Taking Ownership:

Cloud Migration

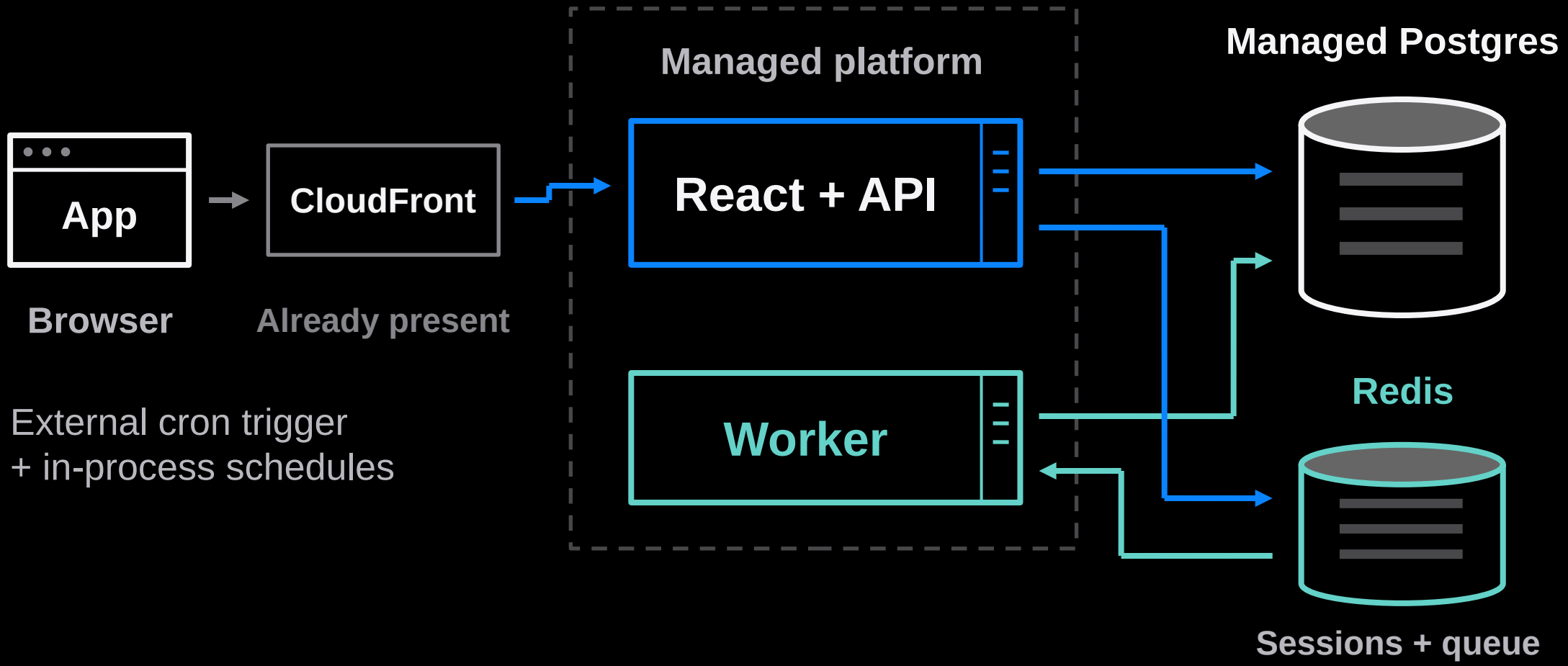
Building and operating a live platform across three teams

Isaac Rowaiye

IR Technical Solutions

SREday Dallas 2026

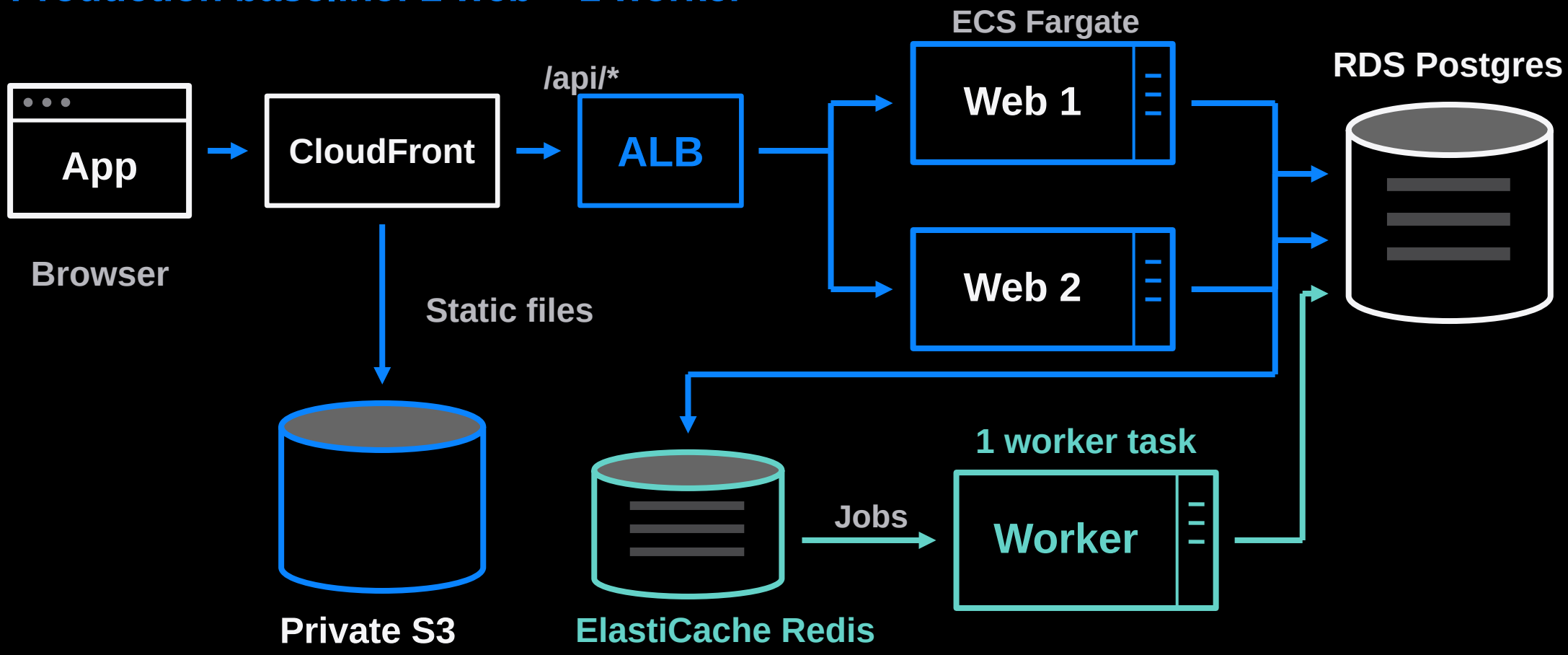
The original architecture



Existing application roles and a separate HTTPS data feed.

Mapping the move to AWS

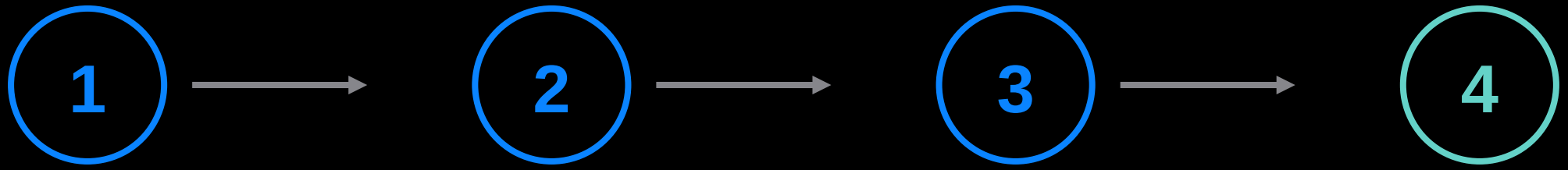
Production baseline: 2 web + 1 worker



Kept: application behavior, Postgres, Redis and HTTPS integrations.

Changed: hosting, network access, scaling and release ownership.

The migration sequence



Map

Dependencies
+ owners

Build

Parallel AWS
environments

Validate

Deploy, restore
+ test paths

Cut over

Stop writes
+ final copy

Verify destination. Redirect traffic. Keep operating.

GitHub Actions: image, migration, rollout, stabilization, endpoint check

What changed, what stayed

STAYED

Web + worker responsibilities

Postgres + Redis

API + session behavior

Separate HTTPS data feed

CHANGED

React to S3 + CloudFront

Private network + access paths

2 web tasks + 1 worker

Infrastructure + release owners

Three teams. The boundaries moved with the platform.

The restore that did nothing

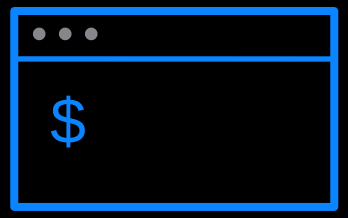
Image ENTRYPOINT `["/bin/bash", "-c"]`

ECS override `["sh", "-c", "<restore script>"]`

COMBINED INVOCATION

```
/bin/bash -c sh -c "<restore script>"
```

Positional arguments. Never executed.



Empty shell

Exit 0

Exit 0. Database unchanged.

GitLab: when recovery failed

2017. Primary deleted during replica recovery.



Replica

Already cleared



Database dumps

Wrong client version



Failure alerts

Emails rejected

Recovery: a manual snapshot, about 6 hours old.

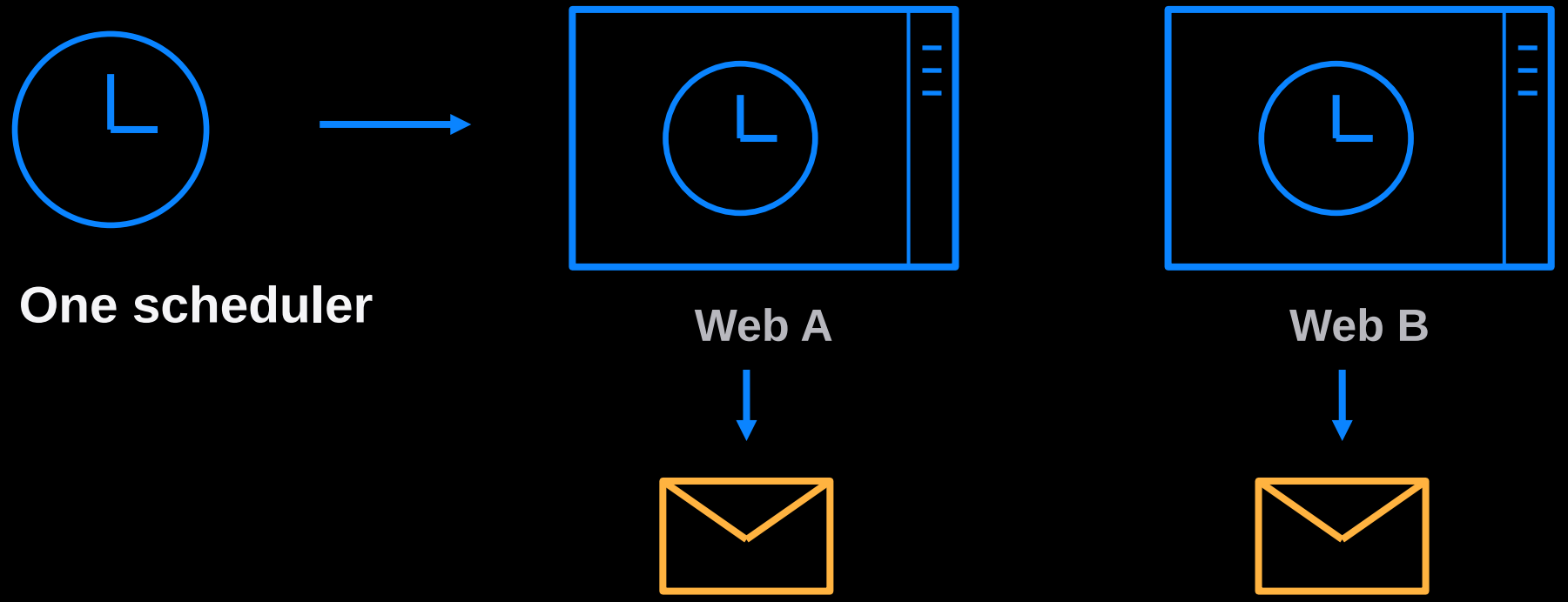
About 18 hours offline. Several hours of database changes lost.

Source: GitLab public postmortem, February 10, 2017

Two owners, one assumption

APP TEAM

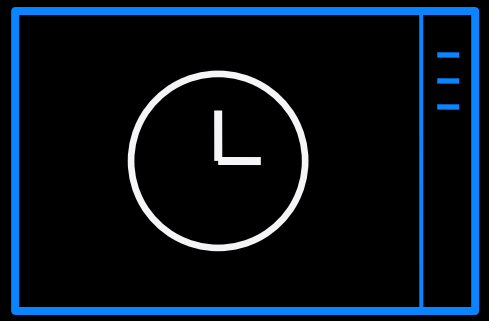
INFRA: TWO WEB CONTAINERS



The scheduler scaled with the app.

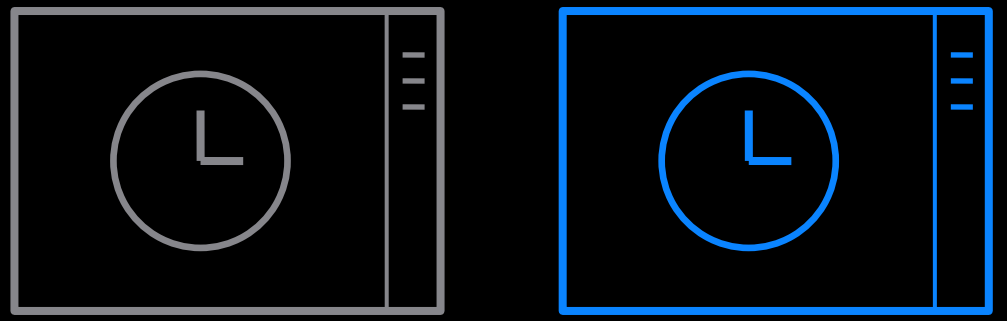
Once per what?

STEADY STATE



One running copy

ROLLING DEPLOY



Old

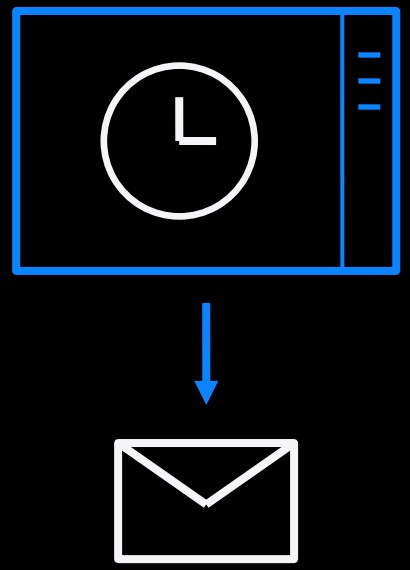
New

Both alive at once

One per container is not one across the service.

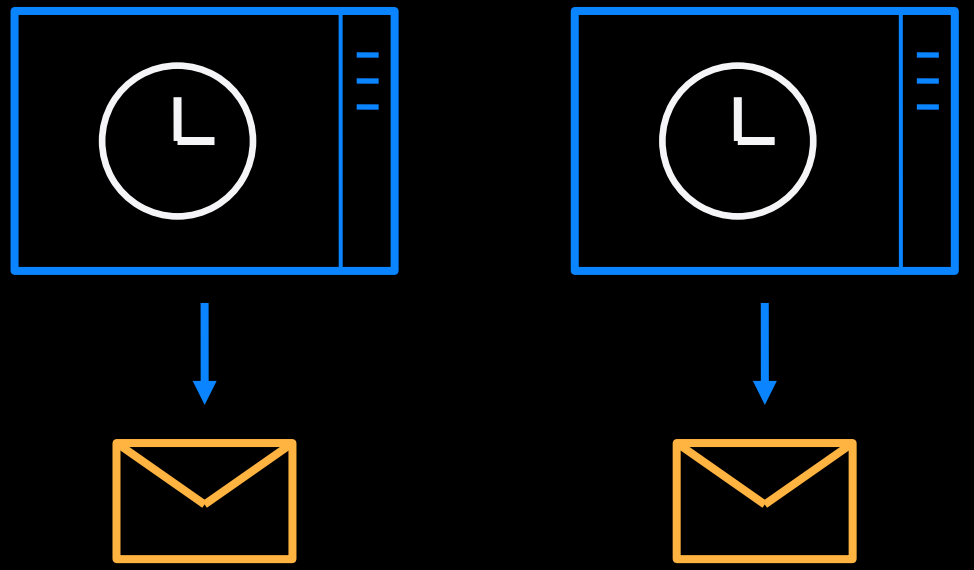
The clue

TEST ENVIRONMENT



One copy. One email.

PRODUCTION



Two copies. Same email, twice.

Same code. Different replica count.

Two writers, one field

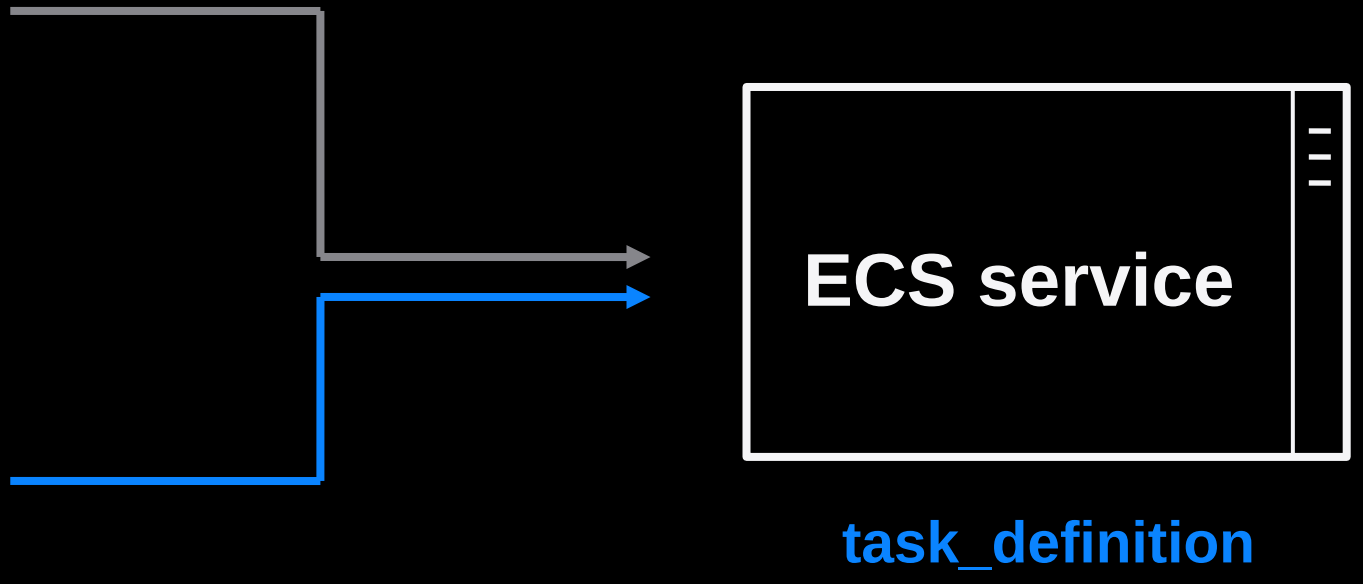
Community option or custom rule. Same ownership decision.

Terraform

Configured revision

Deploy pipeline

New release revision



One owner for the deployed revision.

Staging. The plan as I read it.

```
  # aws_iam_access_key.smtp will be created
  # aws_iam_user.smtp will be created
  # aws_iam_user_policy.smtp will be created
  # module.app.aws_ecs_service.web will be updated in-place
  # module.app.aws_ecs_service.worker will be updated in-place
  # module.app.aws_secretsmanager_secret_version.app must be replaced

Plan: 4 to add, 2 to change, 1 to destroy.
```

Who approves this plan?

Production plan. Stopped before apply.

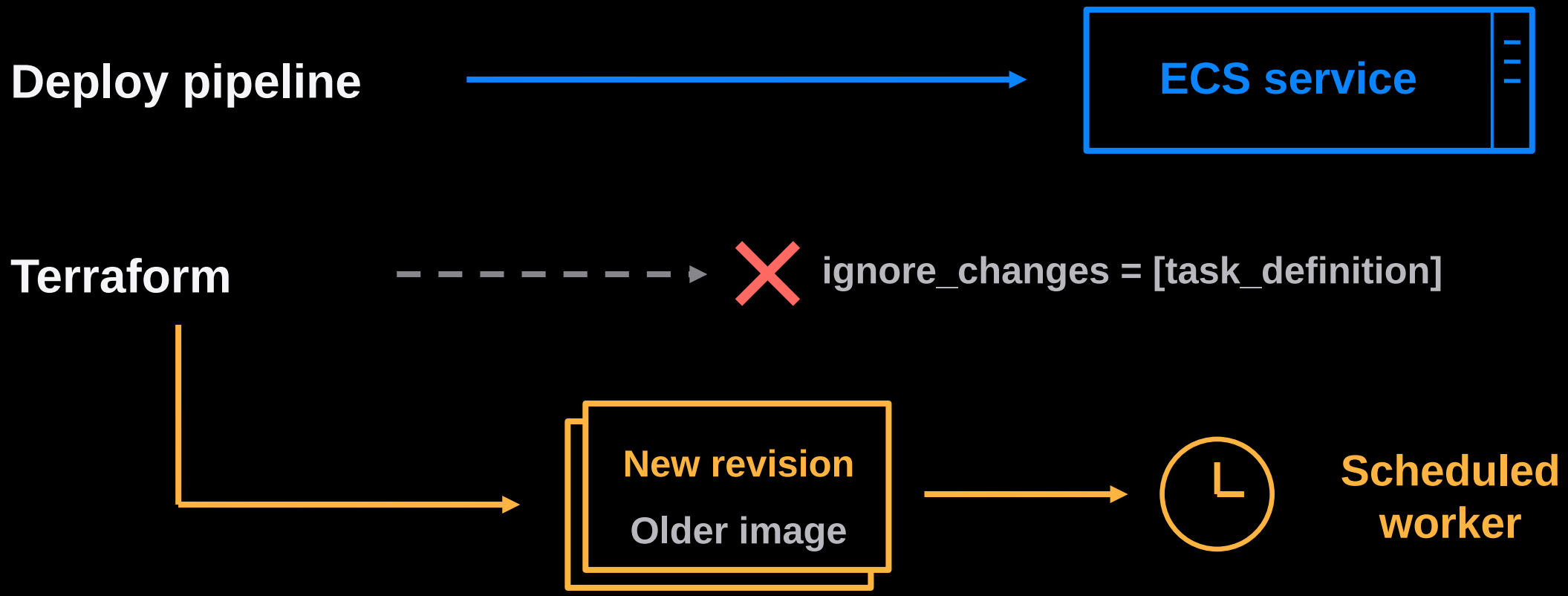
```
  # module.app.aws_ecs_service.web will be updated in-place
  ~ resource "aws_ecs_service" "web" {
        name            = "app-web"
      ~ task_definition = ".../task-definition/app-web:24" -> ".../task-definition/app-web:23"
        # (16 unchanged attributes hidden)
    }

  # module.app.aws_ecs_service.worker will be updated in-place

Plan: 0 to add, 2 to change, 2 to destroy.
```

24 → **23** **Production apply stopped.**

The guard, and the gap behind it



Check every launch path that can select a revision.

The three rules I kept

A check you've never seen go red is decoration.

Every gate ships with the test that fails it.

Replica count is a contract between two teams.

Test at production's count, and ask the other side what assumes there's only one.

Two writers on one field: the last one to run wins.

One owner, in code, and an alarm when what's running isn't what the owner shipped.

If you haven't broken it, you haven't tested it.

Let's go break something.

Isaac Rowaiye

IR Technical Solutions