

How My Homelab Helped Me Understand Kubernetes Cluster Architecture

 medium.com/@pragalva.sapkota/how-my-homelab-helped-me-understand-kubernetes-cluster-architecture-6c56fc7eabd

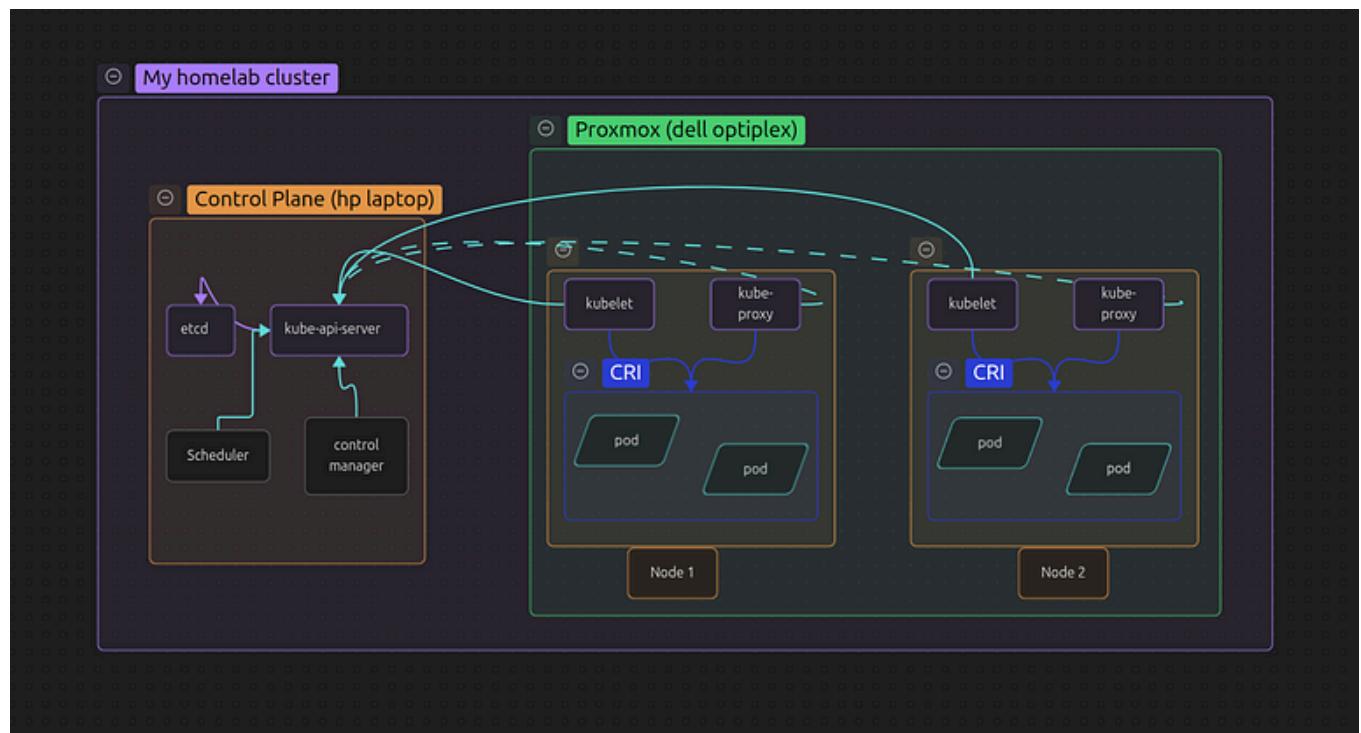
Pragalva Sapkota

December 13, 2025

Introduction

In this blog, I explain the cluster architecture of my Kubernetes homelab.

I walk through how the control plane and worker nodes are structured and how the core components work together in a real setup.



Architecture of Kubernetes Cluster

A Kubernetes cluster consists of a control plane and a set of workers (at least one) called nodes. Control plane acts like the brain of the cluster controlling the worker nodes, while workers performs the application workload.

The control plane does not run applications directly. Instead, it exposes APIs and runs controllers that decide what should happen in the cluster. Worker nodes execute those decisions by running pods and containers.

Components of Control Plane

Control plane components make cluster wide decisions such as scheduling pods and continuously monitoring the state of the cluster. They work to ensure that the current state of the cluster matches the desired state defined by Kubernetes objects like Deployments, Jobs, and Services.

Control plane components can run on one or multiple machines in the cluster. In many setups, they run on dedicated nodes, and user workloads are typically restricted from running on them for stability and security.

kube-apiserver

The kube apiserver is a core component of the Kubernetes control plane that exposes the Kubernetes API. All communication within the cluster, whether from users, controllers, or nodes, goes through the API server.

It acts as the front end of Kubernetes. Components don't talk to each other directly. They interact through the kube-apiserver

It is designed to scale horizontally meaning we can run several instances of it and balance traffic between them.

etcd

etcd is a consistent and highly available key value store used as the backing store for all cluster data. It stores information such as cluster state, configuration, and metadata.

In my setup, etcd was bootstrapped while configuring the Talos control plane. The control plane relies on etcd as the single source of truth for the cluster.

kube-scheduler

The kube scheduler watches for newly created pods that do not have a node assigned. When it detects such pods, it selects an appropriate node for them to run on.

Scheduling decisions are made based on multiple factors such as resource availability, constraints, affinity rules, and policies.

kube-controller-manager

A controller is a control loop that watches the state of the cluster through the kube apiserver and works to move the current state toward the desired state.

The kube controller manager runs multiple controller processes. Each controller is responsible for a specific aspect of cluster behavior.

Types of controllers:

- Responsible for monitoring node health and reacting when a node becomes unavailable.
- Watches Job objects and ensures that the specified number of pods run to completion for one time tasks.
- EndpointSlice objects contain references to a set of network endpoints. They group endpoints based on properties like IP family, protocol, and port numbers. Since pods are ephemeral, this controller continuously updates EndpointSlices so Services know which pods should receive traffic.

- Creates default ServiceAccount objects for new namespaces.

cloud-controller-manager

My cluster does not run a cloud controller manager because it is an on premise homelab.

The cloud controller manager separates components that interact with cloud provider APIs from components that interact only with the cluster. Core Kubernetes components communicate only with the cluster, while cloud specific logic is handled separately.

This design removes unnecessary dependencies on cloud providers and allows Kubernetes to run consistently across on premise and cloud environments.

Node Components

Node components run on every worker node and are responsible for managing pods and containers.

kubelet

kubelet is an agent that runs on each node in the cluster. It ensures that containers described in PodSpecs are running and healthy.

It receives PodSpecs from the API server and uses the container runtime to start and manage containers. kubelet only manages containers that are created by Kubernetes.

kube-proxy (optional)

kube-proxy is a network component that runs on each node. It maintains network rules that allow Services to route traffic to the correct backend pods.

container runtime

The container runtime is responsible for managing the execution and lifecycle of containers within the Kubernetes environment.

Addons

Addons use Kubernetes resources to extend cluster functionality. Namespaced addon resources typically belong to the namespace. Below are two addons that I find interesting:

DNS

Cluster DNS is a DNS service that provides name resolution for Kubernetes Services and Pods.

Although classified as an addon, DNS is required for most workloads to function correctly. Pods automatically receive DNS configuration that points to the cluster DNS service, allowing containers to resolve service names.

Web UI (Dashboard)

Allows web-based UI for the users to troubleshoot applications running on clusters as well as the cluster itself.

Conclusion

Seeing how the control plane and worker nodes interact in my own homelab made the concepts stick far better than just reading docs. This setup gives me a solid base to experiment, break things safely, and understand what is really happening when the cluster behaves the way it does.