# K8sCIFAR

Data-Parallel Training of CIFAR-10 Across Physical Nodes using KUBERNETES

Authors:

**Darpan Bhattacharya** & **Soham Bhattacharya**

Ramakrishna Mission Vivekananda Educational and Research Institute

April 24, 2025

# Agenda

# Introduction

**Definition**: `Data parallel training` is a method used in deep learning to speed up the training of large models by distributing the workload across multiple computing devices, like GPUs or TPUs.

In data parallelism, each device gets **a copy of the model** and is assigned a **different subset of the training data**. All devices perform the forward and backward passes independently and compute gradients on their respective data batches.

# KUBERNETES

## What is it?

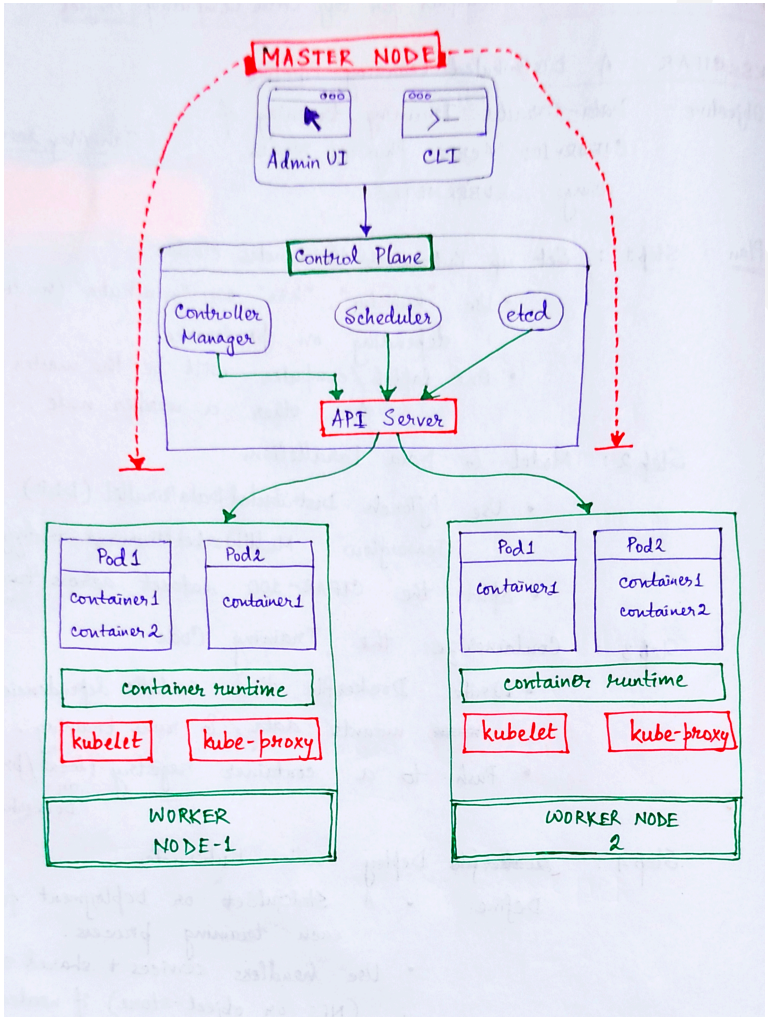
**Kubernetes** (also called K8s) is an open-source container orchestration platform that automates the deployment, scaling & management of containerized applications.

- `"kubeadm"`, `"minikube"`, `"k3s"` are tools used for setting up and managing Kubernetes clusters, but each has a different case scenarios to be served.

# Tools

| Tool | Best for | Cluster Type | Ease of use |
|---|---|---|---|
| `kubeadm` | Manual, full-featured clusters | multi-node, full k8s | Medium |
| `minikube` | local dev & testing | single-node | Easy |
| `k3s` | Edge, Iot, lightweight setups | single/multi-node | Very easy |

# Kubernetes Architecture



# Architecture Nomenclatures

- **Admin UI**: A web-based UI to manage and monitor Kubernetes clusters. "Kubernetes Dashboard" is the official Admin UI provided by Kubernetes. (We haven't used it in our project).
- **Command Line Interface (CLI)**: The CLI tool is called `"kubectl"`. It interacts with the Kubernetes API Server using typed commands. kubectl connects us to the control plane & lets us manage the cluster from the terminal.

# Architecture Nomenclatures

- **Control Plane**: The **brain** of your Kubernetes is called the control plane. It makes all global decisions (like scheduling), detects/responds to cluster events, manages desired states.
  - `kube-apiserver`: front door to the cluster, handles all API requests
  - `etcd`: distributed key-value database that stores all cluster data
  - `kube-scheduler`: decides which node a pod should run on
  - `kube-controllermanager`: runs background controllers (ensures replicas are running)
  - `cloud-controllernanager`: integrates with cloud platform

# Architecture Nomenclatures

- **Pods**: A Pod is the smallest most basic deployable unit in Kubernetes. It is like a tiny box that wraps your application's container(s).
  Think of pod as a wrapper around one or more containers that:
  - share the same network
  - share the same storage
  - are scheduled to run together

## Why not just use one container directly?

Kubernetes doesn't deal with containers - it deals with pods. Containers need some extra stuff (like networking, storage & orchestration) and pods give that structure.

# Architecture Nomenclatures

## Container

Think of a container like a *mini virtual machine*, but much faster and more efficient. It runs an isolated process on the host OS. It includes only the app code + runtime + libraries + dependencies, not a full OS. It's based on a container image (e.g., a Docker image).

# Architecture Nomenclatures

## kubelet

Brain of each worker node that reads pod specifications from API Server.

## kube-proxy

Network traffic-controller on each node.

# Docker File

A **Dockerfile** is like a blueprint or recipe for creating a Docker image - which is basically a snapshot of everything your app needs to run [code, dependencies, OS, etc].
We write a dockerfile to define how to package your app & once the image is built, Kubernetes can use it to spin up containers inside Pods.

**Flow**:

1. Write a dockerfile
2. You use it to build a docker image
3. You push that image to a container registry (like DockerHub)
4. Kubernetes pulls that image when creating Pods & run it as container

# Distributed Data Parallel (DDP)

**Distributed Data Parallel (DDP)** is a technique used in distributed machine learning training, especially in frameworks like PyTorch.

- It splits a model and its training data across multiple GPUs or machines.
- Each process trains a copy of the model on the chunk of the data.
- Gradients are synchronized across the worker nodes during each training step.
- It enables faster training for large datasets or complex models.

*Kubernetes is not a DL-framework - but it orchestrates resources which makes DDP possible at scale.*

# Agenda

**Objective**: Data-Parallel Training of CIFAR-10 across two physical nodes using KUBERNETES.

# PLAN

1. **Step 1**: Set up Kubernetes cluster
   - Use `"kubeadm"`, `"k3s"` or `"minikube"` (multi-node) depending on preference.
   - One laptop/computer will be the master node, the other will be the worker node.
2. **Step 2**: Data Parallelism
   - Use PyTorch DataDistributedParallel (DDP) or Tensorflow's MultiWorkerMirroredStrategy.
   - Split the CIFAR-10 dataset across two nodes.

# PLAN

1. **Step 3**: Containerize the Training Code
   - Write DockerFile that installs dependencies, mounts data & runs training.
   - Push to a container registry (local/private) or DockerHub.
2. **Step 4**: Deploy Kubernetes
   - Define a StatefulSet or Deployment for each training process.
   - Use headless services + shared storage (NFS or object-store) if needed.
   - Set host networking or Service DNS for communication.

Each pod gets one replica of the training script and participates in DDP.

# Observations

## Advantages

1. Automated scaling.
2. Self-healing.
3. Rollouts and rollbacks
4. Declarative configuration (YAML/JSON-based)

## Disadvantages

1. Steep learning curve
2. Complex setup and management
3. Resource overhead
4. Debugging can be tough

# Replication

It addresses high-availability, scalability & fault tolerance.

- It happens through a ReplicaSet. Kubernetes mainly creates replicas of Pods.

## Pod Replication

- Where it happens? ReplicaSet
- Purpose: App availability

## Data Replication

- Where it happens? In storage layer
- Purpose: Data durability

# Fault Tolerance

**Fault Tolerance** refers to a system's ability to continue working even when parts of it fail - like a crashd node, lost network connection, or a container dying.

- Automatic Pod Restart
- Self Healing
- Replica Set

→ If you are doing data science or ML with tabular data and want a fast setup, choosing H2O is fine; but fault tolerance is very limited due to its tight cluster coupling.
→ If you are deploying production grade ML pipelines, especially DL, batch-jobs, or services, **Kubernetes offers superior fault tolerance**.

# End

THANK YOU!

Any questions are welcome.