A build should leave a trail.

Source
revision

→

Build
context + recipe

→

Image
config + layers

→

Evidence
tests + identity

A trustworthy image connects controlled inputs to a tested result.

Author & Course Lead — Badr Tajini · ESILV · 2026–2027

What would you hand to a clean host?

A Dockerfile is copied to a clean host. What is still missing?

> The intended context, locked dependencies, base identity, runtime contract and a test plan.

A Dockerfile is one part of a reproducible build contract.

Author & Course Lead — Badr Tajini · ESILV · 2026–2027

A release claim has four questions.

INPUT

What entered?

context · lock · base

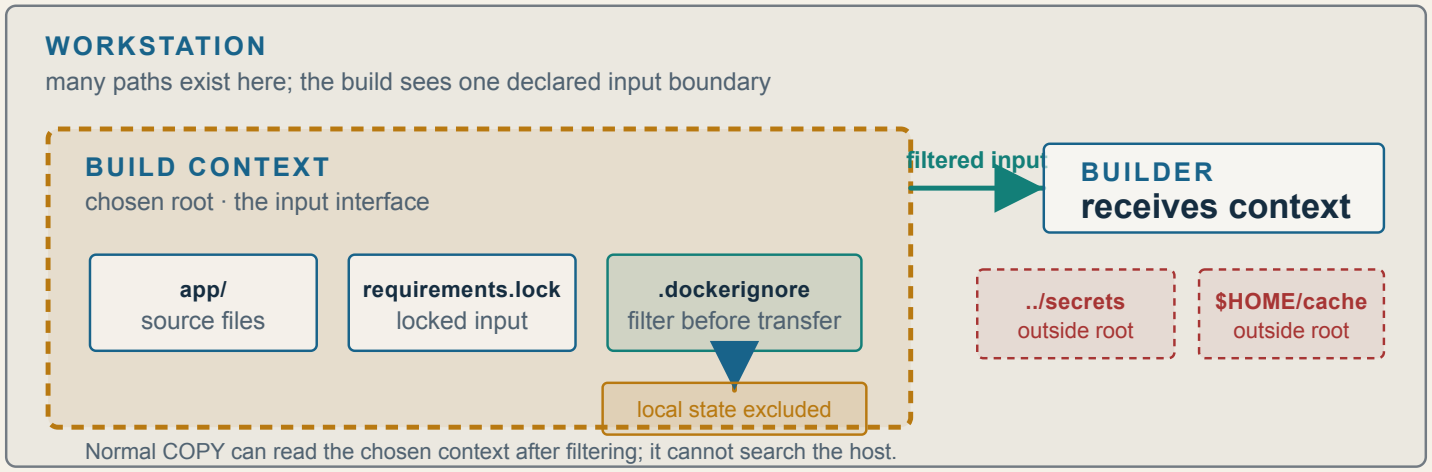
EVIDENCE

What passed?

identity · runtime · endpoint

Ask what entered, what changed, what exists and what passed.

The context is a boundary, not a laptop.



Static boundary: the build context is the chosen root. `.dockerignore` filters local state before transfer; normal COPY cannot read paths outside that root.

Normal COPY can use files in the chosen context, subject to ignore rules.

Which files cross the build boundary?

The root contains `app/`, `requirements.lock`, `.venv/` and `evidence/`. Which belong by default?

> Application and lock inputs are candidates. Local environments and evidence need an intentional exclusion unless the contract explicitly requires them.

Selecting a root is the first reproducibility decision.

COPY reads the context you chose.

AVAILABLE

`app/`

inside context

`requirements.lock`

inside context

UNAVAILABLE

`../secrets`

outside context

`$HOME/.cache`

outside context

A build path is an input declaration; it is not a promise to search the host.

Ignore before you copy.

Docker ignore rules keep local state out before a COPY can capture it.

Git ignore and Docker ignore ask different questions.

GIT

Versioned?

`.gitignore` shapes repository state.

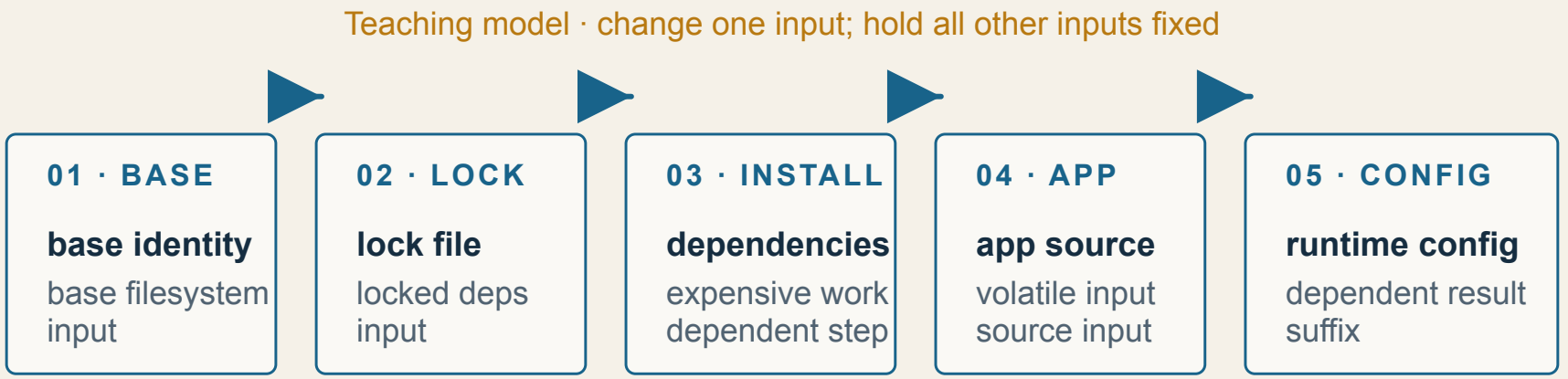
DOCKER

Sent?

`.dockerignore` shapes build input.

Tracked status and build-context membership are separate decisions.

Cache reuse starts with a matching prefix.



Static fallback: BASE → LOCK → INSTALL → APP → CONFIG. Change one input: the first changed input is the first miss, and later dependent steps rebuild. This model reports causality, not elapsed time.

The first changed input marks the first miss; dependent steps rebuild after it.

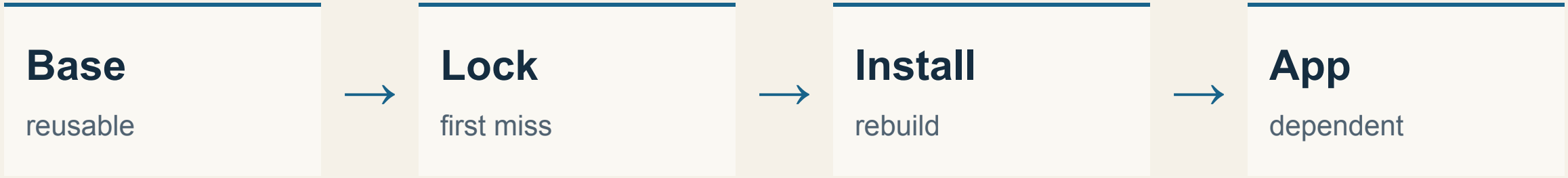
One source edit should preserve the dependency prefix.

Change only `app/__init__.py`. Which steps should remain eligible?

> The base, dependency lock and install steps remain eligible; the first miss is the application copy and its dependent suffix follows.

With lock and base fixed, an application edit starts its rebuild at the app copy.

A lock edit invalidates installation.



Dependency inputs belong before application source when installation is the expensive stable work.

A fast rebuild is useful feedback, not reproducibility proof.

WARM TRACE

Reuse

Eligible prior results are available.

REPRODUCIBLE CLAIM

Control

Inputs, identity and behavior are recorded.

Cache speed describes reuse under current inputs; reproducibility requires controlled identity and evidence.

Author & Course Lead — Badr Tajini · ESILV · 2026–2027

Reproducibility starts before the builder runs.

Context
filtered

→

Dependencies
locked

→

Base
identified

→

Args
stable

→

Record
trace

Lock dependencies, identify the base, stabilise arguments and record the build.

A base tag is a coordinate; identity needs a record.

Does `FROM python:latest` identify the same base forever?

> No. A mutable tag is a coordinate that can resolve differently. Use the controlled base identity required by the course route and record what was used.

A conventional tag can resolve to changing content, so retain the resolved identity in evidence.

Author & Course Lead — Badr Tajini · ESILV · 2026–2027

Dockerfile steps become image content.

Dockerfile
ordered recipe

→

Layers
filesystem deltas

→

Config
CMD · USER · LABEL

→

Image
package

COPY and RUN contribute filesystem layers; CMD, USER and LABEL configure the image package.

Build tools stop at the copy boundary.

Builder

compilers + tooling

→

COPY --from

selected artifacts

→

Runtime stage

serve only

A multi-stage image selects runtime material instead of carrying the builder workspace forward.

Ten minutes to reset.

Pause here.
Return with one question about a build boundary.

Break: resume with cache, layers and runtime identity still visible in the route.

Build time and run time are different clocks.

Build
RUN installs

→

Image
CMD stored

→

Start
process begins

→

Test
observe behavior

RUN executes while building; CMD supplies a default for a later container process.

A foreground command gives the runtime one process to observe.

IMAGE

CMD

default command stored in config

CONTAINER

Process

starts, serves, receives stop and exits

The image declares a command; the container lifecycle supplies the running process and stop behavior.

A non-root declaration needs runtime proof.

The Dockerfile says `USER app`. What evidence is still needed?

> Inspect the image configuration and observe a non-zero runtime UID while the service still reads configuration, binds its port and writes only to intended paths.

USER in a Dockerfile is a claim about the default runtime identity; id output tests the claim.

Runtime behavior includes where the process may write.

DECLARED

UID ≠ 0

default identity

Port

bind succeeds

OBSERVED

Health

endpoint responds

Write test

only intended path

Non-root is a starting control; the service still needs intentional readable and writable paths.

One release has several identities.

Source
Git SHA

→

Image
OCI labels

→

Registry
digest

→

Evidence
later commit

Source revision, image metadata, registry content and evidence commit answer different questions.

Labels and the version endpoint should tell one source story.

IMAGE INSPECT

`revision = SHA`

OCI label in config

RUNTIME CHECK

`/version → SHA`

service-facing observation

Independent image and runtime records should agree on the intended revision.

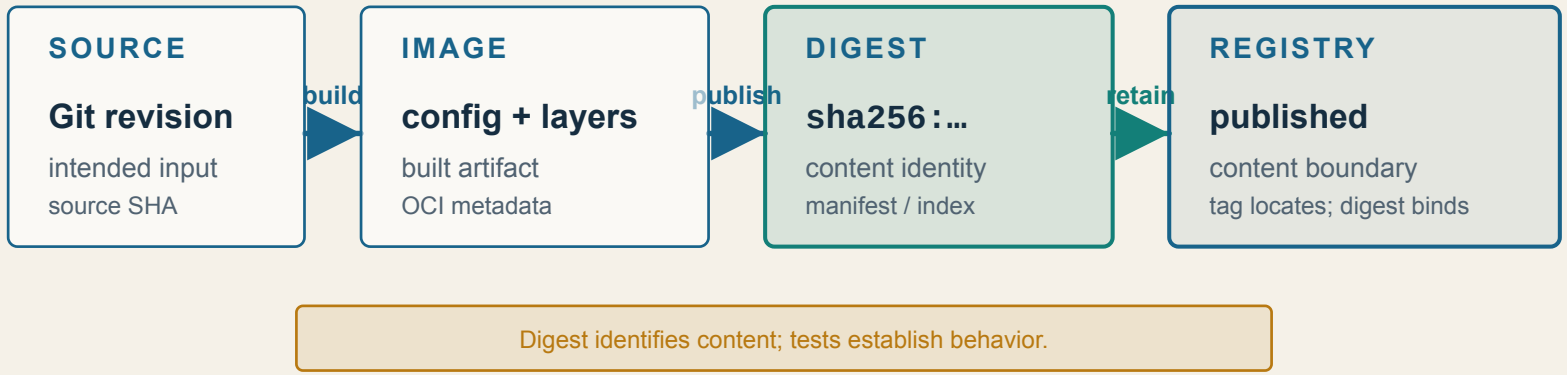
Three identities, three questions.

Match the record: source revision, local image, running instance?

> Git SHA → source revision. Local image/config ID → local image object. Container ID → one runtime instance.

Source, image and runtime identifiers should never be swapped in evidence.

A tag points; it does not bind content.



Identity chain: source revision → image config and layers → registry/platform digest → published content. The digest identifies content; tests are still required for behavior claims.

A Git-SHA-shaped tag remains movable until the published digest is retained.

A digest identifies content, not correctness.

Tag
human locator

→

Digest
manifest/index

→

Pull
exact content

→

Test
behavior evidence

Retain the registry/platform digest, then use tests to support behavior claims.

A registry serves; a daemon runs.

REFERENCE TRANSCRIPT · no registry runtime observed

```
local_image  ->  registry/repository:<full-Git-SHA>
registry      ->  repository@sha256:<manifest-or-index>
digest pull   ->  target daemon creates local image
tests         ->  health · version · readiness · UID
```

Push and pull move image content; the target daemon creates the runtime instance.

A tested image makes a narrower claim than a built image.

/healthz
200

→

/version
source SHA

→

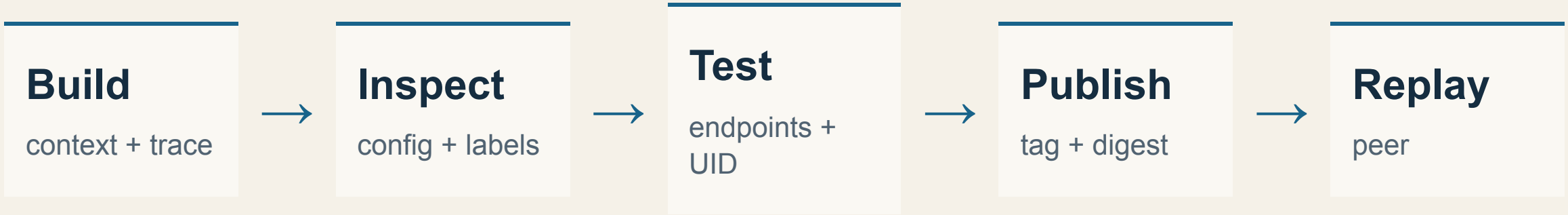
/readyz
503 without DB

→

UID
non-zero

Health, version, readiness and UID observations support distinct parts of the API image contract.

TD2 turns the model into replayable work.



Build, inspect, test, publish and replay are separate links in the evidence chain.

Before TD2, name the image contract.

Complete the sentence:
‘I trust this image enough to test because…’

> The context is intentional, dependencies and base identity are controlled, labels match the source, runtime is non-root, endpoints meet the contract and the result has replayable evidence.

Context, identity, runtime and evidence must agree before publication is meaningful.

Arrive at TD2 ready to predict.

BEFORE A COMMAND

Predict

context · expected result · stop condition

AFTER A COMMAND

Record

observed output · identity · next decision

State the execution context, expected observation, stop condition and evidence path before typing.

Author & Course Lead — Badr Tajini · ESILV · 2026–2027

Use the remaining time to repair one claim.

Context, cache, identity or test? Which boundary still feels unclear?

> Return to the relevant position, collect a new prediction, and ask what observation would change the claim.

Contingency is for interpretation and recovery, not for adding a new topic.