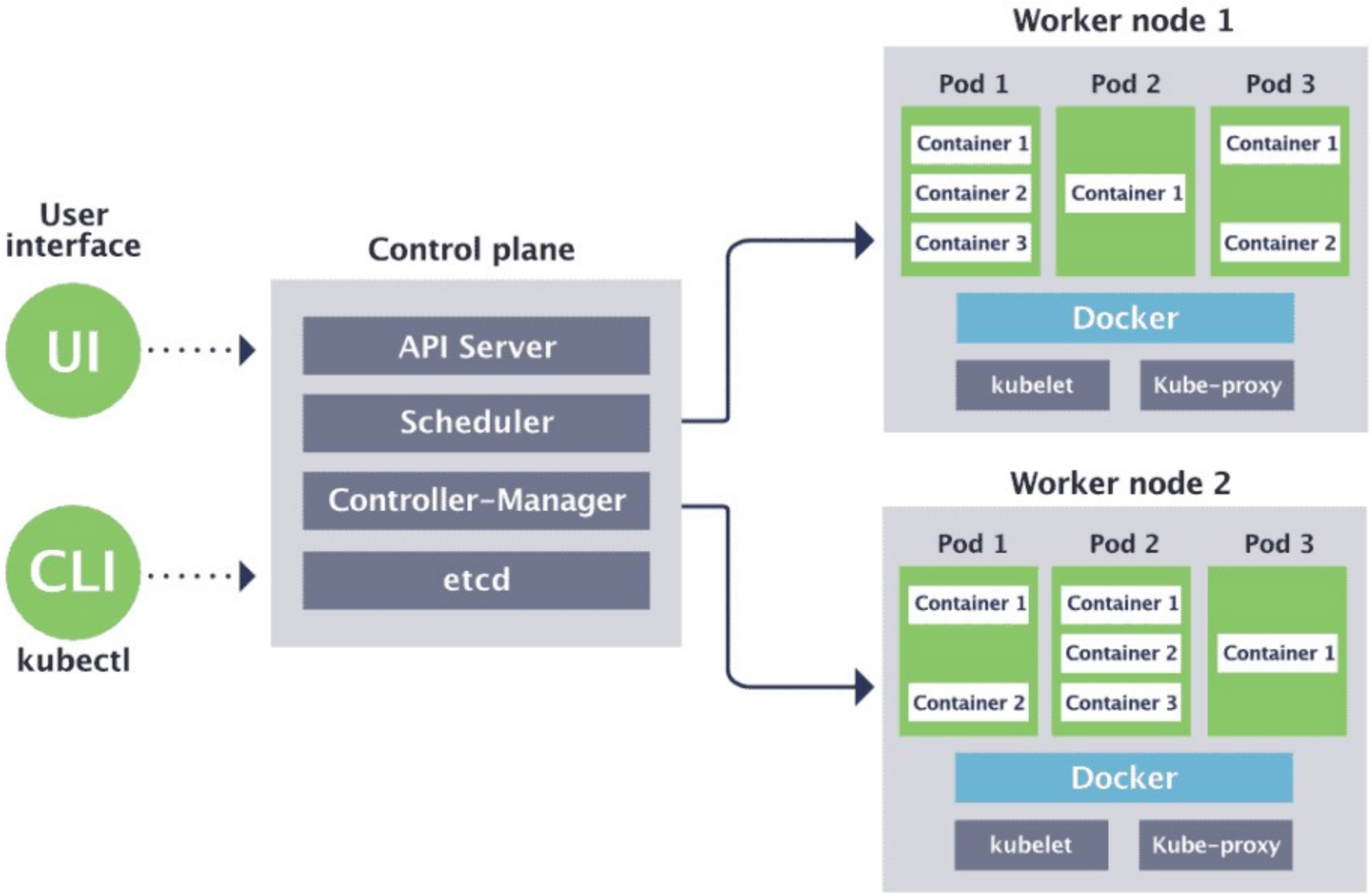
Recall: Docker Swarm



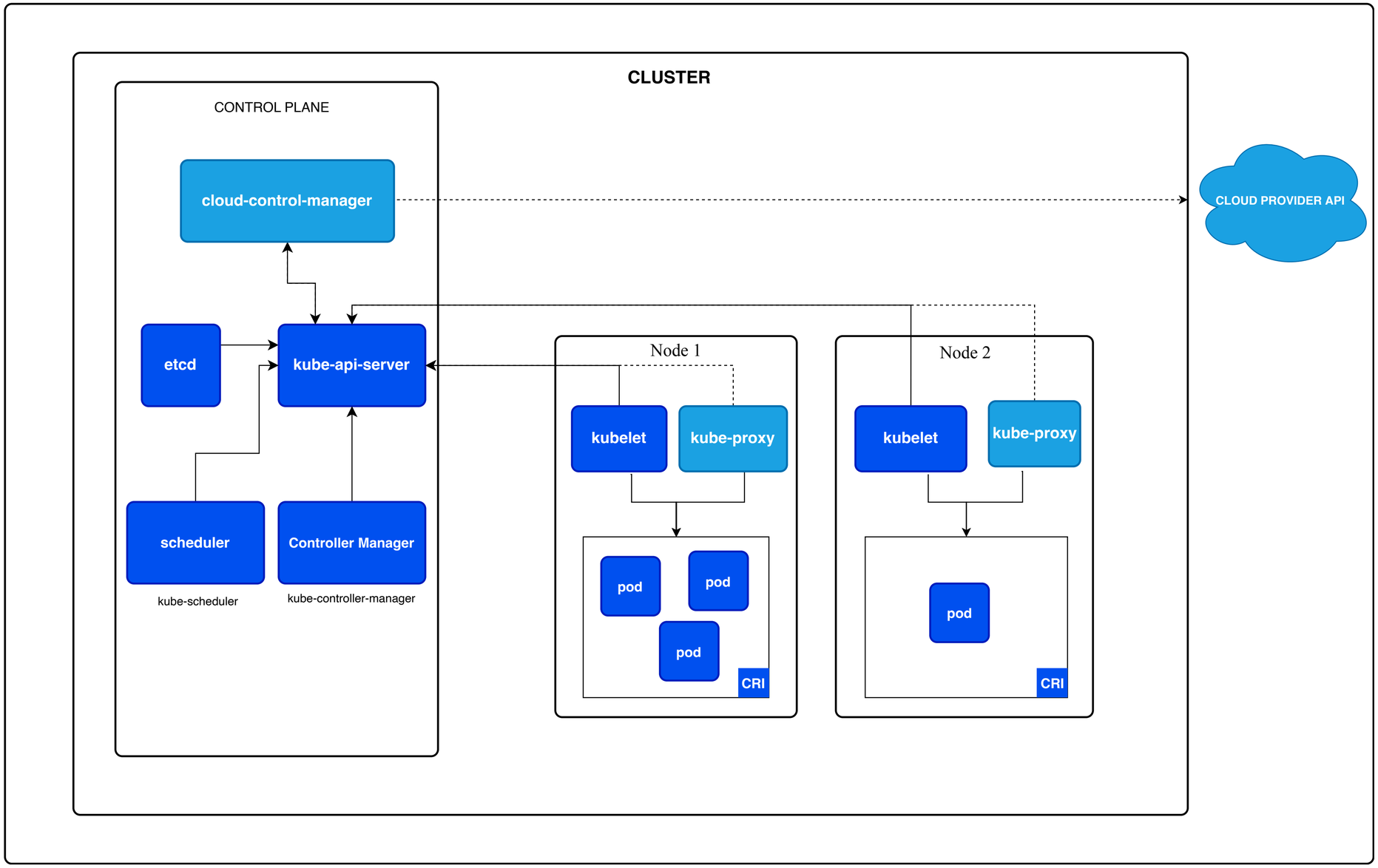
Kubernetes Cluster Architecture



Kubernetes Cluster Architecture



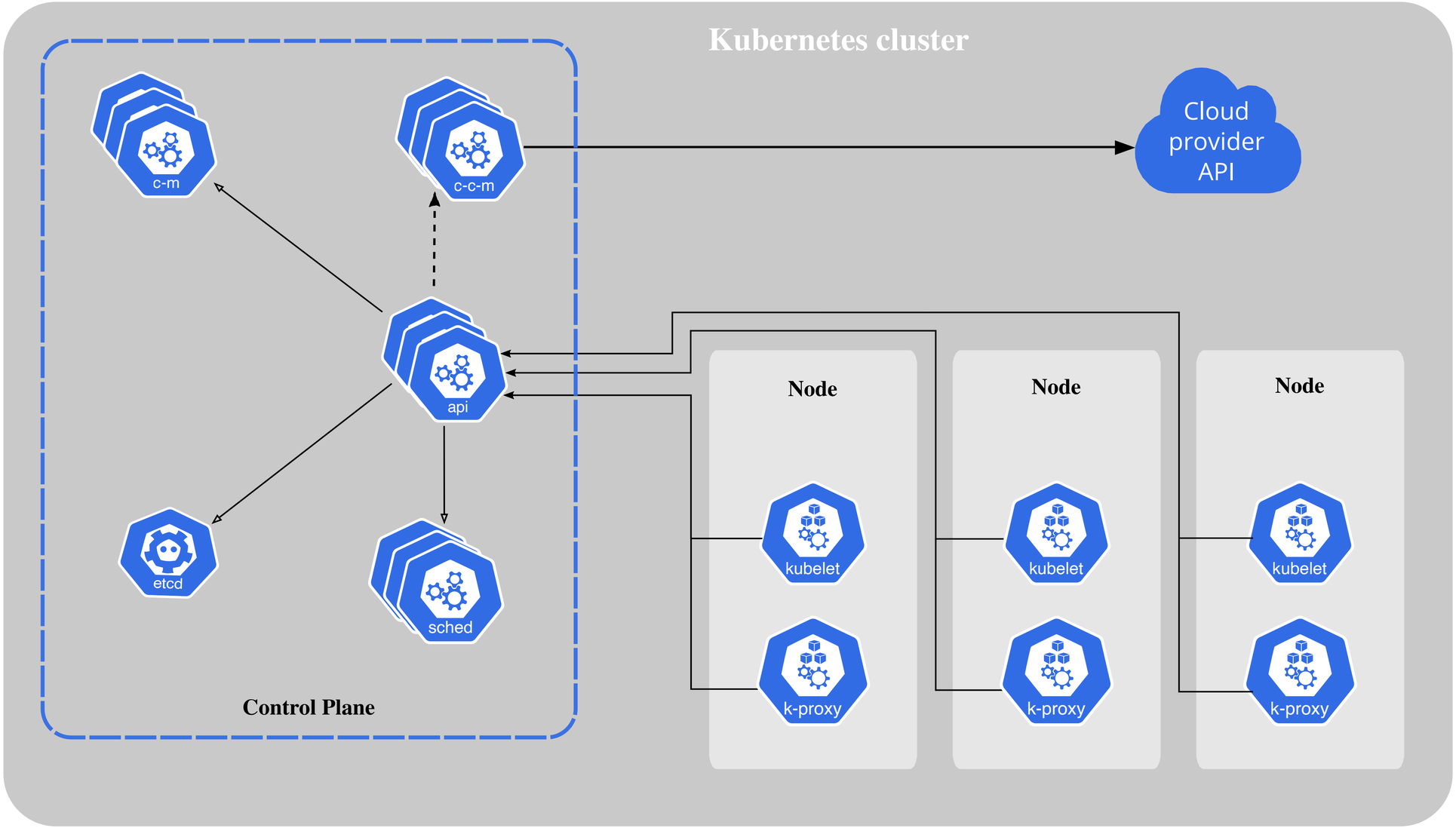
Kubernetes Cluster Architecture



Kubernetes Cluster Concepts

- **Control plane**
  - Manages worker nodes and Pods in the cluster
  - Production: **Replicated** for fault-tolerance and high availability
- Set of **worker nodes** (machines)
  - Run containerized applications
  - Nodes host **Pods**

Kubernetes Components



API server api

Cloud controller manager *(optional)* c-c-m

Controller manager c-m

etcd (persistence store) etcd

kubelet kubelet

kube-proxy k-proxy

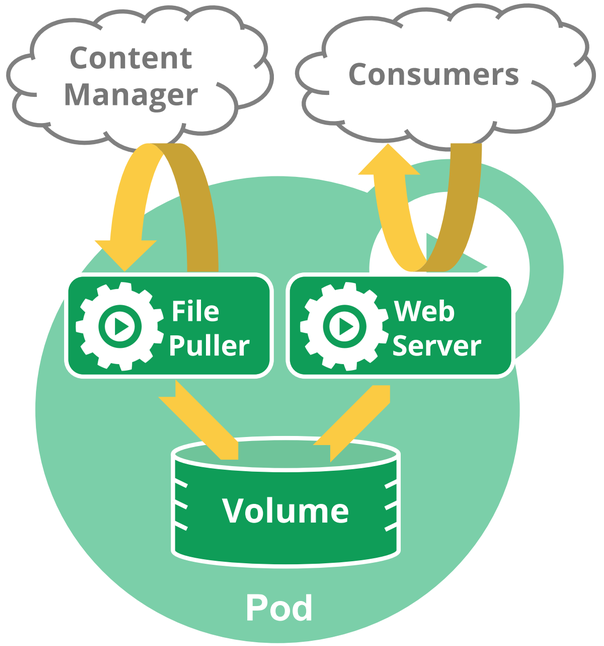
Scheduler sched

Control plane

Node

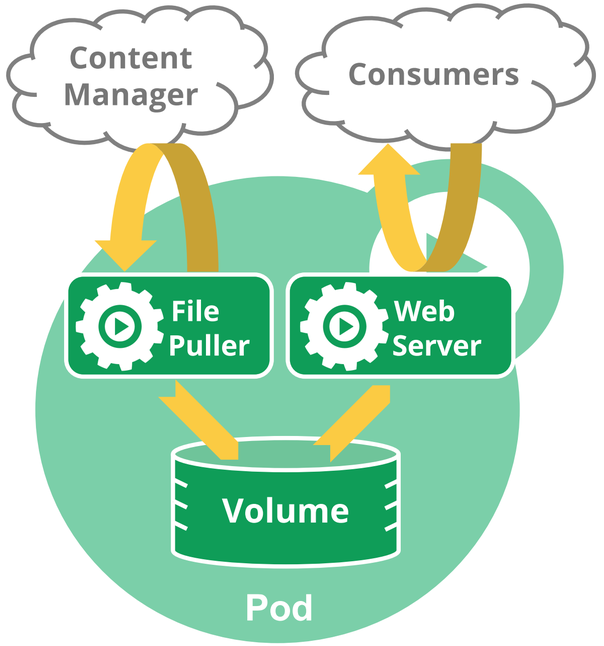
Pod — the smallest deployable unit

- Kubernetes manages Pods (not containers)
- **Pod** — group of one or more (tightly coupled) containers
  - **Shared** storage and network resources
  - **Co-located** and **co-scheduled**
  - On same physical or virtual machine
- To run pods, nodes must install a **container runtime**



Pod — the smallest deployable unit

- One-container-per-Pod model
  - Most common
  - Wrapper around a single container
- Multiple containers working together
  - **Shared** storage and network resources
  - **Co-located** and **co-scheduled**



Joke time

How do Kubernetes containers communicate?

With "Pod-casts" of course!

Why did the pod break up with the container?

Because it couldn't handle the traffic anymore!

Control Plane Components

Control Plane Components

Make global decisions about the cluster

- Scheduling
- Detecting and responding to cluster events
  - E.g., start new pod when Deployment's `replicas` field unsatisfied
- Control plane runs on machines separate from worker nodes (typically)

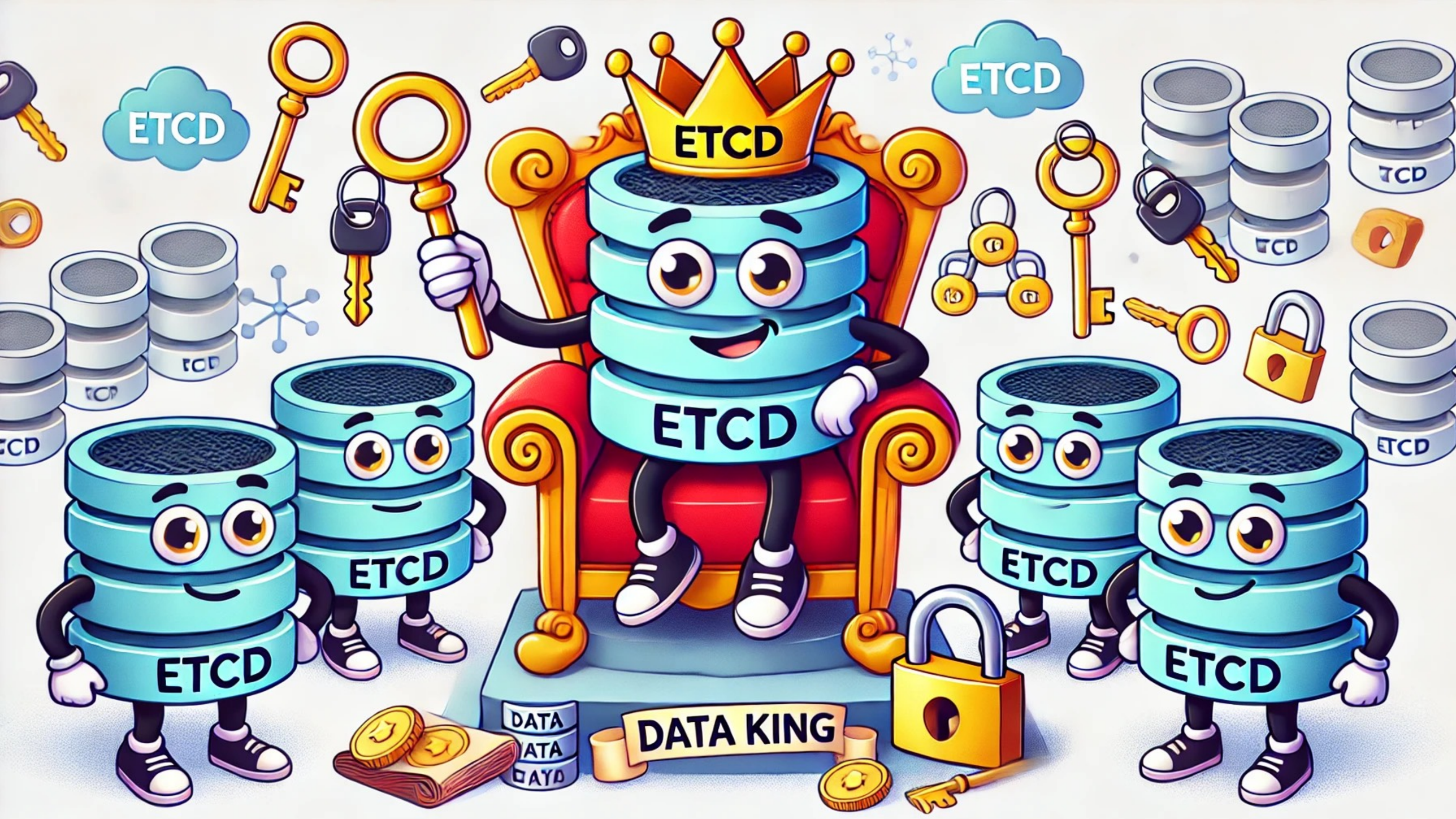
kube-apiserver

The Kubernetes API server

- The Hub in a Kubernetes cluster
- Exposes the Kubernetes API to
  - Worker nodes
  - Control plane components
  - External systems and tools
- Scale horizontally (deploying more apiserver instances)
  - Load balance traffic between instances

The Kubernetes API server

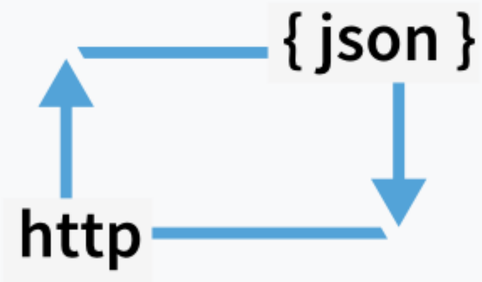
- Services REST operations to access the cluster's shared state
- Authentication, authorization, and request validation
- Validates and configures data for api objects
  - Pods, services, scheduler, replication controllers, etc


ETCD
ETCD
ETCD
ETCD
ETCD
ETCD
ETCD
ETCD
ETCD
DATA
DATA KING

Features

Simple interface

Read and write values using standard HTTP tools, such as curl



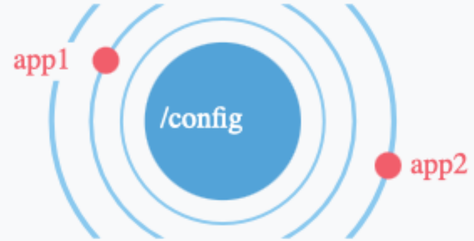
Key-value storage

Store data in hierarchically organized directories, as in a standard filesystem

```
/config
  └─ /database
/feature-flags
  ├─ /verbose-logging
  └─ /redesign
```

Watch for changes

Watch specific keys or directories for changes and react to changes in values



Strongly consistent, distributed key-value store

- Reliable (replicated) data store
- Tolerates **network partitions** and **machine failures**
  - Manual intervention to remain fault-tolerant
- Uses a leader to drive a consensus protocol
  - Raft (see DAT520 Distributed Systems)

Side Track:
My PhD Work

Autonomous Replication Management

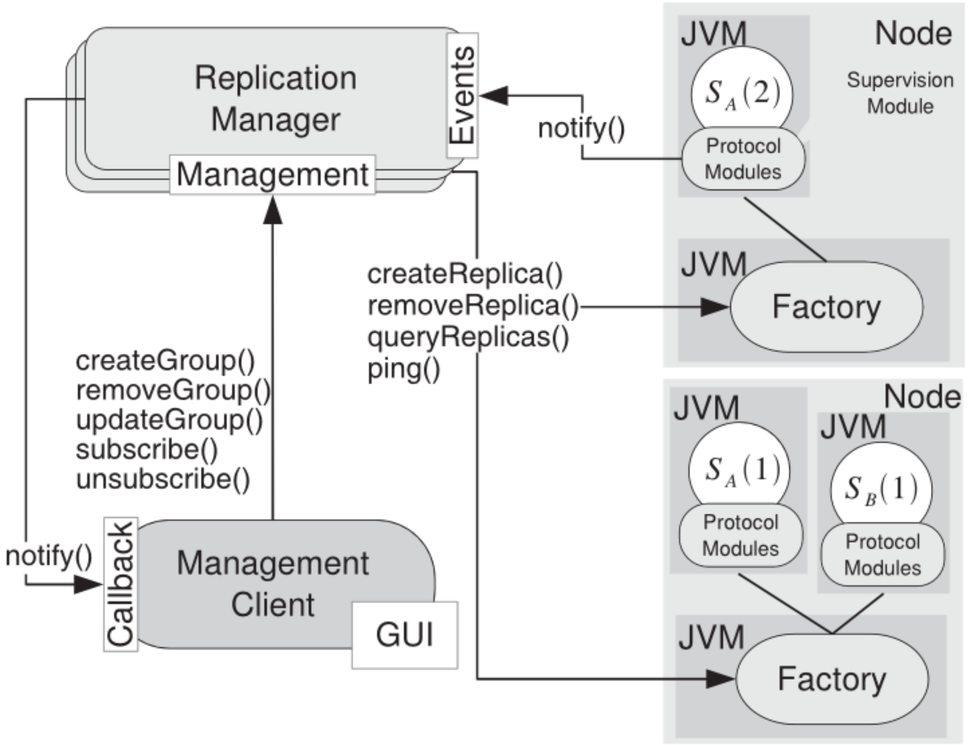


Figure 1. Overview of ARM components and interfaces.

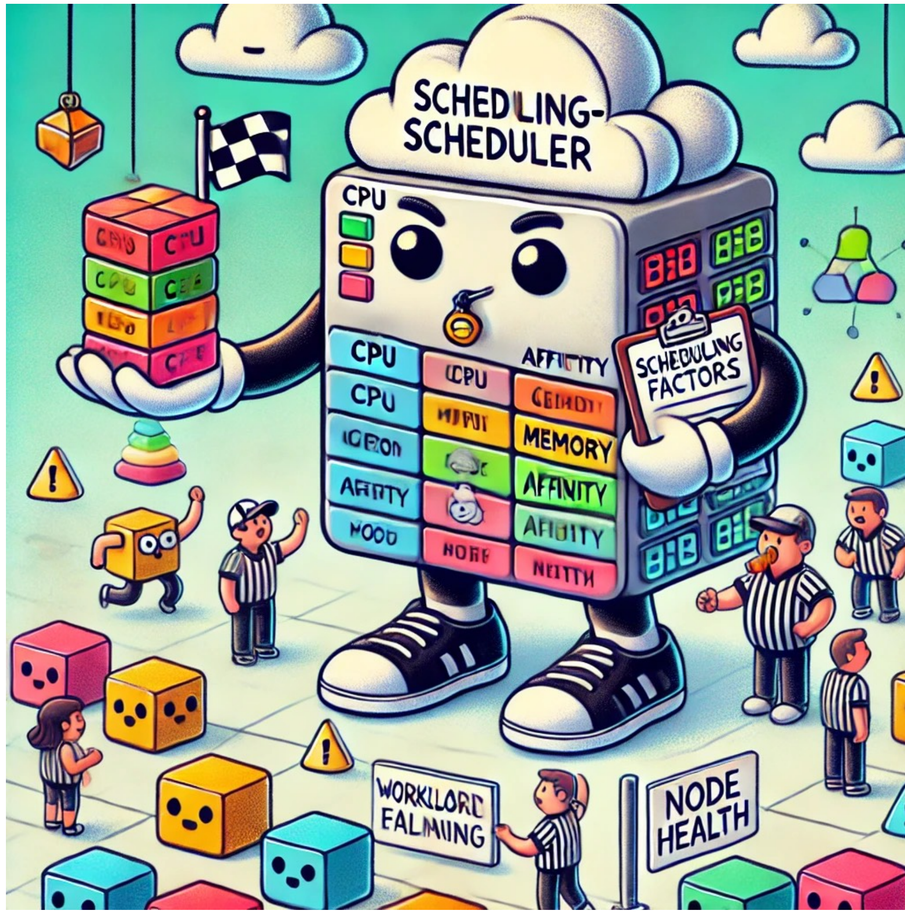
```
<Service name="ARM/ReplicationManager">
  <Param name="ServiceMonitorExpiration" value="3"/>
  <ProtocolModules>
    <Module name="GroupMembership"/>
    <Module name="StateMerge"/>
    <Module name="EGMI"/>
    <Module name="Supervision">
      <Param name="GroupFailureSupport" value="no"/>
      <Param name="RemoveDelay" value="5"/>
    </Module>
  </ProtocolModules>
  <DistributionPolicy class="DisperseOnSites"/>
  <ReplicationPolicy class="KeepMinimalInPartition">
    <Redundancy initial="3" minimal="2"/>
  </ReplicationPolicy>
</Service>
```

Figure 8. A sample service configuration description for the RM.

kube-scheduler

Optimize Pod Placement for Efficient Resource Use and Availability

- Monitors unassigned Pods
- Matches Pods to available worker nodes
- Based on resource requirements, constraints, and policies



Optimize Pod Placement for Efficient Resource Use and Availability

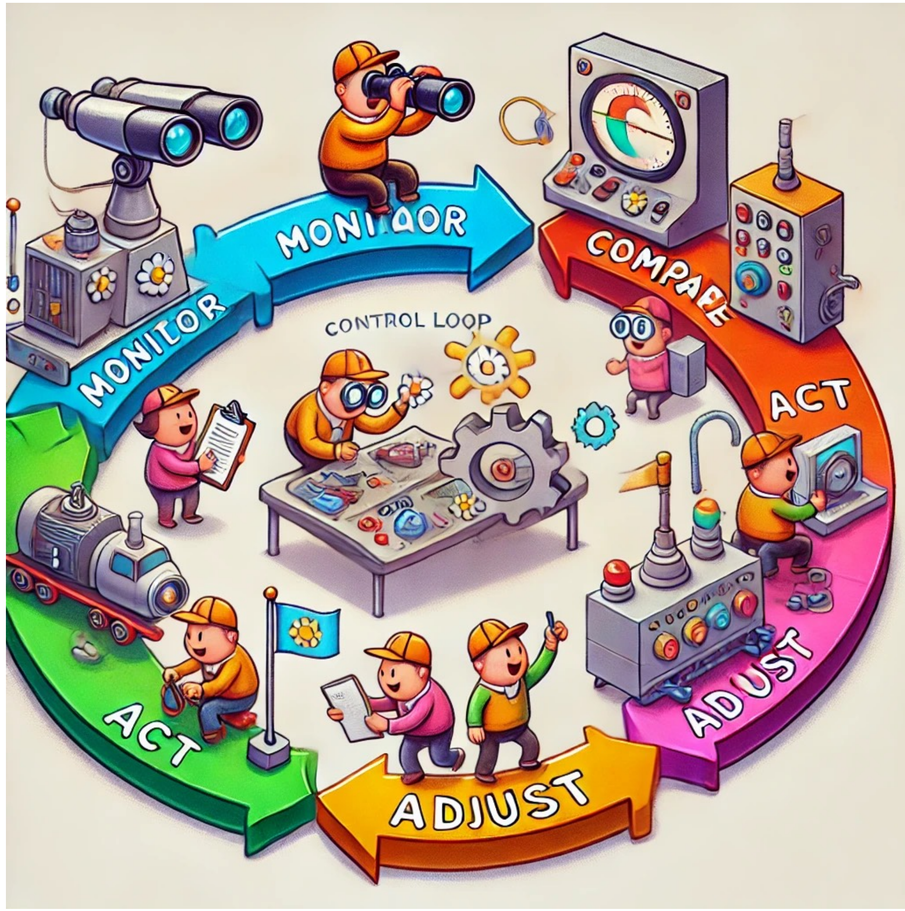
- Ensures that workloads are balanced across the cluster
- Scheduling based on factors like
  - resource utilization
  - data locality
  - inter-workload interference
  - affinity/anti-affinity rules
  - hardware/software/policy constraints
  - deadlines and node health

kube-controller-manager

kube-controller-manager

The Controller Concept — from Robotics and Automation

- Control loop — non-terminating loop to regulate the state of a system
  - System has *current state*
  - Specify the *desired state*
- Example: Thermostat
  - Current temperature
  - Specify desired temperature



kube-controller-manager

Controllers Track Kubernetes Resource Type (Object)

- Objects specify desired state
- Controllers are responsible for
  - moving closer to the desired state
- Action: send messages to API server to trigger changes (side effects) to reach desired state

kube-controller-manager

Example Controllers

- Node Controller
  - Detect and respond when nodes go down
- Job Controller
  - Watches for Job objects (one-off tasks)
  - Create Pods to run tasks to completion

A Job is a Kubernetes resource that runs one or more Pods to carry out a task.

cloud-controller-manager

cloud-controller-manager

Cloud-specific control logic (optional)

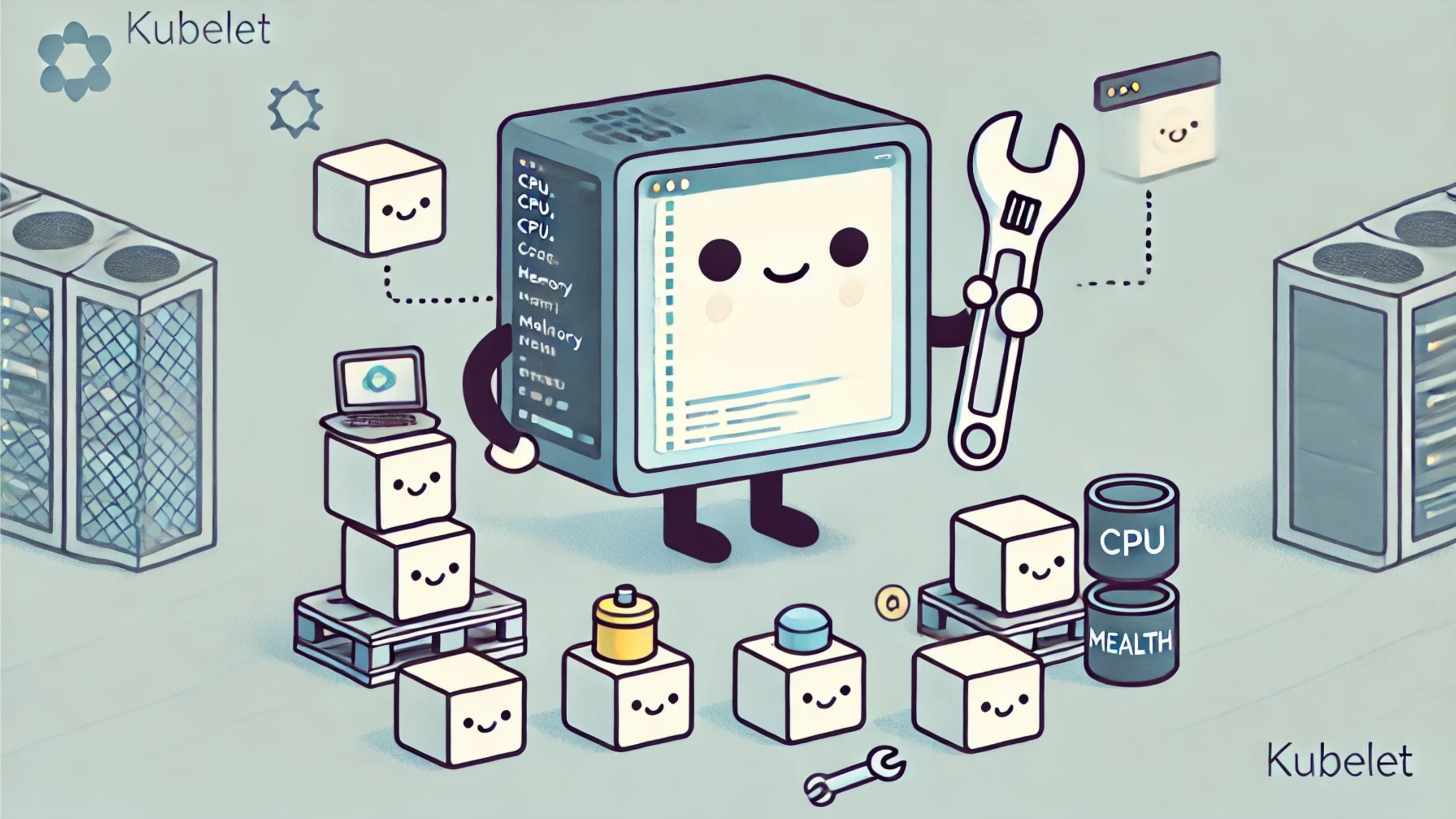
- Runs controllers specific to cloud provider
  - Works similar to kube-controller-manager
- Not used for on-premises or own laptop for learning/testing
- Node Controller
- Route Controller: sets up routes in underlying cloud infrastructure
- Service Controller: creating, updating, and deleting cloud load balancers

Joke time

Why did the DevOps engineer bring a ladder to the data center?

Because they wanted to reach new "heights" in horizontal scaling!

Node Components


Kubelet
CPU
CPU
CPU
CPU
MEALTH
Kubelet

Node Agent — runs on each node

- Register node with apiserver
- Node Resource Monitoring
  - Collects resource usage data like CPU and memory
  - Reports to control plane to help with scheduling and resource management

Node Agent — runs on each node

- Pod Management
  - Monitor pods assigned to kubelet's node
  - **Ensures containers** described in PodSpecs are **running** and **healthy**
- Pod Specification Compliance
  - Specify which containers should be running
  - The kubelet receives pod specifications from control plane

Node Agent — runs on each node

- Container Lifecycle Management
  - Perform container health checks and liveness or readiness probes
  - Start, stop and manage containers
    - Container in a pod crashes — the kubelet can restart it

kube-proxy

kube-proxy

Maintains network rules for the node (optional)

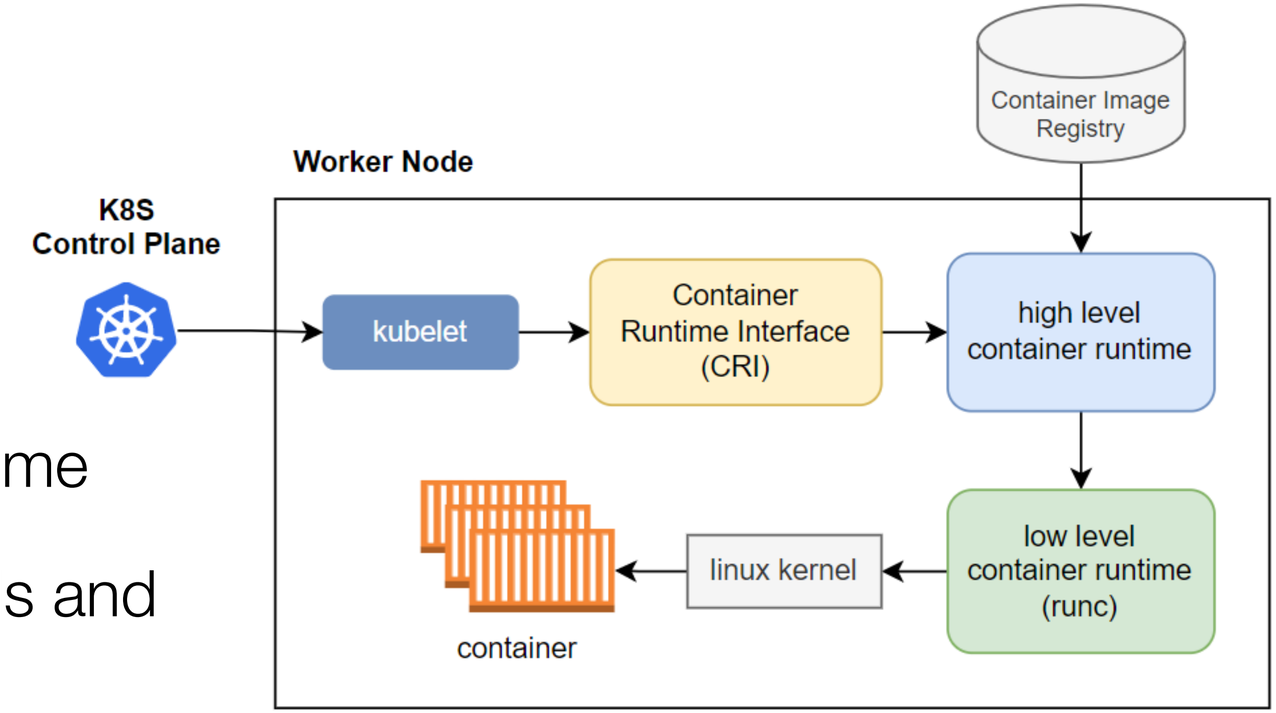
- Allow network communication to Pods
  - inside or outside cluster
- Uses OS packet filtering layer (if available)
- Don’t need kube-proxy if a network plugin implements packet forwarding

Container Runtime Interface

Container Runtime Interface (CRI)

Enable kubelet to use different container runtimes

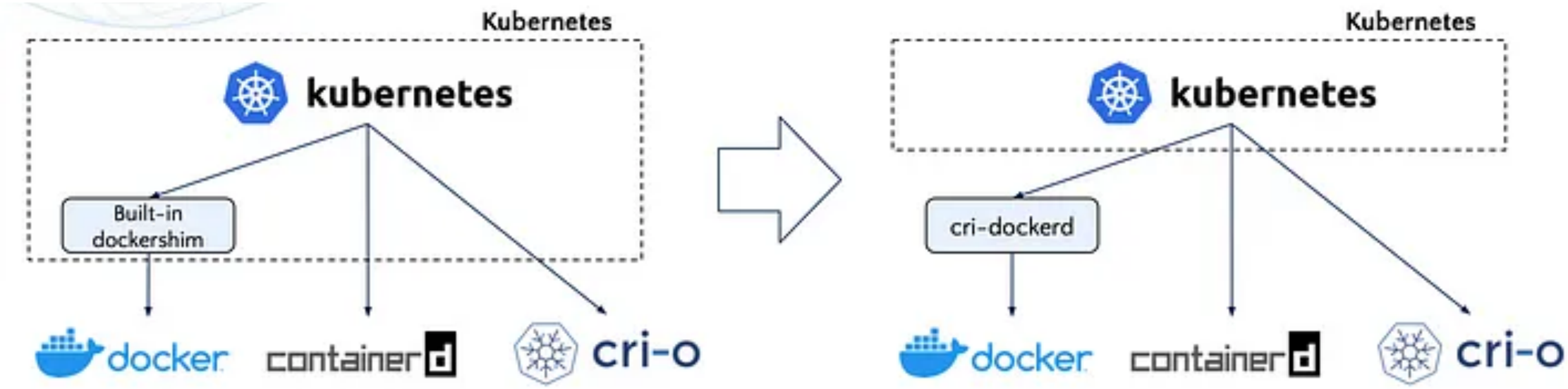
- Each node needs a container runtime
- Kubernetes CRI defines
  - gRPC protocol for communication between kubelet and container runtime
  - The kubelet uses CRI to launch Pods and their containers



Container Runtime Interface (CRI)

Alternatives to Docker as Kubernetes runtime

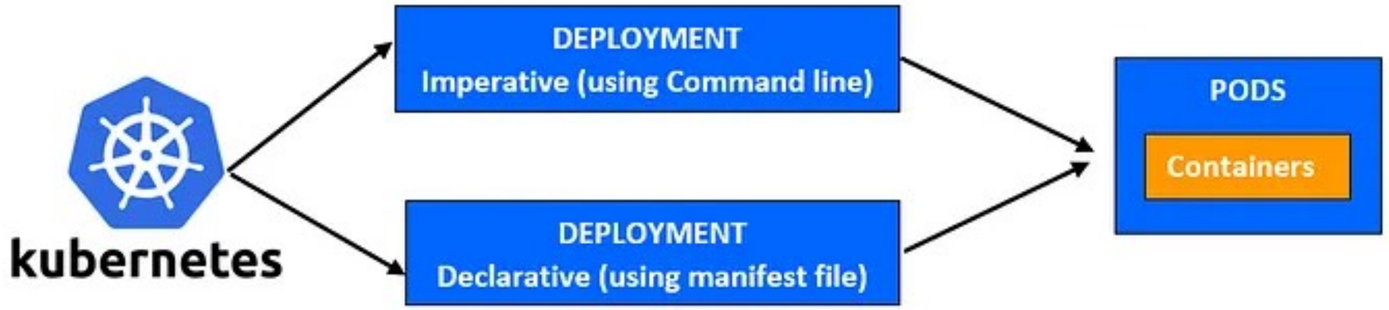
- Kubernetes v0.2 (2014): Docker only supported runtime
- Kubernetes v1.5 (2016): Introduced Container Runtime Interface
- Kubernetes v1.24 (2022): Dropped *built-in* support for Docker



kubectl

Create and manage Kubernetes objects

Management technique	Operates on	Recommended environment	Supported writers	Learning curve
Imperative commands	Live objects	Development projects	1+	Lowest
Imperative object configuration	Individual files	Production projects	1	Moderate
Declarative object configuration	Directories of files	Production projects	1+	Highest



Imperative commands

- `kubectl create deployment nginx --image nginx`
- Advantages:
  - Single word action
  - Single step to make changes to cluster
- Disadvantages:
  - No change review process and audit trail

Imperative object configuration

- `kubectl create -f nginx.yaml`
- Advantages:
  - Config stored in git
  - Change review process and audit trail
- Disadvantages:
  - Must understand object schema (yaml format)
  - Updates must be reflected in config files, or they will be lost

Declarative object configuration

- `kubectl diff -f configs/`
  `kubectl apply -f configs/`
- Advantages:
  - Support operating on directories
  - Automatically detect operation type (create, patch, delete) per object
- Disadvantages:
  - Harder to debug and understand
  - Partial updates mean lead to complex merge and patch operations

Joke time

Why do Kubernetes nodes make terrible comedians?

Because they always lose their connection to the audience!

Scaling Your Service and Your Teams

Decoupled Architecture

Decouple Components from each other

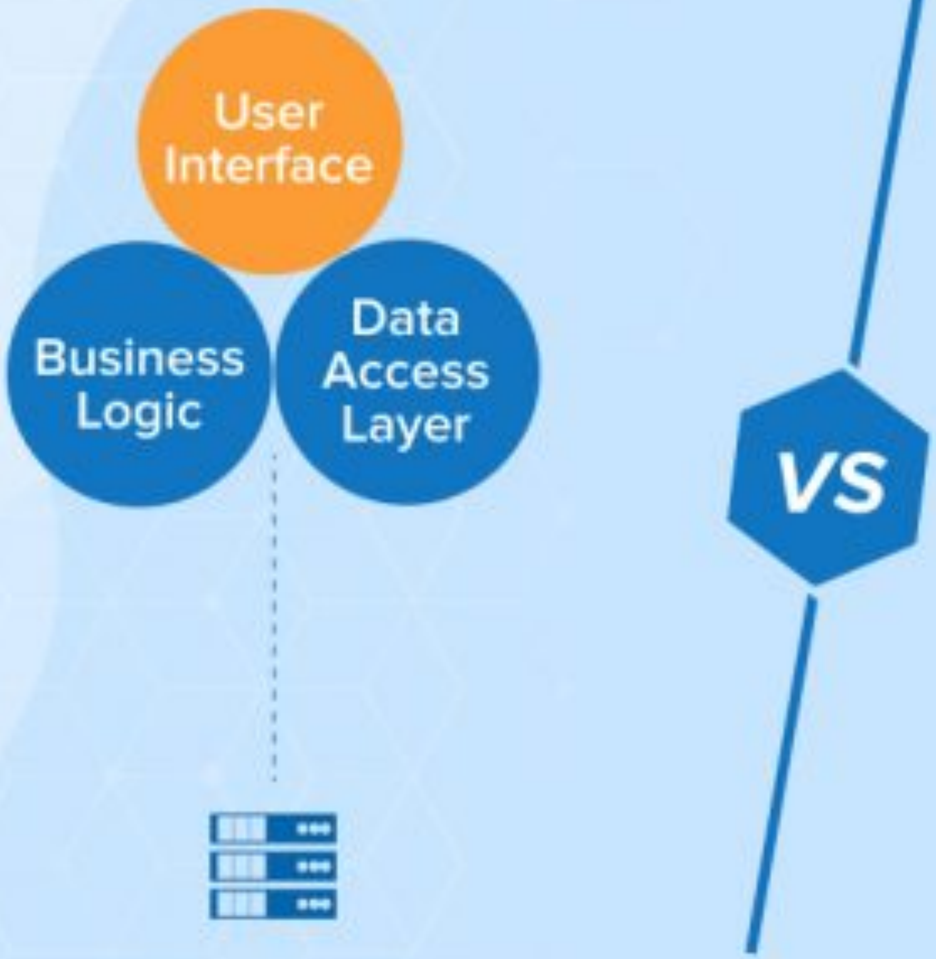
- Component Isolation
  - Make components function independent of each other
  - Define API for component to interact with other (consumer) components
    - Hide internal details
- Allows scaling of individual services without impacting other components

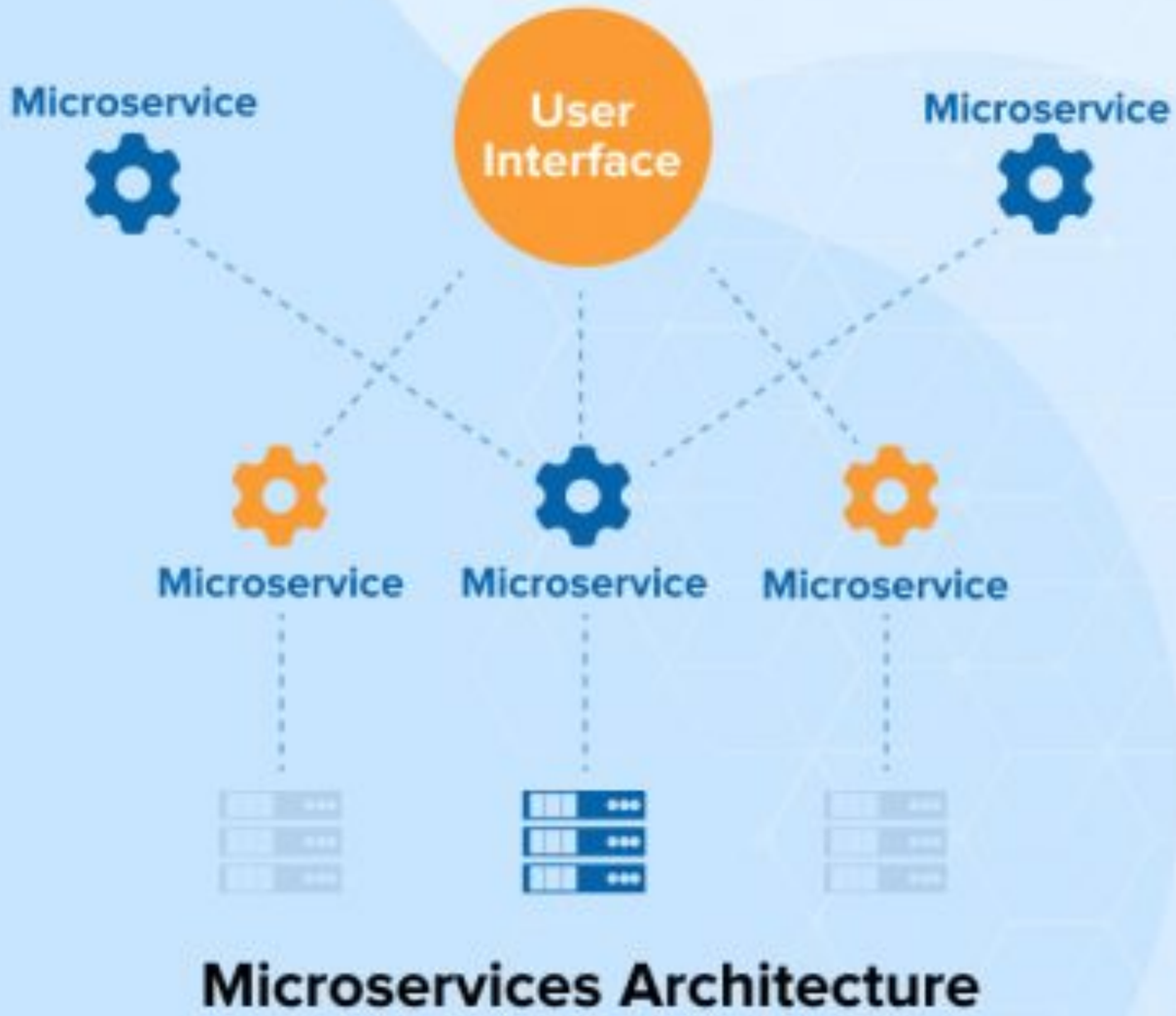
Decoupled Architecture

Team Efficiency

- APIs enable dev teams to focus on smaller, manageable microservices
- Reducing the need for cross-team coordination
- Clear API boundaries lowers the communication demands across teams
- Faster development and deployment

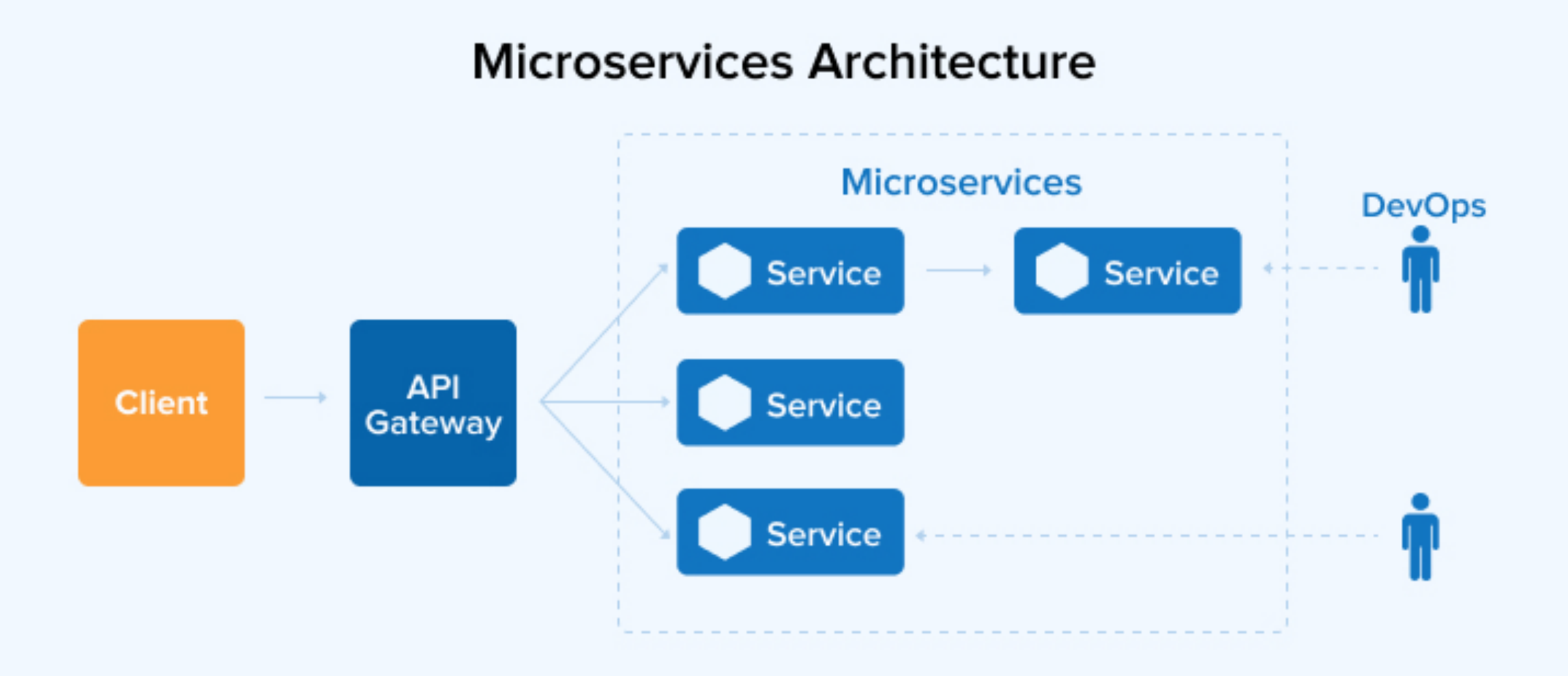
Monolithic vs Microservices Architecture





Monolithic Architecture

Microservices Architecture



Questions?

References

- https://www.cncf.io/blog/2019/08/19/how-kubernetes-works/