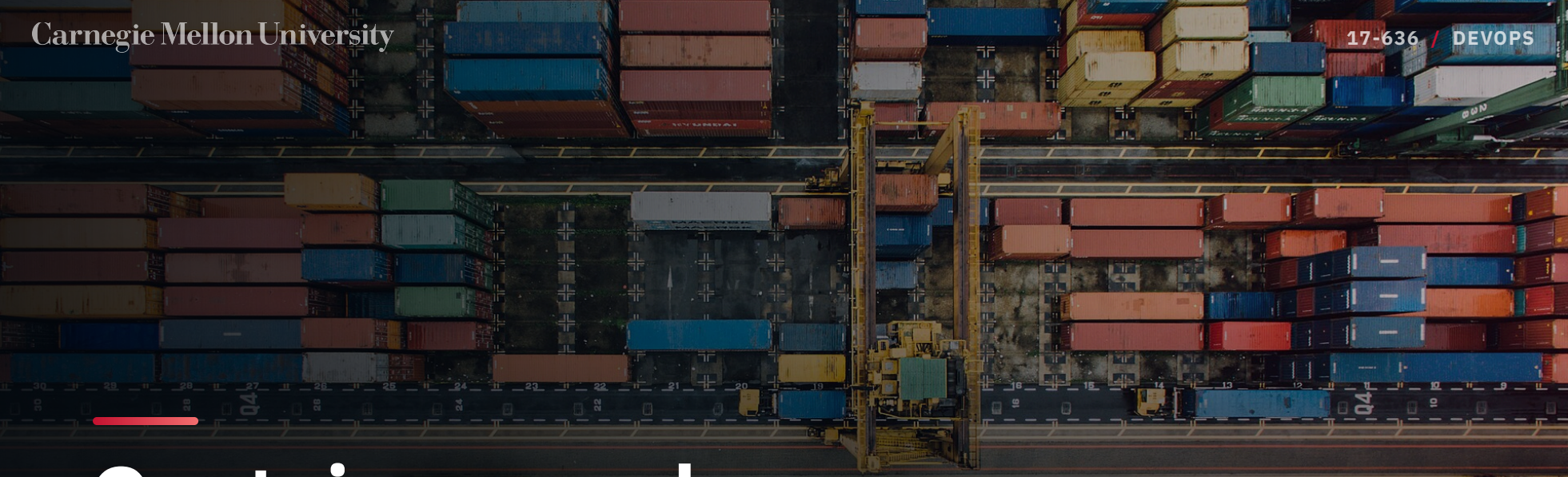


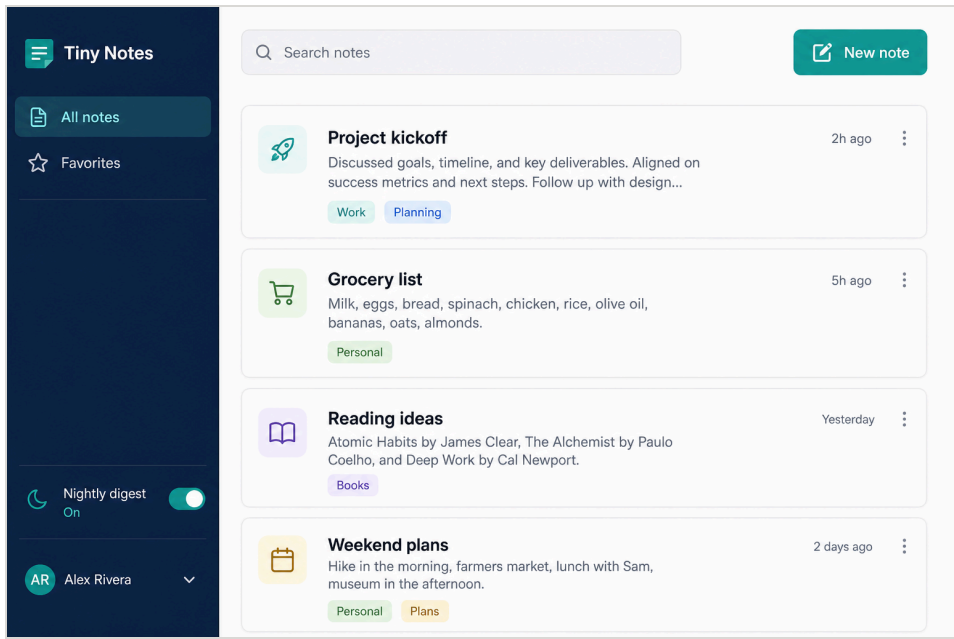
Containers and Orchestration

Chris Timperley · Wednesday, August 26, 2026

🎯 Learning Goals

- Explain how Infrastructure as Code and versioned images make deployments repeatable
- Decide when to use a VM or a container based on isolation, startup time, and state
- Explain how idempotence, convergence, and reconciliation support desired-state automation
- Design a replacement strategy that preserves the state a service needs to recover

Meet Tiny Notes



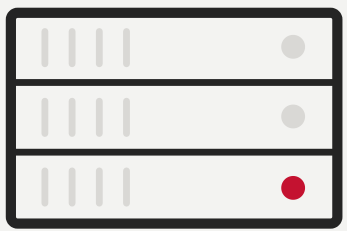
What users do

- create and edit notes
- search their notes
- receive a **nightly digest** (a summary of recent notes)

What the system needs

- a web interface and API
- a database that keeps every note
- a worker that sends the digest

One physical server has accumulated three years of fixes



BARE METAL

notes-prod-1

one physical computer
OS installed directly on it

DAY 1	Install Tiny Notes, Node 20, port 443
MONTH 7	Open port 8443 during an outage
MONTH 12	Upgrade Node directly on the server
MONTH 18	Add a nightly job after missed digests
TODAY	It works—but only with all four changes

The nightly digest works because of one line stored on one server



VERSIONED SOURCES

Digest program

The repository has the code; setup notes omit scheduling.

+



ONLY ON NOTES-PROD-1

`0 2 * * *`

Cron runs send-digest every day at 02:00.

→



PRODUCTION BEHAVIOR

Nightly digest

The job runs only while both pieces exist.

Production behavior can live beyond the application repository.

KEY CONCEPT

Snowflake server

A one-of-a-kind server whose behavior depends on local changes no one recorded well enough to repeat.

Difficult to reproduce

The repository and setup notes do not capture every change, so a replacement may behave differently.

Difficult to change safely

The server is fragile: without a complete record of what changed or why, an edit can break hidden behavior.

A new machine cannot recreate changes that were never recorded.

Martin Fowler | Snowflake Server

Drift is one of the leading causes of Snowflake servers



CONFIGURATION DRIFT

The settings changed.

Someone edited the database URL and opened a firewall port.



DEPENDENCY DRIFT

The ingredients changed.

A package update pulled in a newer Node runtime and system library.



PROCESS DRIFT

The operating procedure changed.

The nightly digest now depends on an undocumented cron job.

Intended system ≠ running system

Documentation can drift, too

EIGHT MONTHS AGO

Setup notes match production

RUNTIME	Node 20
NETWORK	port 443
DIGEST	enabled in the app

Notes = machine

TODAY

The machine no longer matches

RUNTIME	Node 22
NETWORK	ports 443, 8443
DIGEST	OS cron job

Notes ≠ machine

The setup notes no longer describe the running system

The machine died—and there is no disk copy. What would it take to rebuild Tiny Notes?

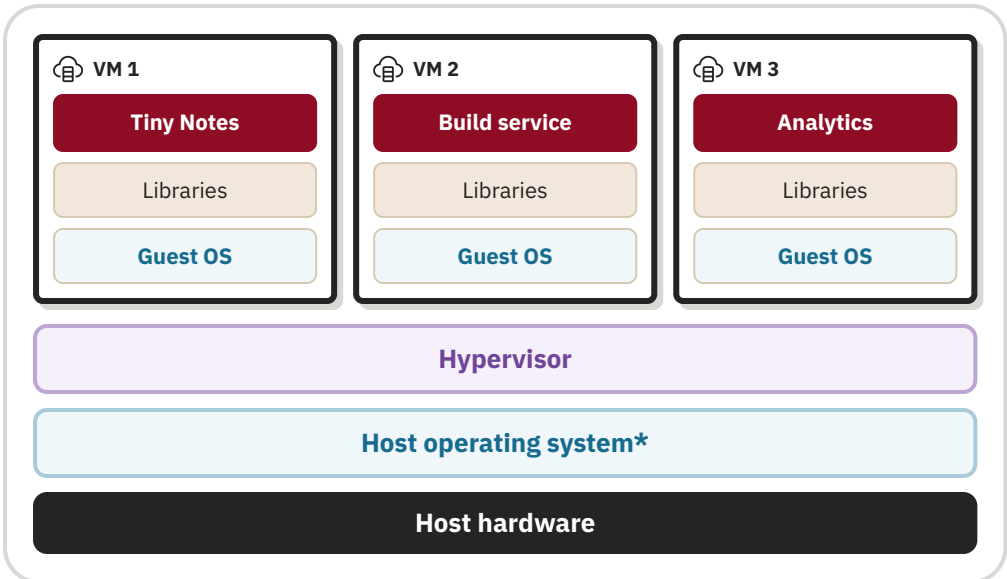


Missing setup details make a server hard to replace

OPERATING MODEL	REPLACEMENT HOW LONG?	CONFIGURATION TRUTH WHERE DOES IT LIVE?	3AM RECOVERY WHAT HAPPENS NEXT?
NEW RUNG 01 **Hand-built bare metal**	**Days**	On the machine and in people's memories	Investigate, reconstruct, and hope
02 Next rung	?	?	?

Virtual Machines, Pets, and Cattle

VMs provide isolation and are easy to copy



Isolation

Each VM gets its own kernel and view of the hardware.

Consolidation

Several operating systems share one physical host.

Mobility

Disks and configuration become files that can be copied or moved.

VMs can be copied instead of rebuilt from scratch



Snapshot
Capture a **known point**

Clone
Copy **disks and settings**

Boot
Start **another guest**

Replace
Retire the **broken VM**

A VM can be easy to copy but unsafe to replace

A pet must be preserved; cattle can be replaced



NAMED VM
apollo

NO · PRESERVE IT

Pet

Preserve or repair this specific machine.

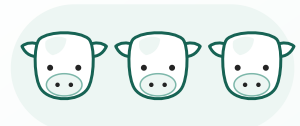
It holds something important that no replacement has.

EXAMPLES

- Hand-tuned server
- VM with local files

OR

YES · REPLACE IT



Cattle

Replace it with an equivalent machine.

`api-042` `api-044`

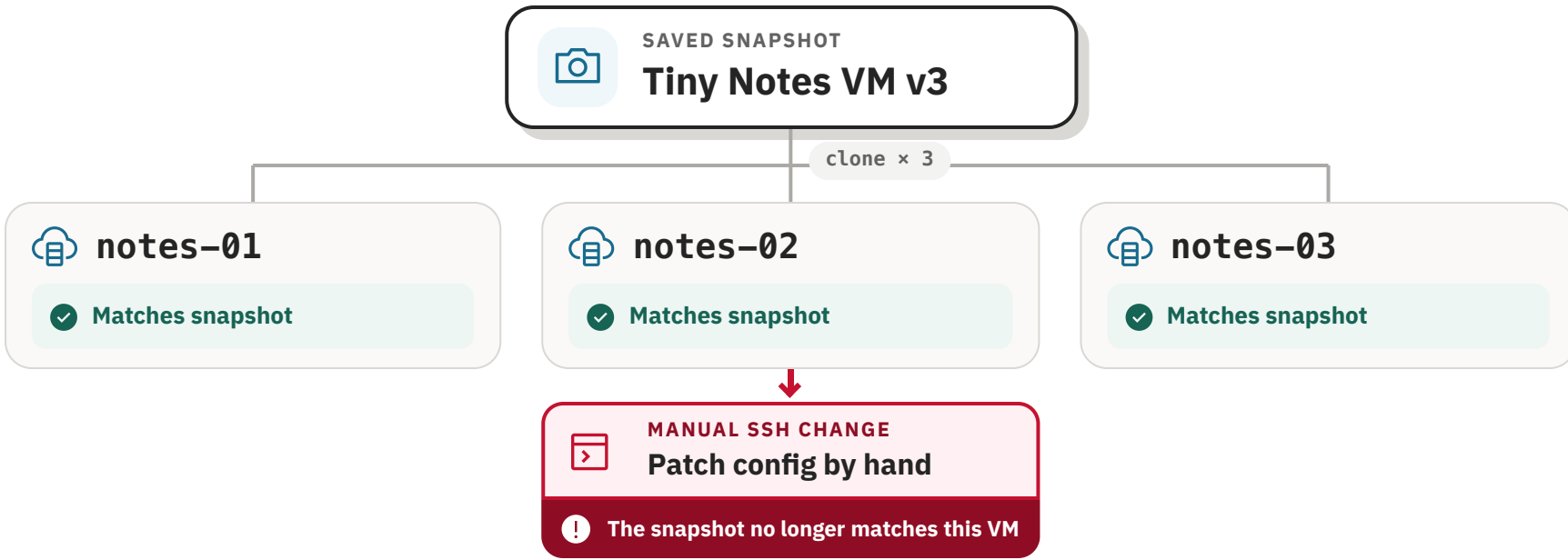
Their setup is repeatable, and important data lives elsewhere.

EXAMPLES

- API server from a snapshot
- Temporary server

Safe replacement lets teams treat critical machines as cattle

A manual change makes one VM different from its snapshot



`notes-02` failed. Which version would you use?

Discuss with a neighbor and then we will vote.

A

Use the snapshot · Create a new VM from the VM v3 snapshot.

B

Copy its disk · Create a new VM from a copy of its disk.

What would make the next replacement safer?



A VM is safe to replace only when it holds nothing unique

Copying a VM is easy; keeping every copy the same is harder

OPERATING MODEL	REPLACEMENT HOW LONG?	CONFIGURATION TRUTH WHERE DOES IT LIVE?	3AM RECOVERY WHAT HAPPENS NEXT?
01 **Hand-built bare metal**	**Days**	On the machine and in people's memories	Investigate, reconstruct, and hope
NEW RUNG 02 **Virtual machine**	**Minutes to hours**	In an image or template—until the guest drifts	Restore, repair, or replace
03 Next rung	?	?	?

Provisioning at Scale

A tiny weekly chore becomes a staffing problem across a fleet



m minutes

human attention per server, per week

×



TINY NOTES FLEET

100 servers

patches · disk checks · configuration fixes

=



IF M = 8

13 h 20 m

every week, before incidents!

A single declaration can provision a fleet of equivalent machines

```
# OpenTofu / Terraform
resource "aws_instance" "notes_api" {
  count         = 100
  ami           = data.aws_ami.tiny_notes.id
  instance_type = "t3.small"

  tags = { Service = "tiny-notes-api" }
}
```

1. **Horizontal scale: declare 100 equivalent machine resources**
2. **Machine shape: choose the image and capacity class**
3. **Operational identity: tag every instance as part of Tiny Notes**

Infrastructure as Code (IaC)

Describe infrastructure in versioned files that software can apply through provider APIs.

IaC can target different cloud providers



Shared intent, provider-specific details
The goal can stay the same, but resource types, settings, and APIs change.

IaC can provision more than virtual machines



CONFIGURATION DRIFT

Network controls

Version subnets, routes, firewall rules, and identity policies.



DEPENDENCY DRIFT

Managed services

Version database engines, storage, queues, and their settings.



PROCESS DRIFT

Traffic and schedules

Version load balancers, DNS records, and scheduled jobs.

IaC can reduce infrastructure drift

Versioned intent exposes hidden manual changes; preview and apply can correct some drift.

IaC makes machine lifecycle changes reviewable



Declare

Version the infrastructure that **should exist**

Preview + review

Inspect proposed creates, changes, and destroys

Apply

Invoke **provider APIs** to make the reviewed change

```
$ tofu plan
Plan: 100 to add, 0 to change, 0 to destroy.

$ tofu apply
Apply complete! Resources: 100 added, 0 changed, 0 destroyed.
```

What happens when we run this setup script twice?

```
#!/usr/bin/env bash
set -e

useradd notes
mkdir /opt/tiny-notes
echo 'NOTES_PORT=8080' >> /etc/tiny-notes.env
systemctl enable --now tiny-notes
```

1. **Execution policy: stop at the first failed command**
2. **Replay actions: create the account, then the directory**
3. **Replay more actions: append configuration, then start the service**

The first time we run the script...

It creates the account and directory, writes the configuration, and starts the service.

The second time we run the script...

`useradd: user 'notes' already exists`

With `set -e`, execution stops here. The remaining commands never run.



KEY CONCEPT

Idempotence

Running the same operation again leaves the system in the same resulting state.

```
apply(apply(state)) = apply(state)
```

A run may inspect reality again without changing the outcome. One operation can be **idempotent**; repeated corrections can **converge**; a controller can **reconcile continuously**.

Ansible describes the account and service state we want

```yaml
# Ansible
- hosts: notes_api
  become: true
  tasks:
    - ansible.builtin.user:
        name: notes
        state: present
    - ansible.builtin.systemd_service:
        name: tiny-notes
        state: started
        enabled: true
```



1. **Scope: converge the Tiny Notes hosts with elevated privileges**
2. **Desired identity: the service account must be present**
3. **Desired process: the service must be enabled and running**

Declarative tools converge actual state toward intent



Observe reality

Inspect what is true now

Compare with intent

Find **what differs** from the declaration

Change the gaps

Apply **only the changes** that are needed

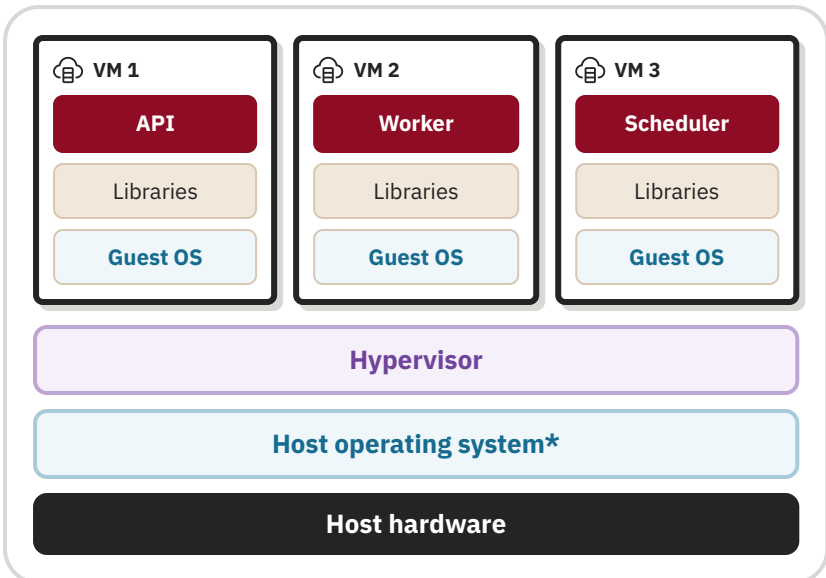
On the second run...
The existing user is left alone. The service starts only if needed.

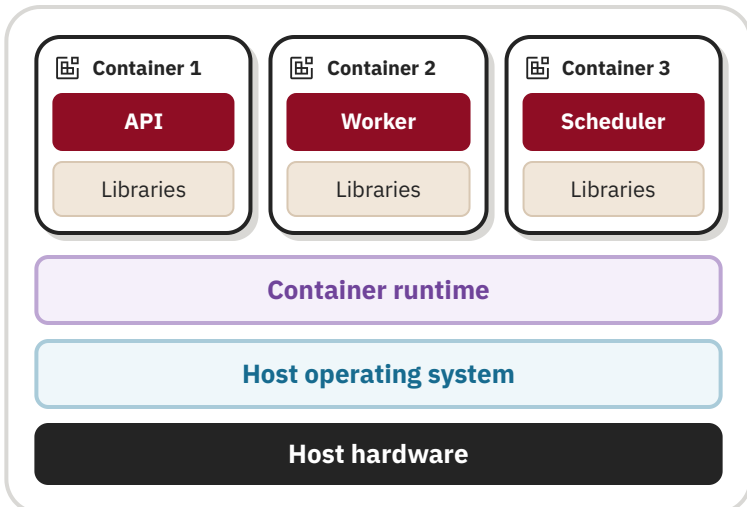
Versioned files record how machines should be configured

OPERATING MODEL	REPLACEMENT HOW LONG?	CONFIGURATION TRUTH WHERE DOES IT LIVE?	3AM RECOVERY WHAT HAPPENS NEXT?
01 **Hand-built bare metal**	**Days**	On the machine and in people's memories	Investigate, reconstruct, and hope
02 **Virtual machine**	**Minutes to hours**	In an image or template—until the guest drifts	Restore, repair, or replace
NEW RUNG 03 **Cloud-provisioned VM**	**Minutes**	In a versioned provisioning declaration	Request a replacement; reapply intent
04 Next rung	?	?	?

Containers and Images

VMs repeat the operating system; containers share it





Sharing the host kernel lowers container overhead

VIRTUAL MACHINES

Independent guests

Each VM carries and boots a complete guest operating system.

Pros
Strong machine boundary; each VM can run a different guest OS or kernel.

Cons
More memory and storage overhead; another operating system to boot and manage.

CONTAINERS

Shared host kernel

Each container packages userspace while sharing the host kernel.

Pros
Smaller artifacts, fast startup, and denser use of a host.

Cons
The kernel is a shared trust boundary; host compatibility still matters.

Containers often package microservices

4

APPLICATION ARCHITECTURE

Services by capability

Tiny Notes now has four services that can be deployed independently.

Web UI
Node.js

Notes API
Python

Digest worker
Native library

Database
PostgreSQL

PACKAGING MODEL

One image per service

The boundaries often align, but neither model requires the other.

Package
Each service with its dependencies.

Deploy
Scale and release services independently.

Replace
Start small units quickly.

Isolate
Keep one service's libraries away from another's.

A Dockerfile makes a service environment reproducible

```
FROM node:22-alpine
WORKDIR /app

COPY package*.json ./
RUN npm ci --omit=dev
COPY . .

USER node
CMD ["node", "server.js"]
```

1. **Choose the base runtime and working directory**
2. **Install production dependencies into the image**
3. **Add the application source**
4. **Run as a non-root user and set the startup command**



What could go wrong here?

```
FROM node:22-alpine
WORKDIR /app

COPY package*.json ./
RUN npm ci --omit=dev
COPY . .

USER node
CMD ["node", "server.js"]
```

Docker executes a recipe; Nix requests a package set



```
FROM alpine:3.21

RUN apk add --no-cache \
    nodejs \
    npm \
    git
```

Imperative
Execute these installation steps in this order.



```
{ pkgs }:

pkgs.mkShell {
  packages = [
    pkgs.nodejs_22
    pkgs.git
  ];
}
```

Declarative
Request the packages that must be present in the environment.

Build the image once; configure containers at startup



Build

Turn the Dockerfile and source **into an image**

Store

Publish the **identified image** to a registry

Run

Start the **same bytes** with runtime configuration

One image can create many containers

BUILD OUTPUT

Image

One immutable, content-identified artifact.

Identity
`tiny-notes-api@sha256:7c…`

Contents
Code, runtime, and dependencies.

Startup
The default command to execute.

RUNTIME INSTANCES

Containers

The image plus configuration supplied at launch.

API-1
`db-prod` · port `8080`

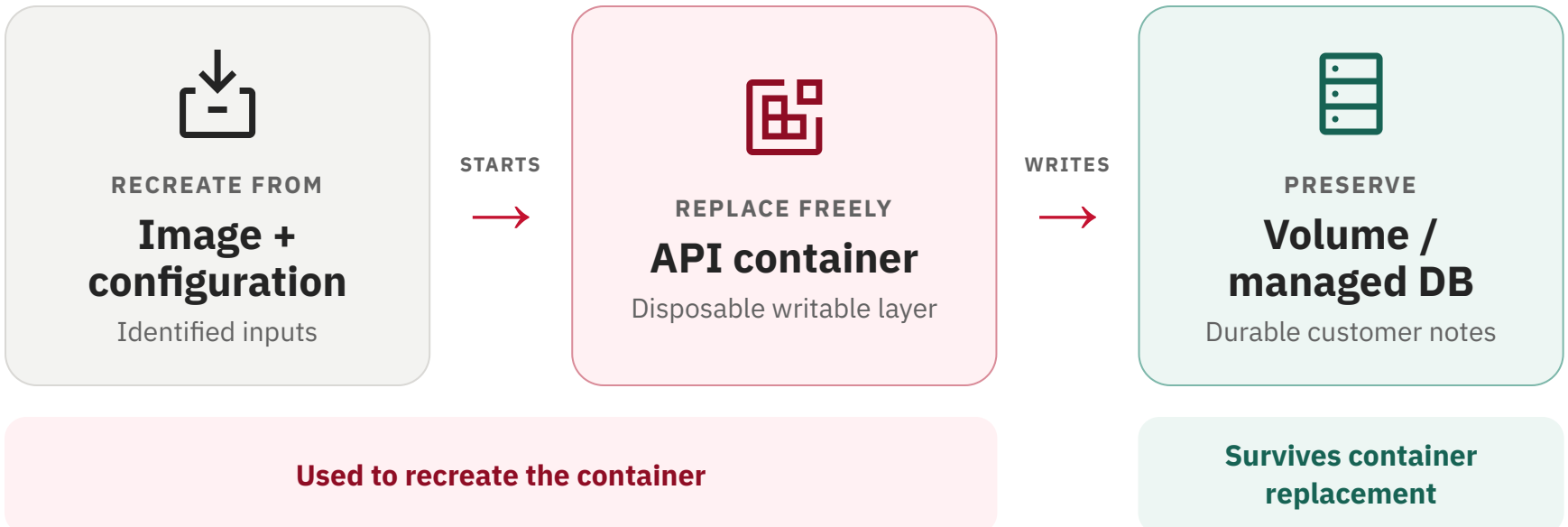
API-2
`db-prod` · port `8080`

API-3
`db-test` · port `9090`

Container inputs and outputs have different lifetimes

Boundary	Tiny Notes example	Changes when...	Must survive a container replacement?
Artifact	API code + dependencies	a new version is built	Recreated from the identified image
Runtime configuration	`NOTES_DB_URL`	the deployment environment changes	Yes. Supplied again at start
Writable container layer	`/tmp`, cache, local edits	this execution changes files	**No. Treat it as disposable**
Persistent state	notes on a volume or managed database	a user edits a note	**Yes. Stored beyond the container**

Replace containers; keep durable state elsewhere



A tag labels an image; a digest identifies it

```
$ docker build -t notes-api:stable .
```



YOU CHOOSE

Tag / label

`notes-api:stable`

A readable pointer. Reusing the tag can make it point to different bytes.



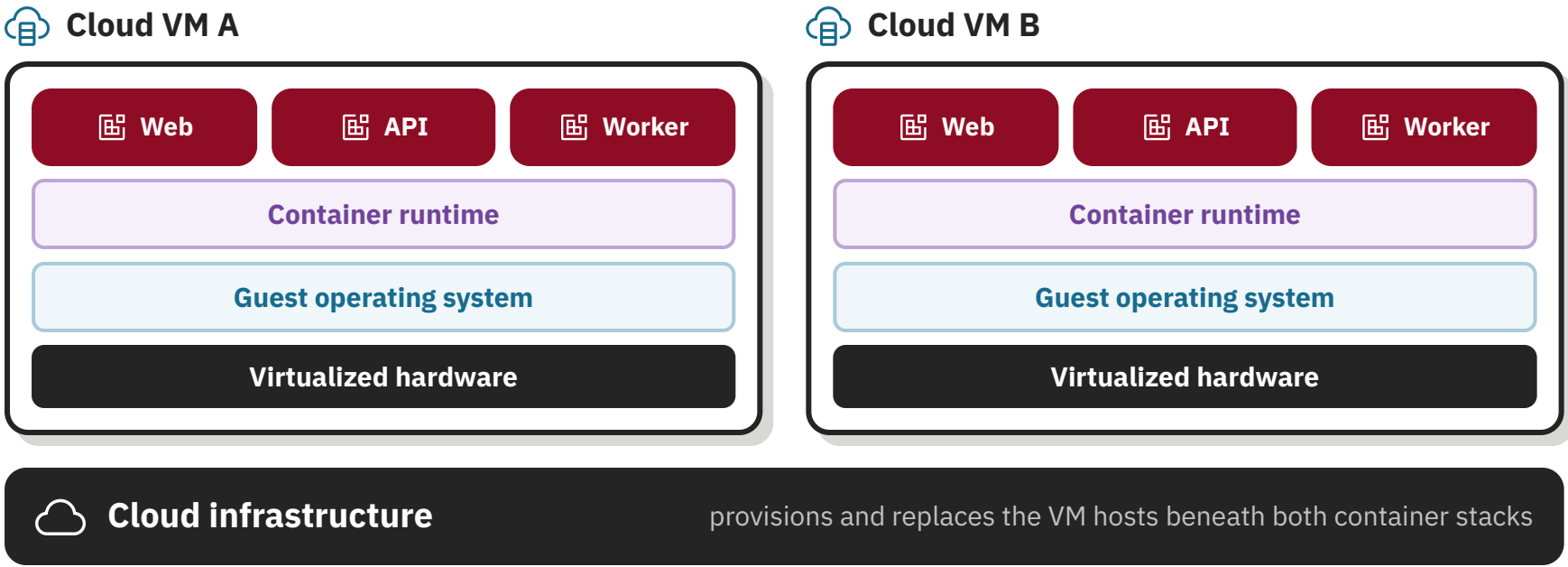
CONTENT-DERIVED

Digest

`sha256:7c4a…91e2`

An exact content identity. The same digest always names the same bytes.

Containers commonly run inside VMs

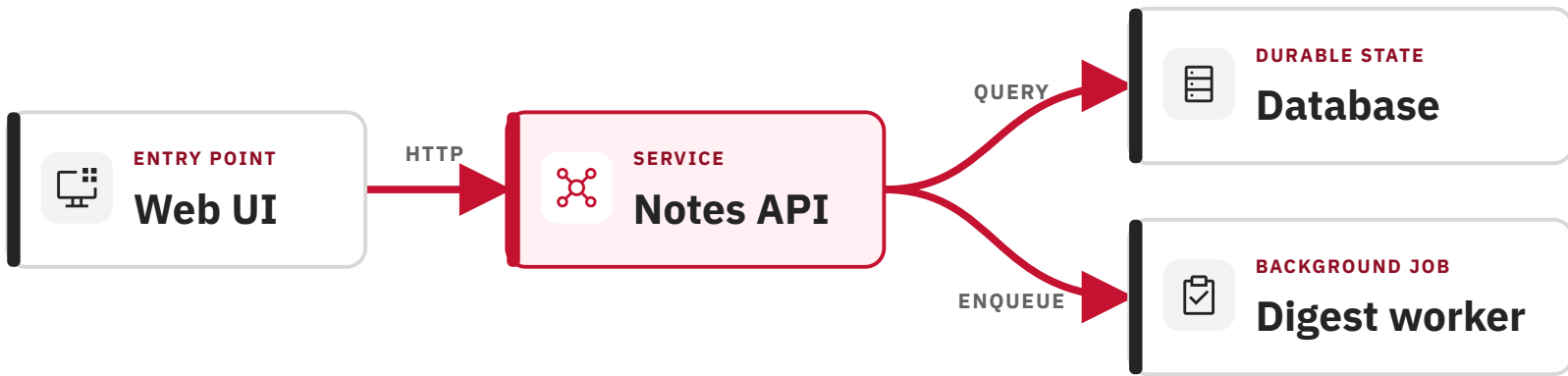


Container images cut replacement time to seconds

OPERATING MODEL	REPLACEMENT HOW LONG?	CONFIGURATION TRUTH WHERE DOES IT LIVE?	3AM RECOVERY WHAT HAPPENS NEXT?
01 **Hand-built bare metal**	**Days**	On the machine and in people's memories	Investigate, reconstruct, and hope
02 **Virtual machine**	**Minutes to hours**	In an image or template—until the guest drifts	Restore, repair, or replace
03 **Cloud-provisioned VM**	**Minutes**	In a versioned provisioning declaration	Request a replacement; reapply intent
NEW RUNG 04 **Container**	**Seconds**	In the image plus configuration supplied at start	Start another; recover state separately

Orchestration and Reconciliation

Cheap containers create a coordination problem at production scale



4 services × **3** copies each = **12** containers

spread across **5 machines** that fail independently

Place + connect
where it runs and how services reach it

Configure
which image and runtime inputs it needs

Keep services running
restart containers that stop working; send traffic only to containers that are ready

Commands stop describing the system after the first failure

10:00

Commands create three replicas

```
$ start API-1 on host A
$ start API-2 on host B
$ start API-3 on host C
$ connect APIs to database
```

Desired **3** Actual **3**

HOST B FAILS

10:07

The system no longer matches

✓	**API-1**	host A
×	~~**API-2**~~	~~host B~~
✓	**API-3**	host C

Desired **3** Actual **2**

The commands describe what happened at 10:00.
They do not describe what should be true now—or restore the missing replica.

A Kubernetes declaration describes what should be running

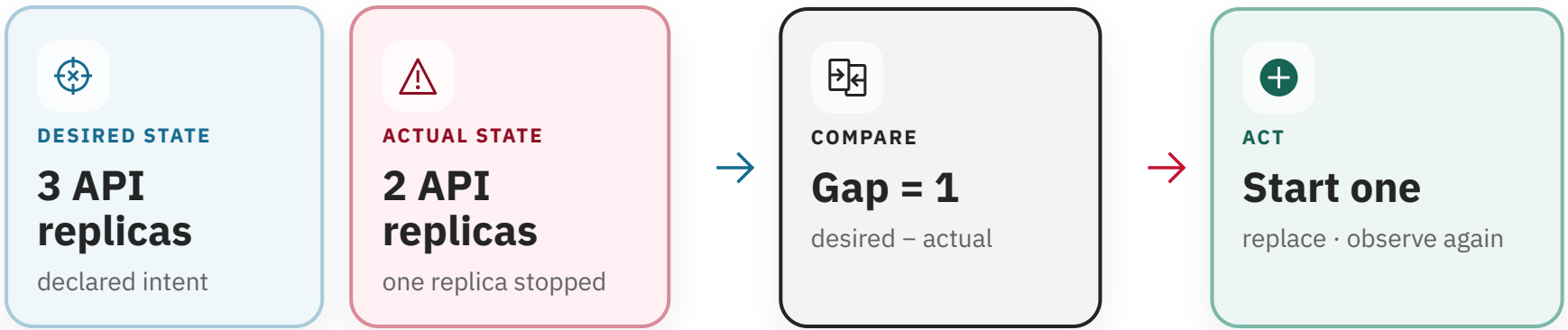
```yaml
apiVersion: apps/v1
kind: Deployment
metadata: { name: notes-api }
spec:
  replicas: 3
  template:
    spec:
      containers:
        - name: api
          image: registry.example/tiny-notes@sha256:7c4a...91e2
          envFrom:
            - secretRef: { name: notes-prod }
```

Pod
A Pod groups one or more containers that must run together. Kubernetes schedules and replaces the group as one unit.

1. **Desired count: the Deployment keeps three Pods running**
2. **Exact artifact: each Pod runs these content-identified bytes**
3. **Runtime input: each Pod loads production configuration by reference**



Reconciliation closes the desired–actual gap



Compose describes how containers run together on one machine

```yaml
# compose.yaml
services:
  api:
    image: registry.example/tiny-notes-api:v4
    environment:
      NOTES_DB_URL: postgres://db/notes
    depends_on: [db]
  db:
    image: postgres:17
    volumes:
      - notes-data:/var/lib/postgresql/data

volumes:
  notes-data: {}
```

1. **Services: run the API and database from named images**
2. **Runtime wiring: configure the API and connect it to the database**
3. **Durable state: mount a named volume outside the database container**

Continuous reconciliation can replace a failed unit before a human is paged

OPERATING MODEL	REPLACEMENT HOW LONG?	CONFIGURATION TRUTH WHERE DOES IT LIVE?	3AM RECOVERY WHAT HAPPENS NEXT?
01 **Hand-built bare metal**	**Days**	On the machine and in people's memories	Investigate, reconstruct, and hope
02 **Virtual machine**	**Minutes to hours**	In an image or template—until the guest drifts	Restore, repair, or replace
03 **Cloud-provisioned VM**	**Minutes**	In a versioned provisioning declaration	Request a replacement; reapply intent
04 **Container**	**Seconds**	In the image plus configuration supplied at start	Start another; recover state separately
NEW RUNG 05 **Orchestrated containers**	**Seconds, automatically**	In artifacts and declared state in version control	It was replaced before anyone was paged

Replacement got faster as configuration moved out of the machine

OPERATING MODEL	REPLACEMENT HOW LONG?	CONFIGURATION TRUTH WHERE DOES IT LIVE?	3AM RECOVERY WHAT HAPPENS NEXT?
01 **Hand-built bare metal**	**Days**	On the machine and in people's memories	Investigate, reconstruct, and hope
02 **Virtual machine**	**Minutes to hours**	In an image or template—until the guest drifts	Restore, repair, or replace
03 **Cloud-provisioned VM**	**Minutes**	In a versioned provisioning declaration	Request a replacement; reapply intent
04 **Container**	**Seconds**	In the image plus configuration supplied at start	Start another; recover state separately
DESTINATION 05 **Orchestrated containers**	**Seconds, automatically**	In artifacts and declared state in version control	It was replaced before anyone was paged

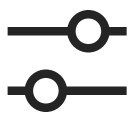
Replacement days → seconds

Truth machine → declaration

Recovery investigation → reconciliation

Faster replacement requires stronger declarations

DAYS



Machine

Describe one **configured machine**

MINUTES



Image

Describe a **reproducible artifact**

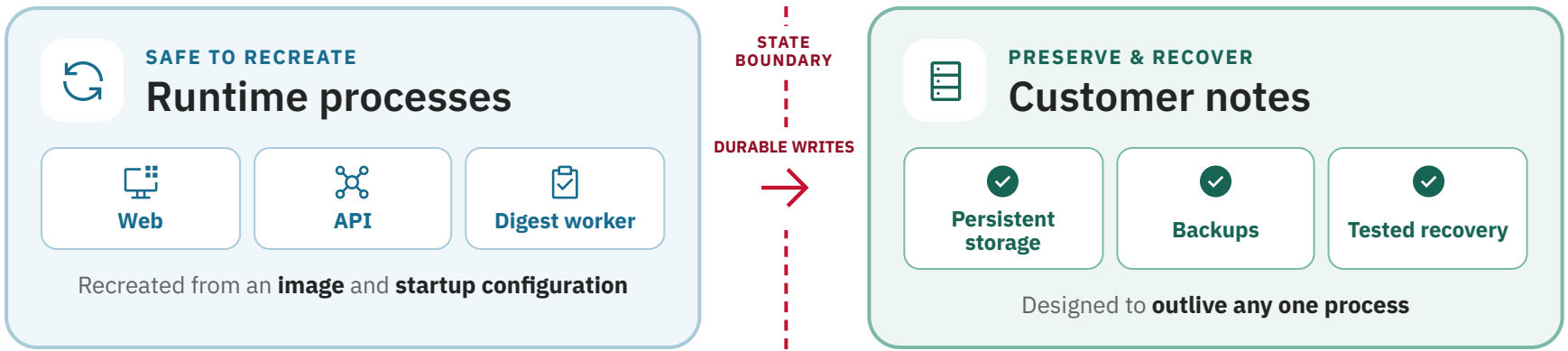
SECONDS



Desired system

Declare the shape that **should be running**

Replace processes while preserving durable data



What’s wrong with this picture?

Each scenario contains a subtle operational problem.

FIND THE RISK

Diagnose

What is wrong with this design?

DESIGN THE REPAIR

Redesign

What would you do instead?

A restart changes production

```
# compose.yaml
services:
  api:
    image: tiny-notes-api:stable
```

The database was replaced successfully

```
services:
  api:
    image: notes-api@sha256:7c4a...
    environment:
      NOTES_DB_URL: postgres://db/notes

  db:
    image: postgres:17
```

The team removes and recreates both containers for every deployment.

The new database starts cleanly.

Every customer note is gone.

Test connects to production

```
FROM node:22-alpine
WORKDIR /app
COPY . .
COPY .env.production .env
RUN npm ci --omit=dev
CMD ["node", "server.js"]
```

Recovery sent the digest twice

```
services:
  digest-worker:
    image: notes-worker@sha256:91e2...
    command: ["node", "digest.js"]
    restart: always
```

The worker sends the email.

It crashes before recording completion and restarts.

The same customer receives the digest again.

🚀 Today, we learned how to ...

- Explain how Infrastructure as Code and versioned images make deployments repeatable
- Decide when to use a VM or a container based on isolation, startup time, and state
- Explain how idempotence, convergence, and reconciliation support desired-state automation
- Design a replacement strategy that preserves the state a service needs to recover