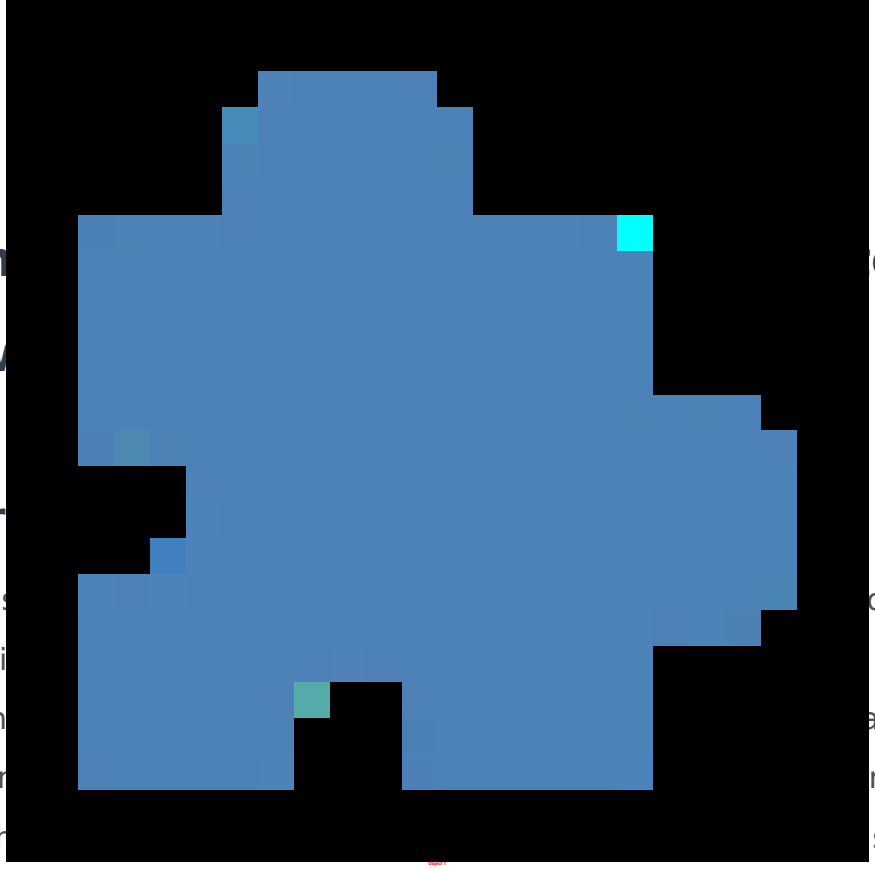


Kubern or Framev

Summar

So far, in this on using Docker and i release process is th ation in production in nsume the product at th scale. It is impossible to manually manage thousands of containers, keeping these are up to date with the latest code changes, in a healthy state, and accessible. As a result, a **container orchestrator framework** is necessary.

A container orchestrator framework is capable to create, manage, configure thousands of containers on a set of distributed servers while preserving the connectivity and reachability of these containers. In the past years, multiple tools emerged within the landscape to provide these capabilities, including Docker Swarm, Apache Mesos, CoreOS Fleet, and many more. However, **Kubernetes** took the lead in defining the principles of how to run containerized workloads on a distributed amount of machines.

Kubernetes is widely adopted in the industry today, with most organizations using it in production. Kubernetes currently is a graduated CNCF project, which highlights its maturity and reported success from end-user companies. This is because Kubernetes solutionizes portability, scalability, resilience, service discovery, extensibility, and operational cost of containers.

Portability

Kubernetes is a highly portable tool. This is due to its open-source nature and vendor agnosticism. As such, Kubernetes can be hosted on any available infrastructure, including public, private, and hybrid cloud.

Scalability

Building for scale is a cornerstone of any modern infrastructure, enabling an application to scale based on the amount of incoming traffic. Kubernetes has in-build resources, such as HPA (Horizontal Pod Autoscaler), to determine the required amount of replicas for a service. Elasticity is a core feature that is highly automated within Kubernetes.

Resilience

Failure is expected on any platform. However, it is more important to be able to recover from failure fast and build a set of playbooks that minimizes the downtime of an application. Kubernetes uses functionalities like ReplicaSet, readiness, and liveness probes to handle most of the container failures, which enables powerful self-healing capability.

Service Discovery

Service discovery encapsulates the ability to automatically identify and reach new services once these are available. Kubernetes provide cluster level DNS (or Domain Name System), which simplifies the accessibility of workloads within the cluster. Additionally, Kubernetes provides routing and load balancing of incoming traffic, ensuring that all requests are served without application overload.

Extensibility

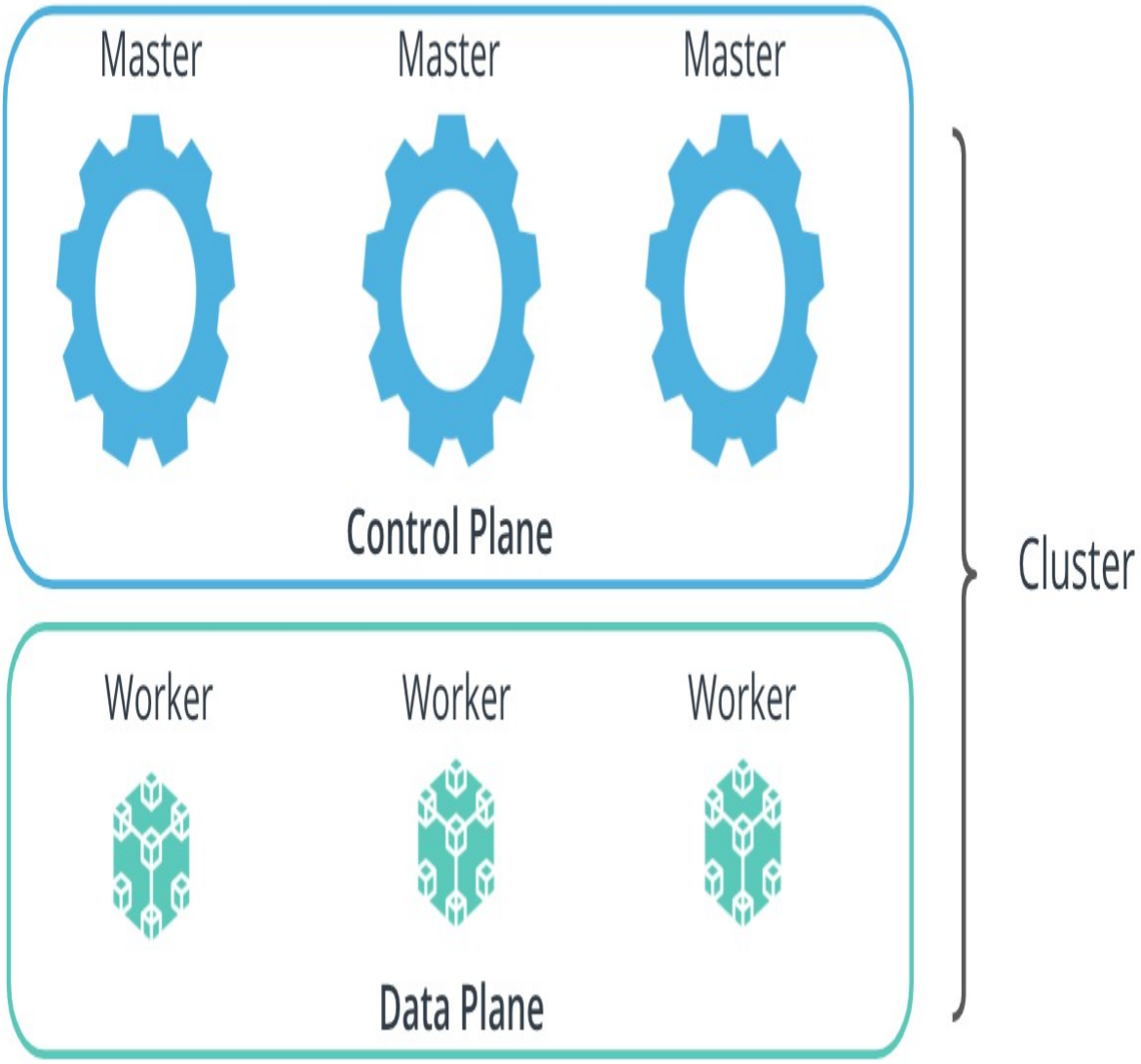
Kubernetes is a highly extensible mechanism that uses the building-block principle. It has a set of basic resources that can be easily adjusted. Additionally, it provides a rich API interface, that can be extended to accommodate new resources or CRDs (Custom Resource Definitions).

Operational Cost

Operational cost refers to the efficiency of resource consumption within a Kubernetes cluster, such as CPU and memory. Kubernetes has a powerful scheduling

mechanism that places an application on the node with sufficient resources to ensure the successful execution of the service. As a result, most of the available infrastructure resources are allocated on-demand. Additionally, it is possible to automatically scale the size of the cluster based on the current incoming traffic. This capability is provisioned by the cluster-autoscaler, which guarantees that the cluster size is directly proportional to the traffic that it needs to handle.

Kubernetes Architecture

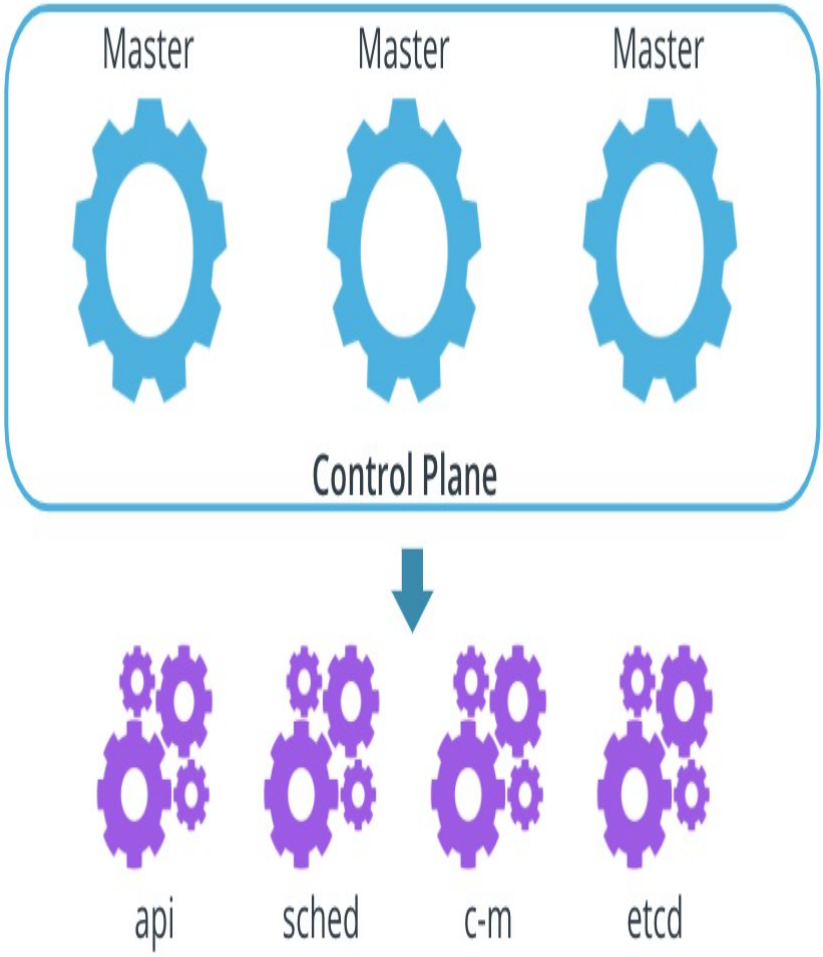


Kubernetes architecture, composed of control and data planes

A Kubernetes cluster is composed of a collection of distributed physical or virtual servers. These are called **nodes**. Nodes are categorized into 2 main types: master and worker nodes. The components installed on a node, determine the functionality of a node, and identifies it as a master or worker node.

The suite of master nodes, represents the **control plane**, while the collection of worker nodes constructs the **data plane**.

Control Plane



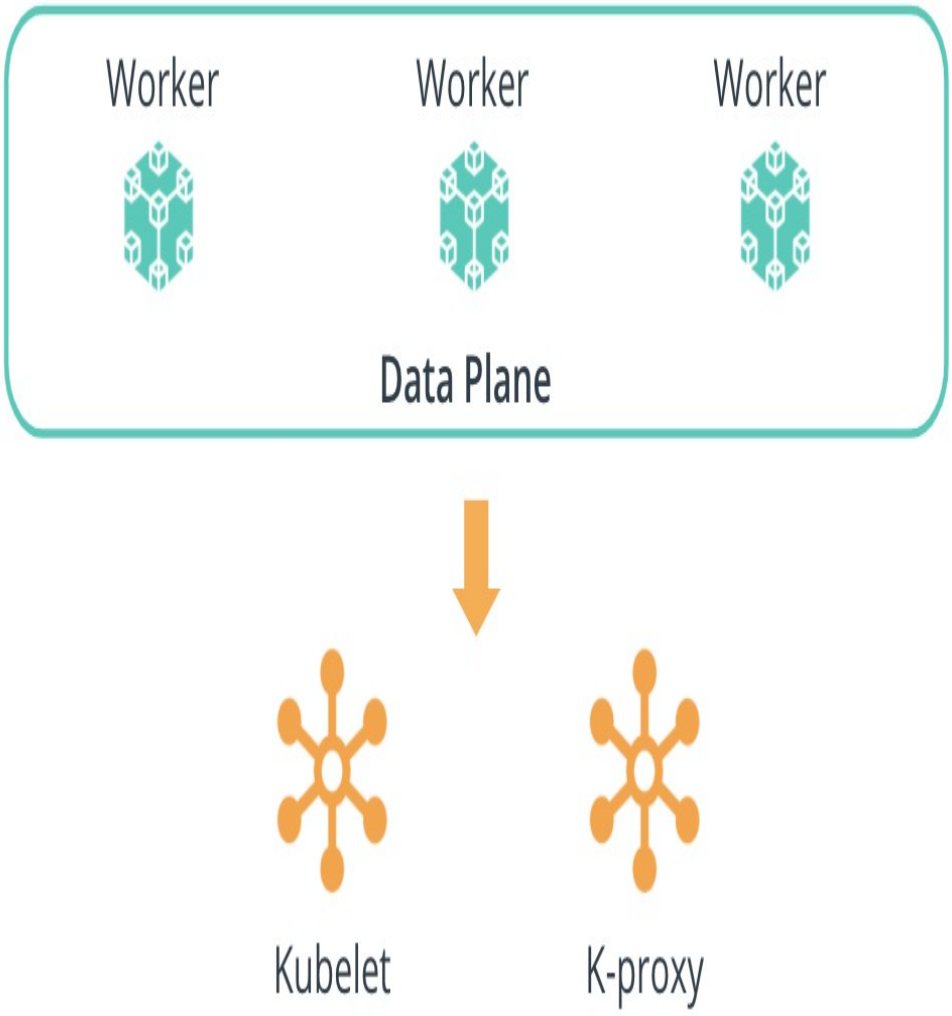
Control Plane components

The control plane consists of components that make global decisions about the cluster. These components are the:

- **kube-apiserver** - the nucleus of the cluster that exposes the Kubernetes API, and handles and triggers any operations within the cluster
- **kube-scheduler** - the mechanism that places the new workloads on a node with sufficient satisfactory resource requirements
- **kube-controller-manager** - the component that handles controller processes. It ensures that the desired configuration is propagated to resources
- **etcd** - the key-value store, used for backs-up and keeping manifests for the entire cluster

There are two additional components on the control plane, they are **kubelet** and **k-proxy**. These two are special and important as they are installed on **all** node. You can see the Data Plane below for more details.

Data Plane



Data Plane conponents

The data plane consists of the compute used to host workloads. The components installed on a worker node are the:

- **kubelet** - the agent that runs on **every** node and notifies the *kube- apiserver* that this node is part of the cluster
- **kube-proxy** - a network proxy that ensures the reachability and accessibility of workloads places on this specific node

  Important Note: The **kubelet** and **kube-proxy** components are installed on **all** the nodes in the cluster (**master and worker** nodes). These components

keep the *kube-apiserver* up-to-date with a list of nodes in the cluster and manages the connectivity and reachability of the workloads.

New terms

- **CRD** - Custom Resource Definition provides the ability to extend Kubernetes API and create new resources
- **Node** - a physical or virtual server
- **Cluster** - a collection of distributed nodes that are used to manage and host workloads
- **Master node** - a node from the Kubernetes control plane, that has installed components to make global, cluster-level decisions
- **Worker node** - a node from the Kubernetes data plane, that has installed components to host workloads

Further Reading

Explore Kubernetes features:

- Kubernetes DNS for Services and Pods
- Kubernetes CRDs
- Kubernete Cluster Autoscaler
- Kubernetes Architecture and Components