1 Software as a Service (SaaS)
2 Platform as a Service (PaaS)
CLOUD COMPUTING
3 Infrastructure as a Service (IaaS)
1 On-demand Self-service
2 Resource Pooling
3 Broad Network Access
4 Rapid Elasticity
5 Measured Services

PMCA506L: Cloud Computing

Module 5 : Orchestration

Courtesy : *Ming Lian , Dogules E Comer & Other Sources of Internet*

Cloud Orchestration

- Industry uses the term *orchestration* to refer to an automated system that coordinates the many subsystems needed to configure, deploy, operate, and monitor software systems and services.

- Cloud orchestration is likewise useful in cloud provisioning, empowering organizations to automate the delivery of important cloud resources to their end users.

- Orchestration in which an automated system configures, controls, and manages all aspects of a service.

A move to containers

- **Rapid creation**
  less time to create a container than to create a VM

- **Short lifetime**
  Unlike a VM that remains in place semi-permanently once created, a container is ephemeral.

  A container resembles an application process: a typical container is created when needed, performs one application task, and then exits.

- **Replication**
  Replication is key for containers.

When demand for a particular service increases, multiple containers for the service can be created and run simultaneously, analogous to creating multiple concurrent processes to handle load.

When demand for a service declines, unneeded container replicas can be terminated.

Orchestrator

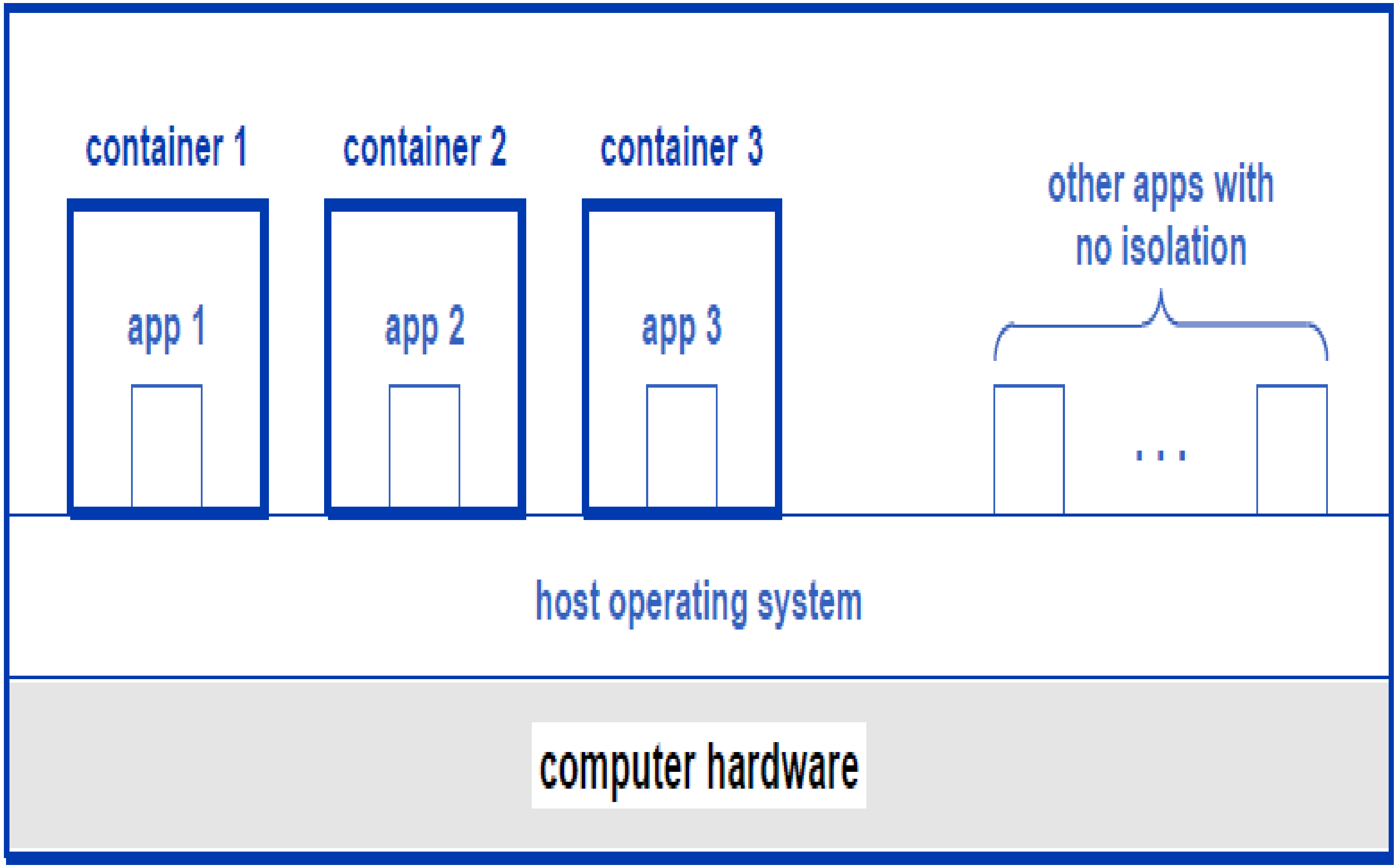
- Dynamic scaling of services
- Coordination across multiple servers
- Resilience and automatic recovery

Container Approach For Isolated Apps

- When an app uses operating system mechanisms to enforce isolation, the app remains protected from other apps.

- *Each container provides isolation, which means an application in one container cannot interfere with an application in another container.*

Container Approach For Isolated Apps



Docker

- Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers.
- **Docker provides the ability to package and run an application in a loosely isolated environment called a container.**
- Using Docker, you can quickly deploy and scale applications into any environment and know your code will run.
- **Docker is primarily a platform for building, packaging, and running containerized applications.**
- **It provides tools for creating and managing container images, as well as for running containers on a single host.**

Docker Containers

- Tools that enable rapid and easy development of containers.
- An extensive registry of software for use with containers.
- Techniques that allow rapid instantiation of an isolated app.
- Reproducible execution across hosts.

Development tools

- The Docker technology provides an easy way to develop apps that can be deployed in an isolated environment.
- Docker uses a high-level approach that allows a programmer to combine large pre-built code modules into a single image that runs when a container is deployed.
- A separate image file must be created for each app. When an imagefile runs,a container is created to run the app. We say the app has been *containerized*.

Extensive registry of software

- *Docker Hub*, an extensive registry of open source software that is ready to use.
- The registry enables a programmer to share deployable apps.
- A user (or an operator) can combine pieces from the registry in the same way that a conventional program uses modules from a library.

Rapid instantiation

- Docker uses an early binding approach that combines all the libraries and other run-time software that will be needed to run the container into an image file.
- Unlike a VM that must wait for an operating system to boot, a container can start instantly.
- Early binding approach means a Docker container does not need the operating system to perform extra work when the container starts.
- As a result, the time required to create a container is impressively small.

Reproducible execution

- Once a Docker container has been built, the container image becomes *immutable.*
- The image remains unchanged, independent of the number of times the image runs in a container.
- Furthermore, because all the necessary software components have been built in, a container image performs the same on any system.
- As a result, container execution always gives reproducible results.

Docker Containers

- *Docker technology makes app development easy, offers a registry of pre-built software, optimizes the time required to start a container, and guarantees reproducible execution.*

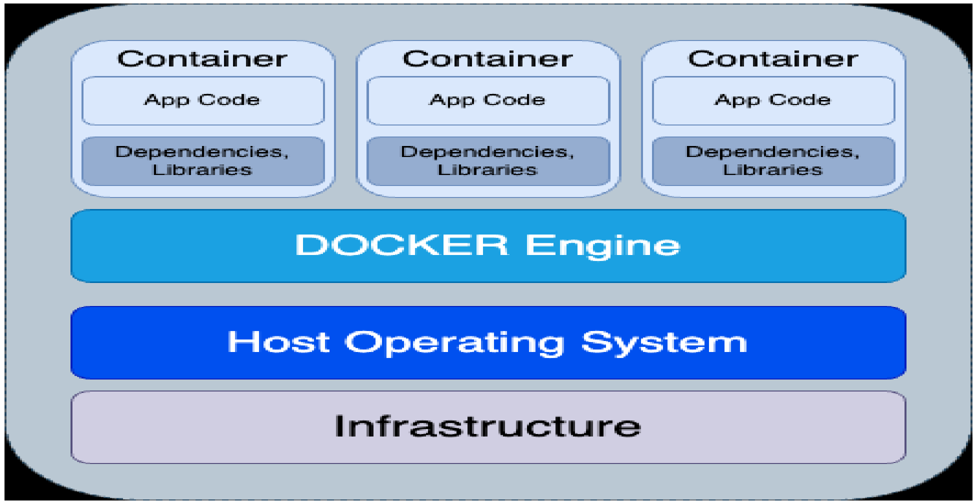
Docker Terminology And Development Tools

Docker Term	Meaning	Analogous To
image	A file that contains binary code for a container along with its dependencies	a.out file (MacOS/Linux) .exe file (Windows)
layer	One piece of software added to an image as it is built	a software module
container	An instance of an image	an executing process
Dockerfile	A specification for how to build an image	Makefile (Linux)
docker build	A command that constructs an image according to a Dockerfile	"make" program (Linux)
docker run X	A command that runs image X as a container	launch app X

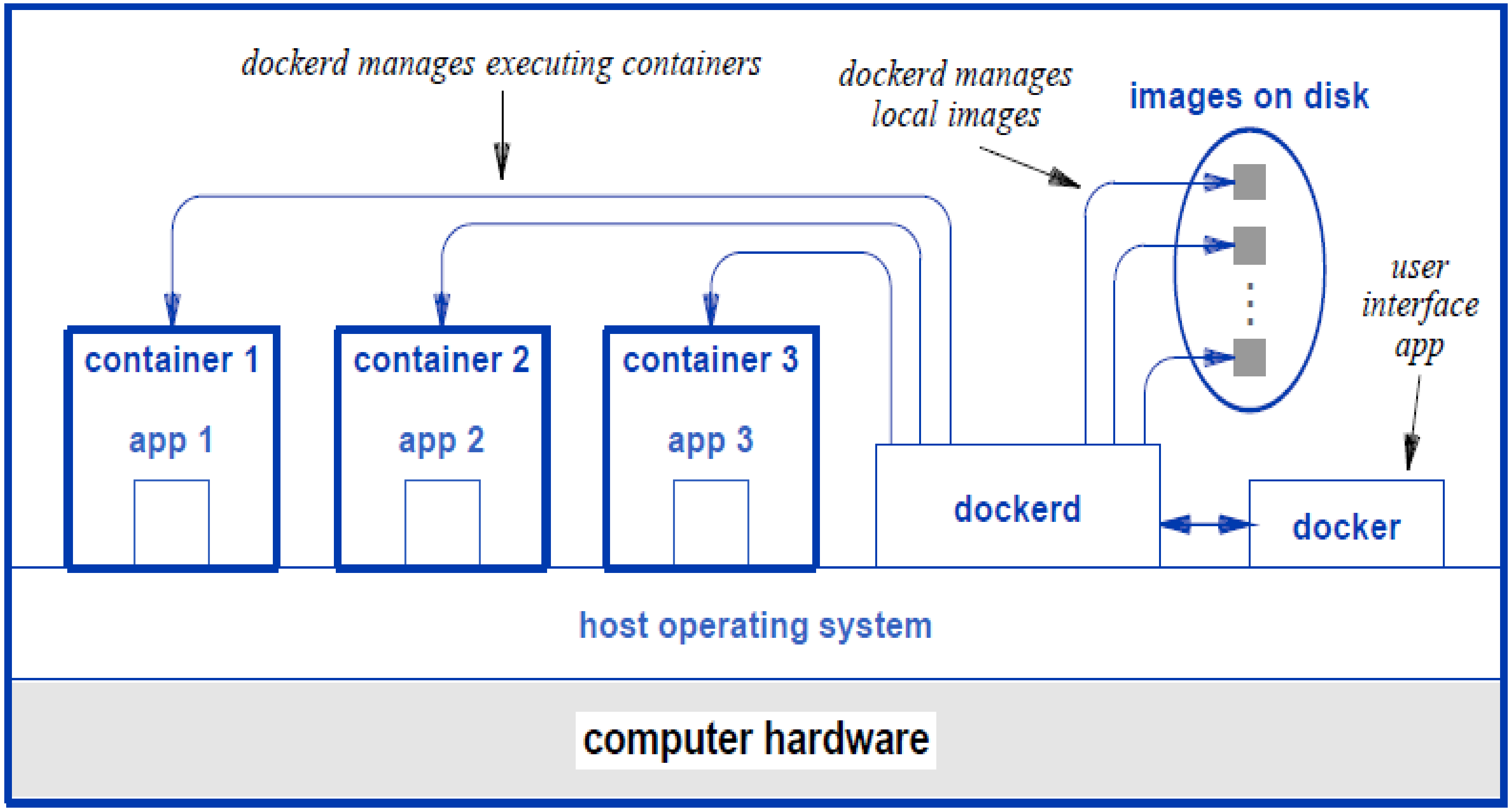
Docker Software Components

- Containers operate under the control of an application known as a *Docker daemon* (*dockerd*).

- Docker provides a user interface program, Collectively, the software is known as the ***Docker Engine*.**



Docker daemon, *dockerd*



Docker daemon, *dockerd*

- The dockerd program, which remains running in background at all times, contains several key subsystems.

- dockerd provides two interfaces through which a user can make requests and obtain information

  RESTful interface

  CommandLineInterface(CLI)

Docker commands

Command	Meaning
docker build .	Read Dockerfile in the current directory and follow the directives to construct an image
docker images	List all images on the local machine
docker run	Create and start a container running a specified image
docker ps	List all currently running containers on the system
docker pull	Download a copy of an image from a registry
docker stop	Stop one or more running containers
docker start	Restart one or more stopped containers

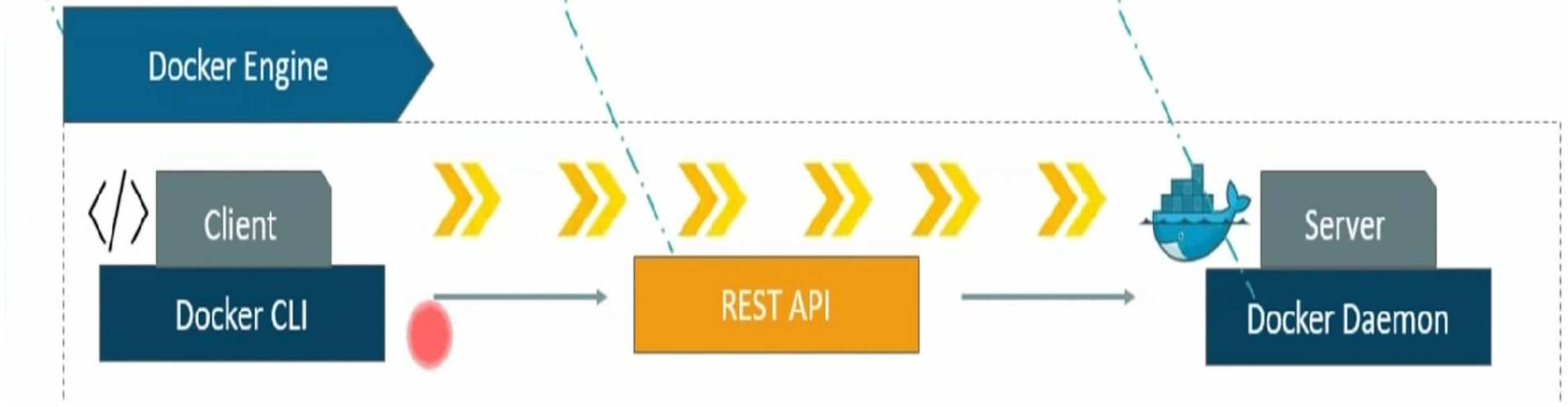
How Does Docker Work?

Docker Engine uses a Client Server Architecture.

Docker Engine is simply the docker application that is installed on your host machine.

The Client- Server architecture communicates using a REST API.

Docker Daemon checks the requests to manage the containers.



Kubernetes (K8s)

- Kubernetes, often abbreviated as K8s(kates), is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications. across a **cluster of machines**.

- Originally developed by Google and later donated to the Cloud Native Computing Foundation (CNCF)

- It focuses on automating container orchestration, scaling, and high availability.

Key features and concepts of Kubernetes

- **Container Orchestration**:
- Kubernetes provides a way to manage and orchestrate containers, primarily Docker containers, but it can also support other container runtimes.
- It handles the deployment and scaling of containers, ensuring high availability and efficient resource utilization.

Cluster Management

Kubernetes organizes containers into clusters, which are made up of one or more nodes (physical or virtual machines).

Clusters provide a framework for deploying and managing containerized applications.

Service naming and discovery

- Kubernetes allows a service to be accessed through a domain name or an IP address.
- Once a name or address has been assigned, applications can use the name or address to reach the container that runs the service
- Typically, names and addresses are configured to be global, allowing applications running outside the data center to access the service.

Load balancing

- Kubernetes does not limit a service to a single container.
- Instead, if traffic is high,
- Kubernetes can automatically create multiple copies of the container for a service, and use a *load balancer* to divide incoming requests among the copies.

Storage orchestration

- Kubernetes allows an operator to mount remote storage automatically when a container runs.

- The system can accommodate many types of storage, including local storage and storage from a public cloud provider.

Optimized container placement

- When creating a service, an operator specifies a cluster of servers(called *nodes*) that Kubernetes can use to run containers for the service.
- The operator specifies the processor and memory (RAM) that each container will need.
- Kubernetes places containers on nodes in the cluster in a way that optimizes the use of servers.

Automated recovery

- Kubernetes manages containers.
- After creating a container, Kubernetes does not make the container available to clients until the container is running and ready to provide service.
- Kubernetes automatically replaces a container that fails, and terminates a container that stops responding to a user-defined health check.

  (**Self-Healing**)

Rolling Updates and Rollbacks

- Kubernetes supports rolling updates, enabling zero-downtime updates for applications.
- A user can create a new version of a container image, and tell Kubernetes to start replacing running containers with the new version
- If issues arise during an update, Kubernetes allows for rollbacks to the previous working version.

Management of configurations and secrets

- Kubernetes separates management information from container images, allowing users to change the information needed for configuration and management without rebuilding container images.

- Kubernetes allows one to store sensitive information, such as passwords, authentication tokens, and encryption keys.

Limits On Kubernetes Scope

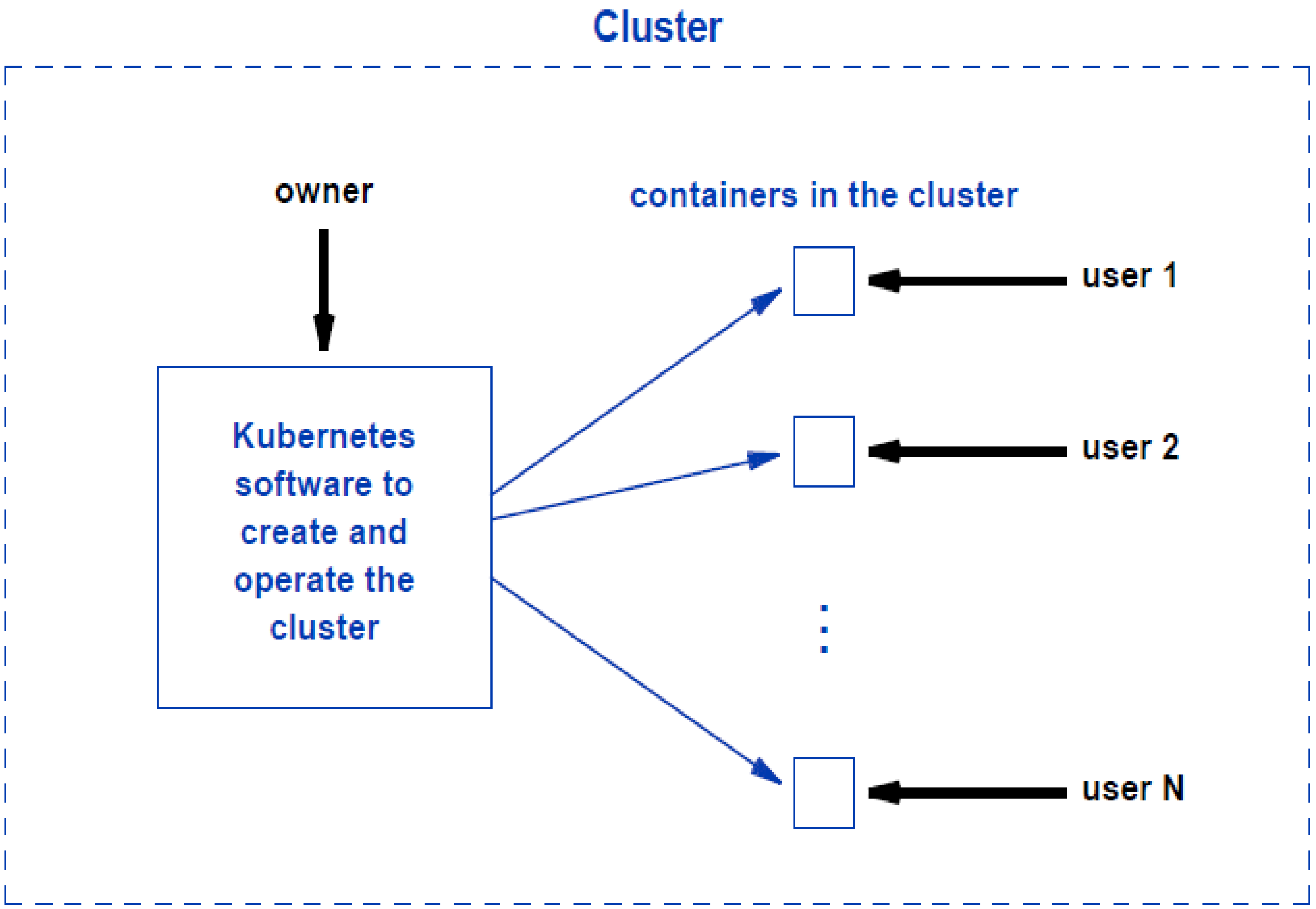
- Does not focus on a specific application type
- Does not manage source code or build containers
- Does not supply event-passing middleware
- Does not handle monitoring or event logging

Kubernetes Cluster Model

- Kubernetes uses the term *cluster* to describe the set of containers plus the associated support software used to create, operate, and access the containers.

- The number of containers in a cluster depends on demand, and Kubernetes can increase or decrease the number as needed.

Conceptual organization of a Kubernetes cluster


Cluster
owner
containers in the cluster
Kubernetes software to create and operate the cluster
user 1
user 2
user N

Kubernetes Pods

- “Pod" is the smallest deployable unit that can hold one or more containers.

- Pods are the basic building blocks of the Kubernetes object model and represent a single instance of a running process in a cluster.

- Containers within the same Pod share the same network namespace, which means they can communicate with each other using local host.

- It's used to group one or more containers that should be deployed together on the same host. These containers share the same network and storage resources, making them tightly coupled.

Pod Creation, Templates, And Binding Times

- Kubernetes uses a late binding approach in which a programmer creates a *template* for the pod (sometimes called a *pod manifest*) that specifies items to use when running the pod.

- A template assigns a name to the pod, specifies which container or containers to run, lists the network ports the pod will use, and specifies the version of the Kubernetes API to use with the pod.

- A template can use *yaml* or *json* formats

An example template for a pod that runs one container and uses port 8080.

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  labels:
    purpose: web-server
spec:
  containers:
  - name: example-pod
    image: f34cd9527ae6
    ports:
    - containerPort: 8080
```

Init Containers

- An init container is a specialized type of container that runs and completes its tasks before the main application containers within the same Pod start.
- **Init containers are designed to perform initialization and setup tasks that are necessary for the proper functioning of the application containers.**
- Init containers are useful for ensuring that your application's prerequisites are met before it begins its execution.
- They help manage complex initialization processes and dependencies, making it easier to run applications with specific requirements in Kubernete.
- **Eg : An init container can check the external storage and either exit normally if the storage is available, or exit with an error status if the storage is unavailable.**

Kubernetes Terminology: Nodes And Control Plane

- Kubernetes uses the term *Control Plane* (also known by the term *master*) to refer to the software components that an owner uses to create and operate containers.

- When the control plane software runs on a node, the node is known as a ***master node***.

- Kubernetes uses to run containers **a *Kubernetes node***. Some sources use the term ***worker node***.

master node and worker node

Kubernetes cluster

- Kubernetes cluster runs a single copy of the control plane components on a single master node.
- It is possible to create a high-availability cluster by running multiple copies of the control plane software on multiple master nodes.

Control Plane Software Components

Component	Purpose
API server	Provides an interface to the control plane
Scheduler	Chooses a node that will run a pod
Cluster State Store	Stores state information for the cluster
Controller Manager	Handles controllers that operate the cluster
Cloud Controller Manager	Handles cloud provider interactions

Control Plane Software Components

- *API server* : labeled the *kube-api-server*, the APIserver provides an interface to all the control plane components.
- *Scheduler* :*kube-scheduler*, the Scheduler handles the assignment of pods to nodes in the cluster.
- *Cluster State Store* : The Cluster State Store holds information about the cluster, including the set of nodes available to the cluster, the pods that are running, and the nodes on which the pods are currently running.
- When something changes in the cluster, the Cluster Store must be updated.
- **etcd**: A distributed key-value store that stores the cluster's configuration data, state information, and metadata. Etcd helps maintain consistency and reliability in the cluster.

Control Plane Software Components

- Controller Manager : **kube-controller-manage**r, the Controller Manager component consists of a daemon that remains running while the cluster exists.

- Kubernetes includes useful controllers, such as a *Replication Controller*, *Endpoints Controller*, and *Namespace Controller*

- Controller Manager performs housekeeping functions such as garbage collection of events, terminated pods, and unused nodes.

Control Plane Software Components

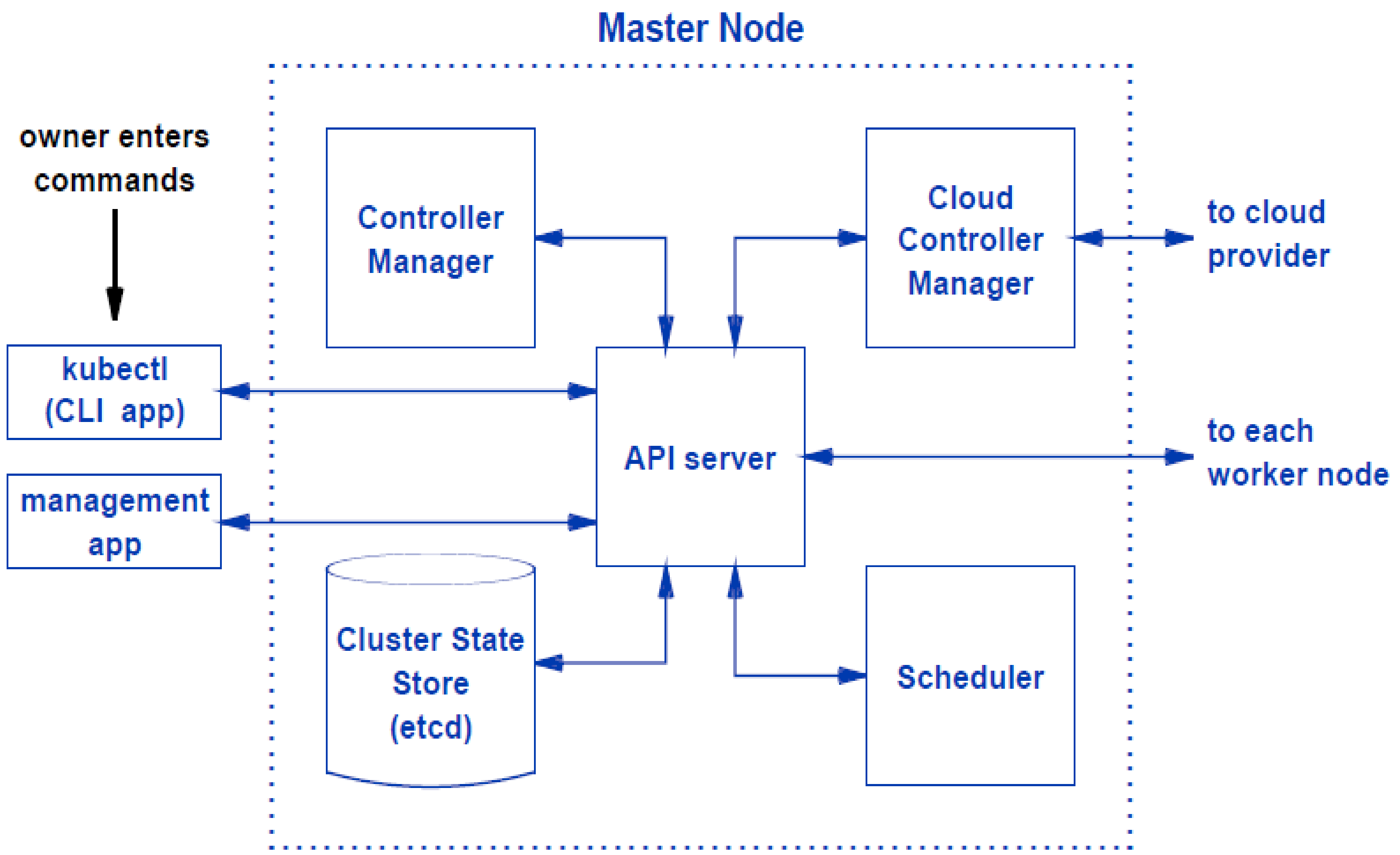
- The **Cloud Controller Manager (CCM)** is a component in a Kubernetes cluster responsible for interacting with a cloud provider's APIs to manage and control cloud-specific resources and functionalities.

- CCM is an optional component, and its use depends on whether your Kubernetes cluster is running in a cloud environment and needs to integrate with that cloud provider's services.

Command-Line Interface (CLI) (*command-line app*)

- Perform various tasks related to Kubernetes cluster management, resource configuration, monitoring, and troubleshooting.
- These command-line tools are used by administrators, developers, and operators to interact with and manage Kubernetes clusters.
- **kubectl** is the primary command-line tool for interacting with Kubernetes clusters. It allows users to create, update, delete, and manage Kubernetes resources, inspect cluster status, and perform various administrative tasks.

Communication Among Control Plane Components



Worker Node Software Components (also known as Minions or Nodes)

- Each worker node runs software components that control and manage the pods running on the node.

Component	Purpose
Service Proxy	Configures the network on the node (i.e., iptables)
Kubelet	Uses container software to run and monitor pods
Container Runtime	Standard container software (e.g., Docker Engine)

Service Proxy

- Also known as a **"Kube Proxy" or "Kubernetes Proxy,"** is a network service responsible for facilitating network communication to and from Pods within the cluster.

- Its primary role is to enable internal network connectivity among Pods, expose services to the network, and manage routing and load balancing for those services.

- iptables is a powerful and flexible Linux kernel-level firewall tool that can be configured to filter, modify, and route network packets.

- iptables helps in managing network traffic and implementing various network-related features and policies within the cluster.

Kubelet

- The Kubelet is a critical component in Kubernetes that ensures that containers are running as expected and that the desired state, as defined in the Pod specifications, is maintained.

- It communicates with the control plane components, such as the API server, to synchronize its actions with the cluster's desired state.

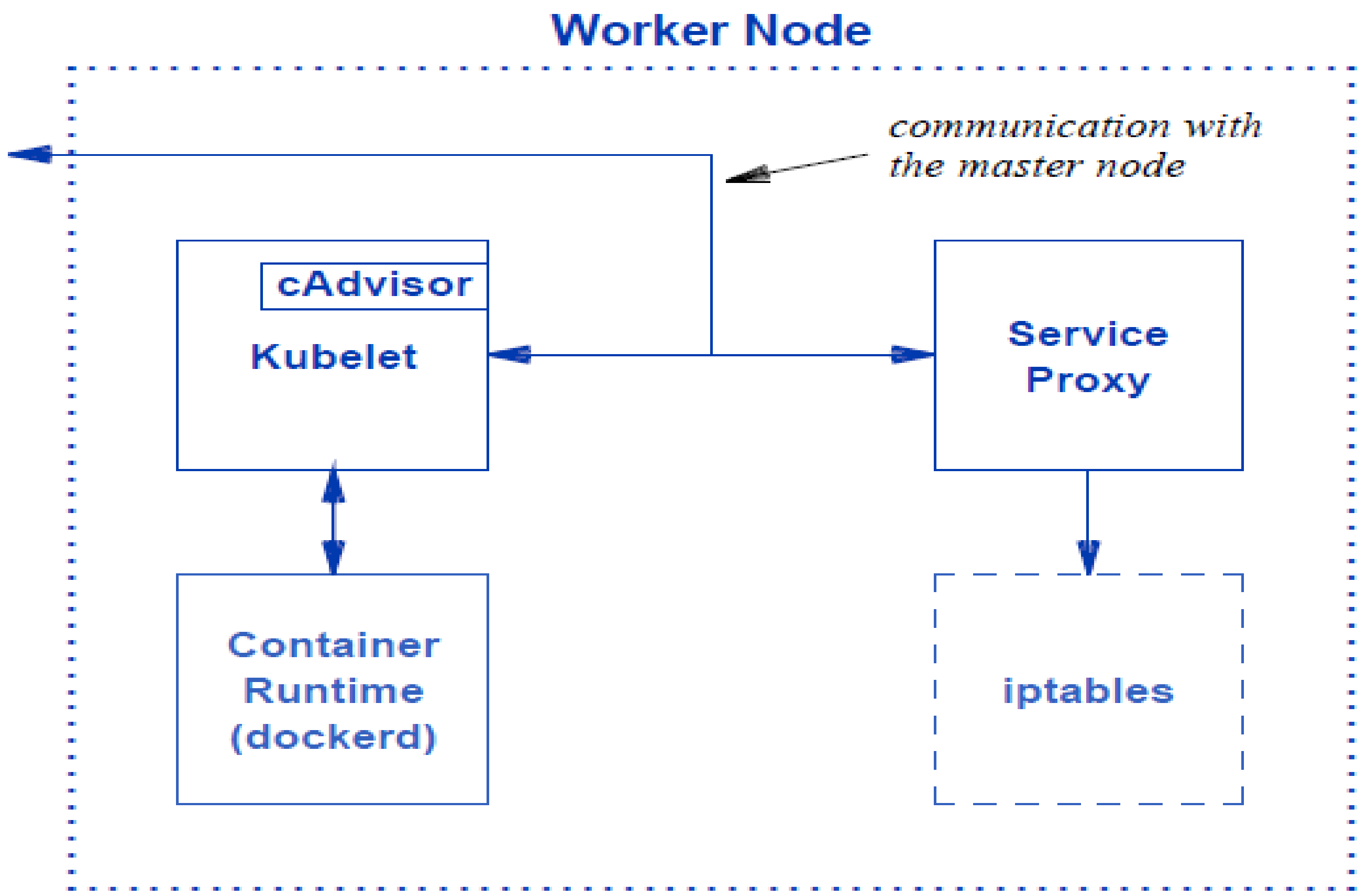
- The Kubelet is an essential part of the orchestration and management of containers in a Kubernetes cluster.

- Kubelet includes a copy of the *cAdvisor* software that collects and summarizes statistics about pods.

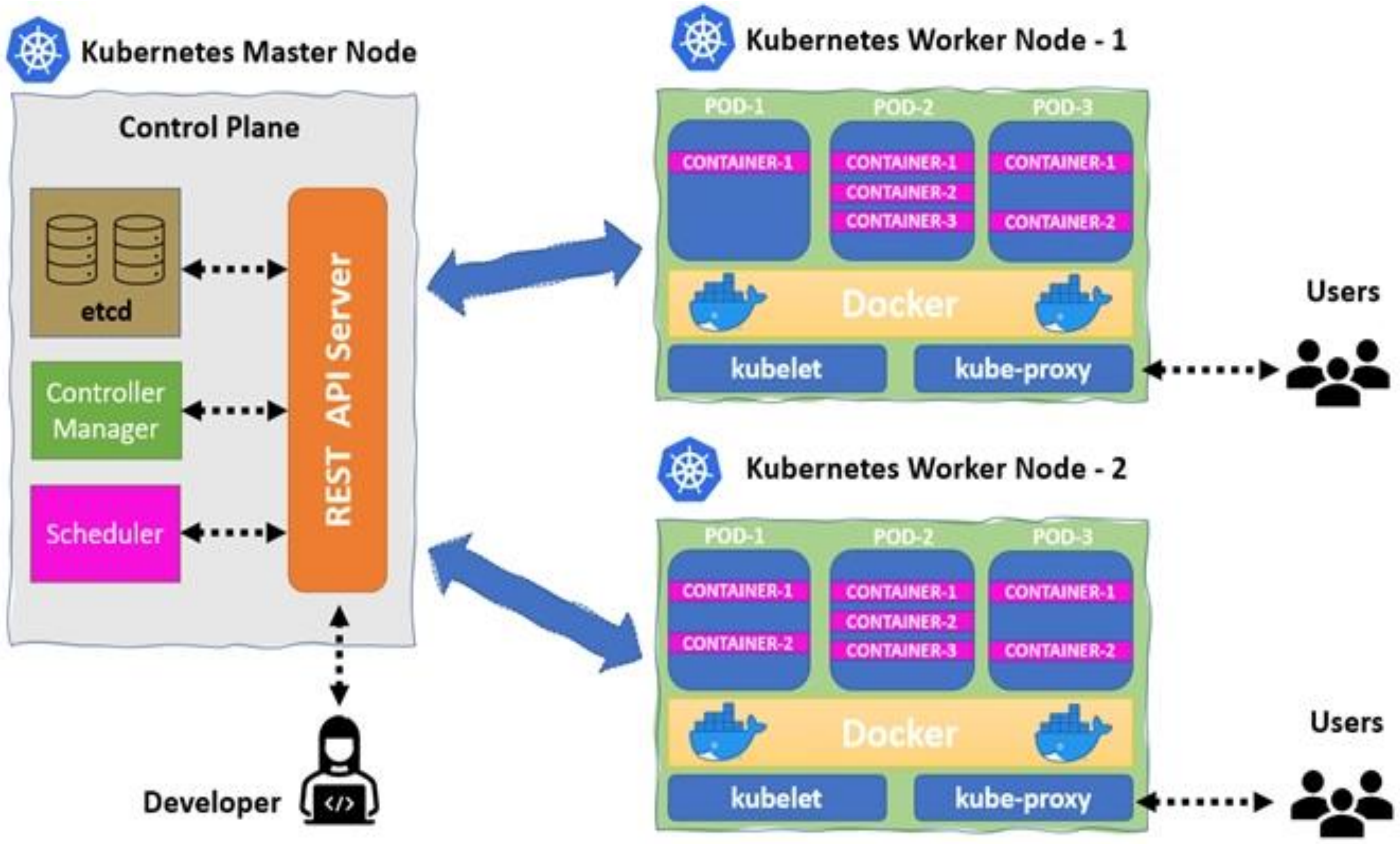
Container Runtime

- **When Kubernetes needs to deploy containers, Kubelet interacts with the Container Runtime system to perform the required task.**

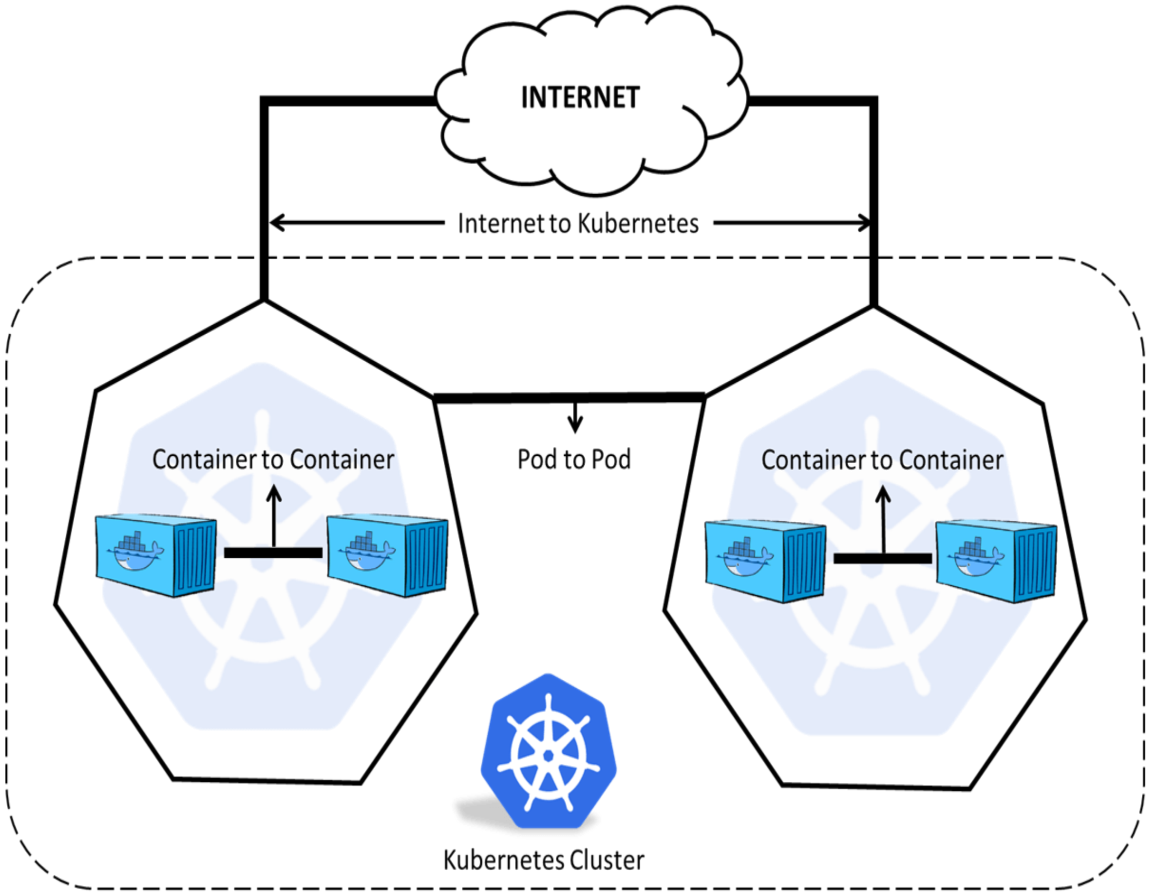
Software components on a worker node and the communication among them.


Worker Node
communication with the master node
cAdvisor
Kubelet
Service Proxy
Container Runtime (dockerd)
iptables

Overview of Kubernetes Architecture



Kubernetes Networking Model



SUMMARY

- CONTAINERS
- DOCKER
- KUBERNETES
- CLUSTER MODEL
- MASTER NODE AND WORKER NODE
- KUBERNETES ARCHITECTURE