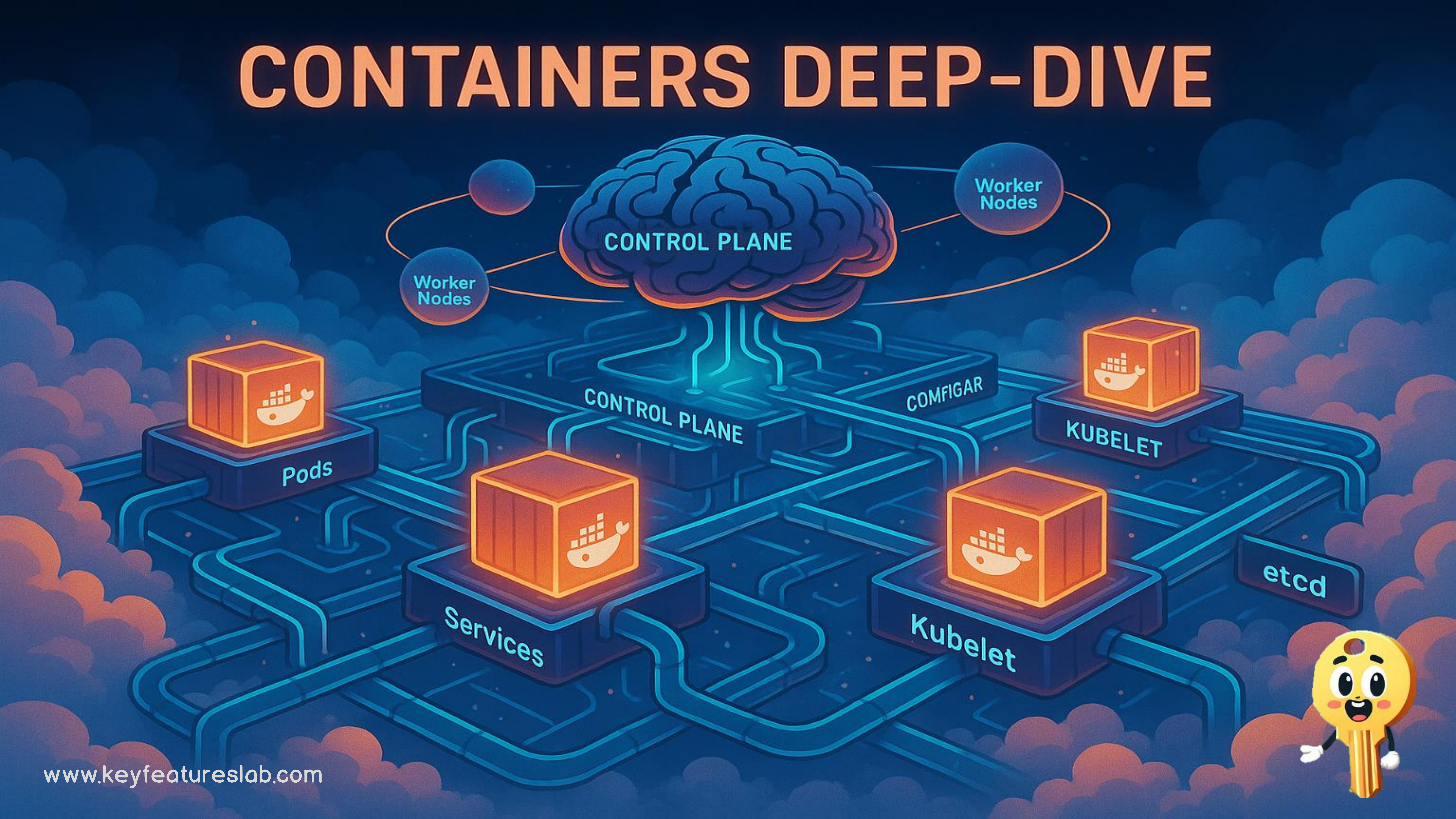
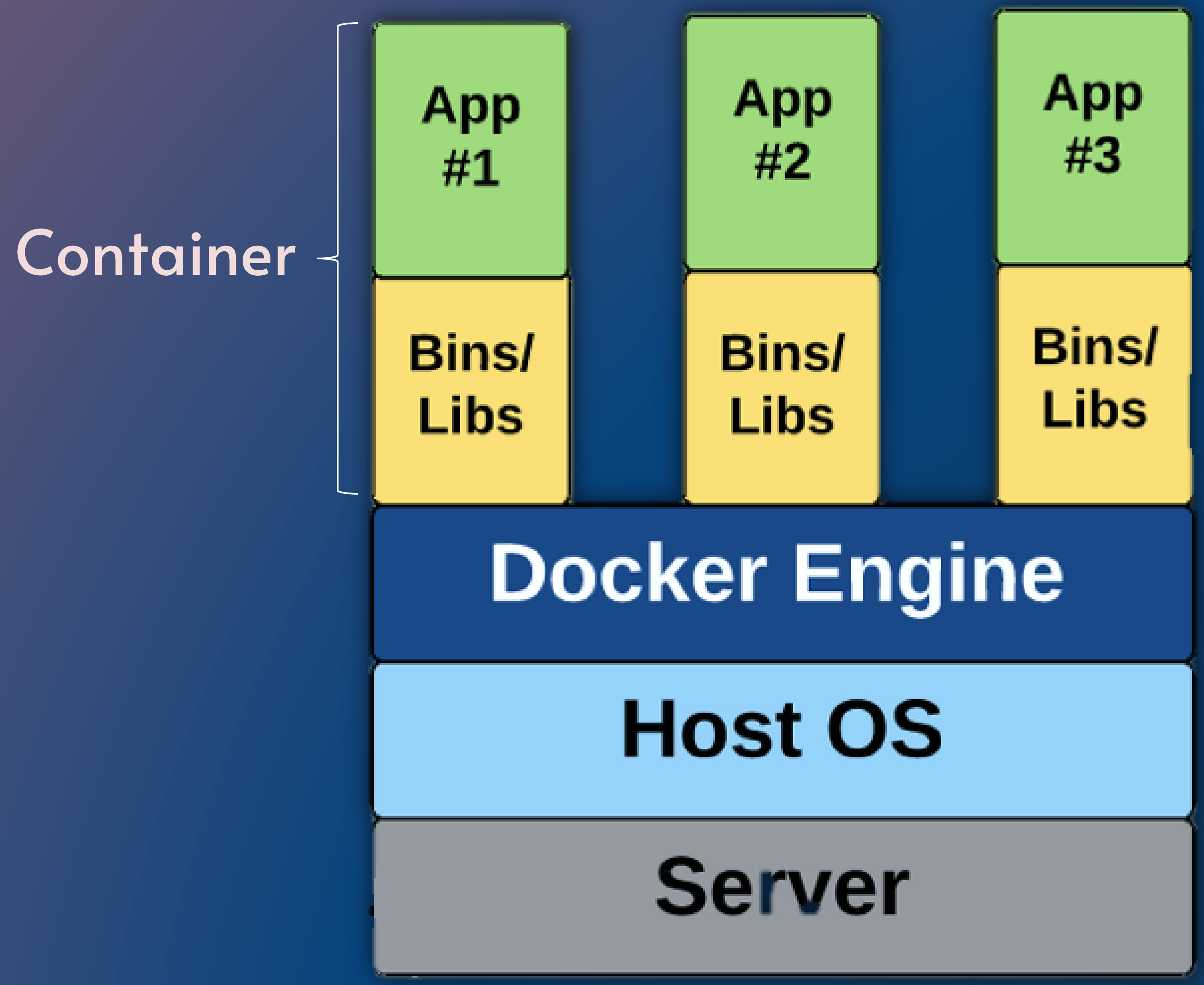

CONTAINERS DEEP-DIVE
Worker Nodes
CONTROL PLANE
Worker Nodes
CONTROL PLANE
COMFIGAR
KUBELET
Pods
Services
Kubelet
etcd
www.keyfeatureslab.com

Today's agenda:

- ✓ Introduction to Kubernetes
- ✓ CoreDNS
- ✓ Identities in Kubernetes
- ✓ Kubernetes Security Posture Assessment
- ✓ Q&A

Container Architecture

- ✓ Containers are lightweight, portable, and self-sufficient software packages that contain everything needed to run an application, including code, runtime, system tools, libraries, and dependencies.
- ✓ Each container instance, has a limited allocated set of resources that it cannot exceed: CPU, memory, disk I/O, network, …)
- ✓ Docker leverages Linux's namespaces to isolate containers from the host as well as from one another. Containers cannot see other containers' files, processes, and network information, unless the user gives them permission to do so.



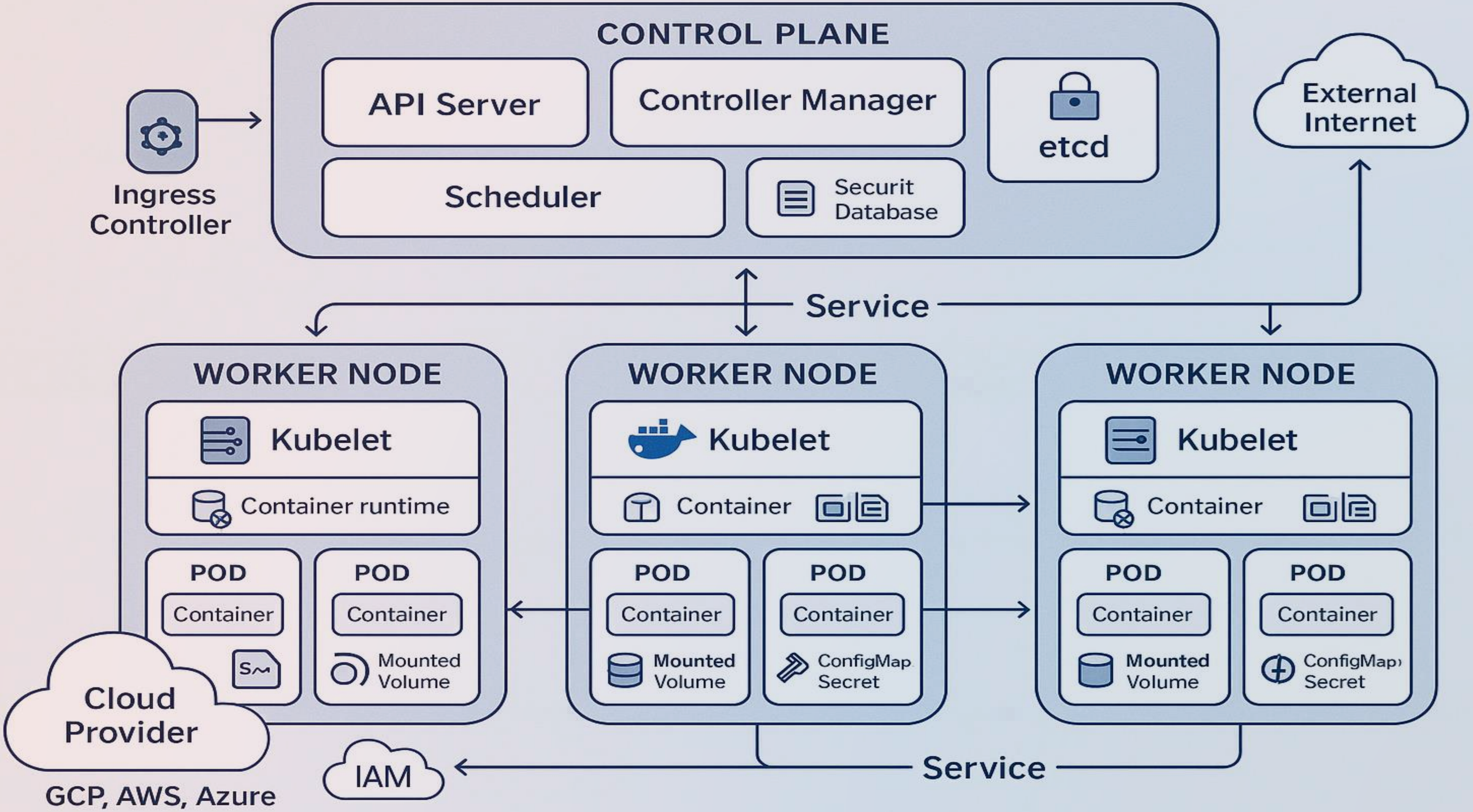
VMs vs. Containers

- **Architecture:**
  - VMs virtualize hardware using a **hypervisor**, allowing multiple OS instances to run independently on a host machine.
  - Containers virtualize at the operating system level, sharing the host OS kernel but isolating applications in separate user spaces.
- **Resource Consumption:**
  - VMs are heavier because each runs its own full operating system, consuming more CPU, memory, and storage.
  - Containers are lightweight, using shared system resources more efficiently and reducing overhead.
- **Isolation:**
  - VMs provide **strong isolation** by running entirely separate OS instances, including their own kernels.
  - Containers **isolate at the process level**, sharing the host kernel while keeping file systems, networks, and processes separate.
- **Boot Time:**
  - VMs have slower boot times due to the need to start a full OS.
  - Containers start almost instantly since they use the already-running host OS, making them ideal for rapid scaling and microservices.
- **Deployment:**
  - VMs are deployed as independent units with their own OS, which adds complexity and maintenance overhead.
  - Containers are **deployed from lightweight images**, enabling fast, consistent, and scalable deployments across different environments.

Docker & Kubernetes

- Docker is an open-source project based on Linux containers. It uses Linux kernel features like namespaces and control groups to create containers on top of an operating system.
- Docker image containers can run natively on both Linux and Windows.
- Docker is also the name of the company that collaborates with major cloud providers such as Amazon, Google, and Microsoft to advance and promote containerization solutions.

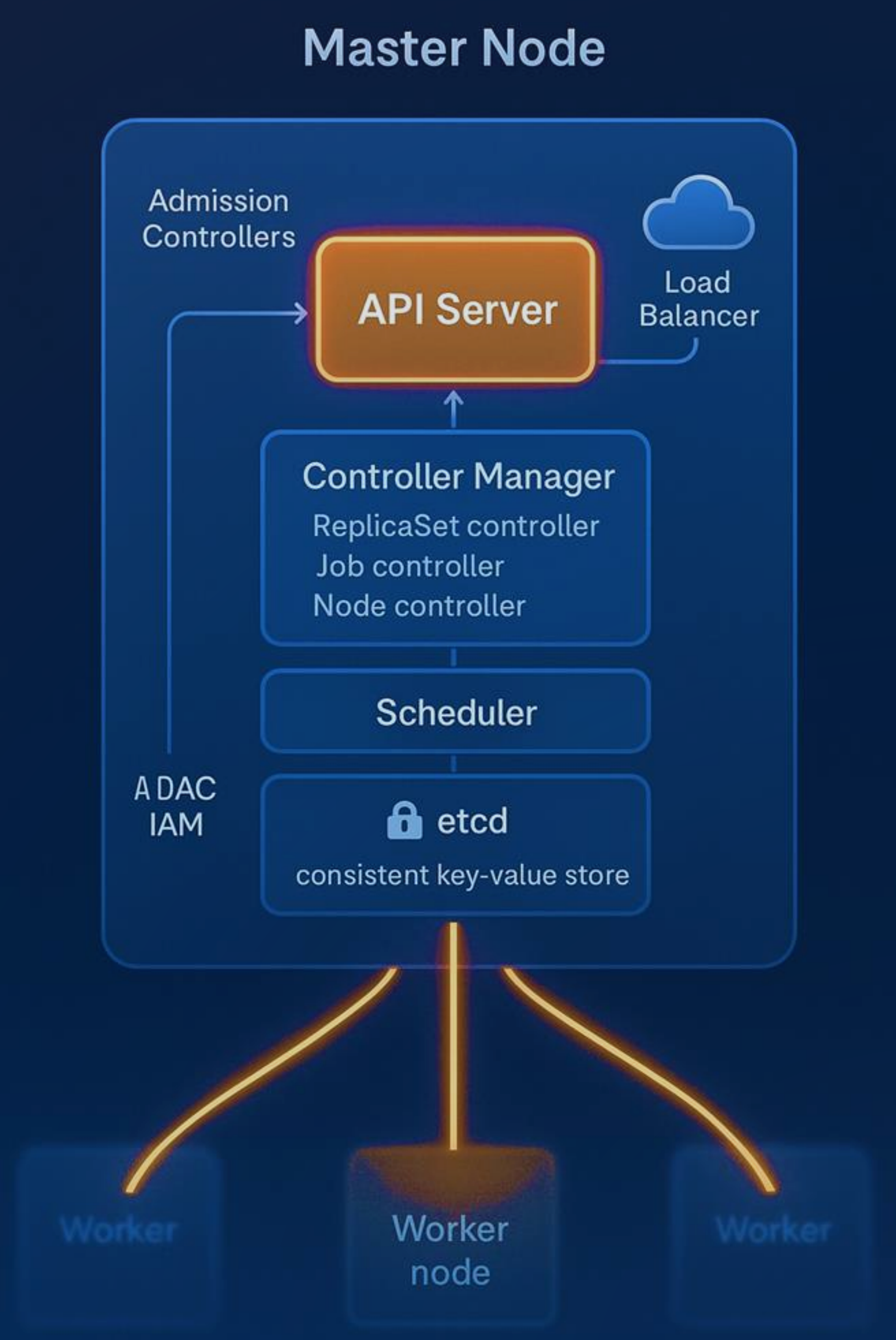
- Kubernetes is an open-source container management system developed by Google.
- It helps developers automate the deployment, scaling, and management of containerized workloads.
- It is based on a master-slave model where a master node controls all the containers running on the other nodes.



CONTROL PLANE
API Server
Controller Manager
etcd
Scheduler
Securit
Database
Ingress
Controller
External
Internet
Service
WORKER NODE
Kubelet
Container runtime
POD
Container
Mounted
Volume
WORKER NODE
Kubelet
Container
POD
Container
Mounted
Volume
POD
Container
ConfigMap
Secret
WORKER NODE
Kubelet
Container
POD
Container
Mounted
Volume
POD
Container
ConfigMap
Secret
Cloud
Provider
GCP, AWS, Azure
IAM
Service

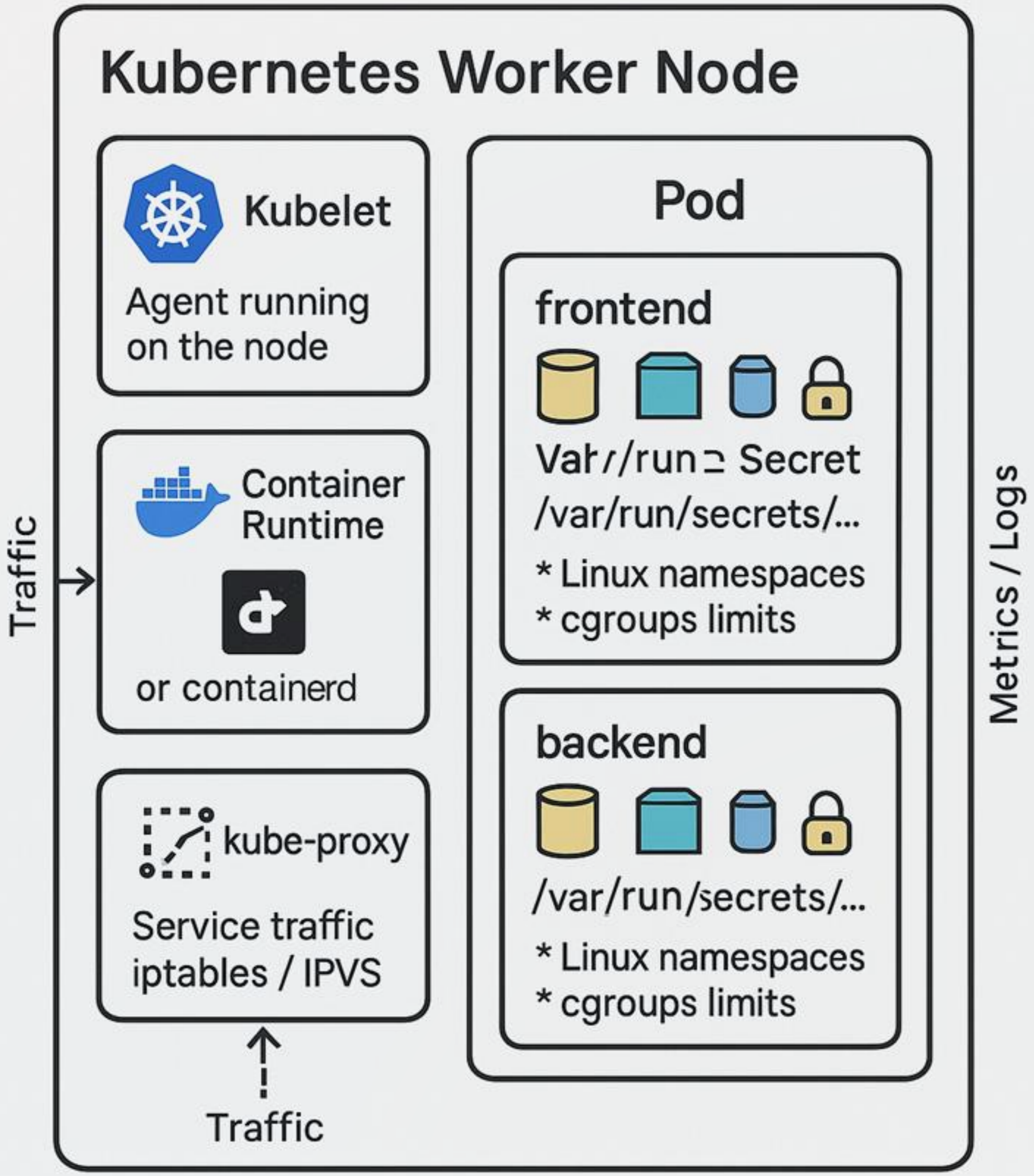
Master Node

- API Server: the central brain of the control plane. Whether you create, read, update, or delete resources like pods, services or deployments, we'll have to do so through the API Server. It is also responsible for client and user authentication.
- Scheduler: when a new deployment is received and it requests to create a new pod, the API Server contacts the scheduler then evaluates all available nodes and picks the best one for that pod.
- Controller Manager: it monitors all the controllers and is responsible for collecting and sending information through and by the API Server. Moreover, it takes automated actions to maintain the desired state of the cluster.
- etcd: a distributed key-value store used by Kubernetes to store desired state of the cluster.



Worker Node

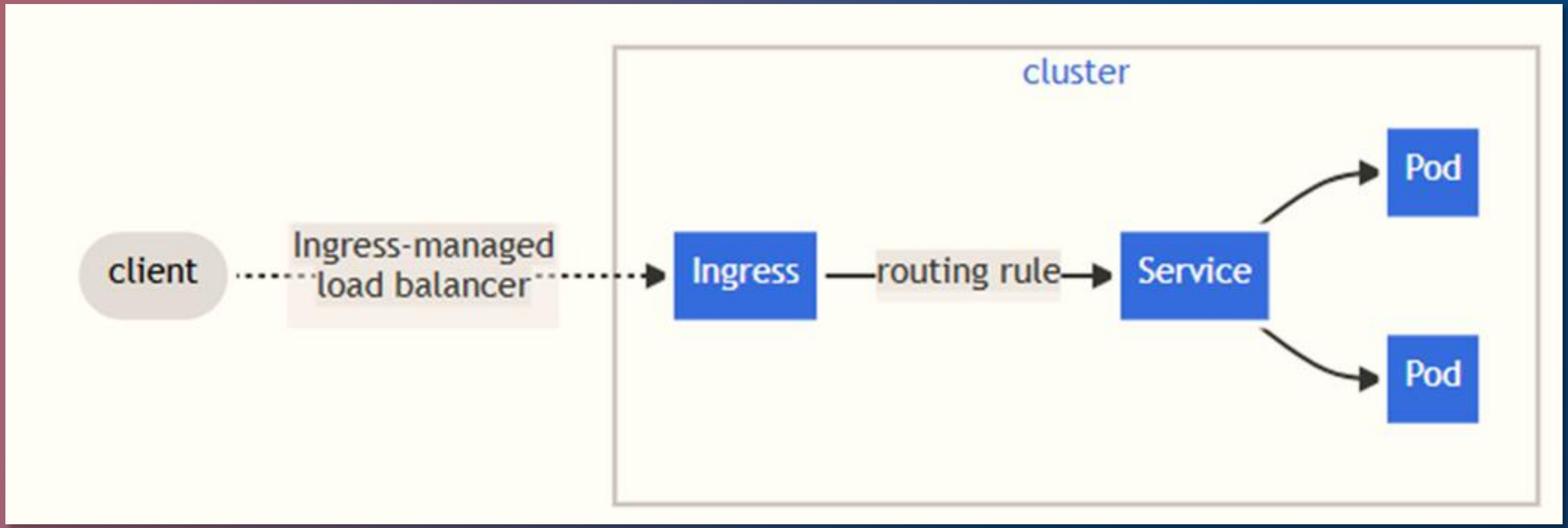
- Kubelet: An agent on each worker node that communicates with the control plane. It receives pod specs from the API server, starts the containers, and ensures they stay healthy. If a pod fails, the Kubelet tries to restart it on the same node.
- Kube-proxy: handles network traffic within the cluster. It enables communication between services and pods and distributes incoming requests to the appropriate pods on the same or different nodes.
- Pods: a group of containers that are deployed together on the same host.
- Docker: the execution environment of the containers.
- Container Runtime: the software on each worker node that pulls container images, starts and stops containers, and manages their execution. Common runtimes include containerd, CRI-O, and formerly Docker (deprecated now).

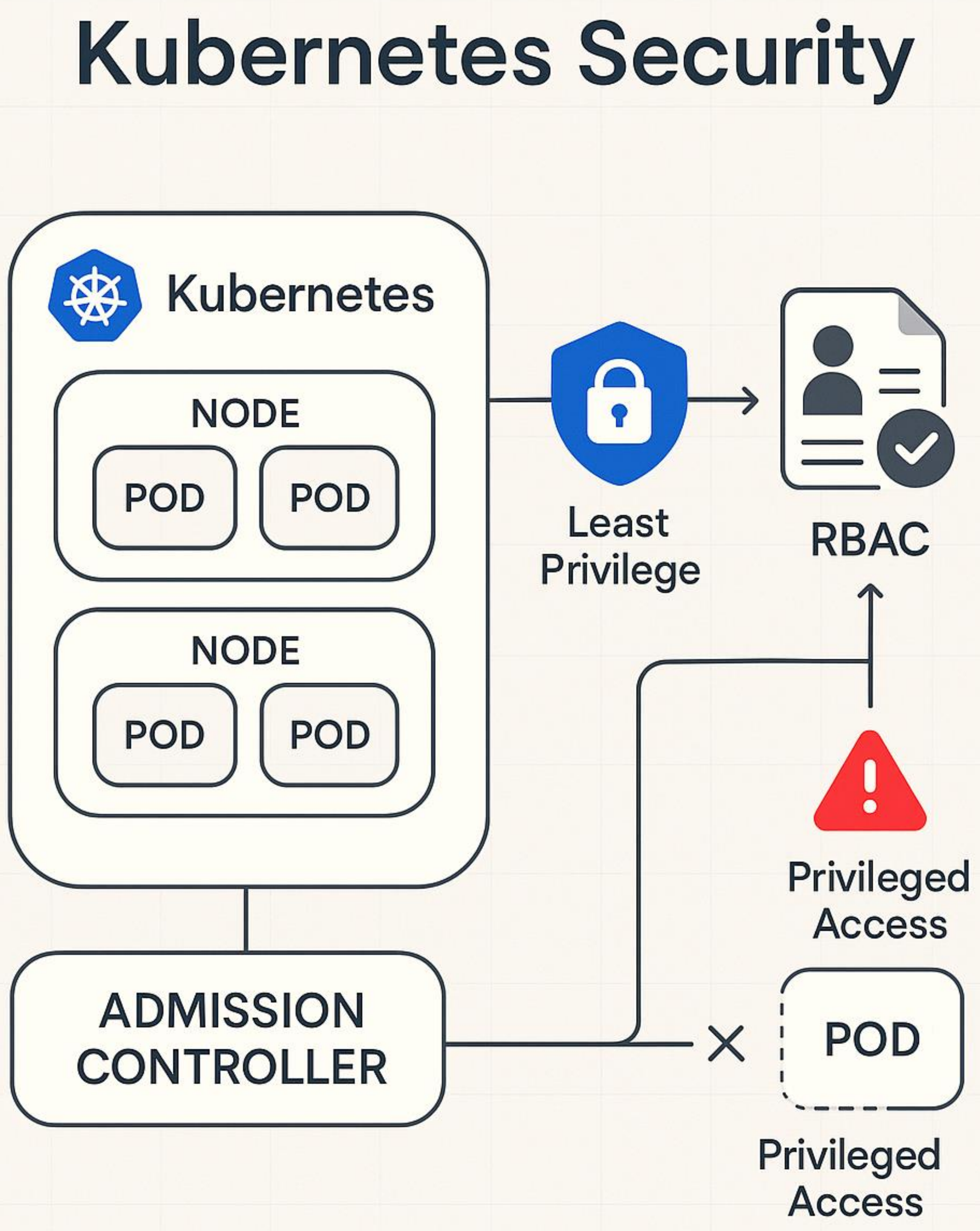


Services

- In Kubernetes a Service is a method for exposing a network application that is running as one or more Pods in your cluster. Expose an application running in your cluster behind a single outward-facing endpoint, even when the workload is split across multiple backends.

- Each Services is assigned a ClusterIP (virtual IP address) that is not tied to any physical network interface.





Users, Roles & Least Privilege

- Just like in any other secure environment, also in Kubernetes it is crucial to applying the principle of least privilege when assigning permissions. To do this we leverage RBAC (Role-Based Access Control).
- Yet, Kubernetes does not manage users natively: users are created externally via user lists or client certificates. Access is enforced by binding roles to certificates.
- To prevent pods from having excessive permissions, each cluster hosts an Admission Controller. This controller runs in the API Server and blocks pods with excessive permissions before their even admitted in the cluster.

Intra-Pod Communication

- If services/applications can be allocated in any pod in the cluster, how does a pod know how to route internal communication? Here's a step-by-step simulation:

1. The client pod initiates communication by sending a request to a Kubernetes Service using its name, which is then resolved by CoreDNS running as a Deployment within the cluster.

2. CoreDNS responds with the Service's ClusterIP, a stable virtual IP that does not belong to any specific pod or node but is managed internally by Kubernetes.

3. The client pod sends traffic to the received ClusterIP, which is intercepted and handled by kube-proxy on the same node rather than being directly routed to a specific pod.

4. Kube-proxy performs round-robin load balancing by selecting one of the backend pods associated with the Service and forwarding the request, regardless of whether the destination pod is on the same node or a different one.

End-to-End Container Security with Microsoft Defender for Containers

CSSC Security

Code
Supply chain attacks

Injecting malicious code into container images and dependencies

Build
Poisoning the CI/CD pipeline

Embedding malware in automated workflows

Binary drift detection

Ship
Artifact tampering

Modifying or replacing images in public/third-party registries

Deployment
Misconfiguration

Configuring insecure settings and permissions, leaving environments exposed

Kubernetes nodes protection

Vulnerability assessment

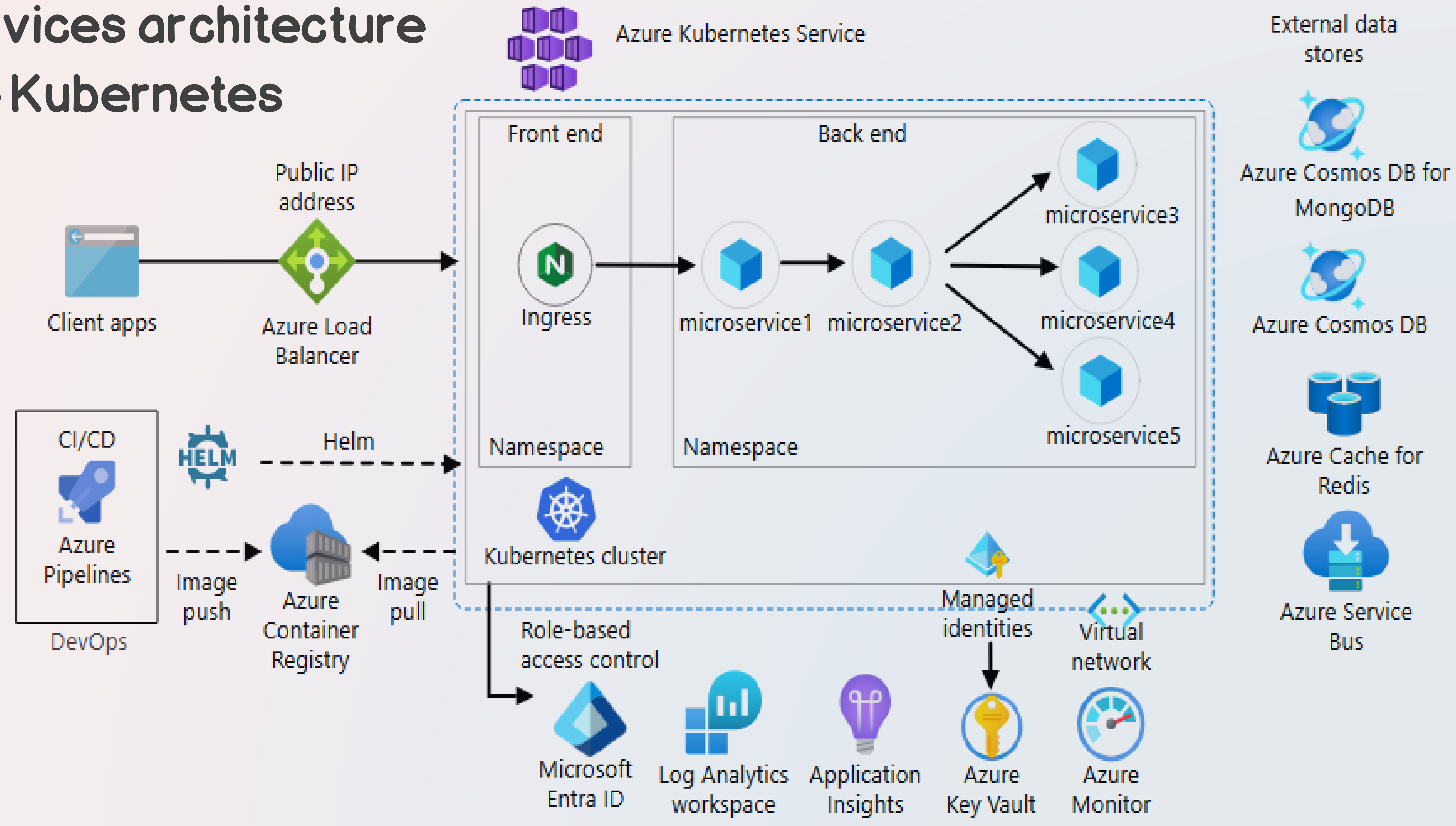
Runtime
Vulnerable/exposed application

Enabling attackers to break out of the container boundaries

Container Security Solutions

Feature / Solution	Defender for Containers	Sysdig Secure	Aqua Security	Prisma Cloud
Cloud-native integration	✅ Excellent (Azure-native)	✅ Good (multi-cloud)	✅ Good	✅ Excellent (all clouds)
Runtime threat detection (eBPF/Falco)	✅ Yes (eBPF-based sensor)	✅ Yes (eBPF + Falco engine)	✅ Yes (proprietary engine)	✅ Yes
Kubernetes audit log monitoring	✅ Yes	✅ With Secure for Cloud	✅ Yes	✅ Yes
Image scanning (CI/CD & registries)	✅ Native with ACR & DevOps	✅ Broad registry support	✅ Advanced scanner	✅ Full coverage
Policy enforcement (RBAC, PSP, runtime)	✅ Azure Policy integration	✅ Advanced runtime policies	✅ Granular runtime controls	✅ Comprehensive (CSPM + CWPP)
SIEM & analytics integration	✅ Native with Microsoft Sentinel	✅ Syslog/Splunk support	✅ Yes	✅ Native with Cortex XSOAR
Auto-deployment of agents	✅ Automatic via Azure	❌ Manual or Helm chart	❌ Manual or custom setup	❌ Requires setup
Open source base (e.g., Falco)	❌ Closed-source	✅ Yes (Falco engine)	❌ Closed-source	❌ Closed-source
Multi-cloud support	⚠️ Azure-first (via Arc for others)	✅ Native multi-cloud	✅ Native multi-cloud	✅ Excellent
License included with CSP tools	✅ Part of the Cloud Workload Protection Plan (CWP)	❌ Separate licensing	❌ Separate licensing	❌ Separate licensing

Microservices architecture on Azure Kubernetes Service



Microsoft Azure