Cloud Week 1

What is virtualization, and how does it contribute to efficient resource management in cloud environments?

1. Virtualization creates an abstraction layer over computer hardware into Virtual Machines.

2. When an application consists of a few large components it is acceptable to use VM. But when there are more components and they are smaller, it creates resource overhead.

3. The abstraction provided by virtualization allows us to run our applications independent of the host systems environment by giving the environment required by the applications.

How do hypervisors work, and what are the differences between Type 1 and Type 2 hypervisors?

Hypervisor divides the physical hardware resources into smaller sets of virtual resources that can be used by the operating system inside each VM.

Applications running inside those VMs perform system calls to the guest OS' kernel in the VM, and the kernel performs instructions on the host's physical CPU through the hypervisor. Type 1 hypervisors do not use a host OS, while type 2 do.

What are the advantages and disadvantages of using virtual machines compared to physical servers?

Setting up and maintaining a physical server is more complex and costly. Virtual servers help distribute computing resources among all running VMs and allow running applications requiring different environments on the same machine, thus improving resource utilization and scalability.

However, running VMs on a device introduces the overhead of running multiple operating systems. Physical servers do not have this overhead because they run a single OS.

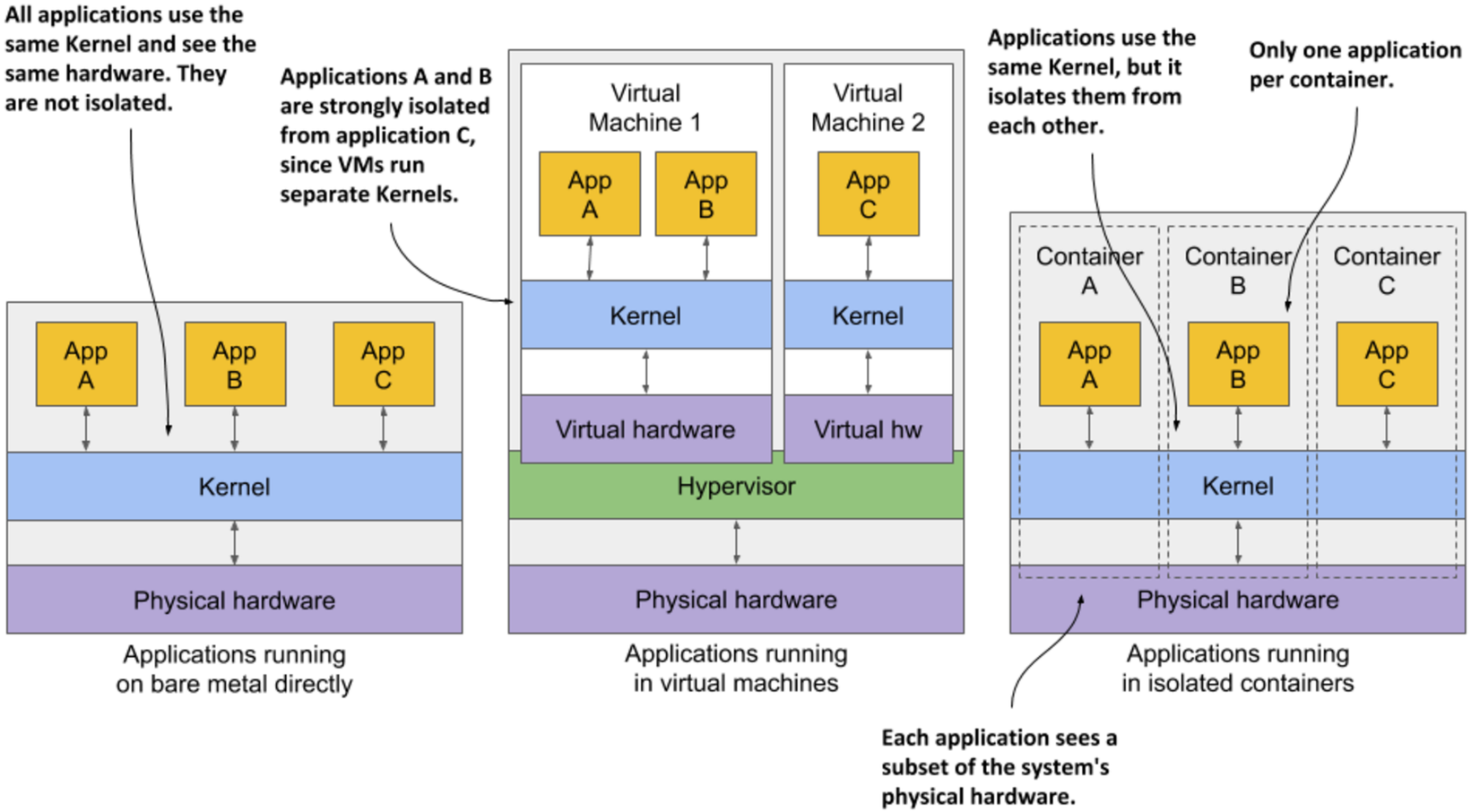
Explain the concept of containers and how they differ from virtual machines.

Containers

Containers use control groups to limit the amount of resources the process can consume and namespaces to make sure each process sees its personal view of the system like its own files and processes.

Virtual Machines

While VMs running on the same machine can have different operating systems by using hypervisors, containers perform system calls on the same kernel running on the host OS.


All applications use the same Kernel and see the same hardware. They are not isolated.
App A
App B
App C
Kernel
Physical hardware
Applications running on bare metal directly
Applications A and B are strongly isolated from application C, since VMs run separate Kernels.
Virtual Machine 1
Virtual Machine 2
App A
App B
App C
Kernel
Kernel
Virtual hardware
Virtual hw
Hypervisor
Physical hardware
Applications running in virtual machines
Applications use the same Kernel, but it isolates them from each other.
Only one application per container.
Container A
Container B
Container C
App A
App B
App C
Kernel
Physical hardware
Applications running in isolated containers
Each application sees a subset of the system's physical hardware.

Why are containers more lightweight than VM?

1 Architecture

Containers share the host systems OS kernel, they only package the application and its dependencies

VMs include a full guest OS kernel, drivers , and libraries

2 Boot time

With **containers** there is no need to boot a seperate OS , it is just a single isolated process running on the host OS

VMs require booting an entire OS before the application can start
-more time

3 Isolation & Portability

Containers use OS-level virtualization provided by technologies like cgroups and namespaces to isolate processes

VMs rely on a hypervisor (e.g., VMware, Hyper-V, KVM) to provide hardware-level isolation

What kind of impact do containers have on software deployment and development?

Consistency across environments

Rapid Iteration and Deployment

Microservices Architecture

Portability

Isolation and Resource Efficiency

What role does container orchestration play in modern cloud infrastructure?

Container orchestration tools automate life cycle management and operational tasks based on the container definition file, including:

- Provisioning and deployment
- Scaling containers up or down and load balancing
- Allocating resources between containers
- Moving containers to another host to ensure availability if there’s a shortage of resources or an unexpected outage
- Performance and health monitoring of the application
- Allocating resources between containers
- Service discovery

How does Kubernetes manage containerized applications at scale? Discuss its core components.

Kubernetes abstracts container orchestration by offering functionalities such as self-healing, load balancing, scaling, and service discovery.
To achieve this, it relies on a set of core components that work together to manage containers efficiently.

Control Plane

The master node hosts the Kubernetes Control Plane that controls and manages the whole Kubernetes system.Control plane holds and controls the state of the cluster,

Worker Nodes

The worker nodes run the applications deployed

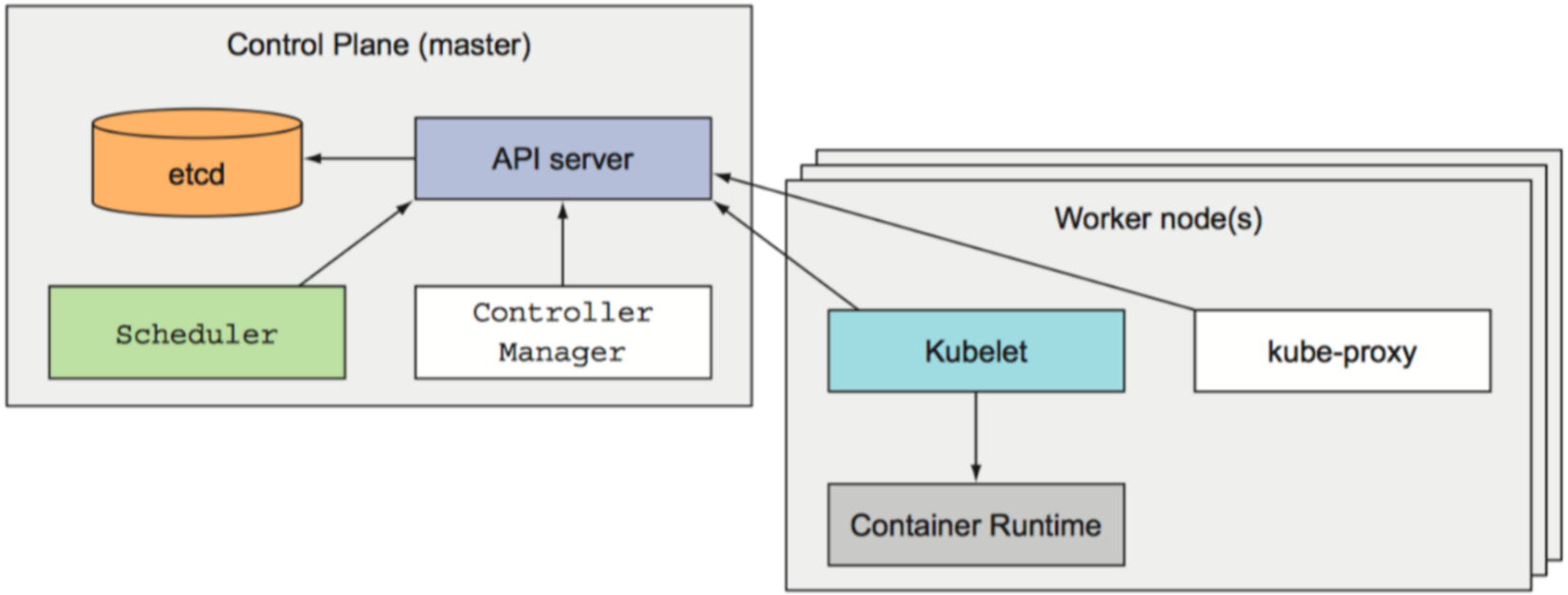


Figure 1.9 The components that make up a Kubernetes cluster

control plane

API Server

Processes API requests from clients, including kubectl, communication between other Control Plane components and other nodes.

Etcd

Holds the entire state of the Kubernetes cluster, including configuration data, the state of the pods, services, and more.

Controller manager

Performing cluster-level functions: replicating components, keeping track of worker nodes, handling node failures

Scheduler

schedules apps by assigning a worker node to each deployable component of applications.

worker nodes

kubelet

receives pod specifications (PodSpecs) from the API server
ensures that the specified containers are running and health

pods

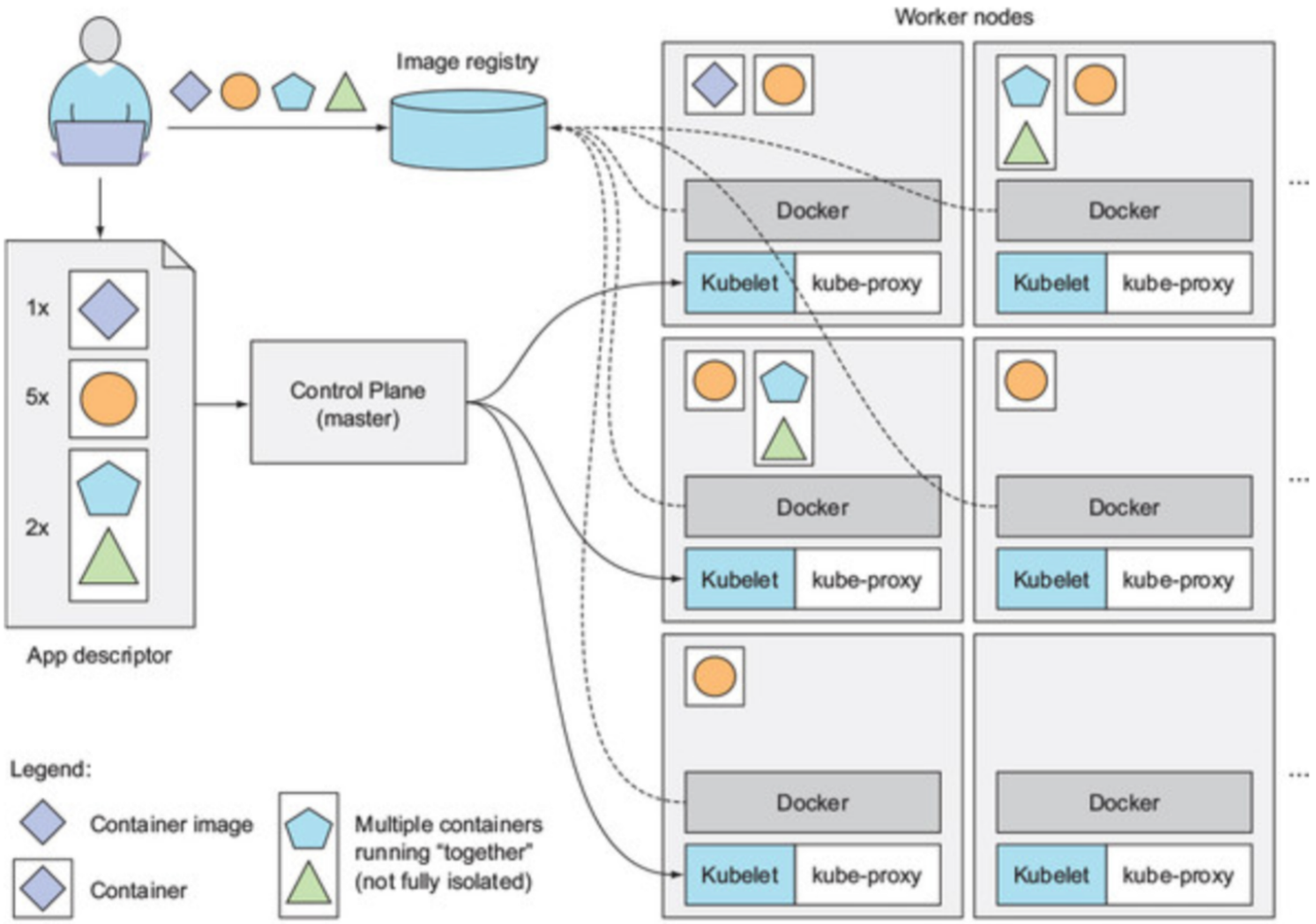
The smallest deployable units in Kubernetes, encapsulates containers that share the same network and storage

container runtime

Interacting with the underlying OS to allocate resources and ensure containers run as expected.
Such as Docker,CRI-O

kube-proxy

Manages network rules to allow pods to communicate with each other, as well as to expose services to external clients.
Load-balancing the network traffic between application components.


Worker nodes
Image registry
Docker
Kubelet
kube-proxy
1x
5x
2x
Control Plane
(master)
App descriptor
Legend:
Container image
Container
Multiple containers
running "together"
(not fully isolated)

THANK yOU