Kubernetes Workflow Explained: The ”Restaurant” Analogy A breakdown of Docker, Minikube, and Kubectl

Project Notes for Amer Syed

November 7, 2025

1 The Big Picture: The Restaurant Analogy

These are not separate tools; they are **layers of a single system**. Here's how they work together:

- **Docker (The "Product"):** This is your application, perfectly packed in a "kitchen-in-a-box." The `Dockerfile` is the packing list, the **Image** is the flat-packed box, and the **Container** is the assembled, running kitchen.
- **Kubernetes (K8s) (The "Manager"):** This is the **Operating System** for your restaurant chain. It's the "manager" that knows how to order new kitchens, set them up, handle 10,000 customers, and automatically replace a kitchen if it catches fire.
- **Minikube (The "Sandbox"):** This is a **toy model of the restaurant** you build on your desk. It's a tiny, all-in-one K8s cluster (manager + kitchen) that runs on your Mac so you can practice being the manager.
- **kubectl (The "Phone"):** This is the **phone you use to call the manager.** It's the command-line tool you use to give orders (`apply`, `get`, `delete`) to *any* K8s cluster, whether it's your `minikube` toy model or a giant production cluster on AWS.

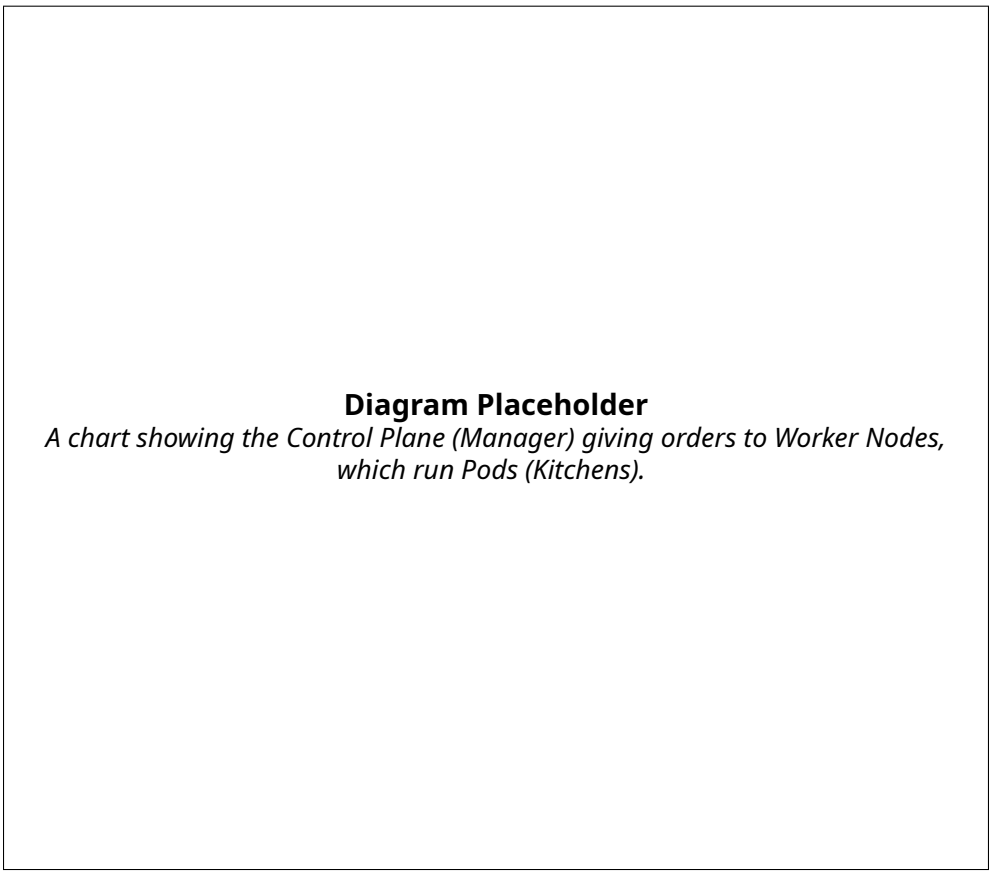

Figure 1: High-Level Kubernetes Architecture

2 The Commands: An Explanatory Flow

Here is the "story" of your command list, broken down by phase.

2.1 Phase 1: Building the "Practice Restaurant" (Minikube)

These commands are for `minikube` itself. You are setting up your local "sandbox."

`minikube start`
- **Theory:** This is the "on" switch. It boots up the all-in-one K8s cluster (Control Plane + Worker Node) inside a single container on your Mac.
- **Use:** This is the *environment* where you will *use* `kubectl`.

`minikube status`
- **Theory:** A simple "health check."
- **Use:** "Is my minikube cluster running?"

`e start --nodes=2`
- **Theory:** An advanced `start`. This builds a cluster with one "Manager" (Control Plane) and *two* "Worker Nodes."
- **Use:** This lets you test real distributed computing, like watching K8s place one app on Node 1 and another on Node 2.

2.2 Phase 2: Stocking the Kitchen (Docker & Minikube)

This is the tricky part. Your `minikube` cluster is a "box" inside your Mac. Your Docker images are "boxes" on your Mac. You have to move your app image into the `minikube` box.

`docker images`
- **Theory:** Lists all images on your **Mac's** "loading dock" (your local Docker Desktop).

`inikube image list`
- **Theory:** Lists all images *inside* the **Minikube** cluster's "storage room."

`e load ...:latest`
- **Theory:** This is the **"local dev shortcut."**
- **Use:** "Take the `k8s-test-app` image from my Mac's loading dock and copy it directly into Minikube's internal storage room."
- **Relation:** This *replaces* the need for a cloud registry (like Docker Hub or your AWS ECR) for simple local testing.

2.3 Phase 3: Giving Orders to the Manager (kubectl)

You are now speaking `kubectl`, the universal language of Kubernetes.

`deployment.yaml`
- **Theory:** The **master command**. You're handing the "Manager" (`kubectl`) your instruction manual (`deployment.yaml`) and saying, "Make the restaurant look exactly like this."
- **Relation:** K8s reads this, sees you need `replicas: 2` of the `k8s-test-app` image. It goes to its *internal* storage (where you just loaded the image), and builds two **Pods** (the running containers) to match.

`-A | get pods -A`
- **Theory:** "Status update" commands.
- **Use:** `get pods -A` means "Show me every app (`Pod`) running in every department (`-A` for All Namespaces)."

`e pod <pod-name>`
- **Theory:** The **Fault-Tolerance Demo**.

- **Use:** "Fire that *one specific chef*."
- **Relation:** The `ReplicaSet` (created by your `Deployment`) immediately sees this. Its rule is "I *must* have 2 pods." It sees there is only 1, so it instantly spins up a *new* pod to replace the one you killed. This is how K8s "self-heals."

2.4 Phase 4: Visiting the Restaurant (Service)

`kube service ...`

- **Theory:** This is a *Minikube-only* shortcut.
- **Use:** "I'm a customer. I want to visit the restaurant."
- **Relation:** It automatically finds the `Service` you defined in your YAML, finds its IP and port, and opens your Mac's browser to that address.

`ectl get service`

- **Theory:** Shows you the "public counter" K8s created (the `ClusterIP` or `LoadBalancer`).

2.5 Phase 5: The "Stress Test" (Load Balancing)

`logs -f <pod-id>`

- **Theory:** "Follow" (`-f`) the real-time logs of a *specific* pod.
- **Use:** You run this in two separate terminals, one for `pod-1` and one for `pod-2`. You are "listening" to both chefs.

10 Users → 1 min)

- **Theory:** You are sending many customers per second to the single `Service` (the "public counter").
- **The "Aha!" Moment:** By watching your two log terminals, you will see the requests (`GET / ...`) being split between them.
- **This *proves* load balancing.** K8s is automatically spreading the traffic across all your running `replicas` to ensure no single one gets overloaded.