CareerSwami · Cloud Computing & DevOps

FREE TWO-WEEK INTERNSHIP

Project 1 — Make a tiny service reproducible on a machine that is not yours

Three endpoints and about forty lines of code. The service is trivial on purpose. The project is whether a stranger can clone it, build it, and get a byte-identical result — and whether it can tell them when it is not healthy.

2 weeks	1 project	6–8 hrs
START TO SUBMIT	ONE SUBMISSION	PER WEEK

WHAT IS IN THIS BRIEF

1. The situation — the problem you are being handed
2. The four things that make a service reproducible — the cases your work must survive
3. What you are given — your data and starting files
4. What finished looks like — the two things you hand in
5. How the two weeks run — week by week
6. Before you start — rules, tools and your starter kit
7. How the work is judged — the marking scheme
8. How to submit — one link, pasted into your dashboard

Read 1 to 4 before you do anything. They take five minutes and they are what the marking scheme is built on. Sections 5 to 8 are the how — come back to them once you know what you are solving.

1 The situation

Treat this as a real client brief. The service is deliberately tiny. Everything being marked lives around it, which is also true of most real platform work.

You are building a status service with three endpoints. */health* returns ok and a version. */ready* reports whether the thing it depends on is actually available. */* displays the status in a readable page.

The dependency is simulated — a local file that may or may not be present, or a second small process. The point is that readiness has to reflect something outside the service, because a service that reports ready purely because its own process is alive tells an operator nothing.

What is actually being assessed is the difference between "it runs on my machine" and "it runs". Pinned dependency versions. Configuration that comes from the environment rather than the source. Logs that say something useful. No secrets committed. And evidence that it started cleanly from nothing, not just that it is currently running on your laptop.

WHAT YOU ARE BEING ASKED TO DO

Build the status service and the machinery around it so that someone with only your repository and your README can produce the same running service you have, and can tell from outside whether it is healthy.

2 The four things that make a service reproducible

Each of these is a thing that works on your machine and fails on a clean one. Your clean-start evidence has to show you actually checked, not that you believe it would be fine.

Unpinned dependencies

The same versions today and in six months.

Where it goes wrong — An unpinned requirements file installs whatever is newest on the day someone clones it. The build that worked in March fails in May for a reason nobody can see in the diff. Pin exact versions and commit the lock file.

Configuration in the source

Ports, paths and the dependency location come from the environment.

Where it goes wrong — A hard-coded path means the service only runs from your home directory. A hard-coded port means it cannot run twice. Read from environment variables with sane defaults, and ship a sample env file that contains no real values.

Health and readiness treated as the same thing

Health is "the process is alive". Readiness is "it can do its job".

Where it goes wrong — Make /ready return ok unconditionally and an orchestrator will send traffic to a service whose dependency is gone. The two endpoints exist precisely because they answer different questions, and your readiness check must genuinely probe the dependency on each call.

Logs written for you, not for an operator

Enough to diagnose a failure without attaching a debugger.

Where it goes wrong — Printing "here" and "got it" helps nobody at three in the morning. Log the event, the outcome, and the relevant identifier — and never log a secret, a token or a full request body.

The rollback requirement is the one people skip. Being able to go back to the previous version is what makes shipping safe; without it, every deployment is a one-way door.

3 What you are given

The endpoints and their contracts are given. The language, the build approach and how you simulate the dependency are yours.

The three endpoints

`/health → ok + version` `/ready → reflects dependency readiness` `/ → human-readable status`
`config from environment` `no secrets in source`

The readiness contract, worked, so it is unambiguous

`Dependency`	A local file at the path in DEPENDENCY_PATH
`File present`	GET /ready → 200 {"ready": true, "dependency": "ok"}
`File removed`	GET /ready → 503 {"ready": false, "dependency": "unreachable"}
`Meanwhile`	GET /health → 200 in both cases — the process is alive
`Logged`	one line per check: event, outcome, duration
`Never logged`	paths containing secrets, tokens, or full request bodies
`Evidence`	Capture all four responses, not a description of them

Four things that must be provable

`Builds clean` `Starts` `Reports health` `Rolls back`

No public deployment is required and deploying publicly is not worth extra credit. Keep every secret out of source control, including your commit history. Use placeholder values in the sample env file. Where the brief leaves a choice open, decide, document it, and be consistent.

4 What finished looks like

At the end of two weeks someone with only your repository should be able to produce a running, observable service without asking you anything.

- **The source, an architecture note and a local startup guide** — the three endpoints working; a short note explaining what the service is, what it depends on, and how readiness is determined; and a startup guide with exact commands that assumes nothing is already installed.
- **A build definition, pinned dependency versions and clean-start evidence** — a repeatable build (a Dockerfile, a Makefile or a documented script), every dependency pinned to an exact version, and recorded proof of a start from a clean state: the commands run, the output produced, and the health and readiness responses captured.

"Clean start" means from an empty directory or a fresh container, not from your working folder with the virtual environment already built. That distinction is the entire point.

5 How the two weeks run

Work through it in two parts — but you submit **once**, at the end, not once per week.

PART 1 · WEEK 1

Build the service and say what it depends on

Write the three endpoints, simulate the dependency, and document the architecture and local startup clearly enough that someone else could follow it.

Produce

- Source for /health, /ready and /, with the dependency simulated
- An architecture note: what it is, what it depends on, how readiness is decided
- A local startup guide with exact commands

> Readiness must actually probe the dependency on every request. Remove the dependency file and /ready must change — show that it does, with both responses captured.

PART 2 · WEEK 2

Make it reproducible, then prove it from clean

Pin everything, move configuration into the environment, define the build, and then run it from a genuinely clean state and record what happened.

Produce

- A build definition that someone else can run
- Pinned dependency versions, with a lock file committed
- Clean-start evidence: commands, output, and captured endpoint responses

> No secrets in the repository — check your history, not just the current files. Your sample env file must contain placeholders only. The clean start must be from an empty directory or a fresh container.

6 Before you start

- Pin exact versions and commit the lock file. "Latest" is not a version.
- Health and readiness answer different questions. Do not make one an alias of the other.
- Configuration comes from the environment with documented defaults. A hard-coded absolute path is an automatic mark down.

- Clean-start evidence must be from an empty directory or a fresh container, with the commands and output recorded.
- Describe your rollback procedure concretely — which version you would return to and how. A sentence saying you would roll back is not a procedure.

Your tools

Python or JavaScript, whichever you know better. Docker is useful but not required — a documented script that provably works from clean is equally acceptable. No cloud account, no paid tool and no public deployment is required, and no ID document is ever required.

CS_cloud-computing-devops_Starter_Kit.zip — sample.env, an architecture note template, a clean-start checklist, a runbook skeleton, an evidence log, and a file index.
https://careerswami.com/wp-content/uploads/careerswami-projects/CS_cloud-computing-devops_Starter_Kit.zip

7 How the work is judged

WHAT IS ASSESSED	WEIGHT
Problem and requirements — did you understand what you were solving	20%
The core artefact — the thing you actually made	40%
Validation and evidence — how you know it works	25%
Clarity and reproducibility — can someone else open and follow it	15%

Strong	Complete, correct, evidence-backed and easy to inspect.
Developing	A useful partial result, with the gaps explained.
Needs revision	Core deliverable missing, claims unverified, or the work cannot be opened.

Credit reflects demonstrated work, not visual polish. A small, honest result beats a large, unsupported one.

What counts as evidence

expected result → actual result → the captured terminal output, log line or HTTP response → conclusion. Then say what the evidence does not establish.

Why this one is worth doing. Platform roles do not hire for the forty lines of service code; they hire for the discipline around it. The interview question is some version of "it works for you and not for me — what do we look at?" A repository with pinned versions, environment-driven configuration, a readiness check that genuinely probes, and recorded clean-start output answers it with artefacts instead of opinions.

8 How to submit

One link, pasted into your dashboard

1. Put your work somewhere with a link — Google Drive, Figma, GitHub, Notion, Loom. Whatever fits the work.
2. **Set it so anyone with the link can view.** A locked link is the single most common reason a submission comes back.
3. Include a short README or cover page saying what to open first, and where each deliverable lives.
4. Open your dashboard, paste the link into the submit box, and add a line or two on anything you want the reviewer to know.

Name files `CS_cloud-computing-devops_YourName_v1`. Keep your own copy. A screenshot on its own is not enough for anything interactive. A video is optional — annotated screenshots or narrated slides are fine.

Submitted and heard nothing, or the link came back? Email **support@careerswami.com** with the email address you enrolled with.

Work honestly. Use fictional data and your own work. Credit anything you reused, and disclose AI assistance — including what you checked yourself. Never invent interviews, users, tests or results. Nothing here requires you to spend money, contact real organisations, or deploy anything publicly. Never upload identity documents, real customer records or credentials.

Stuck, confused, or something is broken? Write to us.

`support@careerswami.com`

Or WhatsApp **+91 7383138418**. We answer on working days. Use it for anything at all — a part of this brief that does not make sense, a starter file that will not open, a dashboard link that has stopped working, a deadline you need moved, or a submission you are not sure about.

To get a useful answer fast, spend twenty minutes narrowing the problem down first, then tell us four things: what you expected, what actually happened, what you already tried, and your one specific question. Include your track name and the email you enrolled with.

Asking is not penalised. Submitting partial work with a clearly stated blocker always scores better than a finished answer you copied, and better than a silent no-show.

Optional stretch

Optional, and only once the main work is done — cap it at an hour. Add a basic pipeline that builds and checks on every commit, or structured JSON logging with a request identifier. Label it clearly as a stretch.