hands on
docker & kubernetes





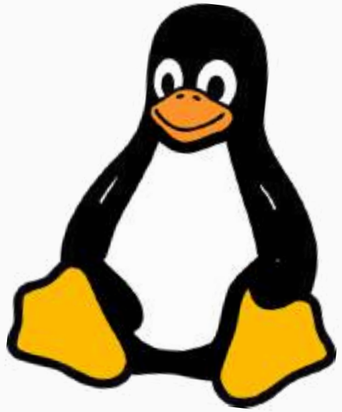


prerequisites

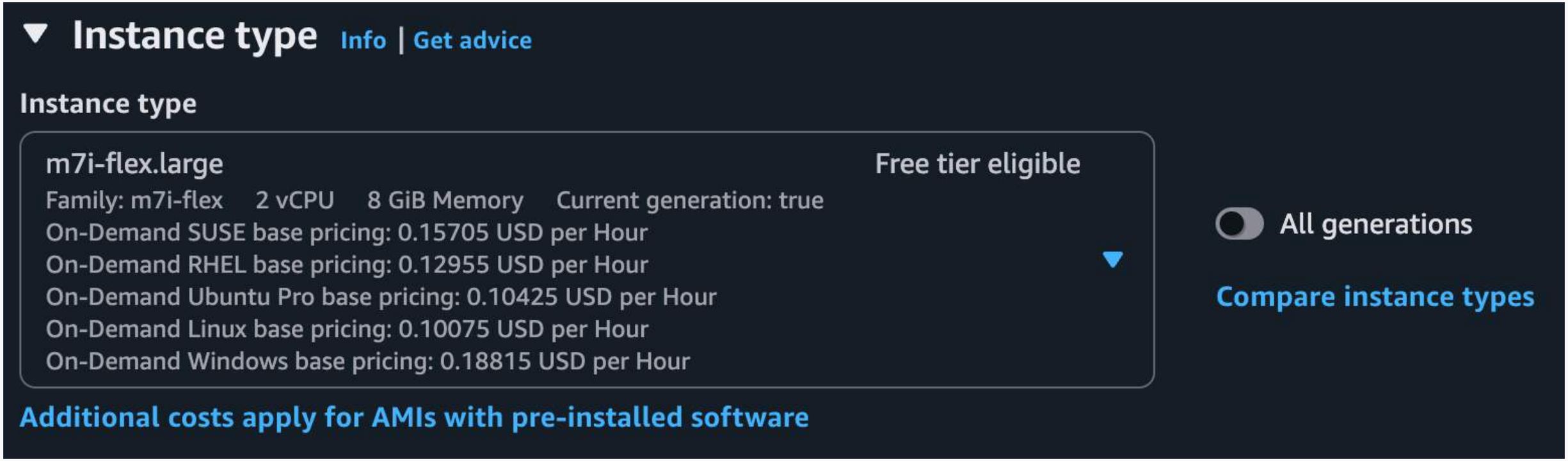
- be able to spin-up EC2
- navigate AWS console
- basic linux commands







prerequisites



Network settings Info

Edit

Network | Info

vpc-03024859285a52f8b

Subnet | Info

No preference (Default subnet in any availability zone)

Auto-assign public IP | Info

Enable

Firewall (security groups) | Info

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

- (•) Create security group
- ( ) Select existing security group

We'll create a new security group called '**launch-wizard-3**' with the following rules:

- [x] Allow SSH traffic from
  Helps you connect to your instance
  Anywhere
  0.0.0.0/0
- [x] Allow HTTPS traffic from the internet
  To set up an endpoint, for example when creating a web server
- [x] Allow HTTP traffic from the internet
  To set up an endpoint, for example when creating a web server

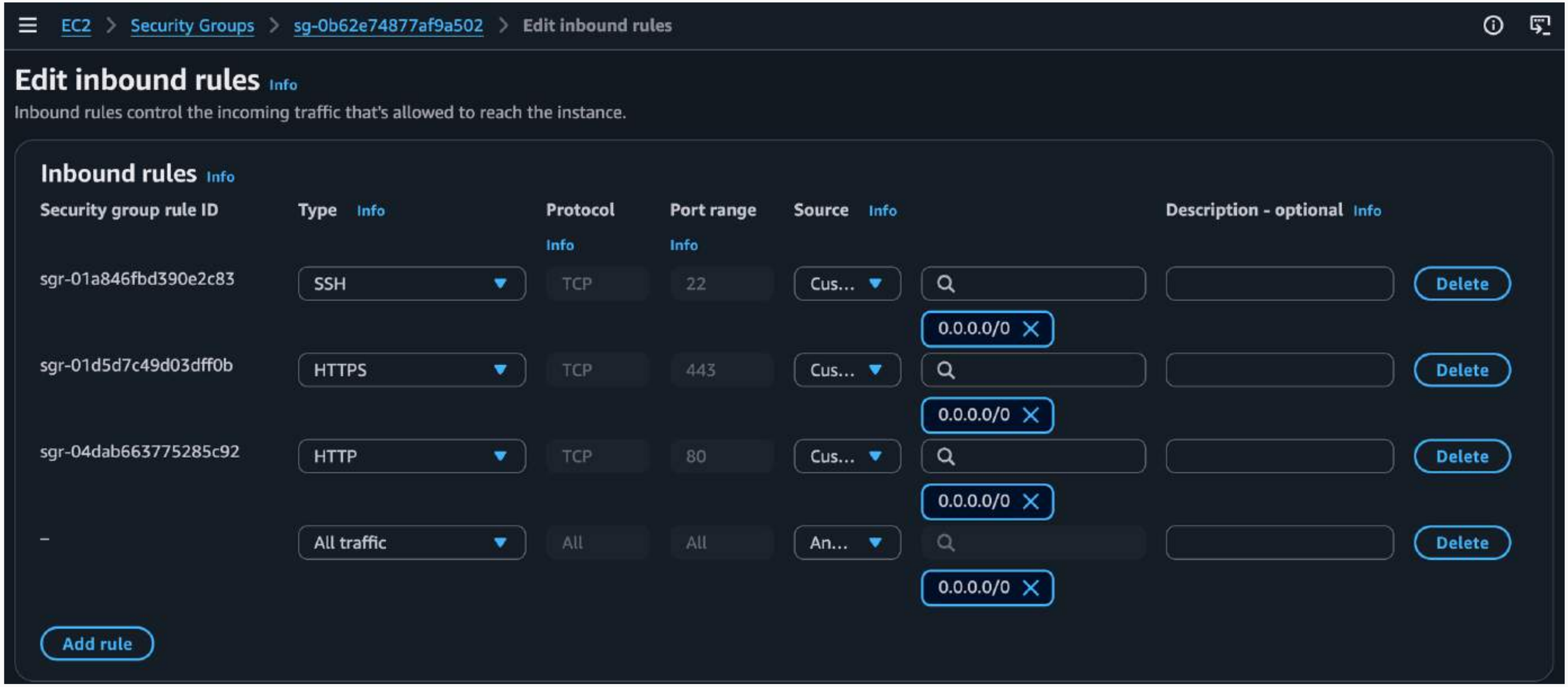
Configure storage Info

Advanced

1x 50 GiB gp3 Root volume, 3000 IOPS, Not encrypted

Add new volume

- just for the demo, so we spend time o
  actual content and not networking.



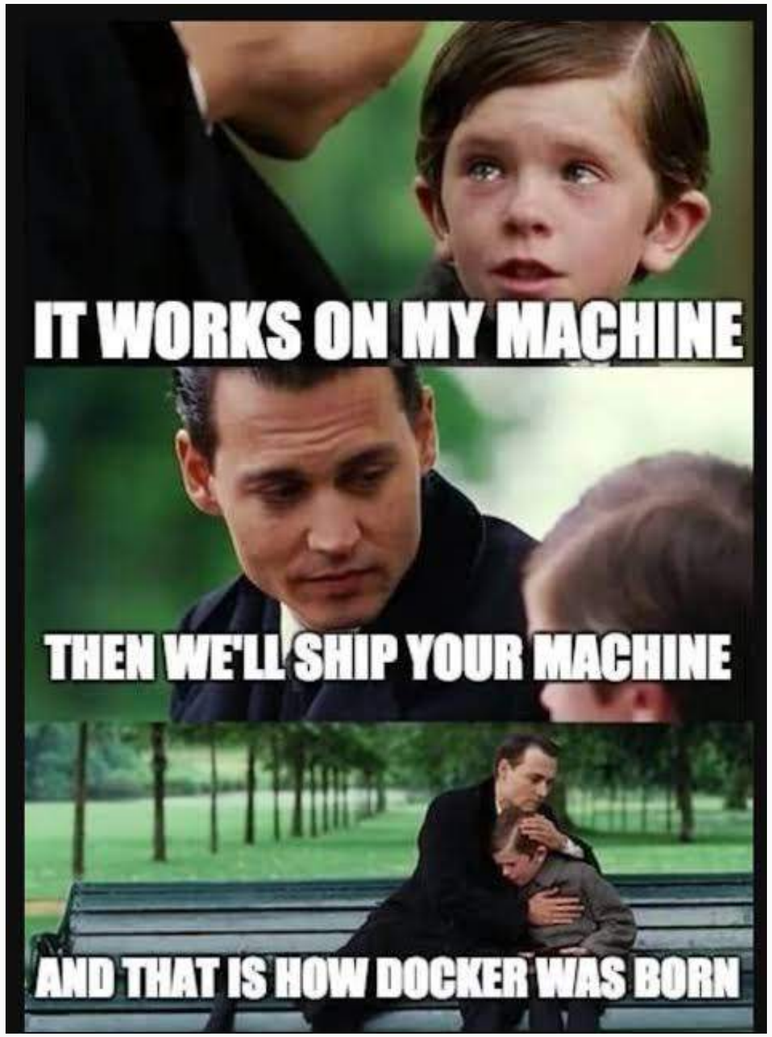
what is docker ?



- Docker is an open-source platform that packages applications and all of their dependencies into standardized, lightweight units called containers.
- bundles code, runtime, system libraries, and settings into a single package called image.
- before docker, software that ran perfectly on a programmer's laptop would fail in production due to different software versions, missing configuration files, or operating system discrepancies.

why use docker ?

- **Consistency Across Environments:** operates the exact same way during development, testing, and production.
- **Isolation and Security:** Each container runs in its own isolated environment, multiple containers run on the same server without conflicting.
- **Resource Efficiency**: Unlike Virtual Machines which require an entire guest operating system, Docker containers share the host machine’s operating system kernel.
- **Scalability and Microservices**: Docker is ideal for microservices, where a large application is split into small, independent parts. Each service can run in its own container, making it easy to scale up specific parts of an app when traffic increases.
- **Simplified CI/CD and DevOps**: It streamlines the deployment pipeline. Teams can "build once, run anywhere," accelerating software shipping speeds and minimizing infrastructure costs.



docker core concepts

- **Dockerfile:** A simple text file containing sequential instructions on how to build a Docker image.
- **Docker Image:** A read-only blueprint or template used to create containers.
- **Docker Container:** A live, running instance of an image.
- **Docker Hub:** A massive public registry where developers share and download pre-built images for databases, web servers, and operating systems.

installing docker on EC2



- we are using Linux because that’s what is used world-wide in production.

- **update system** : sudo yum update -y
- **installing docker** : sudo yum install -y docker
- **start docker process** : sudo systemctl start docker
- **make docker bootup service** : sudo systemctl enable docker
- **use docker w/o sudo** : sudo usermod -aG docker ec2-user (reconnect)
- **verify docker installation** : docker run hello-world

running container using docker



- let’s run a simple container called nginx.
- it’s a popular open-source web server, reverse proxy, and load balancer designed for high performance and low memory use

- **run nginx container** : docker run -d --name nginx -p 8080:80 nginx
- **see all containers** : docker ps / docker ps -a
- **test on EC2** : curl http://localhost:8080
- **test from internet** : http://<EC2_PUBLIC_IP>:8080
- docker stop nginx
- docker rm nginx



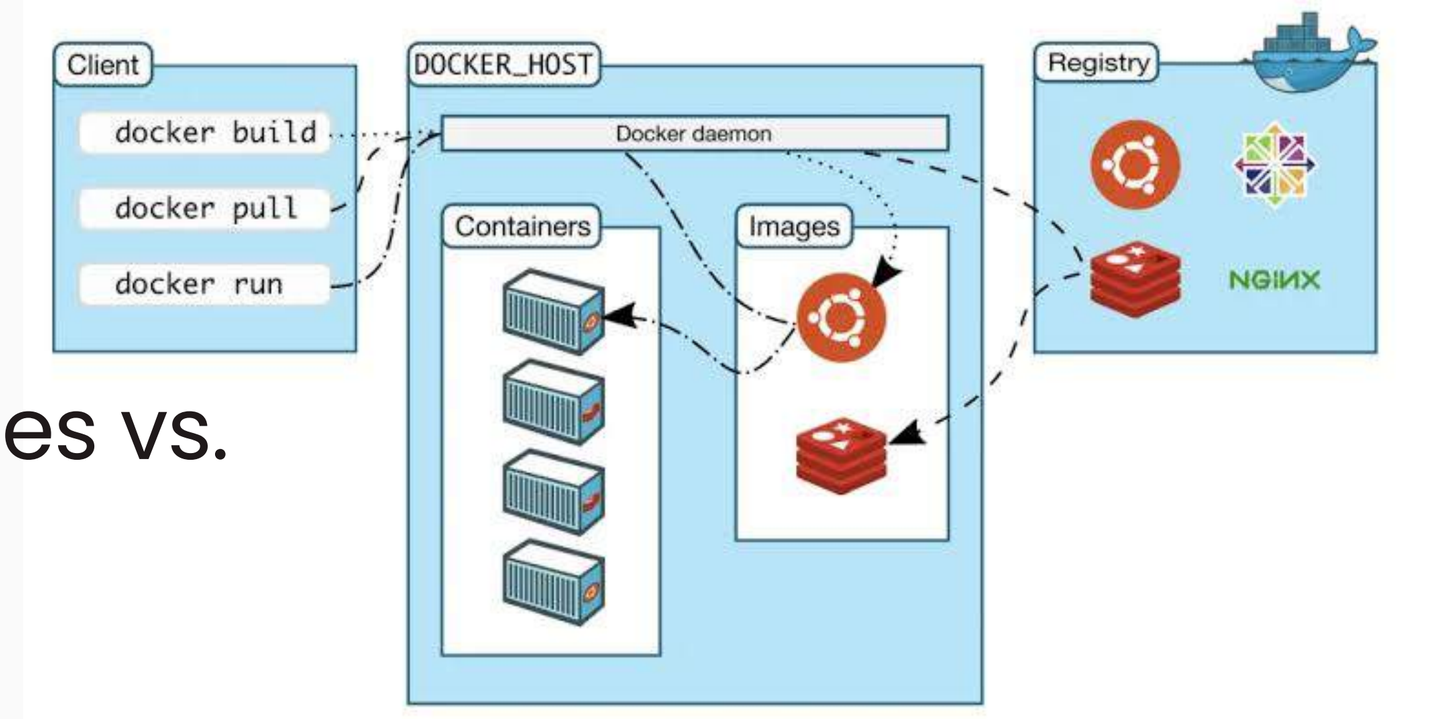
other useful docker commands



- **starting a container :** docker run -d --name **nginx** -p 8080:80 **nginx**
- **see running containers** : docker ps
- **see all containers** : docker ps -a
- **see downloaded images** : docker images
- **stop (pause) a container** : docker stop <container-name>
- **starting existing paused container** : docker start <container-name>
- **remove a container** : docker rm <container-name>
- remove an image : docker rmi <image-name>
- **download an image :** docker pull nginx

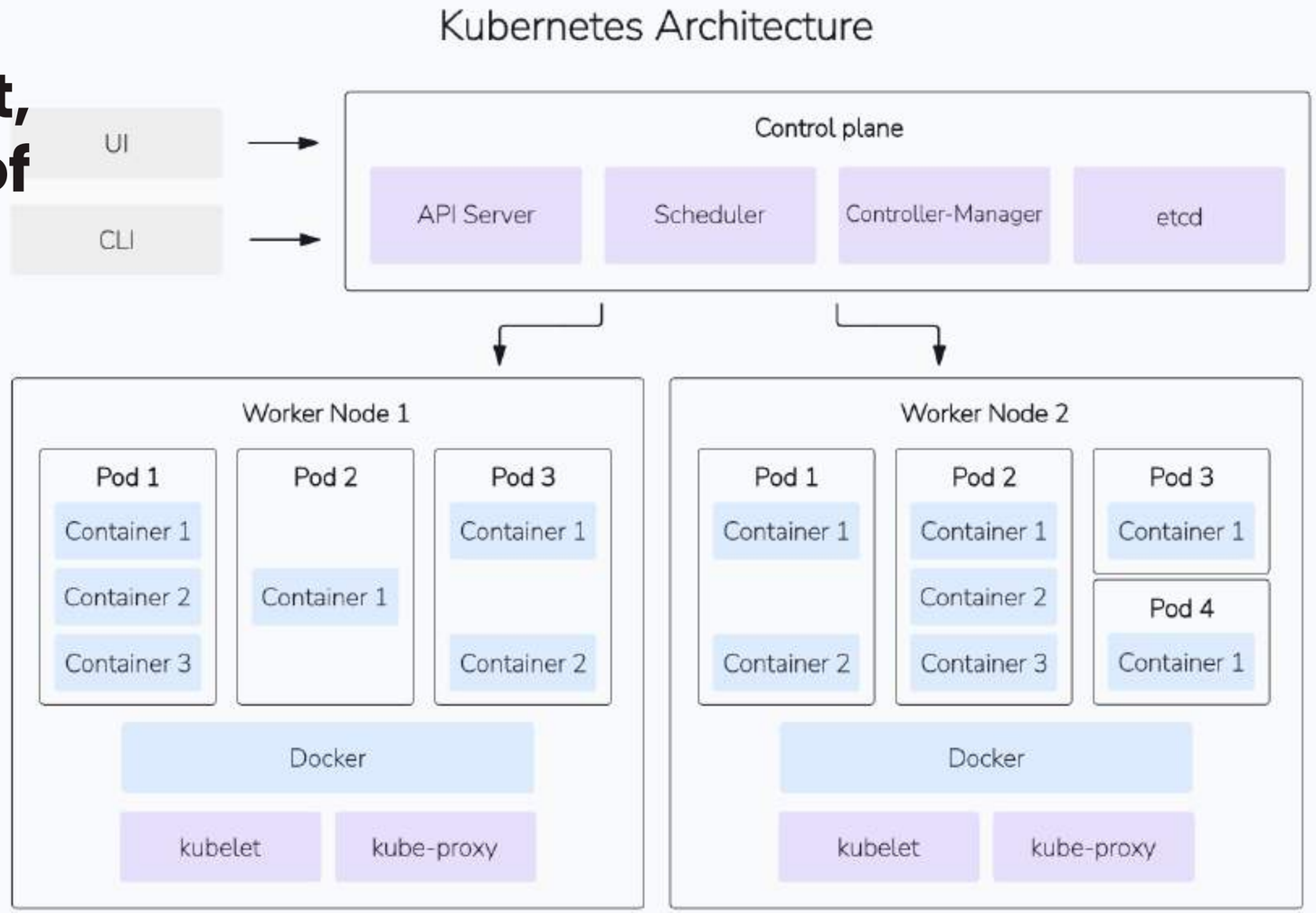
explore beyond this...

- docker networking
- docker compose
- docker volumes
- writing dockerfiles
- understanding images vs. containers
- image optimization
- docker swarms



what is kubernetes ?

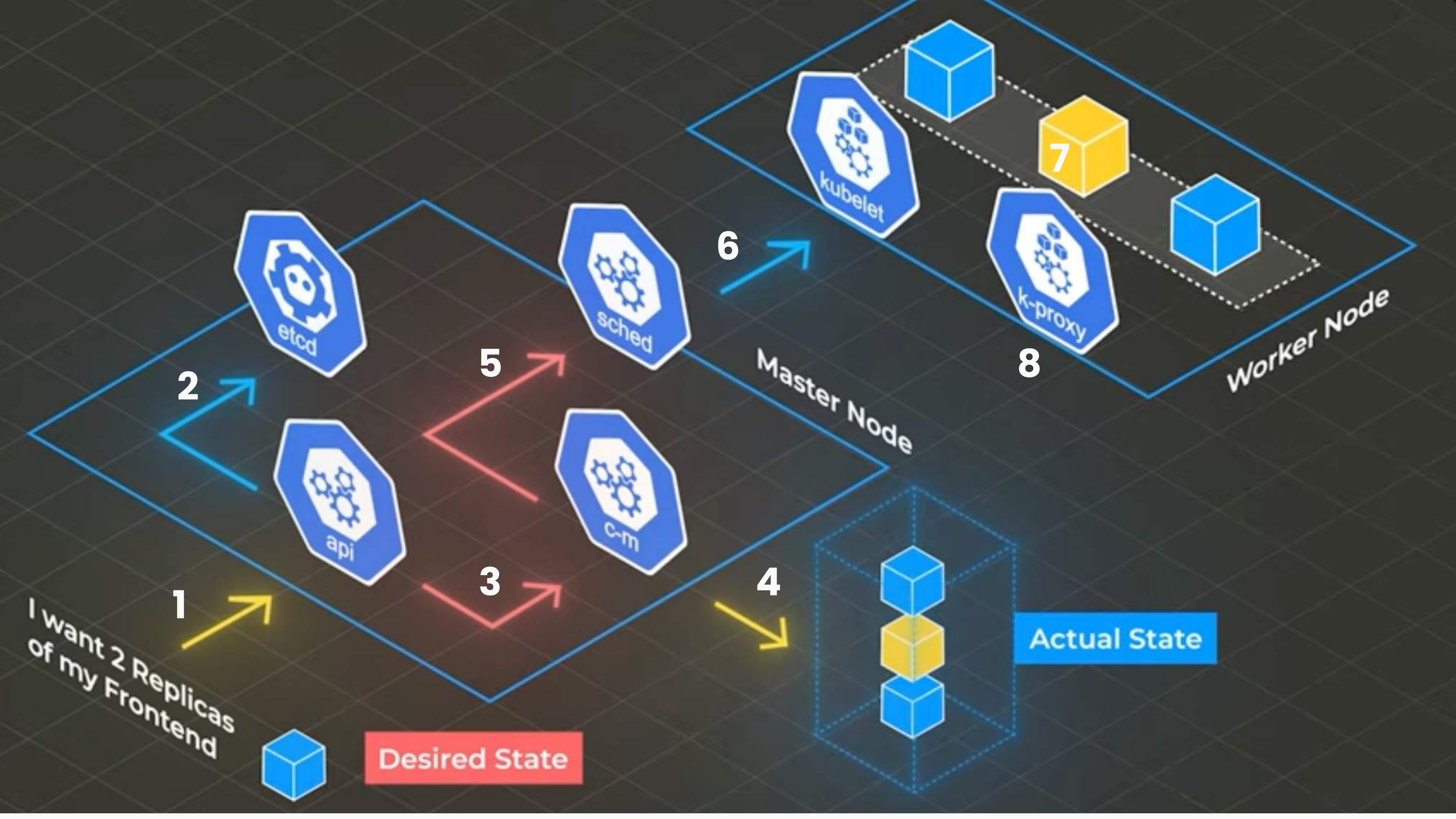
an open-source system for automating the deployment, scaling, and management of containerized applications



- *initially started as Google's internal tool, later open-sourced.*
- *highly scalable.*
- *written in Golang.*



You tell Kubernetes how your application should look (e.g. "run three copies of this web server") & kubernetes continuously matches current state to that goal state / desired state.


etcd
api
sched
c-m
kubelet
k-proxy
7
Master Node
Worker Node
1
I want 2 Replicas
of my Frontend
2
3
4
5
6
8
Actual State
Desired State

features of kubernetes



- **Self-Healing**: Automatically restarts failed containers, replaces unhealthy pods, and reschedules work when nodes die.
- **Auto-Scaling**: Horizontally scales applications up or down based on resource usage or custom metrics.
- **Load Balancing**: Distributes network traffic across multiple application instances to ensure high availability.
- **Portability**: Runs consistently across on-premises data centers, hybrid setups, and public clouds

without kubernetes



1. Create containers manually
2. Monitor containers
3. Detect failures
4. Restart unhealthy containers
5. Scale up/down manually (increasing no. of instances)
6. Manage networking & deployments (connecting different containers to work together as a coherent system)

why use kubernetes when we have docker ?



- A few containers are easy to manage manually.
- At large scale, managing thousands of containers manually becomes complex and unreliable.
- Kubernetes automates container deployment, scaling, networking, and self-healing.
- Kubernetes automates the management of containers at scale.

real world adoption of kubernetes.



- Organizations are adopting Kubernetes for flexibility, automation, and infrastructure abstraction.
- **Tech giants runs kubernetes with thousands of nodes, OpenAI has publicly documented scaling a single Kubernetes cluster to 7,500 nodes.**

kubernetes terminology



- cluster
- nodes
  - master / control plane
  - worker node
- pods
- service
- deployments
- HPA : horizontal pod autoscaler

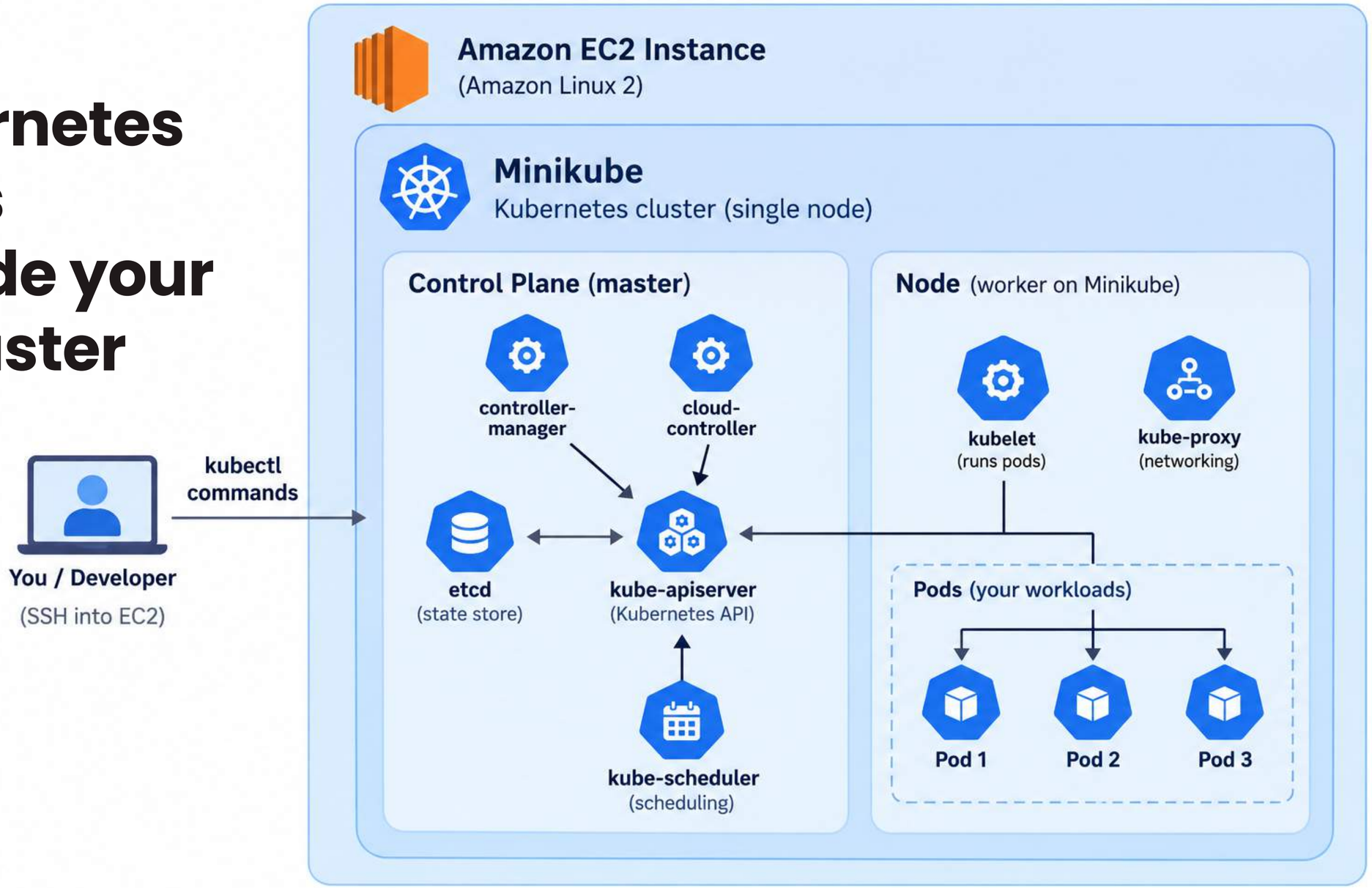
a kubernetes is simply a group of many nodes.



Nodes

Physical Servers
Virtual Machines
Cloud Instances

actual Kubernetes components running inside your Minikube cluster



Cluster

- A group of machines (nodes) that run Kubernetes.
- Managed as a single unit.
- Nodes can be VMs or physical servers.

Node

- A machine (VM or physical server) in the cluster.
- Two types:
  - **Control Plane (Master) Node →** makes decisions, manages the cluster.
  - **Worker Node →** runs your containers (Pods).

kubectl

- Command-line tool to interact with Kubernetes.
- Sends your commands to the kube-apiserver.
- Used to create, view and manage Kubernetes objects.

```
kubectl get pods
```

Kubernetes Objects

- Resources you create and manage in Kubernetes.
- Describe the desired state of your application.
- Examples: Pod, Deployment, ReplicaSet, Service, HPA, etc.

Control Plane Components (Master Node)

kube-apiserver

Entry point

- Unified API gateway for the cluster.
- Handles all requests (from kubectl and other components).
- Reads/writes cluster state in etcd.

etcd

Database

- Distributed key-value database.
- Stores the entire cluster state (desired and actual state).
- Used for backup and disaster recovery.

kube-controller-manager

Reconciles state

- Runs controllers (Deployment, ReplicaSet, HPA, etc.).
- Continuously compares desired vs actual state and takes action to fix any differences (e.g., create missing Pods).

kube-scheduler

Assigns Pods to Nodes

- Decides which worker node should run a new Pod.
- Considers available resources (CPU, memory, storage), constraints, etc.
- Watches for unscheduled Pods and assigns them to the best node.

Worker Node Components

kubelet

Node agent

- Runs on every worker node.
- Communicates with the kube-apiserver.
- Makes sure the Pods assigned to the node are running as specified.

Container Runtime

Runs containers

- Actually creates and runs containers (e.g., containerd).
- Used by kubelet to run the containers defined in Pods.

kube-proxy

Service networking

- Implements Service networking on the node.
- Routes traffic to the correct Pods (using iptables/IPVS).
- Helps expose and load balance traffic to Pods.

Workload & Networking Objects

Pod

Smallest unit

- Smallest deployable unit.
- Can contain one or more containers.
- Gets its own IP.
- Ephemeral (can be deleted and recreated).

ReplicaSet

Maintains replicas

- Ensures a specified number of matching Pods are always running.
- Creates new Pods if some die.

Deployment

Manages ReplicaSets

- Higher-level object to manage ReplicaSets.
- Provides features like rollouts and updates.
- You usually create a Deployment instead of a Pod directly.

Service

Stable endpoint

- Provides a stable IP and DNS name to access a set of Pods.
- Load balances traffic to healthy Pods.
- Types: ClusterIP, NodePort, LoadBalancer.

minikube

what is minikube then?

- **Minikube is a lightweight Kubernetes distribution that runs a Kubernetes cluster on a single machine.**
- fewer requirements : 2 CPUs, 2GB RAM, 20GB free disk space



K3S

K3D

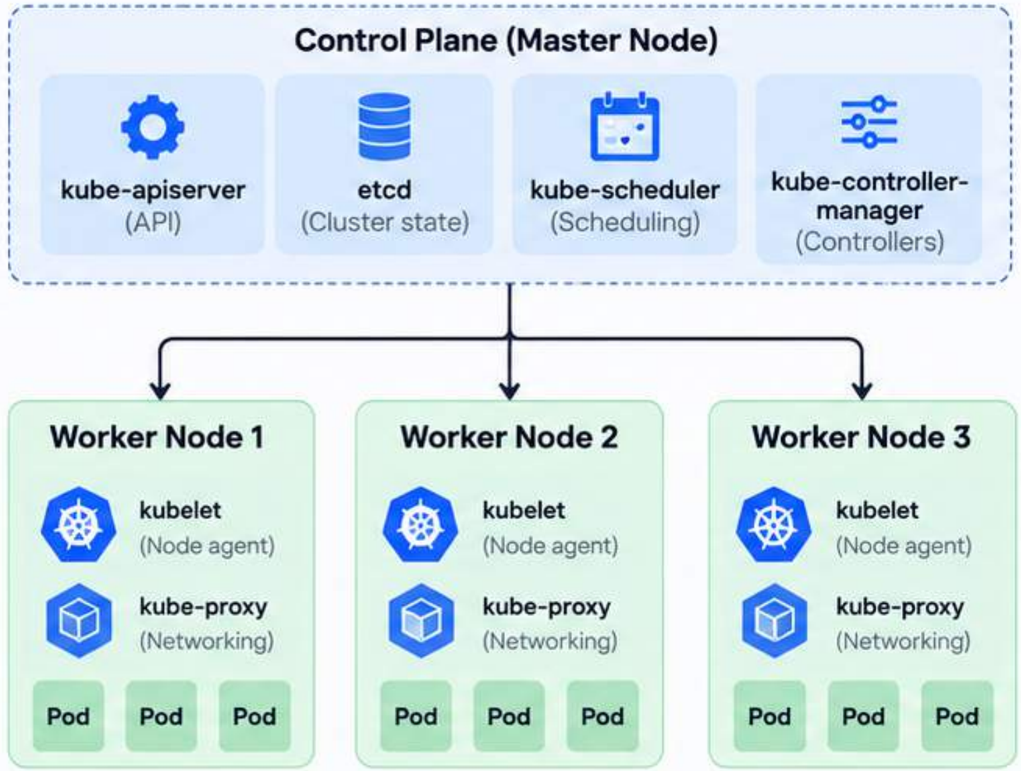
Other lightweight Kubernetes options include Kind, K3s, and K3d.

Kubernetes vs Minikube

Same technology. Different scale.

Kubernetes (Production)

A distributed system running across multiple machines.



Runs on a dedicated machine (or high-availability setup)

Multiple worker nodes across different machines (physical or cloud VMs)

Kubernetes orchestrates containers across many machines,
providing high availibility, scalability, and fault tolerance.

Minikube (Development/Learning)

A lightweight Kubernetes cluster running on a single machine.



Control plane and worker node(s) run on your local machine as a single node cluster.

Minikube makes it easy to run Kubernetes locally
for learning, development, and testing.

lets install a custom python server using docker as container runtime & kubernetes as container orchestrator

install Docker on your EC2 instance

- `sudo yum update -y`
- `sudo yum install -y docker`
- `sudo systemctl enable --now docker`
- `sudo usermod -aG docker ec2-user`

Reconnect to SSH after adding ec2-user to the docker group.

verify docker installation

- `docker --version`
- `docker ps`
- `docker run --rm hello-world`

installing minikube

- `curl -LO https://github.com/kubernetes/minikube/releases/latest/download/minikube-linux-amd64`
- `sudo install minikube-linux-amd64 /usr/local/bin/minikube`
- `rm minikube-linux-amd64`

starting minikube

Run a single-node Kubernetes cluster with the Docker driver.

- `minikube start --driver=docker`
- `minikube status`
- `minikube kubectl -- get nodes`

Pull the server image that we want to deploy

- `docker pull hyperionz9/cpu-demo:1.0`

Create Deployment Configuration file.

```
cat > deployment.yaml <<'EOF'
apiVersion: apps/v1
kind: Deployment
metadata: { name: cpu-demo }
spec:
  replicas: 1
  selector: { matchLabels: { app: cpu-demo } }
  template:
    metadata: { labels: { app: cpu-demo } }
    spec:
      containers:
      - name: cpu-demo
        image: hyperionz9/cpu-demo:1.0
        ports: [{ containerPort: 8000 }]
        resources:
          requests: { cpu: 50m }
          limits: { cpu: 100m }
EOF

minikube kubectl -- apply -f deployment.yaml
minikube kubectl -- get deployment
minikube kubectl -- get pods
```

Create the Service

Expose the Deployment through a Kubernetes Service.

- `minikube kubectl -- expose deployment cpu-demo --type=NodePort --port=8000 --target-port=8000`

- `minikube kubectl -- get service cpu-demo`
- `minikube kubectl -- get endpoints cpu-demo`

installing Metrics Server plugin & starting the HPA

Metrics Server provides CPU data; HPA scales replicas.

```
minikube addons enable metrics-server

minikube kubectl -- rollout status deployment/metrics-server   -n kube-system --timeout=120s

minikube kubectl -- top pods
```

HPA:

- **minimum 1 Pod, maximum 5 Pods, target 50% CPU**

```
minikube kubectl -- autoscale deployment cpu-demo   --min=1 --max=5 --cpu=50%

minikube kubectl -- get hpa cpu-demo -w
```

Expose the Minikube Service to the Internet

Relay EC2:8080 to the Minikube NodePort with socat.

```
NODE_PORT=$(minikube kubectl -- get service cpu-demo   -o
jsonpath='{.spec.ports[0].nodePort}')
MINIKUBE_IP=$(minikube ip)

echo $NODE_PORT
echo $MINIKUBE_IP
```

socat = general-purpose network relay. It bridges EC2 :8080 → Minikube NodePort.

```
sudo yum install -y socat

sudo nohup socat TCP-LISTEN:8080,fork,reuseaddr
TCP:$MINIKUBE_IP:$NODE_PORT   >/tmp/cpu-demo-socat.log 2>&1 &
```

Test if the application is deployed & available at public IP of EC2.

```
curl http://EC2_PUBLIC_IP:8080/
curl "http://EC2_PUBLIC_IP:8080/cpu?duration=2.0"

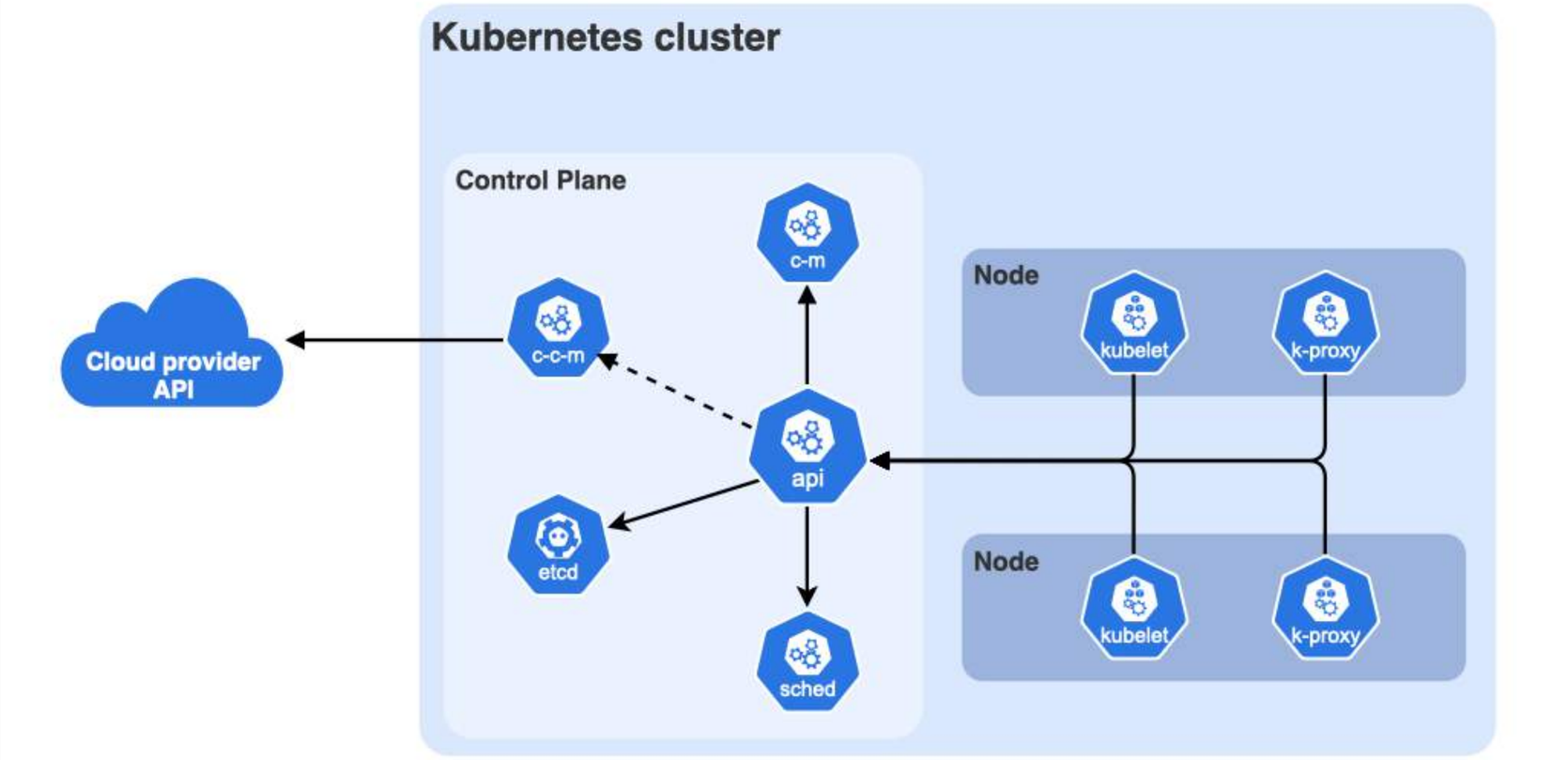

for i in {1..1000}; do
  curl -s "http://EC2_PUBLIC_IP:8080/cpu?duration=2.0" >/dev/null &
done
wait
```

Observe HPA & Pods

```
minikube kubectl -- get hpa cpu-demo -w
```

```
minikube kubectl -- top pods
```

explore beyond slides...



kubernetes is a vast distributed system itself that manages other distributed systems.


I NEED TO KNOW WHY MOVING OUR APP TO THE CLOUD DIDN'T AUTOMATICALLY SOLVE ALL OUR PROBLEMS.
Dilbert.com @ScottAdamsSays
YOU WOULDN'T LET ME RE-ARCHITECT THE APP TO BE CLOUD-NATIVE.
JUST PUT IT IN CONTAINERS.
11-06-17 © 2017 Scott Adams, Inc./Dist. by Andrews McMeel
YOU CAN'T SOLVE A PROBLEM JUST BY SAYING TECHY THINGS.
KUBERNETES.

Thank You