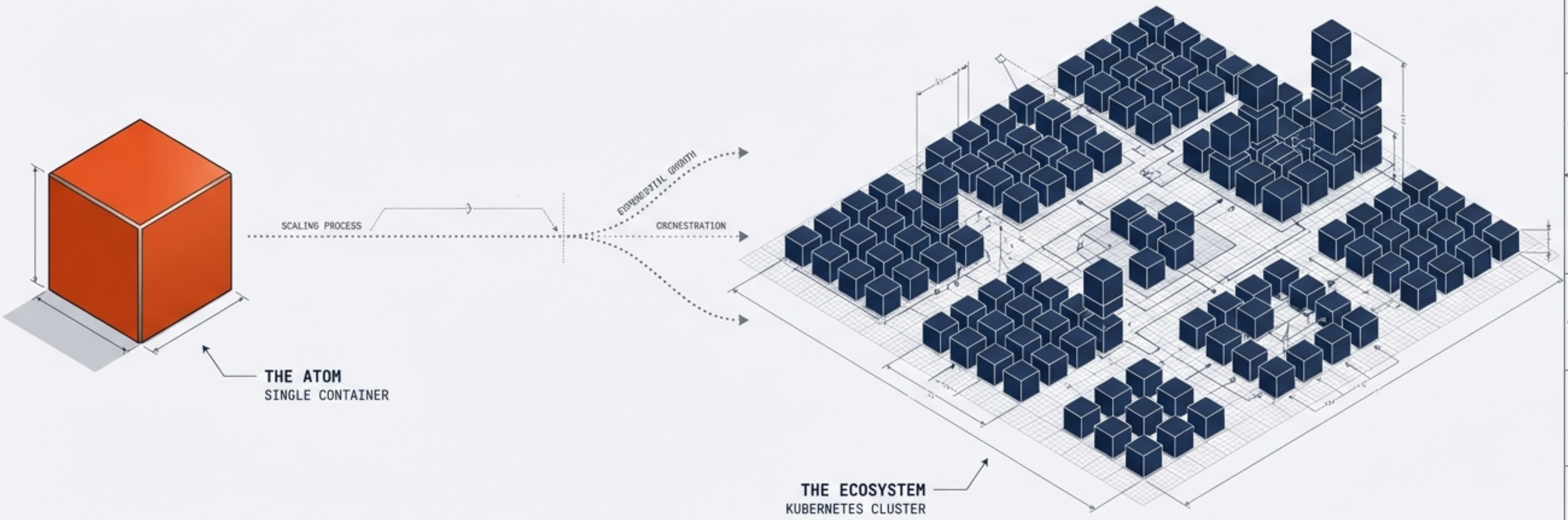
# From Atom to Ecosystem

## The Architecture of Scale: Docker Internals to Kubernetes Orchestration

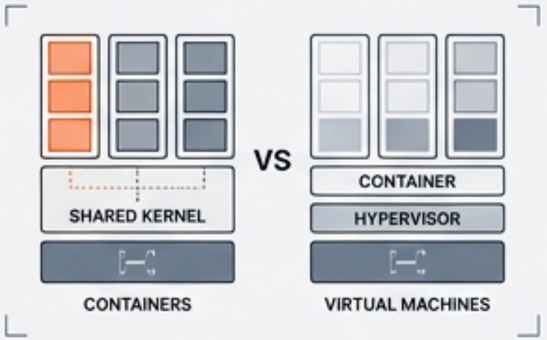


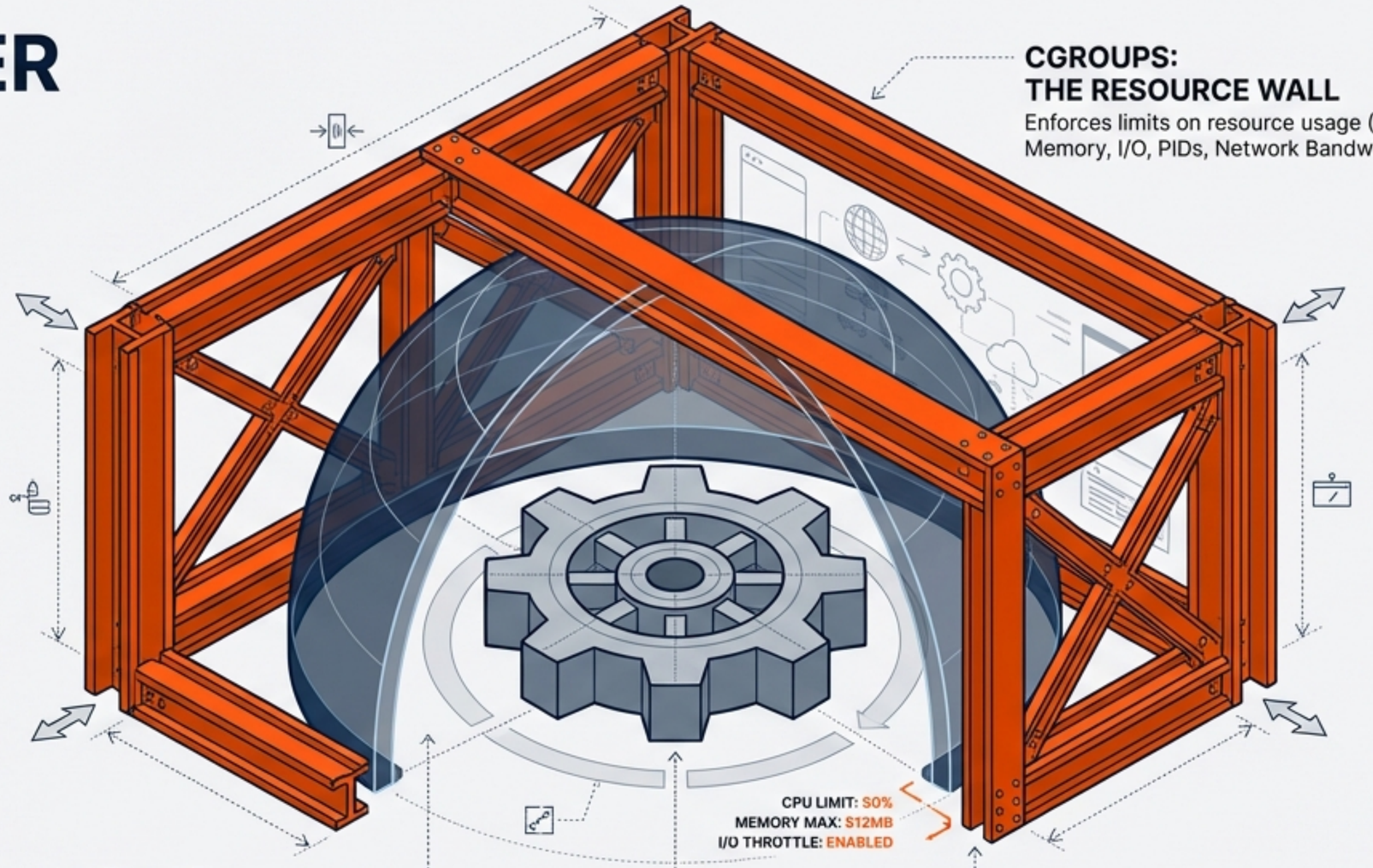

NotebookLM

# THE CONTAINER PRINCIPLE

Constraints (cgroups) and Views (Namespaces)

- NOT a virtual machine.
- Namespace = Isolation (PID, Network, Mount).
- Cgroups = Limits (CPU, Memory, I/O).




**CGROUPS: THE RESOURCE WALL**

Enforces limits on resource usage (CPU, Memory, I/O, PIDs, Network Bandwidth).

**NAMESPACE: THE VIEW**

Restricts visibility of system resources (PID, Network, Mount, IPC, UTS, User).

LINUX KERNEL PROCESS DIAGRAM

**CGROUPS: THE RESOURCE WALL**

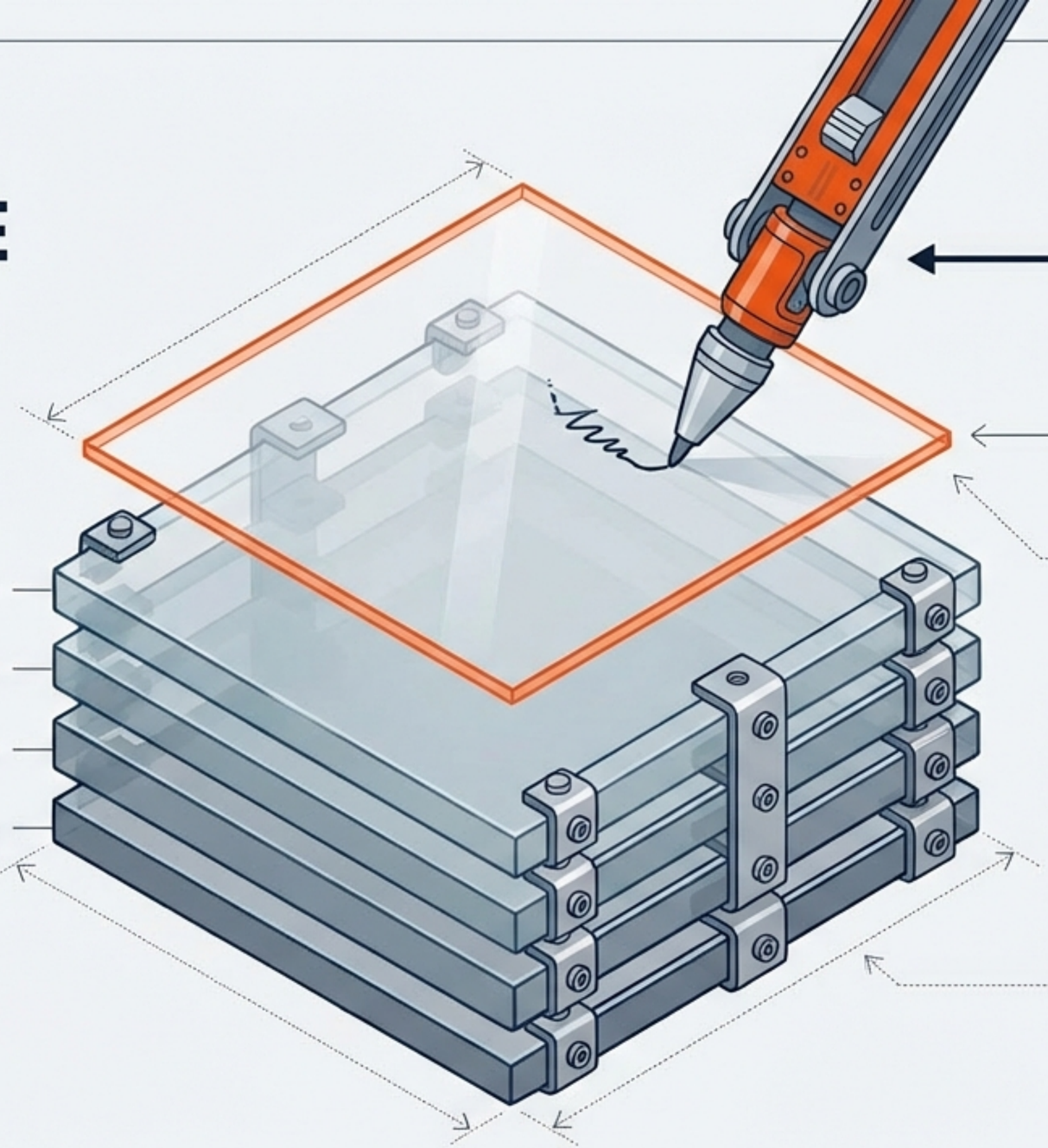
Enforces limits on resource usage (CPU, Memory, I/O, PIDs, Network Bandwidth).



NotebookLM

# ANATOMY OF AN IMAGE

Layers & Copy-on-Write



**Copy-on-Write:** Modifications happen here

Container Layer (Read/Write)

Temporary writable layer for this container instance. All file changes are recorded here.

Read-Only (Immutable)

- App Code
- Libraries
- OS Files
- Base Image

Immutable filesystem layers. Never changed after build.

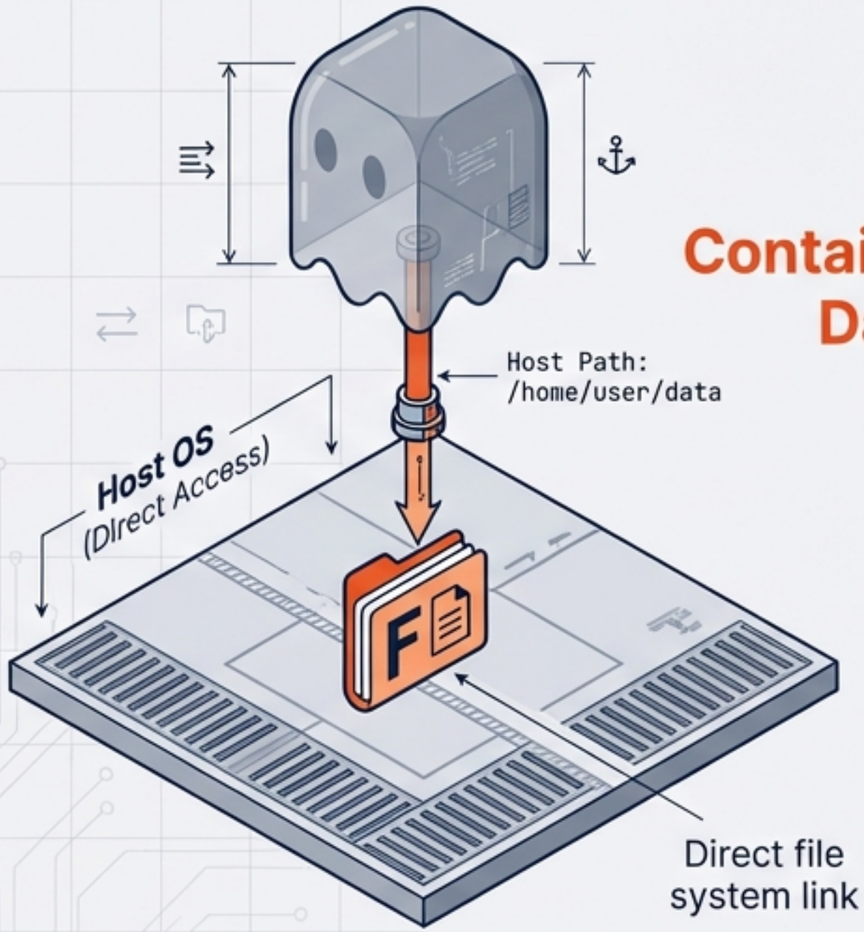


NotebookLM

# DATA PERSISTENCE

Beyond the Ephemeral

## BIND MOUNT




**Containers are disposable.**
**Data is anchored.**

Decouples data from container lifecycle.

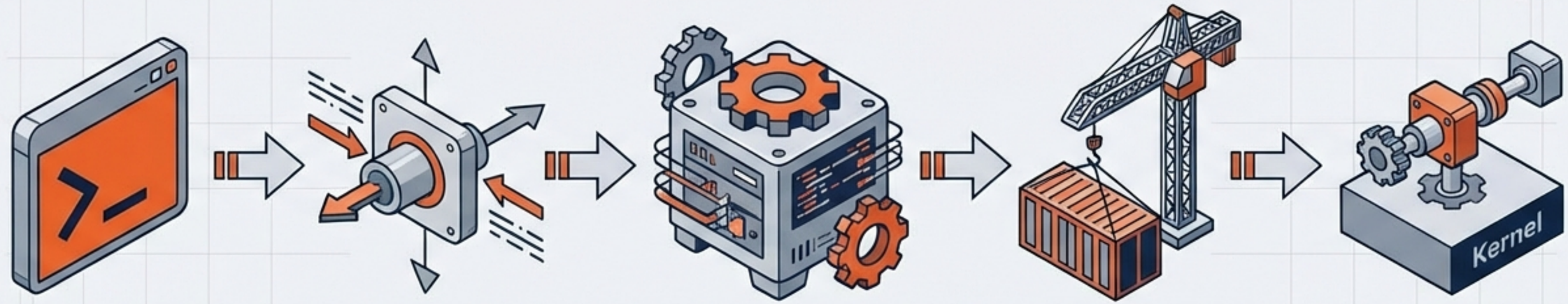
## DOCKER VOLUME




NotebookLM

# THE ENGINE ARCHITECTURE

## The Execution Flow




**Docker CLI** **REST API** **Docker Daemon** **containerd** **runc**

Docker is a collection of specialized tools.
"runc" creates the actual process.



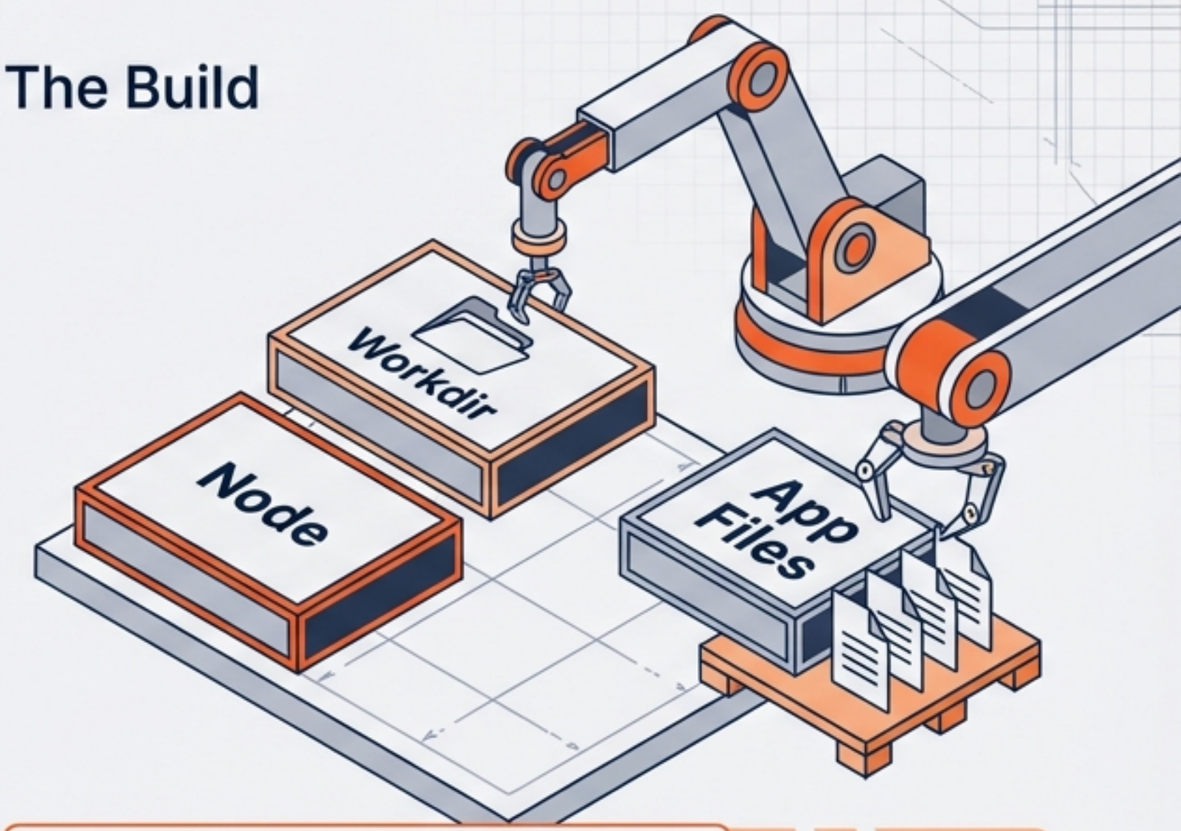
NotebookLM

# The Blueprint: Dockerfile

## The Code

```
1  FROM node:18-alpine
2
3  WORKDIR /app
4  COPY . .
5  RUN npm install
6
7  CMD ['node', 'server.js']
```

## The Build



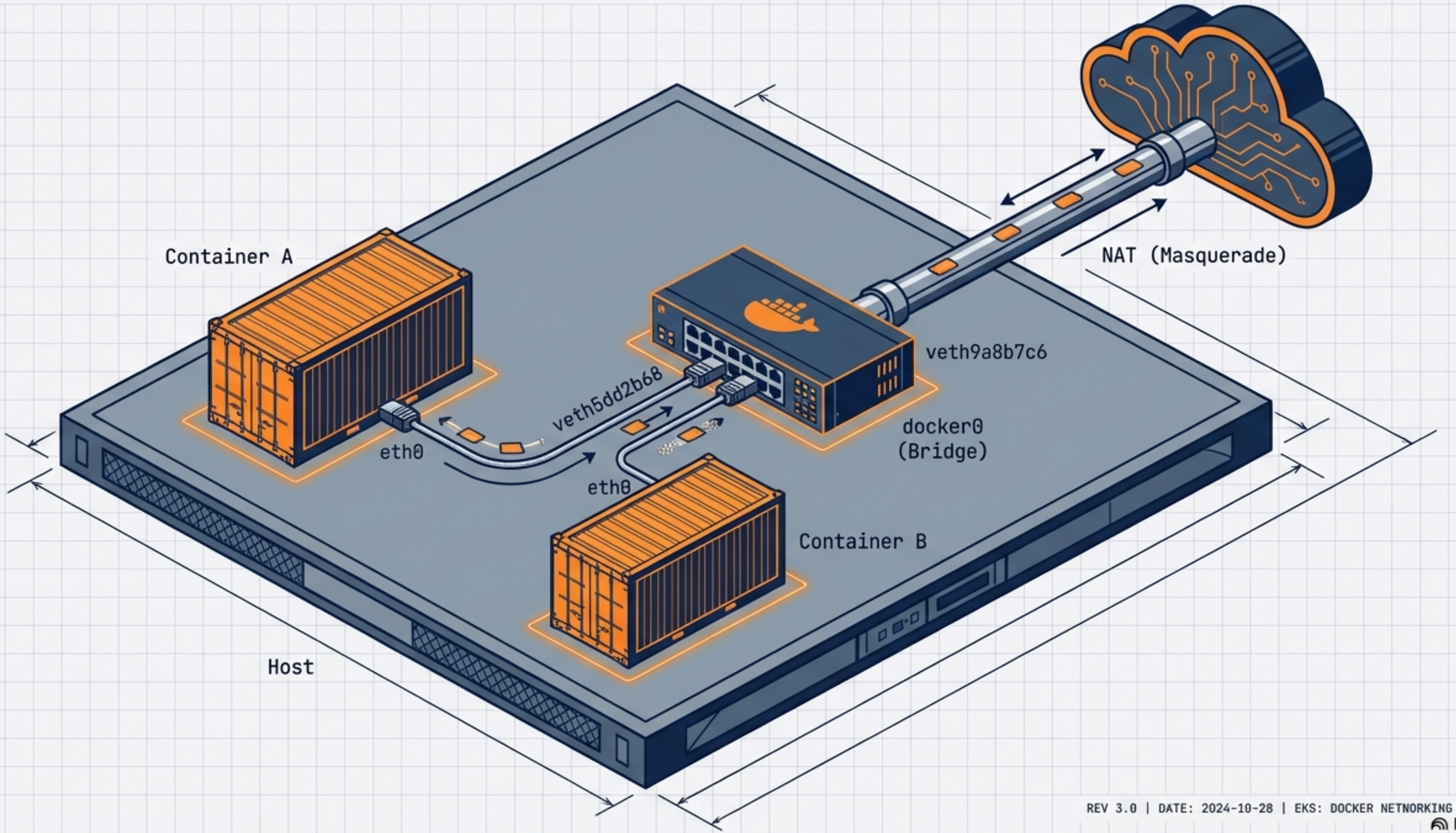

**Optimization Tip:**

Multi-stage builds reduce image size by discarding build-time dependencies.



NotebookLM

# Docker Networking: The Bridge






NotebookLM

# Imperative Toolset

Manual Controls

`docker build`

`docker run`

`docker push`

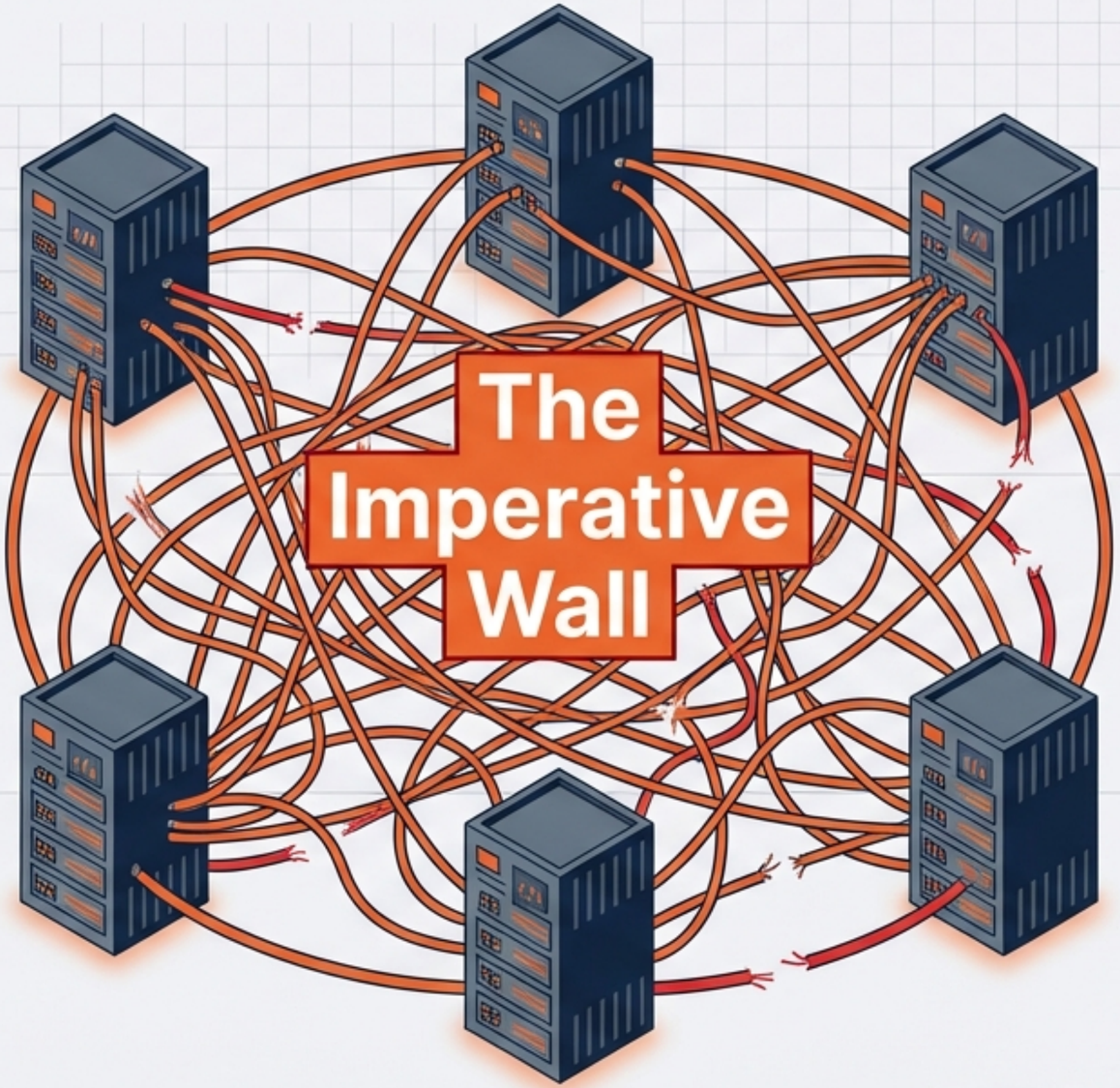
`docker inspect`




NotebookLM

# The Scaling Problem

Spaghetti Diagram




1. Fragile Connectivity

2. Manual Updates = Downtime

3. No Auto-Recovery



NotebookLM

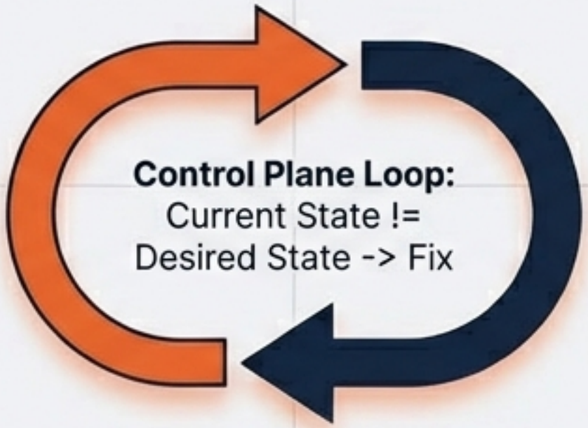
# The Paradigm Shift

Imperative vs. Declarative

## Imperative/Docker

**DO THIS. Then DO THAT.**

(Focus on Steps)




## Declarative/Kubernetes


**MAKE IT LOOK LIKE THIS.**

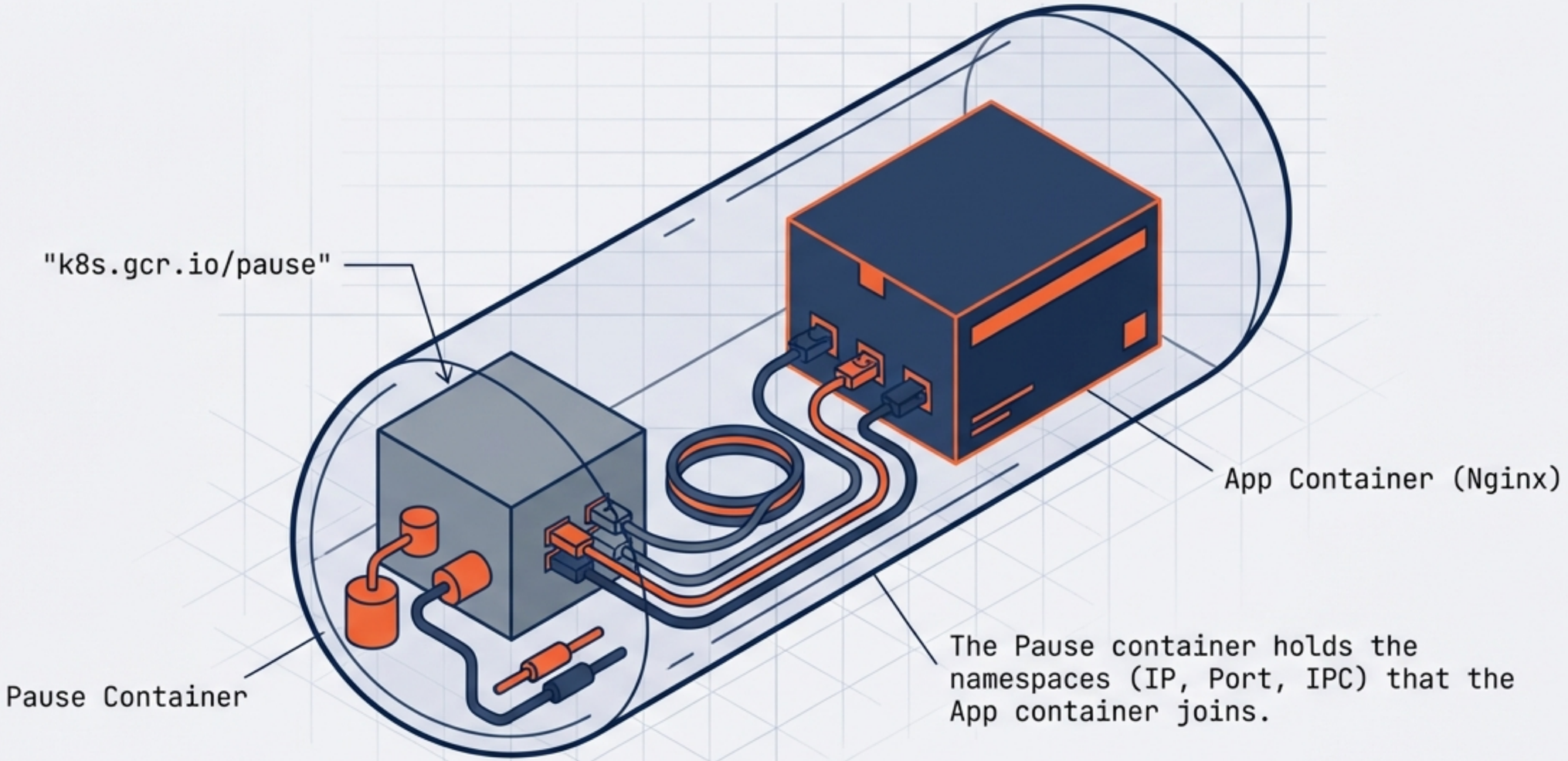
(Focus on State)



NotebookLM

# The New Atom: The Pod

## Shared Context & The Pause Container

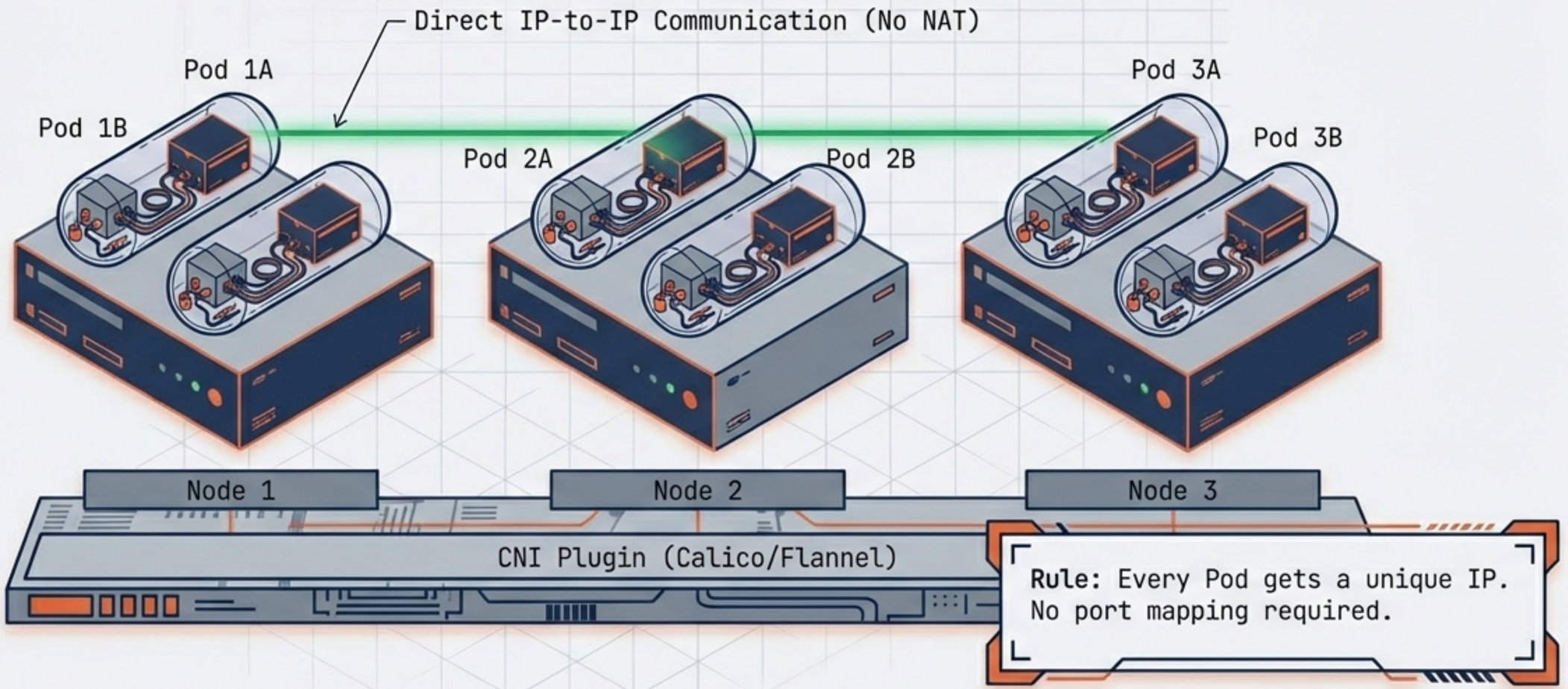





NotebookLM

# Network at Scale

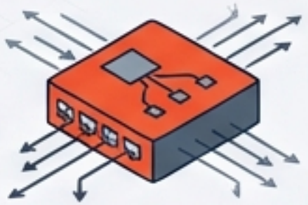
## CNI & The Flat Network






NotebookLM

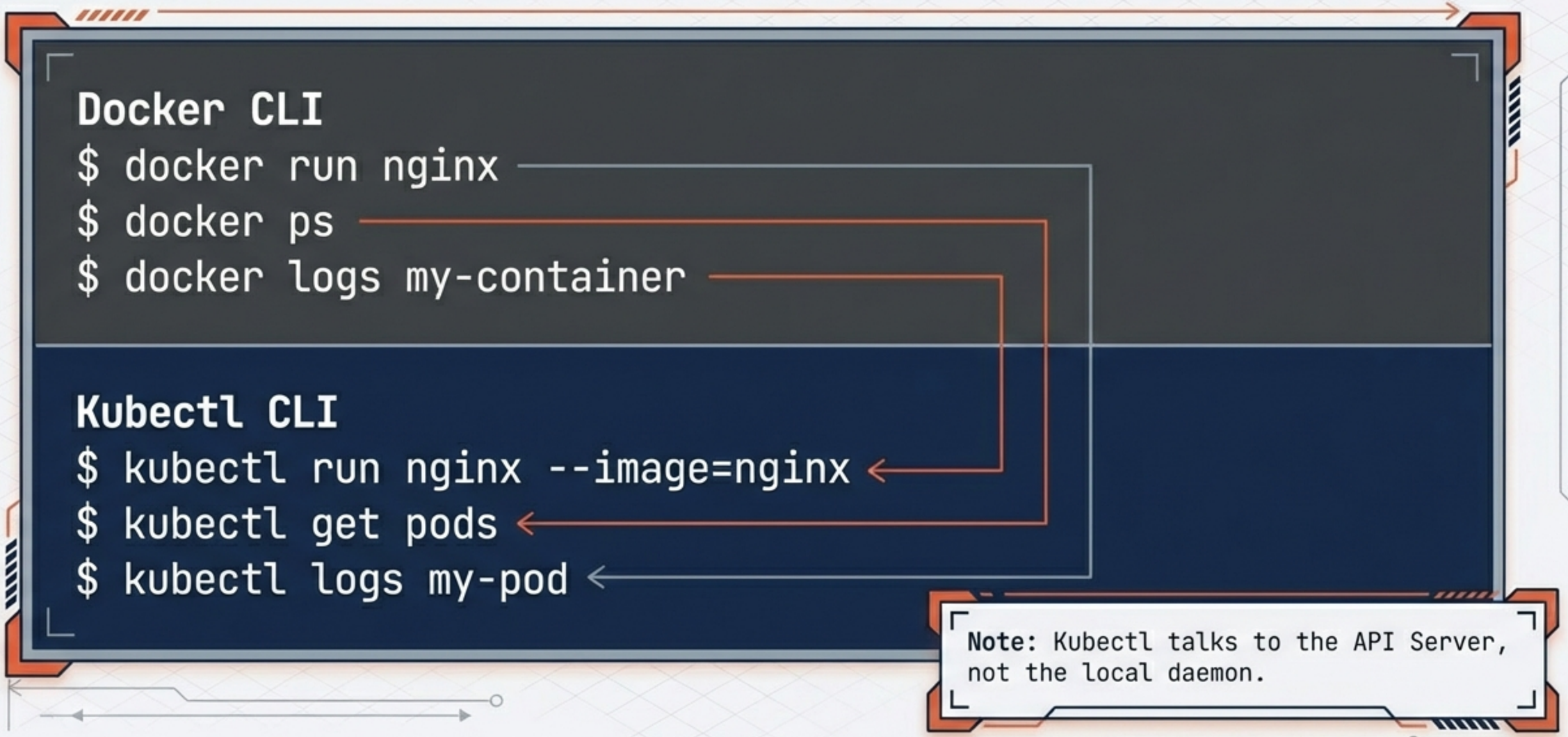
# Translation Guide: Configuration

| Docker Concept | Kubernetes Object |
| --- | --- |
| **Environment Variables** (`-e KEY=VAL`) | **ConfigMap & Secrets**  |
| **Volume / Mount** (`-v /data`) | **PersistentVolumeClaim (PVC)**  |
| **Port Mapping** (`-p 80:80`) | **Service (ClusterIP/LoadBalancer)**  |



NotebookLM

# Translation Guide: CLI Operations






NotebookLM

# Docker Builds the Ship. Kubernetes Manages the Fleet.


**Docker: Packaging & Runtime**
**(The Atom)**

**Kubernetes: Orchestration & Resilience**
**(The Ecosystem)**



NotebookLM