**[K8s Made Easy]**

🌟 **What is Kubernetes?**

Before diving into Kubernetes, make sure you have a basic understanding of **Docker** since itʼs a prerequisite.

🚀 **What is Kubernetes (K8s)?**

Kubernetes, often called **K8s**, is a powerful **container orchestration platform**. Think of it as a tool that helps you **manage containers** efficiently. It handles tasks like creating, updating, and deleting containers automatically.

🛠 **Why Use Kubernetes?**

Here are some common scenarios where Kubernetes is useful:

 **Cloud-Native Deployments:** You have Docker images stored in a registry (like Docker Hub), and you want to deploy them quickly and efficiently in a cloud environment.

 **Self-Healing:** If something goes wrong (like a crash), Kubernetes detects it and **automatically recovers** your application without manual intervention.

 **Auto-Scaling:** Kubernetes can **scale up or down** your application based on traffic, making sure you use just the right amount of resources.

 **Monitoring and Observability:** Kubernetes provides tools and integrations for monitoring your entire system, often with easy-to-use dashboards.

🤓 **Fun Fact:**

Kubernetes is often abbreviated as **K8s** because there are **8 letters** between **K** and **s**.

# 🌐 After Learning Kubernetes

In Kubernetes, your **frontend** and **backend** are all deployed as **Pods** inside a Kubernetes cluster.

## 🔍 Understanding Kubernetes Jargon 🖥 Nodes

A **Node** is a single machine (physical or virtual) in a Kubernetes cluster. There are two types of nodes:

 **Master Node Control Plane):**

This node **manages** the cluster, deciding where to place your applications and ensuring everything runs smoothly.

It handles tasks like **deploying containers**, **auto-healing**, and **listening to commands** from developers.

 **Worker Node:**

These nodes are where your **applications actually run** (backend, frontend, etc.).

They do the heavy lifting by running your application code in containers.

## 📡 Key Components of the Control Plane

 **API Server:**

Acts as the **main communication point** in Kubernetes.

It handles requests from users (via kubectl ), other Kubernetes components, and external applications.

It **authenticates users**, checks permissions, and provides **health checks** and **metrics** for monitoring.

 **etcd:**

A **key-value store** that holds all the data about your Kubernetes cluster (like what Pods are running and where).

It is highly consistent and available, making sure the cluster state is always up-to-date.

 **Kube-Scheduler:**

Decides **which Pod** should run on which node based on resource availability.

It watches for new Pods and assigns them to the right node.

 **Kube-Controller-Manager:**

Manages various **controllers**, which are components that make sure the desired state of the cluster matches the actual state.

Examples of controllers:

**Node Controller:** Keeps an eye on nodes and notices when they go down.

**Deployment Controller:** Manages updates and scaling of your applications Deployments).

**ReplicaSet Controller:** Ensures the correct number of replicas (copies) of a Pod are running.

## 🛠 Key Components of Worker Nodes

 **Kubelet:**

A **small agent** running on each worker node.

It makes sure that containers Pods) are running as expected.

It watches the API server for new tasks, starts and stops containers, and reports the node status back to the API server.

 **Kube-Proxy:**

Handles **networking** within the cluster, ensuring you can communicate with your Pods.

It acts as a **network proxy**, routing traffic to the correct Pod.

 **Container Runtime:**

The software that actually **runs the containers** on a node.

Examples include **containerd**, **CRIO**, and **Docker**.

It works closely with the kubelet to manage the container lifecycle.

## 🔗 Cluster

A **Kubernetes Cluster** is made up of multiple **Master Nodes** and **Worker Nodes** working together. You can easily **add or remove nodes** as needed.

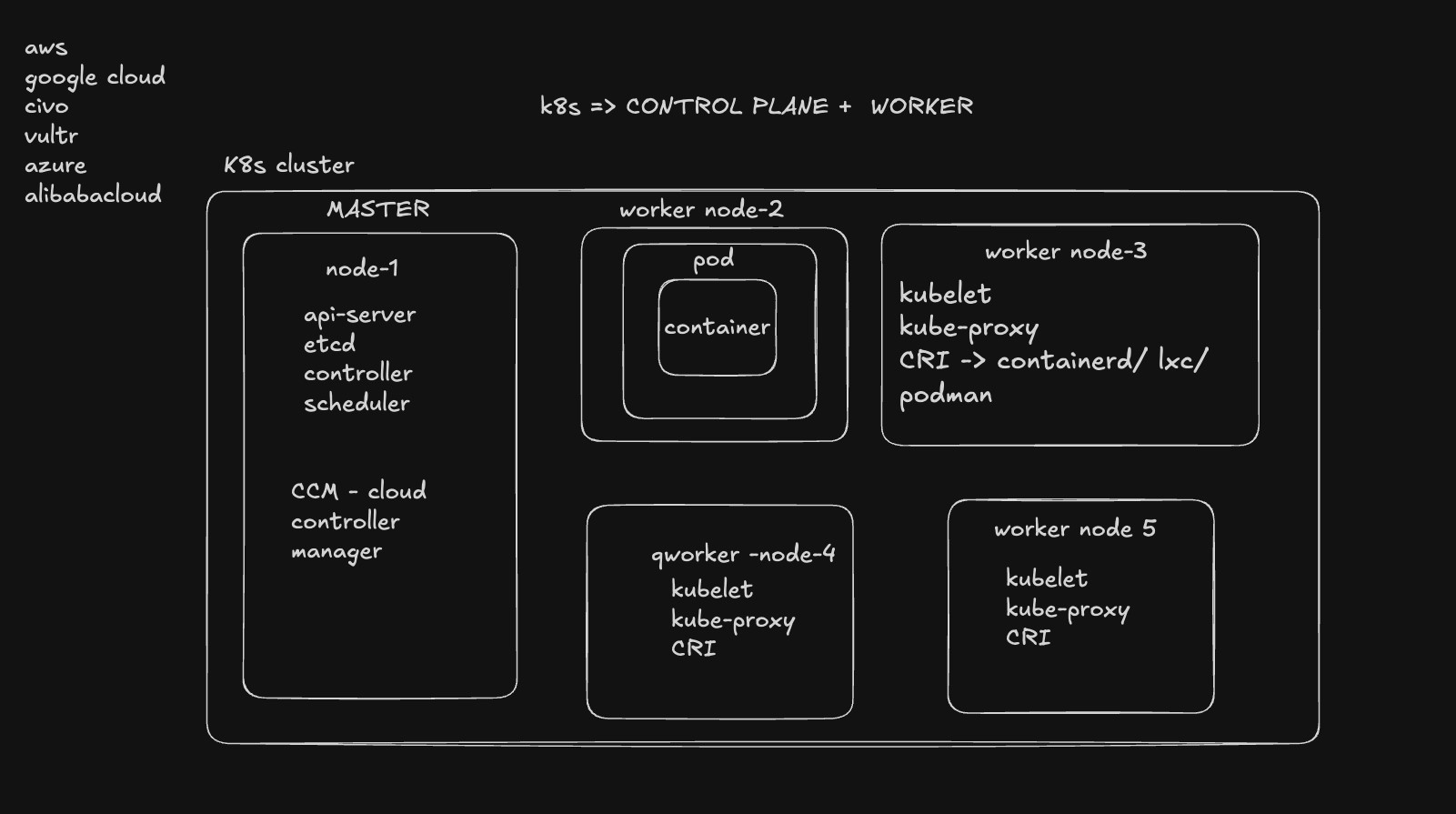
## 🛑 Quick Definitions

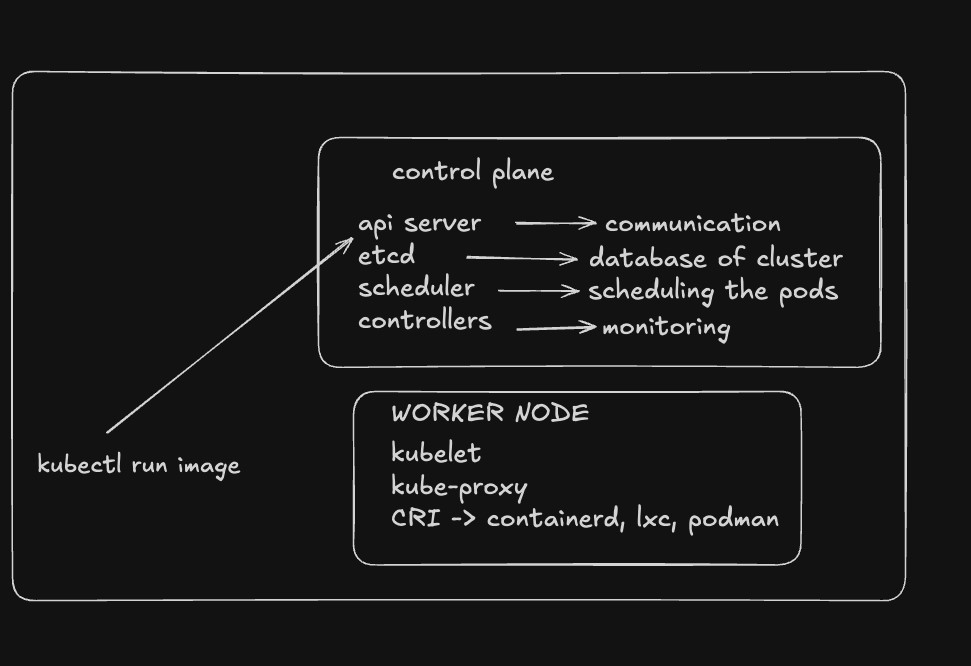
**Images:** A Docker image is a package that contains everything needed to run an application (code, runtime, libraries, etc.).

**Containers:** A container is a running instance of an image (e.g., running docker

run postgres ).

**Pods:** The **smallest unit** you can deploy in Kubernetes. A Pod can contain one or more containers.





# 🛠 How to Create a Kubernetes (K8s) Cluster

A **Kubernetes cluster** is like a team of computers (nodes) working together to run your applications. You can create a cluster **locally** on your computer or use a **cloud provider** (like Google Cloud, AWS to create one.

## 🏠 Creating a K8s Cluster Locally

To set up a Kubernetes cluster on your local machine, youʼll need **Docker** installed. You can use tools like **Minikube** or **Kind** Kubernetes in Docker).

## ⚙ Using Kind (Kubernetes in Docker)

**Kind** is a tool that helps you create a Kubernetes cluster using Docker containers.

 **Install Kind:** Follow the guide here: [Kind Quick Start Guide.](https://kind.sigs.k8s.io/docs/user/quick-start/#installation)

 **Create a Simple Single-Node) Cluster:**

This is like setting up a practice environment on your computer.

kind create cluster --name local

Now, check whatʼs running with:

docker ps

Youʼll see a single container, which acts as the

**control plane**

(the boss that

manages everything).



**Delete the Cluster:**

If youʼre done practicing or want to start fresh, delete it with:

kind delete cluster --name local

## 🛠 Multi-Node Setup (Control Plane + Workers)

In real-world projects, you usually need multiple nodes: one to **manage** (control plane) and others to **run your app** (workers). Hereʼs how you can set it up:

This setup has **1 boss node** (control plane) and **2 worker nodes**.



**Create a**

**clusters.yml**

**File:**

kind: Cluster

apiVersion: kind.x-k8s.io/v1alpha4

nodes:

-

role: control-plane

-

role: worker

-

role: worker

 **Create the Cluster with Multiple Nodes:**

kind create cluster --config clusters.yml --name local

 **Check the Running Containers:**

docker ps

Youʼll see multiple containers: one for the control plane and two for the

workers.

📝 **Real-World Example:**

Think of this like a restaurant:

The **control plane** is the manager who takes orders and decides what needs to be done.

The **worker nodes** are the chefs who do the actual cooking.

The manager (control plane) doesnʼt cook but makes sure everything is running smoothly in the kitchen.

## 🌐 Creating a K8s Cluster on the Cloud

If you want to create a cluster that can be accessed from anywhere, use a **cloud provider**:

 **GKE Google Kubernetes Engine)** Managed by Google Cloud.

 **AWS EKS Elastic Kubernetes Service)** Managed by Amazon AWS.

 **Vultr Kubernetes Service** A simpler option if youʼre looking for a cheaper alternative.

In the cloud, you donʼt have to worry about setting up nodes manually; the cloud provider takes care of it.

## 🏠 Using Minikube Locally

**Minikube** is another popular tool to create a Kubernetes cluster on your local machine.

 **Install Minikube:** Follow the guide here: [Minikube Installation](https://minikube.sigs.k8s.io/docs/start/).

 **Start a Single-Node Cluster:**

minikube start

This will create a single-node cluster (like setting up a small sandbox environment).

 **Check Running Containers:**

Youʼll see a single container running.

docker ps

💡 **Note:** While a single-node setup works for practice, itʼs not ideal for production. In real projects, you donʼt want the control plane (manager) to do the work (run containers). Itʼs like making the restaurant manager cook while also managing the staff.

## 📡 Kubernetes API Server: The Heart of the Control Plane

The **Kubernetes API Server** is like the main entrance to your Kubernetes cluster. Itʼs the central hub that lets you control everything, like starting, stopping, and managing applications (pods). The API Server listens for requests and makes sure the cluster does what you ask.

🏠 **How Does It Work?**

Imagine the API Server as a **reception desk** at a big company:

It listens to requests (from you or other tools).

It checks if youʼre allowed to make those requests (authentication and authorization).

It passes your request to the right department (node) to get the job done.

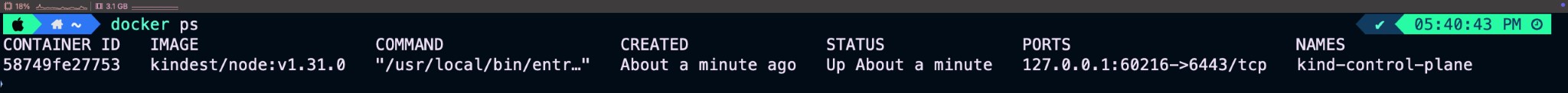
## 🛠 Try Using the Kubernetes API

You can actually interact with the API Server using a browser or command line:

 **Find the Control Plane Container:**

Run the following command to see all running containers:

docker ps



Look for the container that says "control-plane". This is where the API

Server is running.



**Access the API**

Open your browser or use a tool like

curl

to hit the API endpoint.

curl <https://127.0.0.1:60216-/api/v1/namespaces/default/p

ods>

This command asks the API Server to list all the pods in the

**default**

**namespace**

.

📝 **Real-World Example:**

Think of this like checking your bank account using an online banking app:

You log in (authentication).

You ask to see your account balance API request).

The bank system verifies youʼre allowed to view this information (authorization).

It shows your account balance (response).

## 🔐 Security: Authentication and Authorization

The API Server is **secure** and doesnʼt let just anyone access it.

When you send a request, it first **authenticates** you (checks who you are).

Then, it **authorizes** your request (checks if youʼre allowed to do what youʼre asking).

## 🗂 Configuration File

Your credentials and settings to access the cluster are stored in a file called ~/.kube/config . This file contains information like:

The clusterʼs address (where to send API requests).

Your user credentials (how the API Server knows itʼs you).

Which context to use (if youʼre working with multiple clusters).

📂

**Location:**

~/.kube/config



📝 **Note:** You donʼt have to manually update this file. Tools like **kubectl** and **Kind** automatically manage it for you when you create a cluster.

This should make the Kubernetes API Server more understandable and give you a hands-on way to try it out. Let me know if you need more examples or deeper explanations!

## 🛠 kubectl : The Command-Line Tool for Kubernetes

**kubectl** is your main tool for working with a Kubernetes cluster. Think of it as your **remote control** to manage everything in the Kubernetes world. It helps you interact with the Kubernetes API server, making it easy to create, update, delete, and inspect all Kubernetes resources like **pods**, **nodes**, **services**, and more.

## 📥 How to Install kubectl

To get started, you need to install kubectl . Follow the official installation guide based on your system:

🔗 [Install kubectl](https://kubernetes.io/docs/tasks/tools/#kubectl)

## 🔍 Checking if kubectl Works

Once installed, you can run a few basic commands to make sure everything is set up correctly.

 **Check Nodes in Your Cluster:**

kubectl get nodes

This command lists all the nodes (machines) in your Kubernetes cluster.

 **Check Running Pods:**

kubectl get pods

This command shows all the pods (applications) currently running in your

cluster.

## 🛑 Common Issues

If you get an error, make sure your kubectl is configured correctly and your cluster is running.

You might need to set the **KUBECONFIG** environment variable if your configuration file isnʼt found:

export KUBECONFIG=~/.kube/config

## 🔍 See the Exact API Request

Curious about what happens behind the scenes when you use kubectl ? You can add the --v=8 flag to see the HTTP request that kubectl sends to the API server.

kubectl get nodes --v=8

This shows detailed logs, including the exact API request and response. Itʼs useful for **debugging** and understanding how kubectl interacts with the API server.

## 📝 Real-World Example

Imagine youʼre the manager of a food delivery service:

**kubectl** is like your app, where you can check the status of deliveries (pods),

see available delivery drivers (nodes), and update orders (resources).

When you type a command like kubectl get pods , itʼs like checking your app to see which deliveries are in progress.

With kubectl , you can easily manage and monitor your Kubernetes cluster, just like a manager overseeing operations through a control panel.

Let me know if you need more explanations or commands! This should help make

kubectl easier to understand and use.

## 🚀 Creating Your First Pod in Kubernetes

So far, weʼve learned about the **5 key terms** in Kubernetes:

**Cluster** A group of connected machines (nodes) running Kubernetes.

**Nodes** The individual machines in your cluster (master/control plane and worker nodes).

**Images** Pre-built software packages, like templates (e.g., NGINX image).

**Containers** Running instances of images, like apps in action.

**Pods** The smallest deployable unit in Kubernetes, which can contain one or more containers.

🎯 **Letʼs Deploy a Container in a Pod!**

We already have a Kubernetes cluster with 3 nodes running locally or on the cloud. Now, let's deploy an NGINX container inside a pod.

## 🖼 Step 1: Find a Docker Image

Weʼll use the **NGINX image** from Docker Hub. Itʼs a popular web server image.

🔗 [NGINX Docker Image](https://hub.docker.com/_/nginx)

## 🐳 Step 2: Run NGINX Locally (Optional Check)

Before we deploy it in Kubernetes, letʼs see how it works using Docker:

docker run -p 3005:80 nginx

**p 300580**

 This maps port

**80**

inside the container to port

**3005**

on your

machine.

Visit [**http://localhost:3005**](http://localhost:3005/) in your browser to see the NGINX welcome page.

## 🚀 Step 3: Start an NGINX Pod in Kubernetes

Now, letʼs deploy the same NGINX image inside a Kubernetes pod:

kubectl run nginx --image=nginx --port=80

**kubectl run**

 Tells Kubernetes to create a pod.

**-**

**image=nginx**

 Specifies the image we want to use NGINX in this case).

**-**

**port**

**=80**

 Exposes port

**80**

of the container.

## 📊 Step 4: Check Pod Status

See if your pod is up and running:

kubectl get pods

You should see your

**nginx**

pod with a status like

**Running**

or

**Pending**

.

## 📄 Step 5: View Pod Logs

Check the logs of the NGINX pod to see whatʼs happening inside:

kubectl logs nginx

This command gives you a peek into the output of the NGINX web server running inside your pod.

## 🔍 Step 6: Get Detailed Information About the Pod

To see more details about your pod, use:

kubectl describe pod nginx

This provides information like IP address, container image, events, and more.

## 🤔 Common Questions

 **How can I stop the pod?**

kubectl delete pod nginx

 **How can I access the NGINX pod from my browser?**

By default, the pod is **not exposed** to the outside world. Weʼll need to use a **Service** (which weʼll cover later) to make it accessible.

 **How many pods can I start?**

This depends on your cluster resources CPU, memory). Kubernetes can handle thousands of pods, but itʼs good to monitor your resource usage.

## 🛑 Stopping a Pod

When you want to **stop or remove a pod**, you can delete it using kubectl .

## ✂ Step 1: Delete the Pod

Run the following command to stop and delete the NGINX pod:

kubectl delete pod nginx

**delete pod**

 This command tells Kubernetes to stop and remove the pod named

**nginx**.

Once deleted, the pod will **no longer** be running in your cluster.

## 🛑 Step 2: Verify the Pod is Deleted

Check the current status of your pods to make sure itʼs gone:

kubectl get pods

If the deletion was successful, you shouldnʼt see the

**nginx**

pod listed

anymore.

If you still see it with a status like **Terminating**, give it a moment to fully shut down.

## ⚠ Note

In Kubernetes, deleting a pod is typically safe because Kubernetes automatically recreates pods if they are part of a **Deployment**, **ReplicaSet**, or

**StatefulSet**. In our case, since we created the pod manually using kubectl run , it won't be recreated.

Next up, weʼll learn how to create more robust configurations using **Deployments**, which ensure that pods automatically restart if they are deleted or fail.

## 📝 Kubernetes Manifest

A **manifest** in Kubernetes is a YAML file that defines the desired state of a Kubernetes resource (like Pods, Deployments, or Services). It allows you to describe how you want your application or infrastructure to look, and Kubernetes ensures it matches that state.

## Example: Creating a Pod with a Manifest

Instead of running kubectl run , you can create a manifest file YAML to declare the desired state for the pod.

Hereʼs the manifest for deploying an **NGINX** pod:

apiVersion: v1 # Defines the version of the Kubernetes A

PI

kind: Pod # This indicates we're creating a Pod

metadata: # Metadata about the Pod(name,labels,et

c.)

name: nginx # Name of the Pod

spec: # Specification for the Pod's contents containers: # Defines the containers in the Pod - name: nginx # Name of the container image: nginx # Docker image for the container ports: # Ports the container will use - containerPort: 80 # Port the container listens on

## 🛠 Breaking Down the Manifest

**apiVersion: v1**  This tells Kubernetes to use version 1 of the API. **kind: Pod**  We're defining a **Pod** resource. **metadata:**  Metadata provides details like the name of the Pod ( nginx ).

**spec:**  This section defines the containers inside the Pod, which in this case is

an NGINX container using the nginx image.

## 🖥 Applying the Manifest

To create the pod using the manifest:

 Save the above YAML in a file named manifest.yml .

 Apply it using the kubectl apply command:

kubectl apply -f manifest.yml

This will create the Pod as described in the manifest.

## ❌ Deleting the Pod

To delete the pod created from the manifest:

kubectl delete pod nginx

This will remove the

nginx

pod from the cluster.

🌱 **Why Use Manifests?**

Manifests are **declarative**—you define what you want, and Kubernetes will take care of making sure it happens.

They allow you to store the configuration in version control, enabling easier tracking, collaboration, and reuse.

Manifests are a core part of Kubernetes, helping you manage the state of your resources effectively and consistently.