“It works here” is a starting point.

ONE LAPTOP

A successful request.

A TEAMMATE

Can they repeat the result?

A delivery result must survive a change of person and machine.

What must travel with the service?

Send the files.
Is that enough?

> A source revision, a reproducible definition, an intended environment and a test of the result.

A teammate needs a testable contract, not just a folder.

Change → delivery → feedback

Change
source revision

→

Check
test a claim

→

Deliver
defined release

→

Observe
result → next change

Feedback must change the next decision.

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

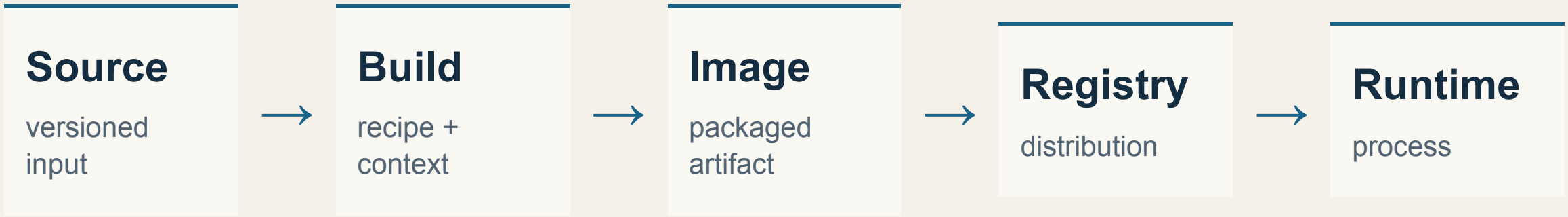
A signal is useful when someone acts.

The release returns errors. Who needs to learn?

> The people responsible for the change, its tests, the release decision and recovery. Assign an owner to the next action.

DevOps connects technical feedback with shared responsibility.

Docker is part of the delivery loop.



Packaging and execution help the loop; they do not define the whole system.

A terminal is a window, not a machine.

Laptop

local shell

→

SSH

remote connection

→

VPS

remote shell

After SSH, the next shell command runs on the VPS.

Which Engine receives the request?

Shell

runs the CLI

→

Docker client

sends request

→

Daemon

manages objects

Shell context and Docker context are separate questions.

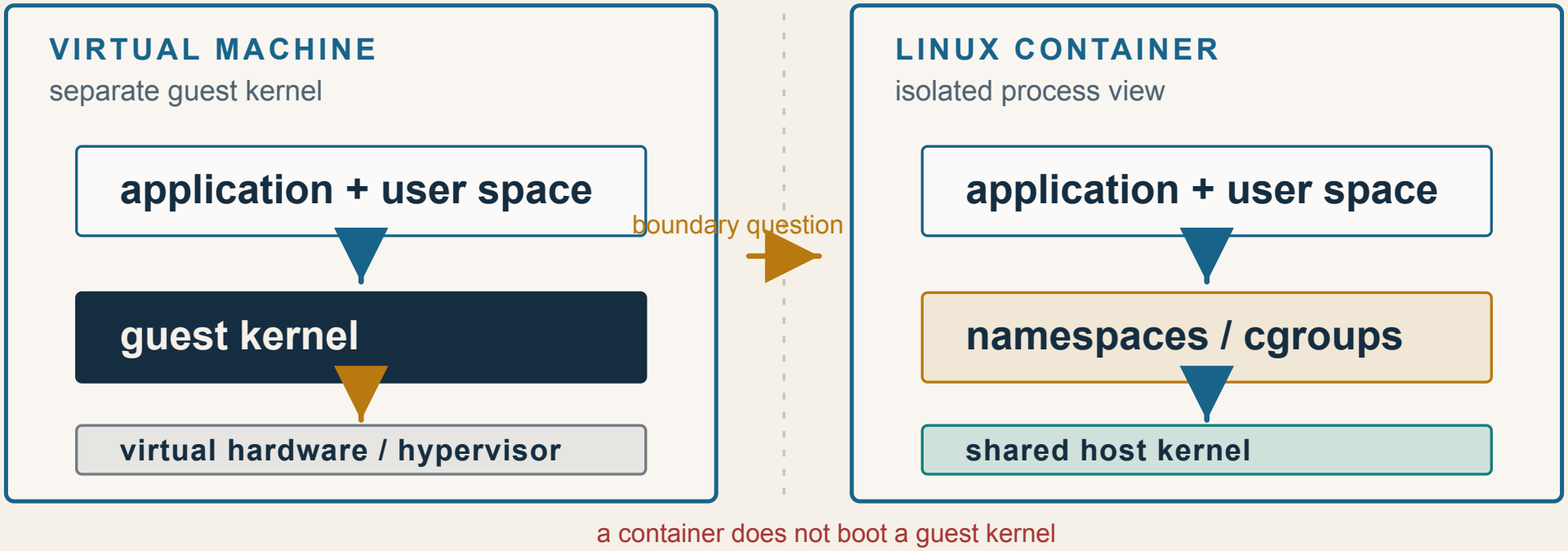
Follow the context change.

After docker exec, where does the new process run?

> Inside the running container. The CLI still sends a request to the selected daemon.

Name the shell, the daemon and the process separately.

Two different kernel boundaries.



Boundary verdict: a virtual machine supplies a guest kernel; a Linux container isolates processes and uses the shared host kernel.

A Linux container isolates processes while sharing the host kernel.

An isolated process still uses a host.

Separate views.

Process tree
Filesystem
Network namespace

Shared foundation.

Host kernel
Host resources
Runtime authority

Isolation is useful; it is not a promise of perfect security.

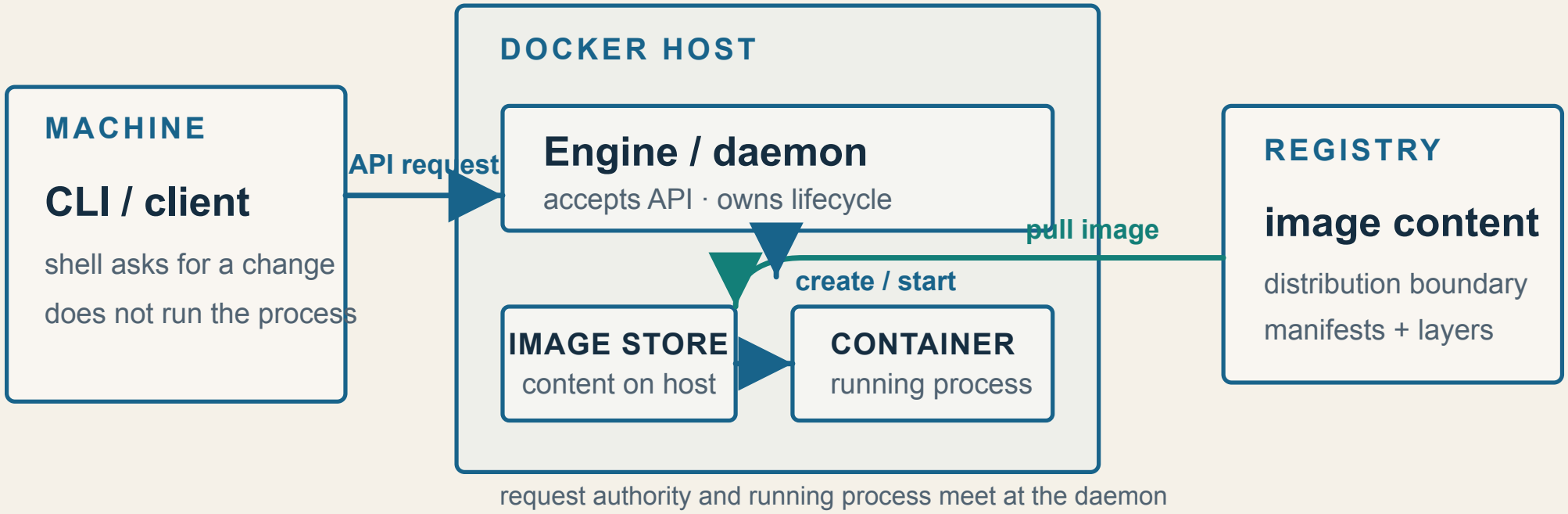
A container “boots its own kernel”?

A classmate says a container is just a small VM. Repair the explanation.

> It is an isolated process tree using the host kernel, with its own runtime configuration and filesystem view.

A useful mental model predicts behavior, not just vocabulary.

The daemon manages the runtime.



Causal trace: the machine sends an API request to the daemon; the registry supplies image content to the host image store; the daemon creates and starts the container process.

The registry distributes images; the host runs containers.

What does an orchestrator add?

SINGLE RUNTIME

Start this container.

ORCHESTRATION

Reconcile desired and observed state.

Kubernetes is a later handoff, not today’s operating environment.

One image. Independent instances.

COMMON INPUT

Image

Immutable content + configuration

TWO INDEPENDENT INSTANCES

Container A + writable layer

Container B + writable layer

Both containers come from the image; B is not created from A.

A name can move. Content has identity.

TAG

`web:release`

A movable reference.

DIGEST

`sha256:…`

Identifies content.

A digest fixes identity; it does not establish trust or correctness.

Which identity answers which question?

What artifact? What local image? What running instance?

> Registry digest → distributed content.
> Local image ID → local image object.
> Container ID → runtime instance.

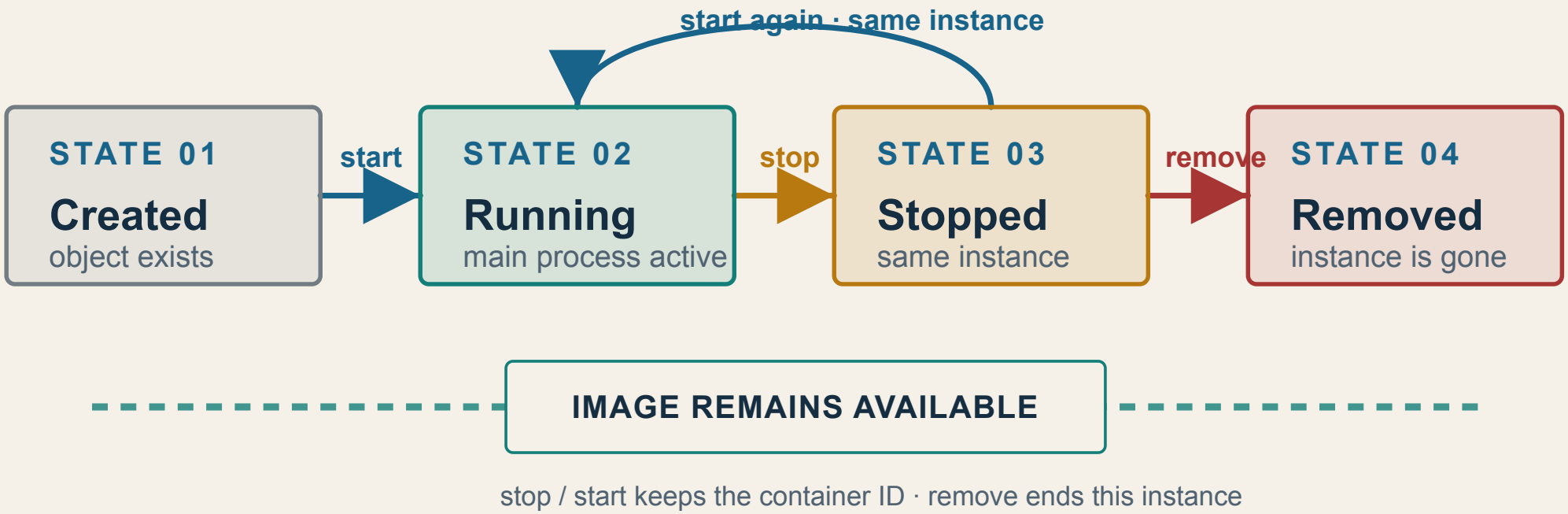
These identifiers answer different questions.

Ten minutes to reset.

Return with one question:
what changes when a container stops?

Resume with the lifecycle model.

Stop is not remove.



State model: create → start → stop → remove. Stop/start retains the container ID; removal ends that instance. The image remains available.

One instance can stop and start; a removed instance cannot restart.

What keeps the container alive?

The main process exits.
Should the container keep running?

> No. A completed main process ends this run. Inspect exit state and logs before deciding whether that completion is a failure.

“Stopped” is a state; the task’s contract determines success.

Observe the lifecycle, one boundary at a time.

REFERENCE TRANSCRIPT · not observed execution

```
fragment $ docker start "$DEMO_ID"
fragment $ docker stop --time 5 "$DEMO_ID"
Inspect: same container ID; state running → exited
```

Predict first. Run the guarded demonstration or use its reference transcript.

State and identity tell different stories.

STOP / START

Same instance.
New process run.

REMOVE / CREATE

New instance.
New writable layer.

Do not mistake a new process run for a new container object.

Publication creates a path, not health.

Host loopback

127.0.0.1:8080

→

Port mapping

host → container

→

Service process

container port 8080

The application must listen and respond on the expected port.

Two loopbacks. One authorized tunnel.

Laptop
local browser

→

SSH tunnel
authorized path

→

VPS loopback
published service

The laptop’s localhost and the VPS’s localhost are different places.

Running. Reachable. Healthy.

docker ps says “Up”.
What have you actually established?

> The container is running. A separate request tests reachability; an appropriate application check tests behavior.

Use one observation for one bounded claim.

Choose the next observation.

Question

what failed?

→

Inspect

configuration + state

→

Logs

bounded process output

→

Test

specific expected result

Collect the smallest observation that distinguishes your hypotheses.

Normal completion is not a crash.

REFERENCE TRANSCRIPT · not observed execution

```
reference $ inspect selected container
Status: exited
ExitCode: 0
```

Exit code zero can be expected for a completed one-shot task.

One symptom. Two plausible causes.

No response arrives.
Name two causes and one discriminating test.

> Example: no published path, or no listening application. Inspect bindings first; test the intended endpoint and examine bounded logs next.

An observed failure is a starting point for reasoning.

A file can outlive a process run.

STOP / START

Writable layer still belongs to A.

REMOVE A

A’s writable layer disappears.

This statement concerns the writable layer, not mounted persistent storage.

Replace the instance. Where is the file?

A writes /tmp/proof.
Remove A. Create B from the same image.
Does B inherit the file?

> No: the file was in A’s writable layer, not in the image or a persistent mount.

Same image does not mean the same mutable runtime state.

Reproducibility needs a definition.

Known image

content

→

Runtime definition

configuration

→

New instance

fresh writable layer

Recreate from a definition; do not rely on undocumented edits.

Make a claim someone else can test.

Define
versioned contract

→

Test
expected outcome

→

Observe
actual result

→

Recover
repeat the definition

A screenshot is supporting evidence, not a replayable definition.

“It is secure” is too broad.

Repair the claim: “The service is secure because it runs.”

> Narrow it: the inspected publication binds only to host loopback. Record the definition, local binding inspection, endpoint test and recovery path.

Bound the claim to what the evidence actually checks.

What would let a teammate disagree?

WEAK

“It worked for me.”

REPLAYABLE

Definition + expected result
Observed result + recovery

Good evidence makes a claim falsifiable and repeatable.

Explain the system without a command list.

Which machine?
Which object changes?
Which observation tests the claim?

> Name the context, predict the object transition, then choose the smallest test that could refute your prediction.

Use these three questions before typing in TD1.

Arrive ready to observe and explain.

Preflight
approved environment

→

Predict
state + identity

→

Run
TD1 exact steps

→

Record
your own evidence

TD1: Server Baseline and Container Lifecycle.

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

Resolve the first unclear boundary.

Context, object, or evidence?

Use the remaining time to revisit one explanation.

If nothing needs recovery, use the exit questions for a second retrieval pass.