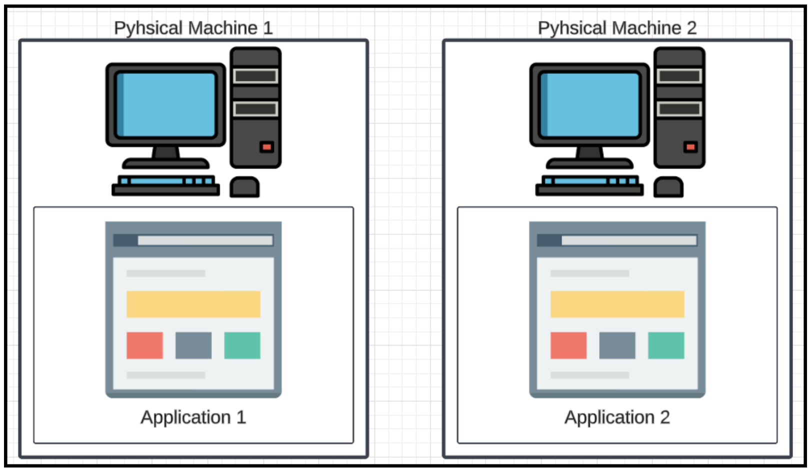
Kubernetes

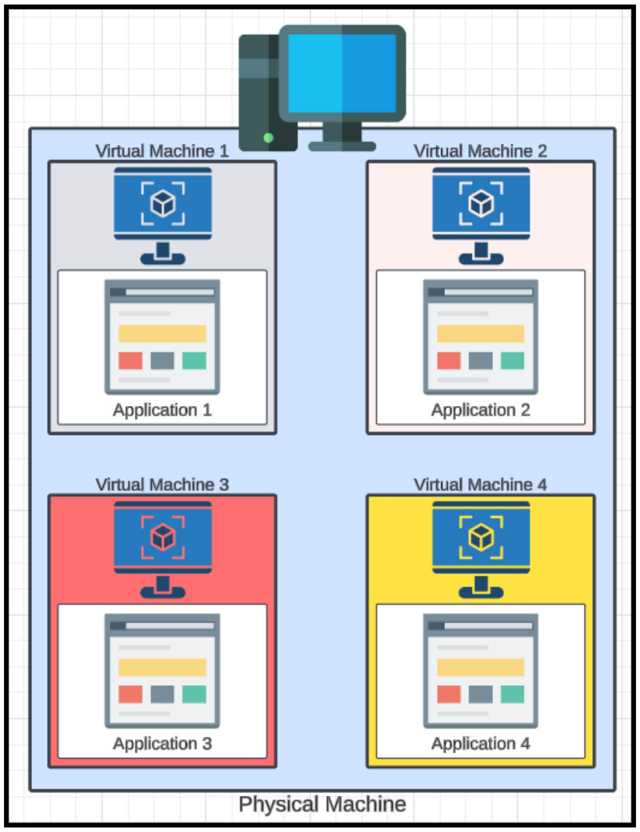
Contents

1. Introduction

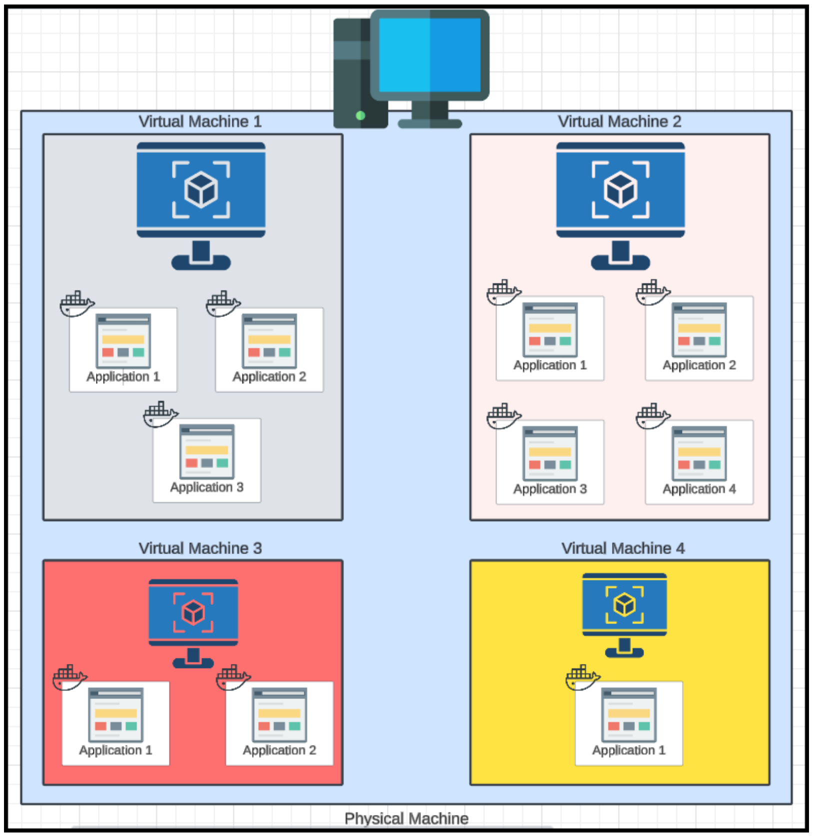
- What is Kubernetes (K8s)?
  - Kubernetes, abbreviated as K8s (since there are 8 letters between "K" and "S"), is a container orchestration tool.
  - It helps manage, scale, and deploy containers efficiently.
- Why Do We Need Kubernetes?



  - Initial Setup (Physical Machines):
    - Initially, applications were run on physical machines. Multiple applications on a single machine caused conflicts and lacked isolation.



  - Virtual Machines (VMs):
    - Virtualization solved the isolation problem by enabling multiple VMs on one physical machine. Each VM acted as an independent entity.



- Containers:
  - Containers introduced further efficiency by enabling multiple isolated applications within a single VM. Containers consume fewer resources compared to VMs.
- Scaling Issues:
  - As the number of containers grew, managing and scaling them became complex, especially with many engineers and applications.

- Challenges with Containers:
  - Manual Management:
    - Without Kubernetes, scaling containers manually is time-consuming and error-prone.
  - Load Balancing:
    - Uneven usage patterns (e.g., more users adding items to a cart than processing payments) lead to inefficiencies.
  - Dynamic Scaling:
    - Events like sales can cause sudden spikes in demand, requiring rapid scaling.

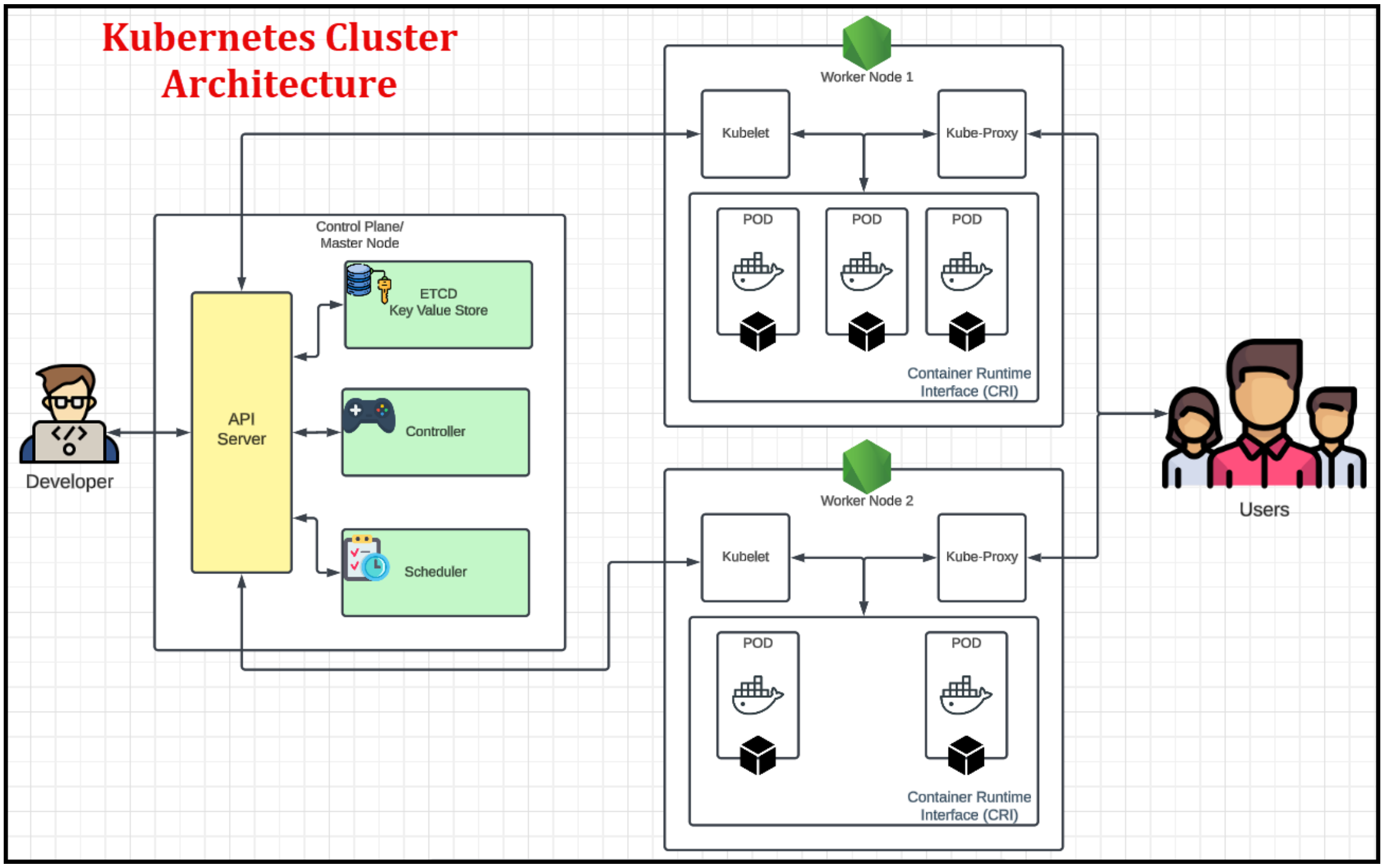
- Kubernetes Features:
    - Container Orchestration:
        - Kubernetes acts as a "project manager," coordinating tasks among containers.
    - Load Balancing:
        - It dynamically adjusts the number of containers based on real-time demand.
    - Autoscaling:
        - Kubernetes automatically scales up or down the number of containers to match workload requirements.
    - Resource Optimization:
        - Ensures optimal allocation of resources across containers.
- Real-World Example (E-Commerce):
    - Different tasks like adding to a cart, reviewing, and processing orders can have varying levels of demand.
    - Kubernetes balances the resources dynamically based on user activity (e.g., scaling up "add to cart" during a sale).
- Conclusion:

  Kubernetes simplifies container management, enabling scalability, resource optimization, and dynamic workload handling. This is critical for modern applications where demand can fluctuate unpredictably.

2. Kubernetes Architecture

Kubernetes architecture can be divided into two main parts:

1. **Control Plane (Master Node)**: The brain of Kubernetes, responsible for managing the cluster and making high-level decisions.
2. **Worker Nodes**: These are where the actual workloads (containers) run, similar to how your body parts execute tasks based on decisions made by your brain.



Control Plane (Master Node)

The control plane manages the entire Kubernetes cluster. Its key components include:

1. API Server
    - Acts as the front-end of the Kubernetes control plane.
    - All communication within the cluster or from users (external tools, scripts) happens through the API server.

- Similar to a "CR - Class Representative," it intermediates all interactions between components.

2. Scheduler
    - Determines **where** to schedule pods based on resource availability and policies.
        - Example: Decides whether a new pod should run on Worker Node 1 or Worker Node 2.
3. Controller Manager
    - Runs various controllers to maintain the cluster's desired state.
    - Types of controllers:
        - **Deployment Controller**: Ensures the desired number of replicas for an application.
        - **Replica Set Controller**: Maintains the specified number of pod replicas.
    - Continuously monitors the cluster state and takes corrective actions via the API server.
4. `etcd`
    - `etcd` is named after the `/etc` directory in Unix systems (for configuration files) and `"d"` for daemon, as stated by the `etcd` maintainers. It is not officially an acronym.
    - The idea behind `etcd`'s name is that it is a "`daemon`" for storing configuration data (similar to the role of `/etc` in a Linux system), but it extends this concept to distributed systems with features like high availability and consensus.
    - A distributed **key-value store** that acts as the database of Kubernetes.
    - Stores all cluster information, including:
        - Number of nodes.
        - Pod configurations.
        - Cluster state.

Worker Nodes

Worker nodes are where the actual work happens. Each worker node consists of:

1. Kubelet
    - Acts as an agent that communicates with the control plane (API server).
    - Ensures that containers are running in pods as per the control plane's instructions.
2. Pods
    - The **smallest deployable unit** in Kubernetes.
    - A pod is a wrapper around one or more containers.
    - Typically, **one container per pod**, but multiple containers can exist if needed.
    - Pods have their own **IP addresses**, which are used for communication. Containers inside a pod share this IP.
    - Pods are ephemeral. They are created, used, and destroyed. If a Pod fails, Kubernetes may create a new Pod with a new IP address to replace it (depending on the configuration).
3. Kube-Proxy
    - Kube-Proxy is a key component of Kubernetes responsible for maintaining network rules and facilitating communication between services and pods in a cluster.
    - It acts as a network proxy and load balancer, ensuring that requests are routed correctly.

Key Points About Pods

- **Pod as a Unit**:
    - All containers within a pod share resources like storage and network.
    - They are tightly coupled and designed to work together.
- **Pod Networking**:
    - Pods, not containers, are assigned IP addresses.
    - Communication with containers happens via the pod's IP.

- **Deployment**:
    - Multiple pods can run on a single node (virtual machine).
    - Pods are managed and scaled by higher-level abstractions like deployments and replica sets.

Summary of Workflow

1. **User Interaction**:
    - Users interact with the **API Server** to submit requests (e.g., deploying an application).
2. **Scheduler**:
    - The scheduler decides where to place the pods (worker nodes).
3. **Controller Manager**:
    - Ensures the desired state of the cluster (e.g., correct number of replicas).
4. **etcd**:
    - Tracks and stores the cluster's current and desired state.
5. **Kubelet**:
    - Executes the control plane's decisions on worker nodes.
6. **Pods and Containers**:
    - Containers run inside pods, and pods perform the actual workload.

This architecture ensures that Kubernetes operates as a robust, scalable, and fault-tolerant system for managing containerized applications.

Kubernetes Types

1. Vanilla Kubernetes

- Refers to the **original and open-source** distributed system.
- Open-source nature allows for modifications and customizations based on requirements.
- Requires **manual installation** on infrastructure using tools like **kubeadm**.
- **Use Case**: Provides full control and flexibility over the Kubernetes environment.
    - Example: AWS's **EKS** uses a modified version of Kubernetes.

        - Suitable for companies that want to wrap Kubernetes with their own changes and distribute it.

2. Kubernetes for Developers

Designed to allow developers to test and play around with Kubernetes locally. Includes:

- **Kind (Kubernetes in Docker):**
    - Runs Kubernetes clusters inside Docker containers.
    - Lightweight and ideal for testing on local machines without incurring additional costs.
- **Minikube and MicroK8s:**
    - Alternatives to Kind.
    - Serve the same purpose of enabling local testing of applications on Kubernetes.
- **Purpose:** All these tools are meant for developers to test applications on Kubernetes before deploying them into production environments.

3. Managed Kubernetes Services

Pre-configured Kubernetes environments offered by cloud providers:

- **Google Kubernetes Engine (GKE)**
- **Amazon Elastic Kubernetes Service (EKS)**
- **Azure Kubernetes Service (AKS)**
- Other options include **IBM Cloud Kubernetes** and **Oracle Kubernetes Engine**.

3. Kubernetes Practice

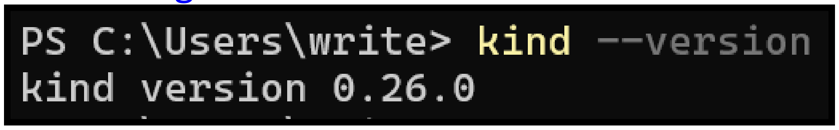
Why Use `kind` (Kubernetes in Docker)?

- Lightweight and Efficient: `kind` runs Kubernetes clusters inside Docker containers, suitable for systems with limited resources.
- Local Development: Perfect for experimenting with Kubernetes on laptops or small machines.
- Resource-Friendly: Can run single-node or multi-node clusters on minimal setups.

Installation Steps for `kind`

- Prerequisites:
  - Ensure Docker is installed and running.
  - On windows, I used…

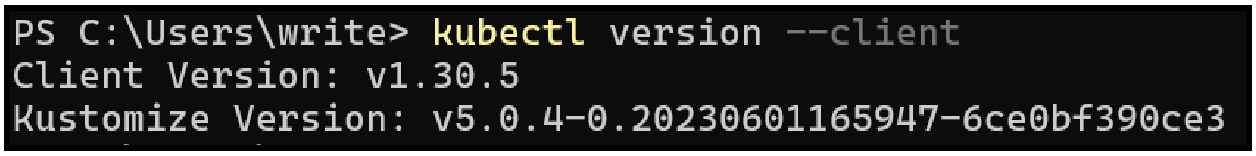
```
winget install Kubernetes.kind
```



```
PS C:\Users\write> kind --version
kind version 0.26.0
```

  - We need to install `kubectl`…

```
curl.exe -LO https://dl.k8s.io/release/v1.32.0/bin/windows/amd64/kubectl.exe
```



```
PS C:\Users\write> kubectl version --client
Client Version: v1.30.5
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
```

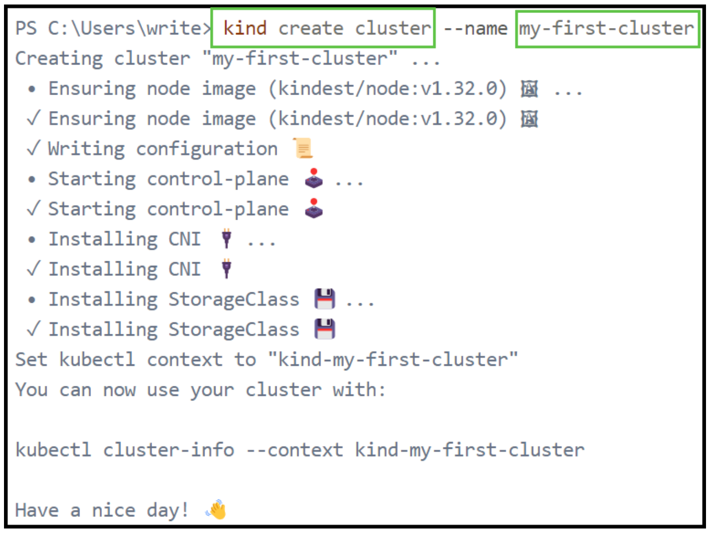
Create a Kubernetes Cluster:

- To create a single-node cluster, use…

```
kind create cluster
```

- Optional: Specify a Cluster Name:

```
kind create cluster --name
my-first-cluster
```

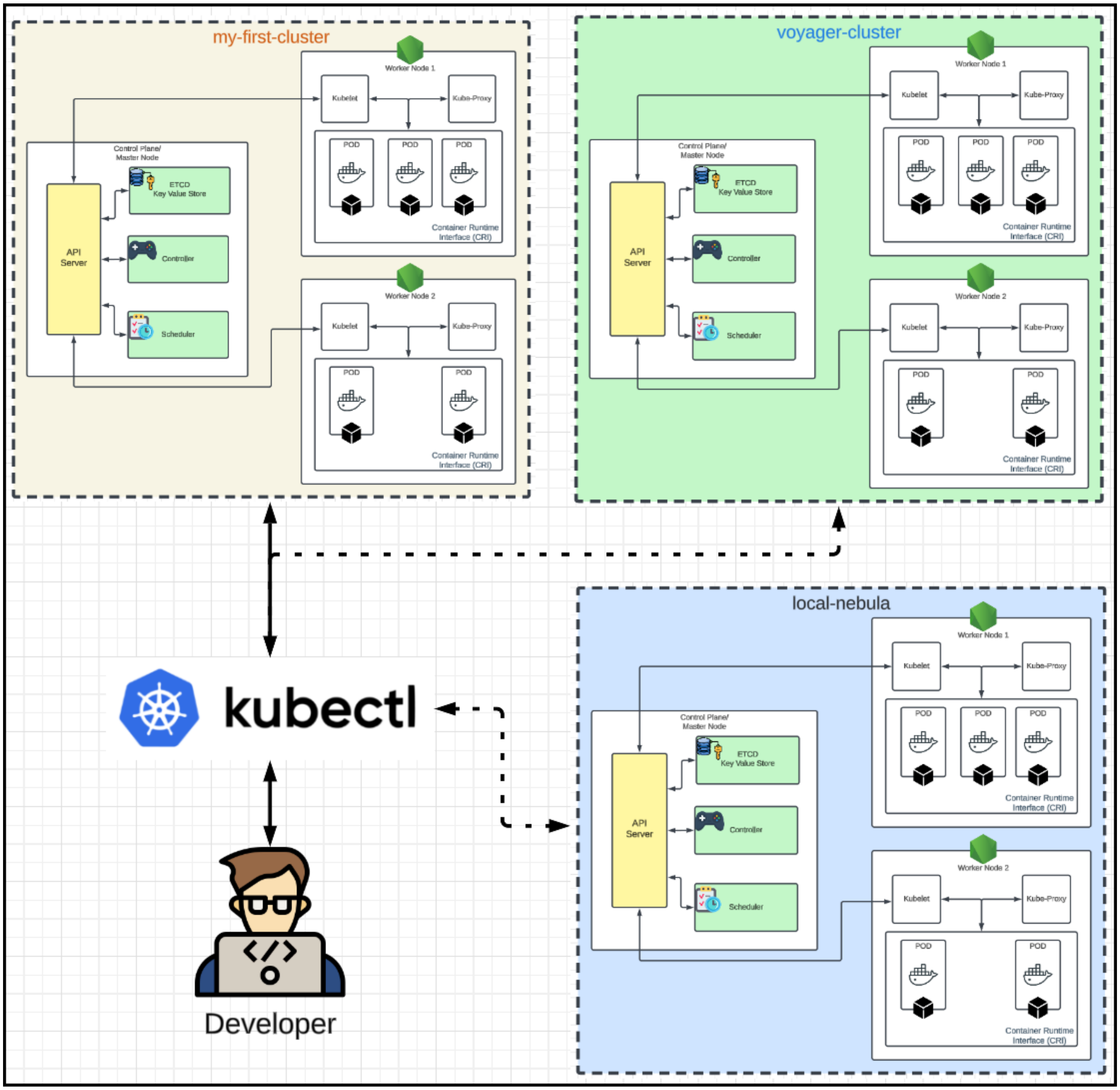


```
PS C:\Users\write> kind create cluster --name my-first-cluster
Creating cluster "my-first-cluster" ...
 • Ensuring node image (kindest/node:v1.32.0) 🖼 ...
 ✓ Ensuring node image (kindest/node:v1.32.0) 🖼
 ✓ Writing configuration 📜
 • Starting control-plane 🕹️ ...
 ✓ Starting control-plane 🕹️
 • Installing CNI 🔌 ...
 ✓ Installing CNI 🔌
 • Installing StorageClass 💾 ...
 ✓ Installing StorageClass 💾
Set kubectl context to "kind-my-first-cluster"
You can now use your cluster with:

kubectl cluster-info --context kind-my-first-cluster

Have a nice day! 👋
```

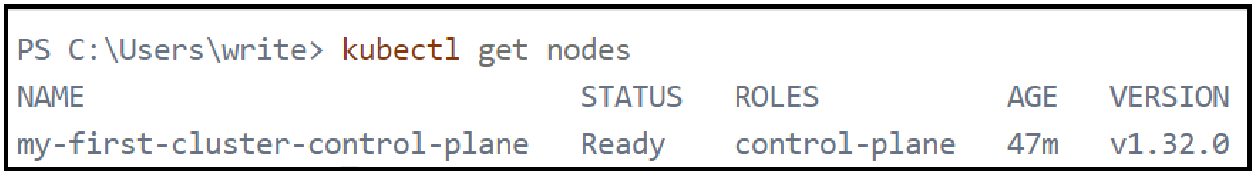
- Developers use kubectl to interact with Kubernetes clusters by creating and managing resources defined in YAML manifests or issuing commands directly.
- kubectl communicates with the Kubernetes API to perform actions like deploying applications, scaling workloads, and monitoring cluster health.



Verifying Cluster and Interaction

- Check Cluster Nodes:

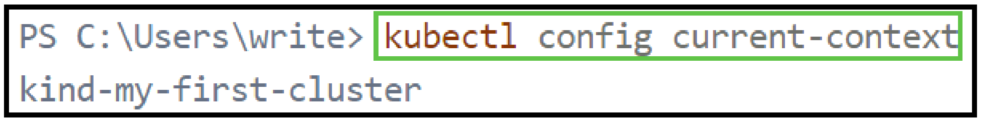
```
kubectl get nodes
```



```
PS C:\Users\write> kubectl get nodes
NAME                           STATUS   ROLES           AGE   VERSION
my-first-cluster-control-plane Ready    control-plane   47m   v1.32.0
```

- Check Current Context:
  - Displays the active cluster context

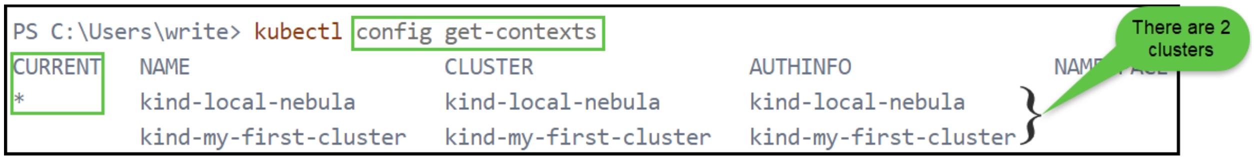
```
kubectl config current-context
```



```
PS C:\Users\write> kubectl config current-context
kind-my-first-cluster
```

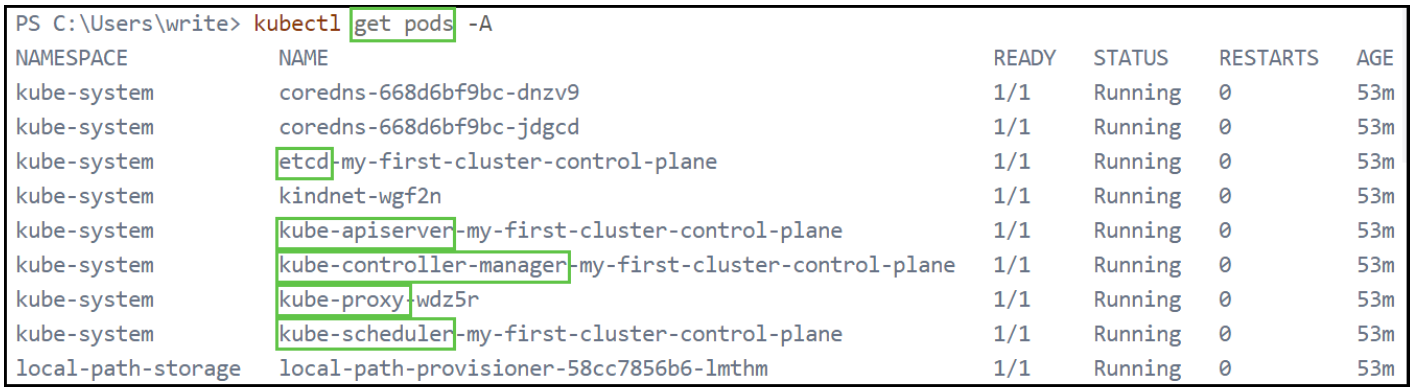
- List Available Contexts:

```
kubectl config get-contexts
```



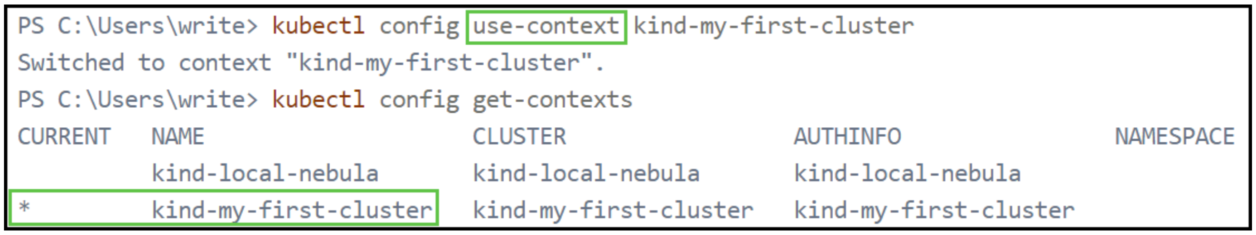
```
PS C:\Users\write> kubectl config get-contexts
CURRENT   NAME                    CLUSTER                 AUTHINFO                NAMESPACE
*         kind-local-nebula       kind-local-nebula       kind-local-nebula
          kind-my-first-cluster   kind-my-first-cluster   kind-my-first-cluster
```

- List all Pods across all namespaces in a cluster:



```
PS C:\Users\write> kubectl get pods -A
NAMESPACE            NAME                                                     READY   STATUS    RESTARTS   AGE
kube-system          coredns-668d6bf9bc-dnzv9                                 1/1     Running   0          53m
kube-system          coredns-668d6bf9bc-jdgcd                                 1/1     Running   0          53m
kube-system          etcd-my-first-cluster-control-plane                      1/1     Running   0          53m
kube-system          kindnet-wgf2n                                            1/1     Running   0          53m
kube-system          kube-apiserver-my-first-cluster-control-plane            1/1     Running   0          53m
kube-system          kube-controller-manager-my-first-cluster-control-plane   1/1     Running   0          53m
kube-system          kube-proxy-wdz5r                                         1/1     Running   0          53m
kube-system          kube-scheduler-my-first-cluster-control-plane            1/1     Running   0          53m
local-path-storage   local-path-provisioner-58cc7856b6-lmthm                  1/1     Running   0          53m
```

- Change the context cluster:



```
PS C:\Users\write> kubectl config use-context kind-my-first-cluster
Switched to context "kind-my-first-cluster".
PS C:\Users\write> kubectl config get-contexts
CURRENT   NAME                    CLUSTER                 AUTHINFO                NAMESPACE
          kind-local-nebula       kind-local-nebula       kind-local-nebula
*         kind-my-first-cluster   kind-my-first-cluster   kind-my-first-cluster
```