Red Hat

Faster, Cheaper, Leaner:

Horizontally Scaling a CI Pipeline

Yorgos Saslis, Software Delivery Engineer

Michal Cichra, Principal Software Engineer

3scale BY RED HAT

CI is a production workload

Customers?

YOU!

Yes, you!

Maintain Flow

CI should sustain flow. Not get in its way.

where are your manners?

yorgos saslis

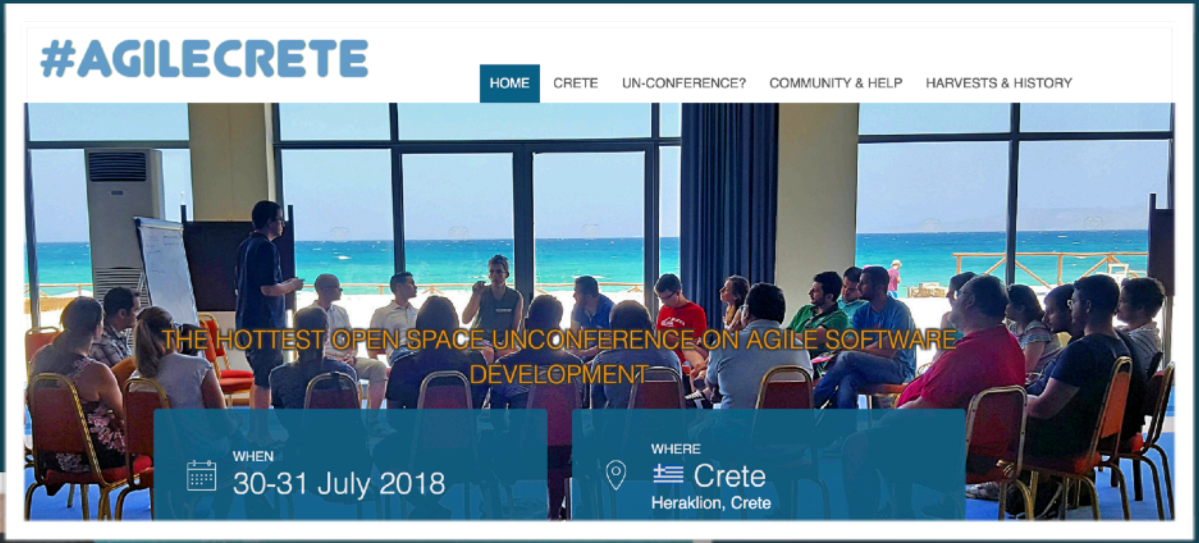
♥

Community

Automation

OSS

Maintainability

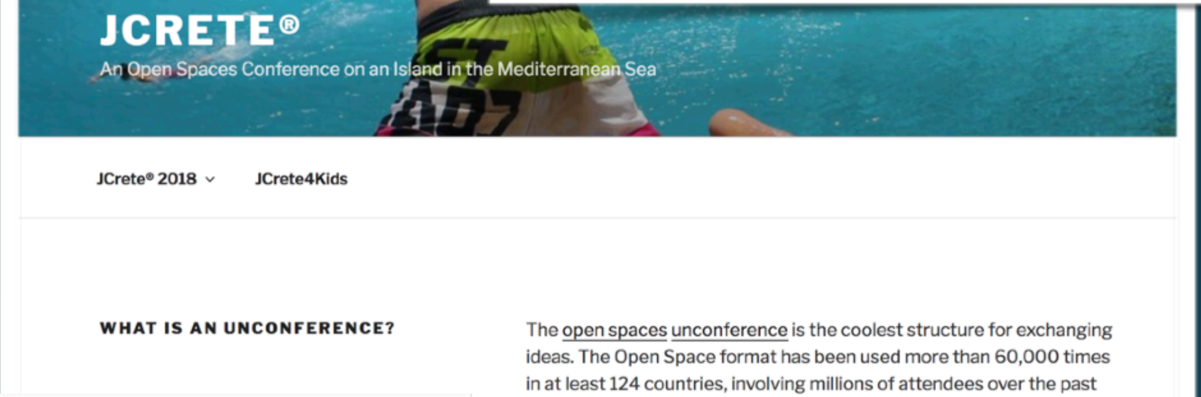


#AGILECRETE

THE HOTTEST OPEN SPACE UNCONFERENCE ON AGILE SOFTWARE DEVELOPMENT

WHEN
30-31 July 2018

WHERE
Crete
Heraklion, Crete

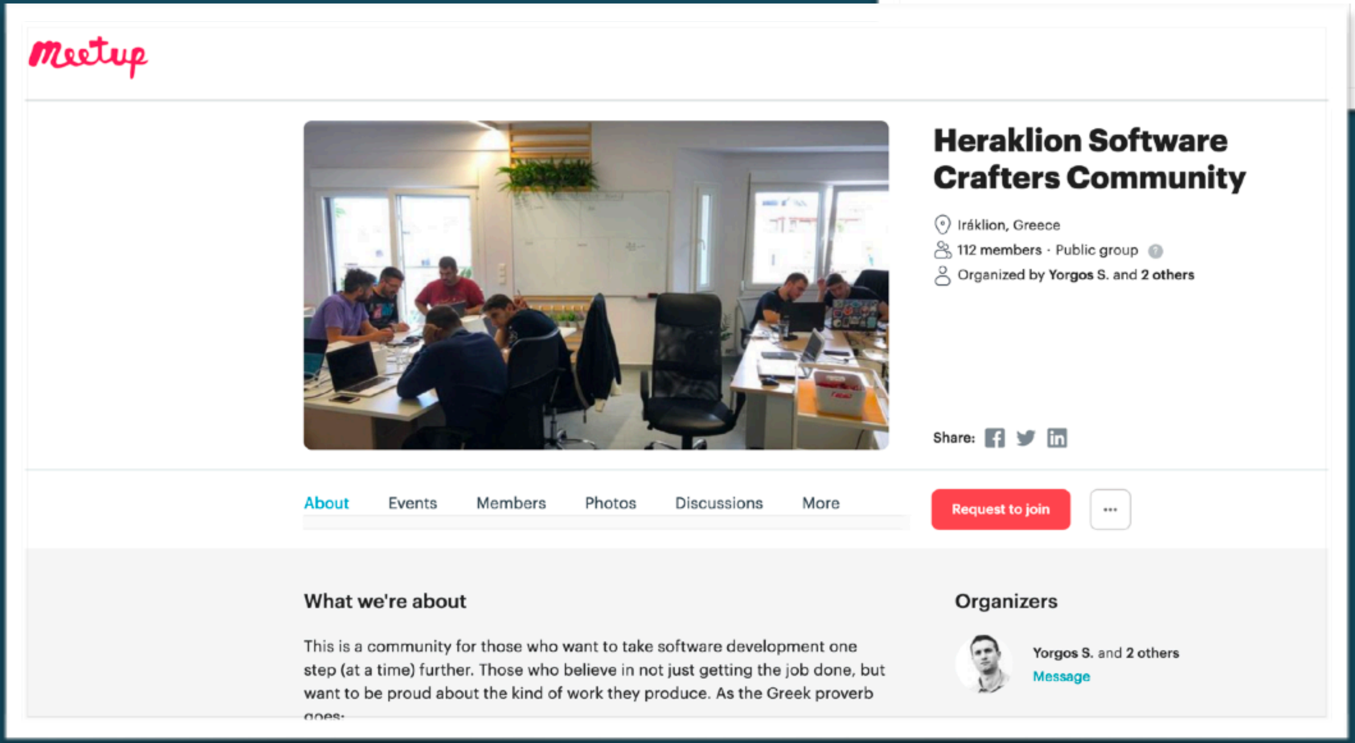


JCRETE®

An Open Spaces Conference on an Island in the Mediterranean Sea

WHAT IS AN UNCONFERENCE?

The open spaces unconference is the coolest structure for exchanging ideas. The Open Space format has been used more than 60,000 times in at least 124 countries, involving millions of attendees over the past 20 years, so it obviously "just works". As JCrete® uses it, it is important to understand what type of event an open spaces unconference is.



Meetup

Heraklion Software Crafters Community

Iráklion, Greece

112 members · Public group

Organized by Yorgos S. and 2 others

What we're about

This is a community for those who want to take software development one step (at a time) further. Those who believe in not just getting the job done, but want to be proud about the kind of work they produce. As the Greek proverb goes:

Organizers

Yorgos S. and 2 others

Message

Red Hat

3scale

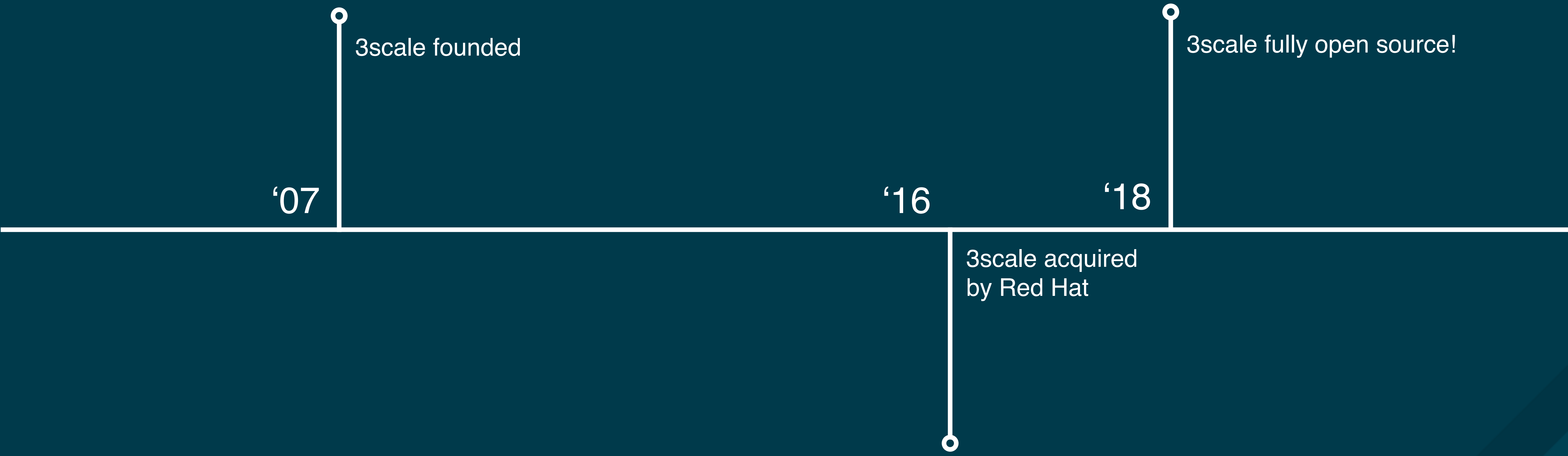
BY RED HAT®

Open Source
API Management

a bit of history...

3scale Timeline

Important milestones



‘07

3scale founded

‘16

3scale acquired by Red Hat

‘18

3scale fully open source!

Open Source projects need CI

All projects need CI. OSS projects need it more!

Hmmm interesting project…

But I just need this extra feature!!

Maybe I can open a pull request…

But how will I know I didn’t break anything with my PR ?

Aha!!

There are a bunch of checks on every PR that will protect me!

Contributing can be daunting

- Daunting task
  - especially for new contributors
  - CI helps lower the barrier-to-entry

What does
CI
for a closed source project
look like?

Red Hat

Single Jenkins Master

EC2 Cloud plugin for provisioning workers

Jenkins master provisioning automated through Makefiles + terraform

SCM Sync plugin used to persist jenkins configuration “as code”, in a github repository.

Job DSL for jenkins jobs in another github repository.

“HA” not so necessary…

Other Important Figures

Get the whole idea

5 person team

(main component)

10-20

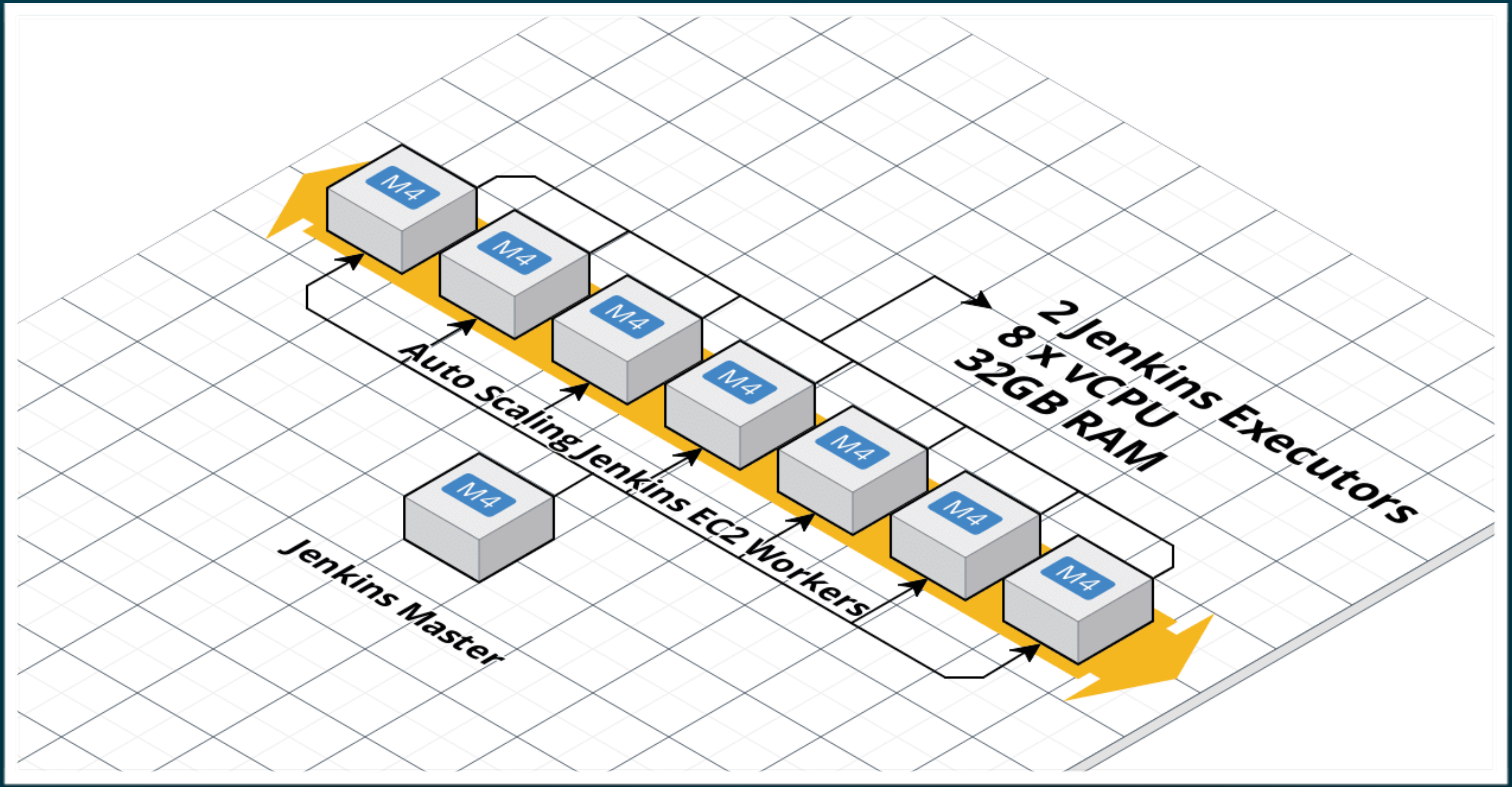
builds per day

2-3

Open PRs per day

Jenkins Worker Nodes

Auto-scaling (both up and down to reduce costs when not used)



Build Time

For “warm” build

~15 minutes

~11 hours CPU time

45 vCPUs
90GB RAM

How do we fit
11 hours
into…
15 minutes?

Bending Space-Time

NOT

(yet…)

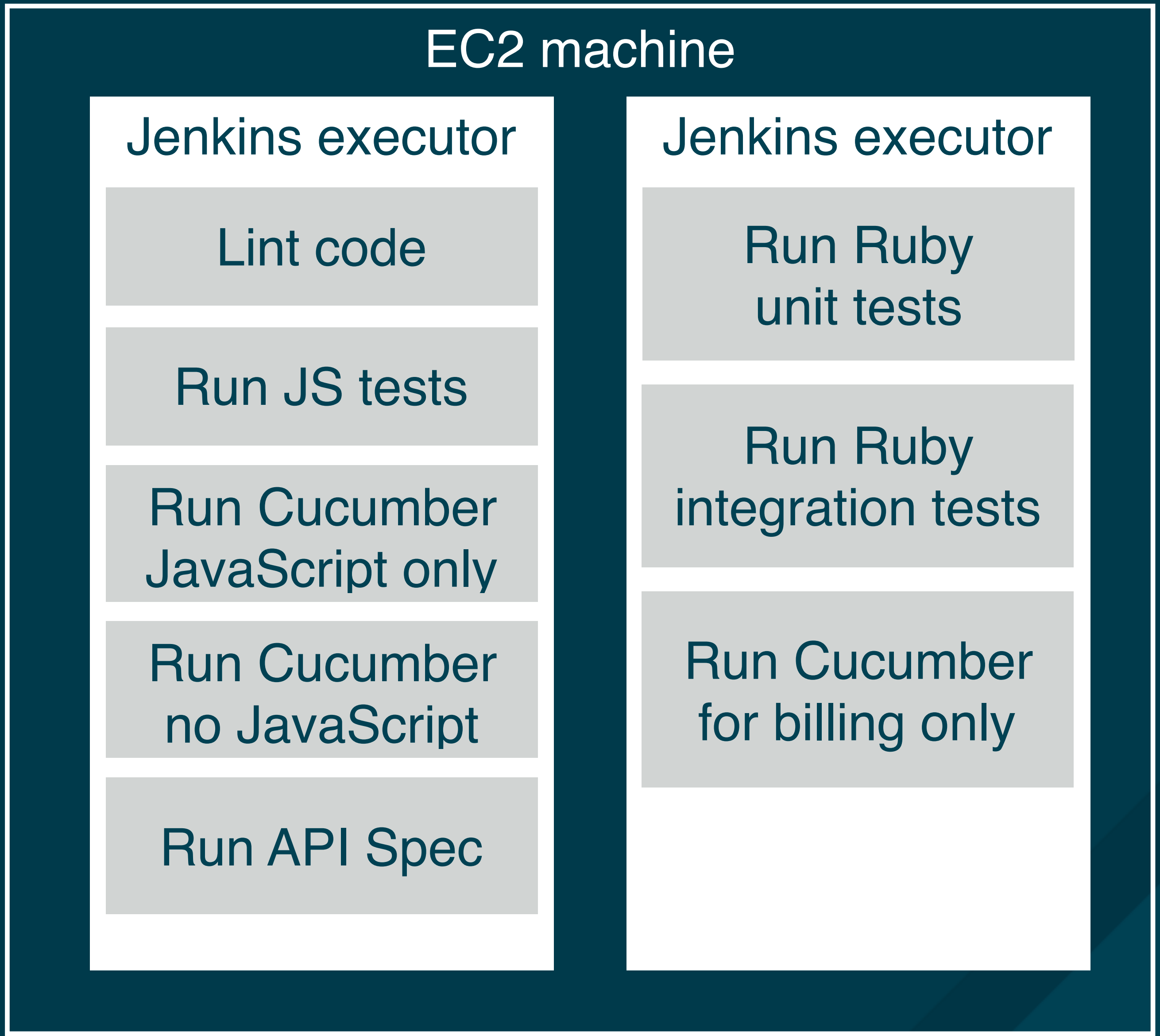
Test suite Parallelization

Homegrown parallelization

15 tasks
manually split

6 executors
for one build

4 languages
to understand
(Groovy, Ruby, Shell, Make)


EC2 machine
Jenkins executor
Lint code
Run JS tests
Run Cucumber JavaScript only
Run Cucumber no JavaScript
Run API Spec
Jenkins executor
Run Ruby unit tests
Run Ruby integration tests
Run Cucumber for billing only

Test Parallelisation

Parallelising Test Execution

Separate **test phases**

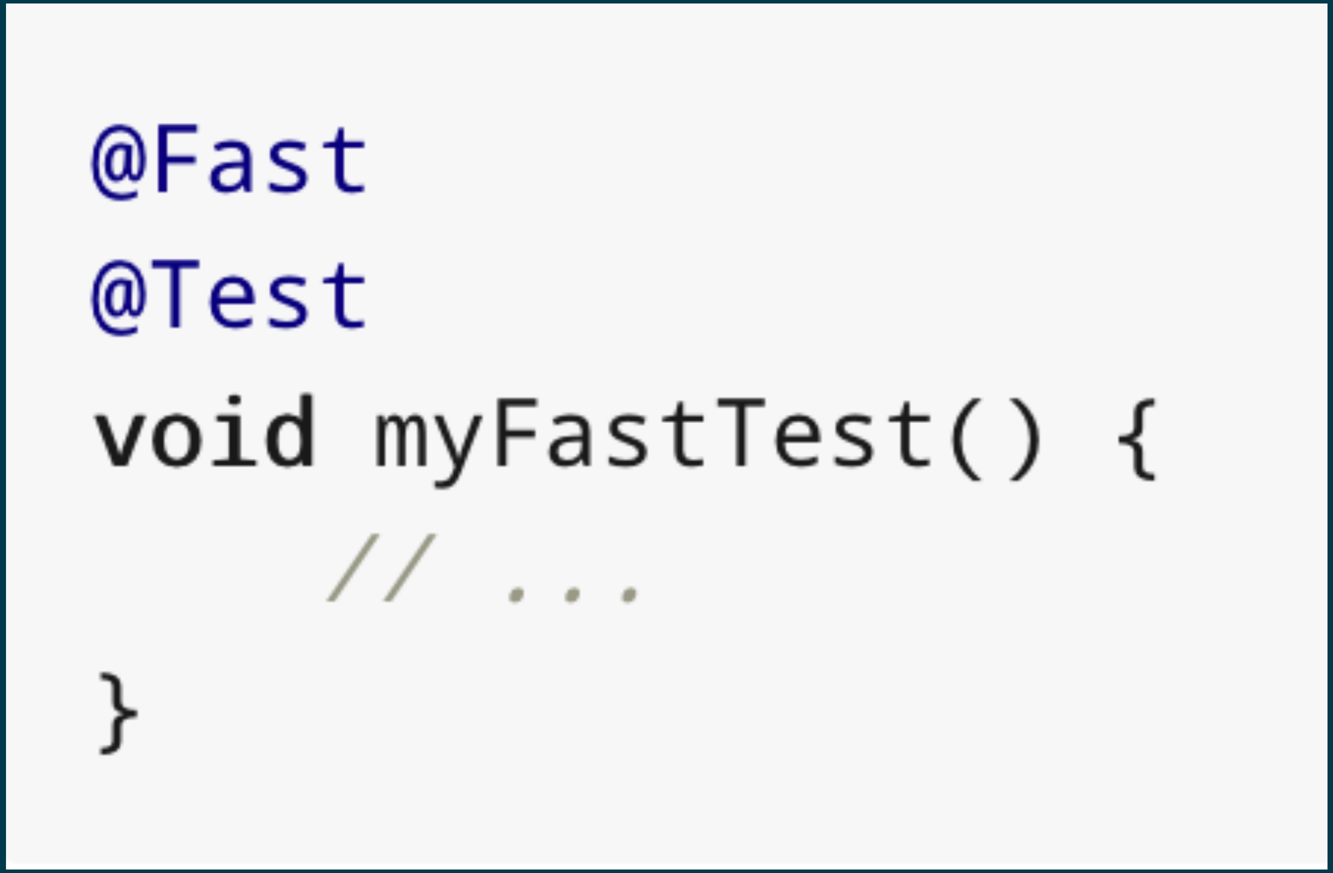
vs.

tests within phase

Separate Test Phases

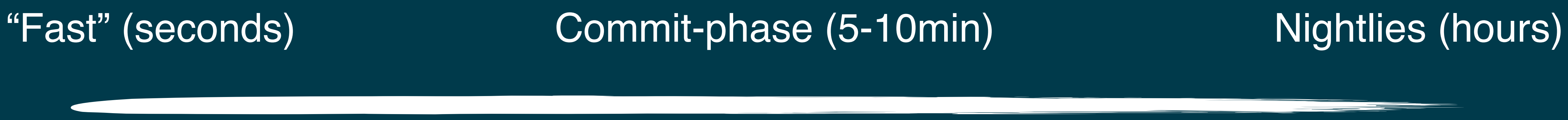
Usually depending on:

- how long tests take
- what environment they run in (how expensive)



```
@Fast
@Test
void myFastTest() {
    // ...
}
```

Separate Test Phases



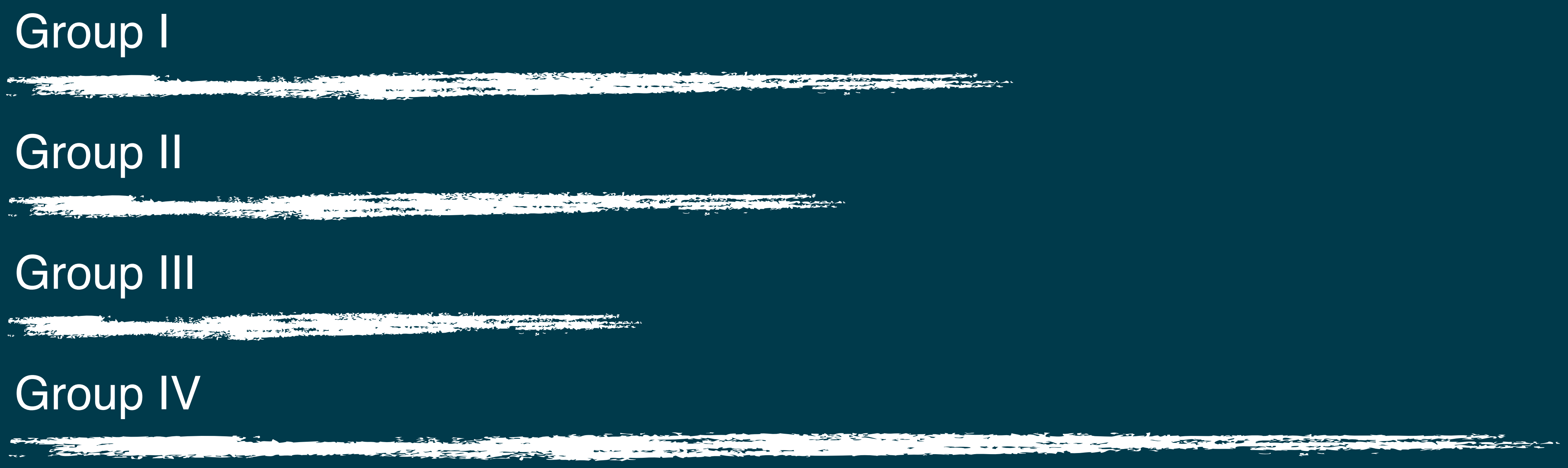
“Fast” (seconds)

Commit-phase (5-10min)

Nightlies (hours)

Parts of same test phase, in parallel

Multiple processes responsible for each running some part of the test suite, aggregating results at the end



Group I

Group II

Group III

Group IV

…but how to group, such that they all end at the same time?

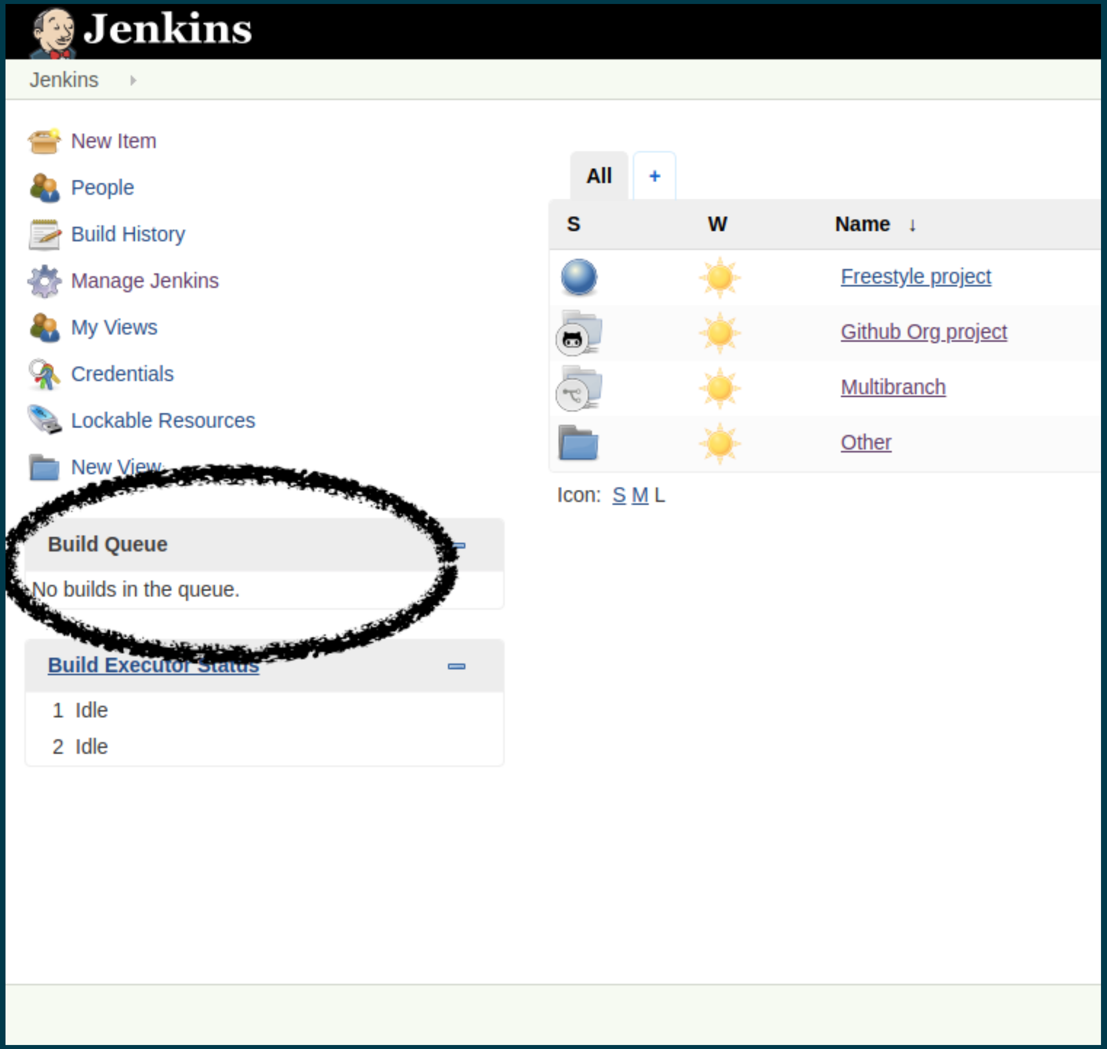
One of those rare moments in life when...

WE CAN FIX SOME TECH DEBT!

Problem 1: Build Queues

Almost never empty.

(during working hours)



Build Queue

No builds in the queue.

Build Executor Status

1 Idle

2 Idle

S	W	Name ↓
●	☀	Freestyle project
	☀	Github Org project
	☀	Multibranch
	☀	Other

Jenkins AWS Plugin did spin up new nodes, but:

- new worker nodes took ~5 minutes just to be provisioned (EC2 + user-data)
- max 7 EC2 instances (4xlarge)
- one build took up several EC2 instances
- Jenkins EC2 cloud plugin scaled up by one at a time
- Typical for cold builds to take > 30 mins

Problem 2: Random test failures

False positives

At least ONE failure per day, not related to actual changes made.

Overcome by always rerunning FULL pipeline on failure.

2-3 runs necessary for build to pass some times.

BAD for team confidence in test suite.

MORE delays…

Problem 3: Jenkins maintenance

Devs are expensive. Devs rely on CI. Therefore, CI is a prod system.

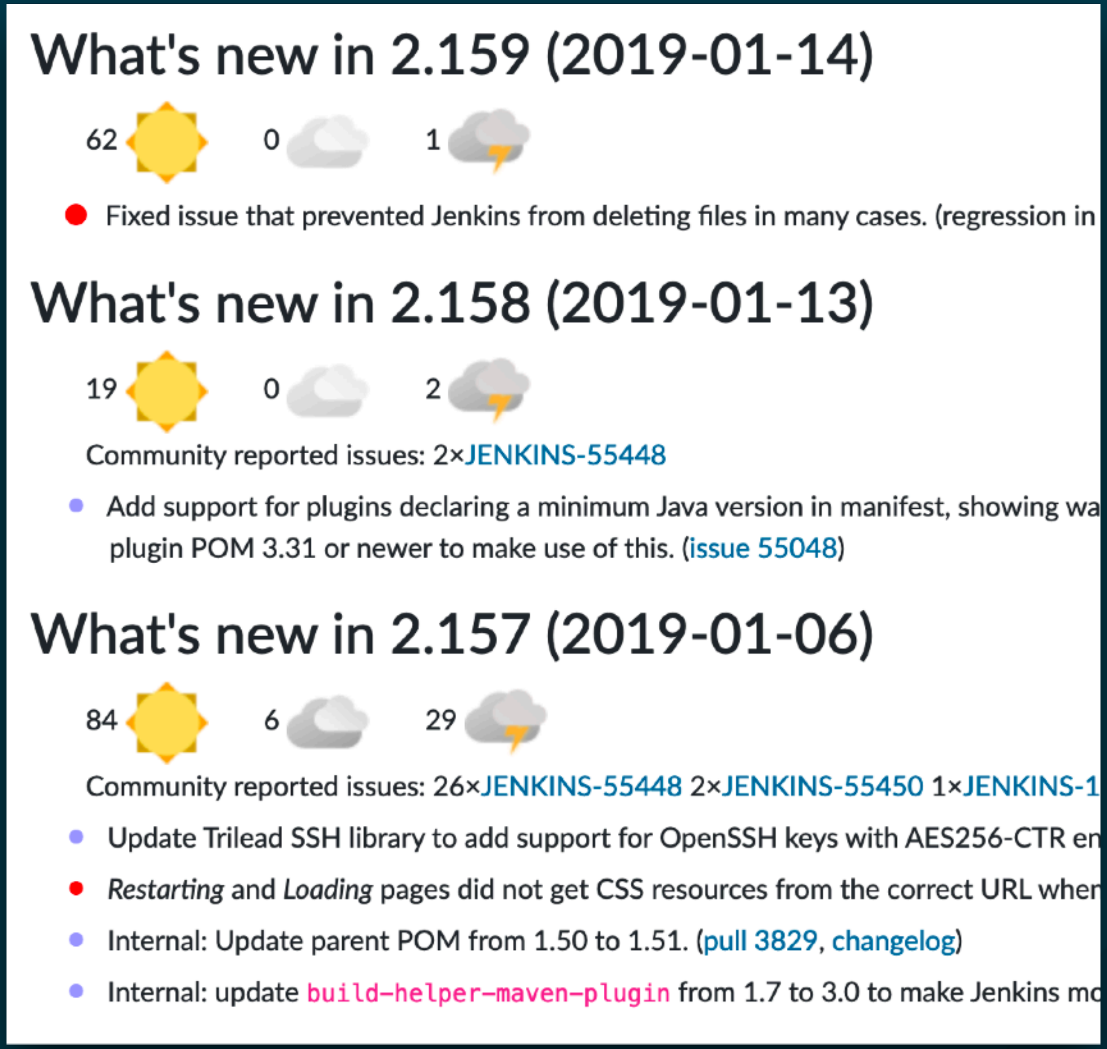
Hosting own CI is like hosting any other production system.

You need to maintain it, test before making changes to it and ensure it is up and running.

Any degradation of the service can block the whole team including production deploys.

Preparing staging environment for verifying any Jenkins core or plugin updates can cost a lot of time.

It felt like security updates happen almost weekly.



What's new in 2.159 (2019-01-14)

62 ☀ 0 ☁ 1 ⛈

- ● Fixed issue that prevented Jenkins from deleting files in many cases. (regression in

What's new in 2.158 (2019-01-13)

19 ☀ 0 ☁ 2 ⛈

Community reported issues: 2×JENKINS-55448

- ● Add support for plugins declaring a minimum Java version in manifest, showing wa plugin POM 3.31 or newer to make use of this. (issue 55048)

What's new in 2.157 (2019-01-06)

84 ☀ 6 ☁ 29 ⛈

Community reported issues: 26×JENKINS-55448 2×JENKINS-55450 1×JENKINS-1

- ● Update Trilead SSH library to add support for OpenSSH keys with AES256-CTR en
- ● *Restarting* and *Loading* pages did not get CSS resources from the correct URL wher
- ● Internal: Update parent POM from 1.50 to 1.51. (pull 3829, changelog)
- ● Internal: update `build-helper-maven-plugin` from 1.7 to 3.0 to make Jenkins mo

Problem 4: AWS Costs

Growing concern, especially as team was expected to grow

~2.5K EUR / month (just for AWS)

Total Costs =
AWS Costs +
Maintenance costs +
Dev team slow-down

Choosing our CI

Concerns

Our shopping list

Created by Alice Design
from Noun Project

Publicly accessible build information
external contributors should be able to see if their build failed and why!

Builds from 3scale team as fast as possible
(willing to pay for that)

Builds from forks should be possible
but not billed on Red Hat (abuse cases in the past)

Upstream CI options

We need to give contributors an easy way to run the test suite

Red Hat

OR

Public CI

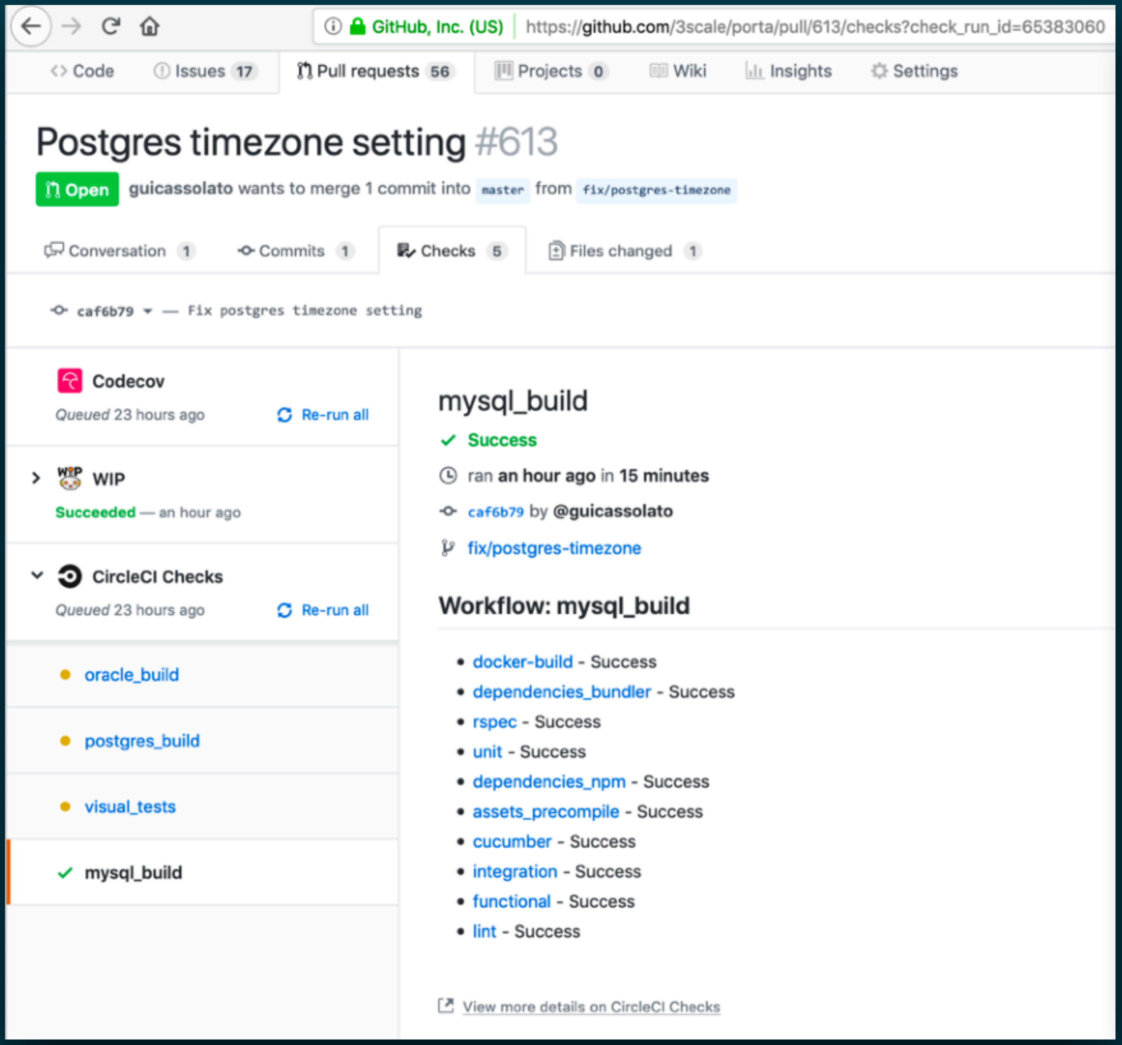
OR

Hybrid

It’s all about Developer Experience…

Public Build Info - Smooth DX

No account needed to access build information.



Postgres timezone setting #613

Open guicassolato wants to merge 1 commit into `master` from `fix/postgres-timezone`

`caf6b79` ▾ — `Fix postgres timezone setting`

Codecov
Queued 23 hours ago — Re-run all

WIP
Succeeded — an hour ago

CircleCI Checks
Queued 23 hours ago — Re-run all

- oracle_build
- postgres_build
- visual_tests
- ✓ mysql_build

mysql_build

✓ Success

ran **an hour ago** in **15 minutes**

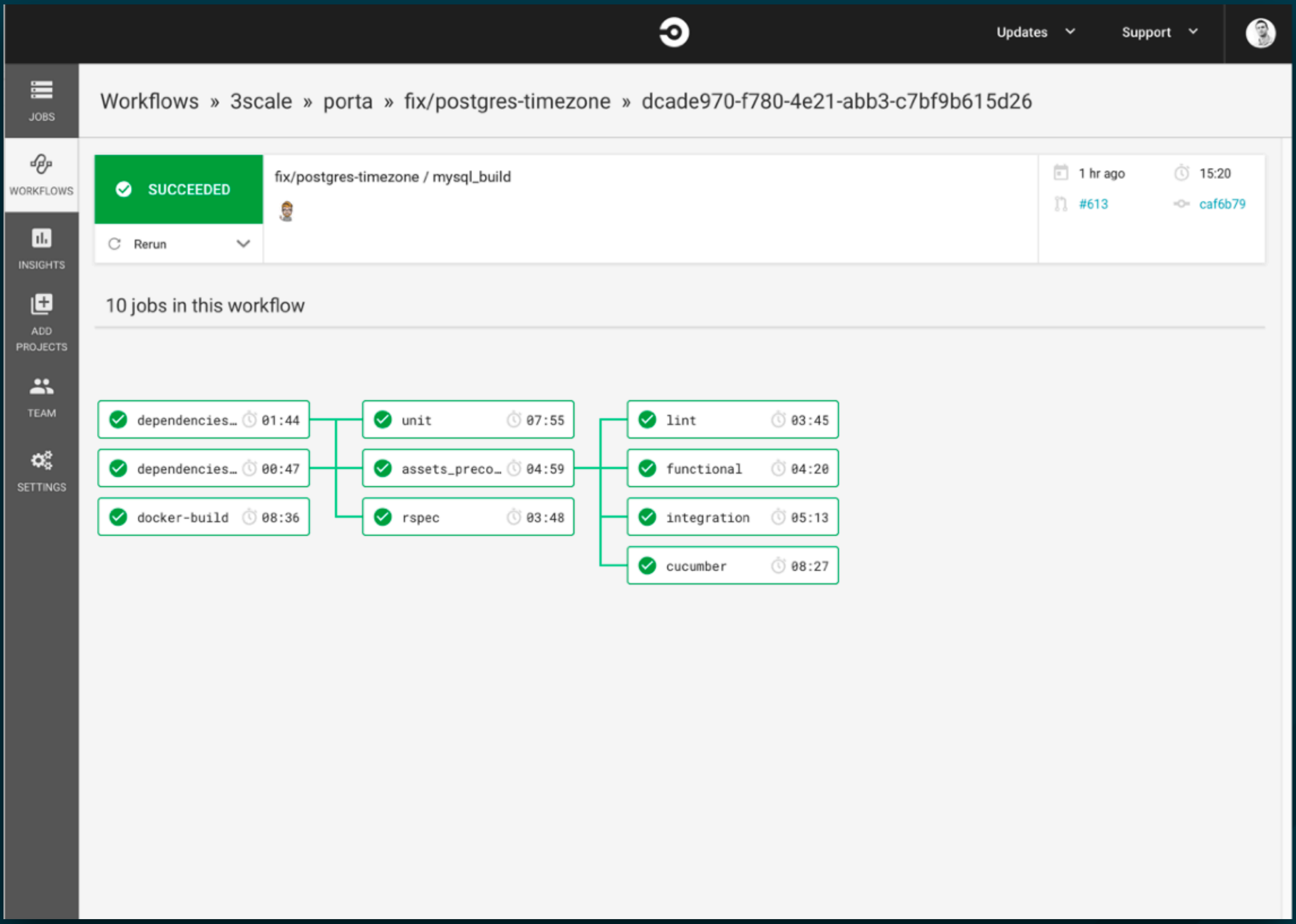
`caf6b79` by **@guicassolato**

fix/postgres-timezone

Workflow: mysql_build

- docker-build - Success
- dependencies_bundler - Success
- rspec - Success
- unit - Success
- dependencies_npm - Success
- assets_precompile - Success
- cucumber - Success
- integration - Success
- functional - Success
- lint - Success

View more details on CircleCI Checks



Workflows » 3scale » porta » fix/postgres-timezone » dcade970-f780-4e21-abb3-c7bf9b615d26

fix/postgres-timezone / mysql_build

1 hr ago — 15:20 — #613 — caf6b79

10 jobs in this workflow

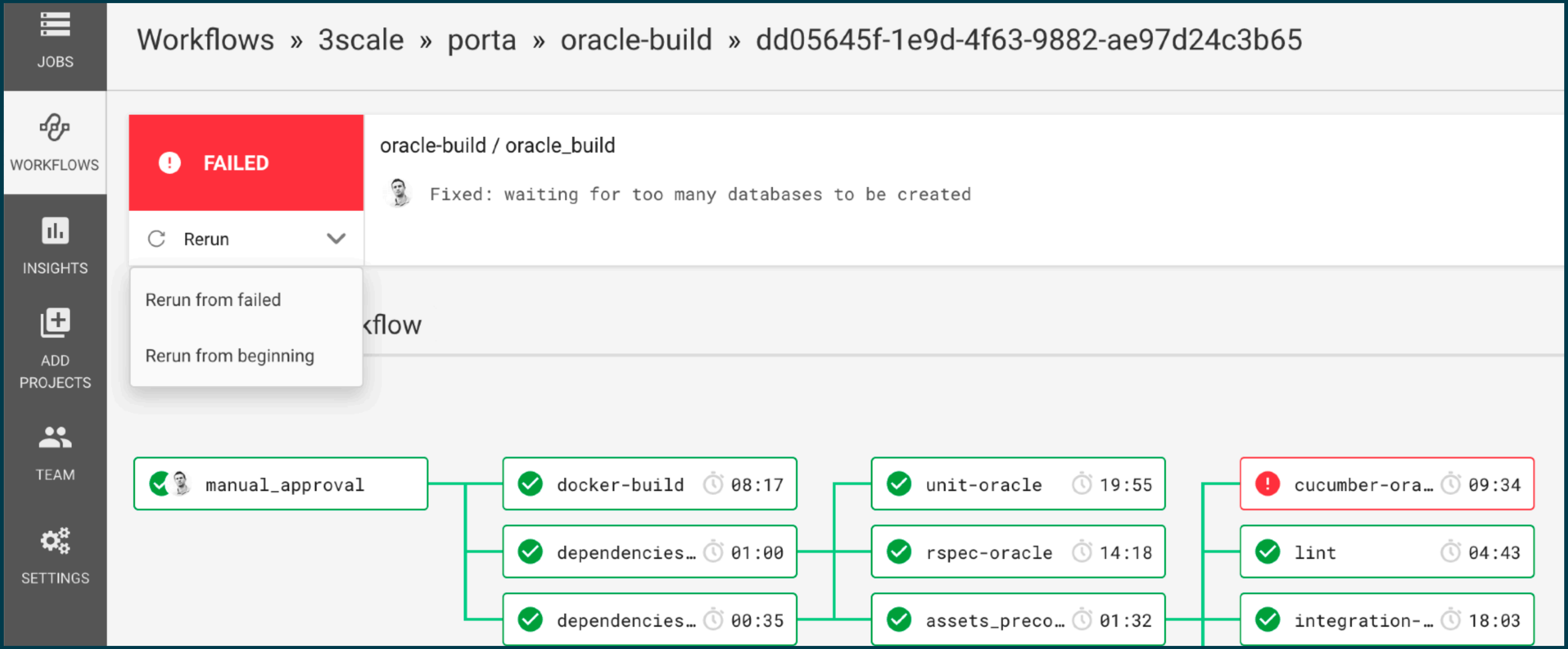
```
✓ dependencies… 01:44    ✓ unit           07:55    ✓ lint          03:45
✓ dependencies… 00:47    ✓ assets_preco… 04:59    ✓ functional    04:20
✓ docker-build  08:36    ✓ rspec          03:48    ✓ integration   05:13
                                                    ✓ cucumber      08:27
```

Accessible right from the GitHub pull request, to dive into detail

No more maintaining CI server!!

Remember: it is a production system!

Rerun from failed stage



Workflows » 3scale » porta » oracle-build » dd05645f-1e9d-4f63-9882-ae97d24c3b65

oracle-build / oracle_build

```
Fixed: waiting for too many databases to be created
```

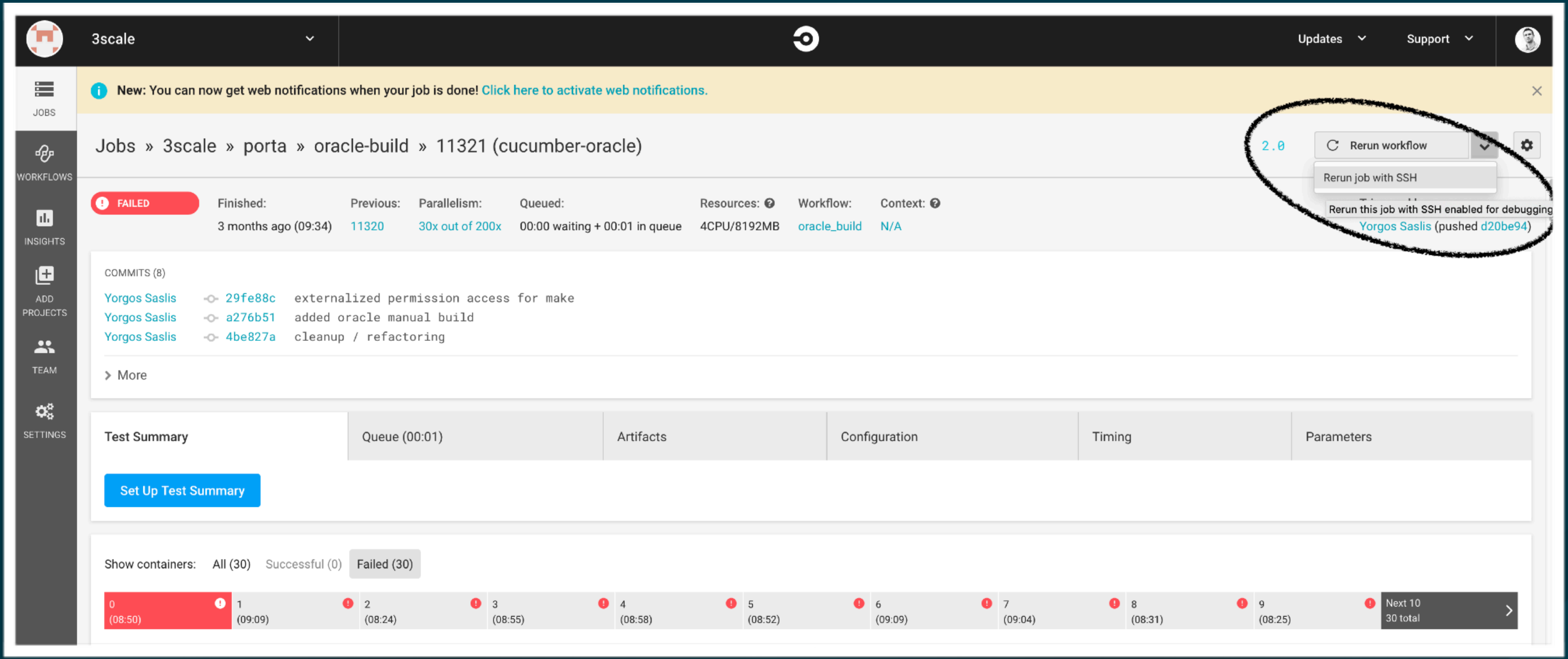
```
✓ manual_approval
✓ docker-build       08:17
✓ dependencies…      01:00
✓ dependencies…      00:35
✓ unit-oracle        19:55
✓ rspec-oracle       14:18
✓ assets_preco…      01:32
! cucumber-ora…      09:34
✓ lint               04:43
✓ integration-…      18:03
```

Pipeline only starts from segment that failed.

No waiting around, no billing for re-running same segments.

Debug CI failures

SSH to container that is running builds (allows us to get builds passing much faster)



New: You can now get web notifications when your job is done! Click here to activate web notifications.

Jobs » 3scale » porta » oracle-build » 11321 (cucumber-oracle)

Rerun this job with SSH enabled for debugging

Finished:	Previous:	Parallelism:	Queued:	Resources:	Workflow:	Context:
3 months ago (09:34)	11320	30x out of 200x	00:00 waiting + 00:01 in queue	4CPU/8192MB	oracle_build	N/A

Yorgos Saslis (pushed d20be94)

COMMITS (8)

Yorgos Saslis	29fe88c	externalized permission access for make
Yorgos Saslis	a276b51	added oracle manual build
Yorgos Saslis	4be827a	cleanup / refactoring

More

Test Summary | Queue (00:01) | Artifacts | Configuration | Timing | Parameters

Show containers: All (30) Successful (0) Failed (30)

0	1	2	3	4	5	6	7	8	9	Next 10
(08:50)	(09:09)	(08:24)	(08:55)	(08:58)	(08:52)	(09:09)	(09:04)	(08:31)	(08:25)	30 total

Bring up the environment to debug the failing build in just a couple of mins

...and price!

Price

It is cheaper because of better resource usage.

Using a fleet of short lived containers is better than VMs

2.5K EUR vs 1.2K EUR

Let’s get back to parallelisation…

How do we parallelize our tests?

If we have to run
${numberOfTests = 1022}
tests,
how do we split them
into
${numberOfContainers = 40}
containers?

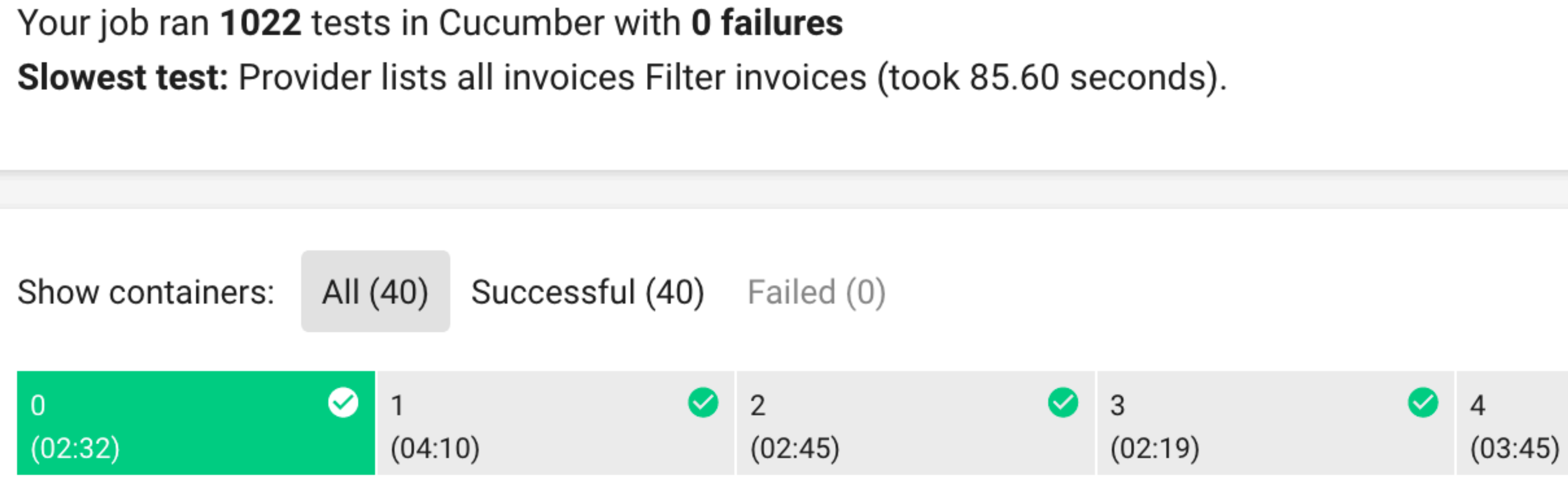
How do we parallelize our tests?

- Alphabetical
- Statically grouped
    - maven phases
    - JUnit Categories
    - filesystem directories
    - …

Wouldn’t it be great if we knew how long each test takes?

Split by timings

Helps orchestrate your test workload, to run in parallel



Your job ran **1022** tests in Cucumber with **0 failures**

Slowest test: Provider lists all invoices Filter invoices (took 85.60 seconds).

Show containers: All (40) Successful (40) Failed (0)

0 ✓ (02:32)	1 ✓ (04:10)	2 ✓ (02:45)	3 ✓ (02:19)	4 (03:45)

extract from `.circleci/config.yml` showing how cucumber tests are split

https://circleci.com/docs/2.0/parallelism-faster-jobs/#using-the-circleci-cli-to-split-tests

http://docs.shippable.com/ci/running-parallel-tests/

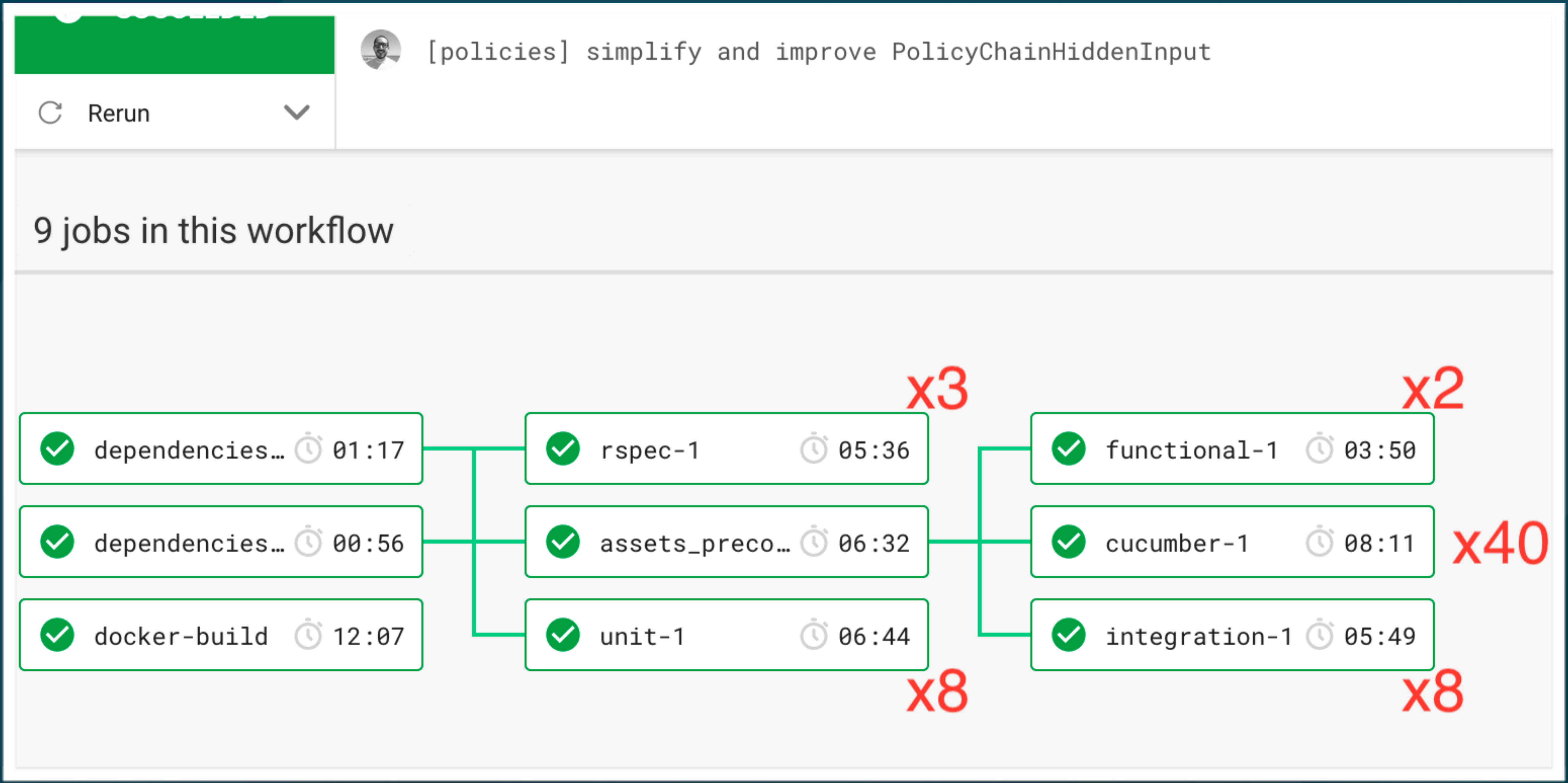
Split by timings - but how?

Helps orchestrate your test workload, to run in parallel

```
- run:
    name: Run cucumber tests
    concurrency: 40
    command: |
     bundle exec cucumber $(circleci tests glob “features/**/*.feature" \
              | circleci tests split –split-by=timings)
```

extract from `.circleci/config.yml` showing how cucumber tests are split

2 levels of parallelism



Rerun

[policies] simplify and improve PolicyChainHiddenInput

9 jobs in this workflow

✓ dependencies… 01:17
✓ dependencies… 00:56
✓ docker-build 12:07

✓ rspec-1 05:36
✓ assets_preco… 06:32
✓ unit-1 06:44

✓ functional-1 03:50
✓ cucumber-1 08:11
✓ integration-1 05:49

Tradeoffs

Nothing comes without sacrifice…

Costs

$$

Less configurable

than Jenkins

External

Dependency

Not

OSS

Not fully Open

Source Software

Enough about CI.
Let’s talk about tests!

flaky tests

Dirty State

If we rely on state for some tests, ensure it’s done properly.

LEFT-OVER STATE FROM PREVIOUS TESTS

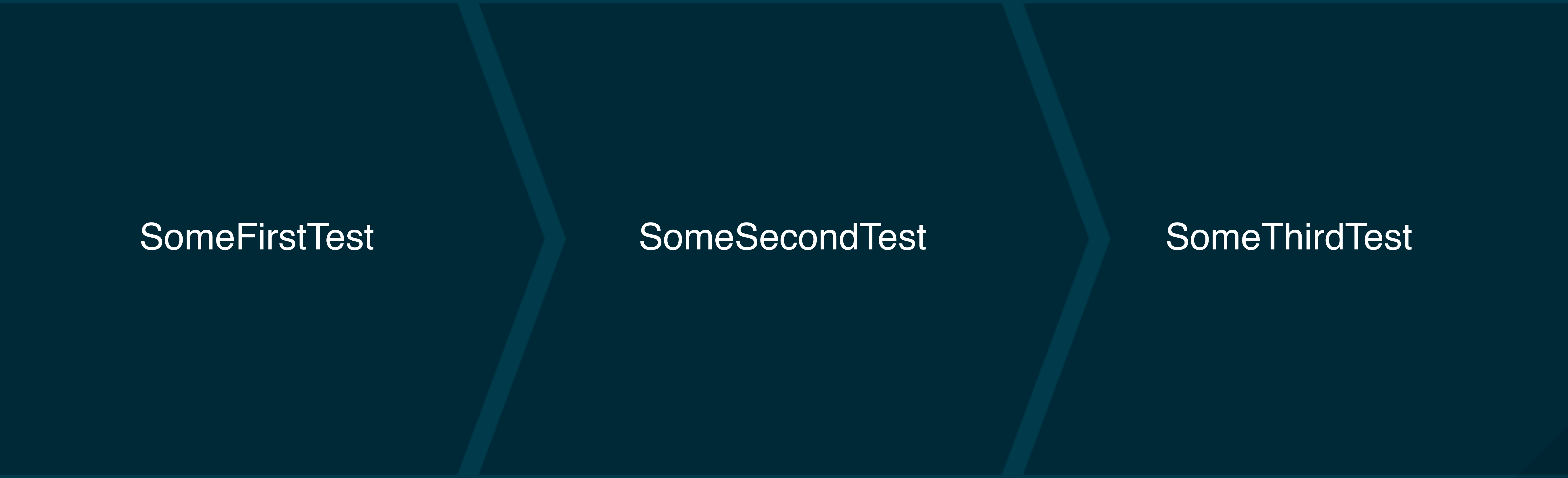
Some tests that rely on bringing the System-Under-Test (SUT) into some “known” state - then running against that - don’t clean up after themselves properly.

BRINGING INTO KNOWN STATE ONLY COVERS SOME PARTS

E.g. if we rely on database for state, we didn’t restore a full database backup before every test (slow), rather we just modified some records in DB — but this does not ensure known state is what we expect it to be.

Reliance on other tests

Symptom: tests only pass if other tests have ran before them.



SomeFirstTest

SomeSecondTest

SomeThirdTest

Example: `SomeThirdTest` passes only when it happens to run after `SomeFirstTest` and `SomeSecondTest`

Tips how ensure test reliability

Discover randomly failing tests early

Randomize

Execute your tests in random order.

Verify you can rerun with the same seed.

Excercise

Run them 10 or 100 times a day if possible.

Not only on merge or pull requests.

Measure

Record test failures and times in machine readable format (JUnit, TAP, ...)

Steps to debug test order dependencies

The process we followed to identify problematic tests whenever a “random” failure occurred.

Reproduce

Run the batch of failing tests and reproduce the failure.

Bisect

Split the test batch in two.

Run only half of the tests.

Repeat

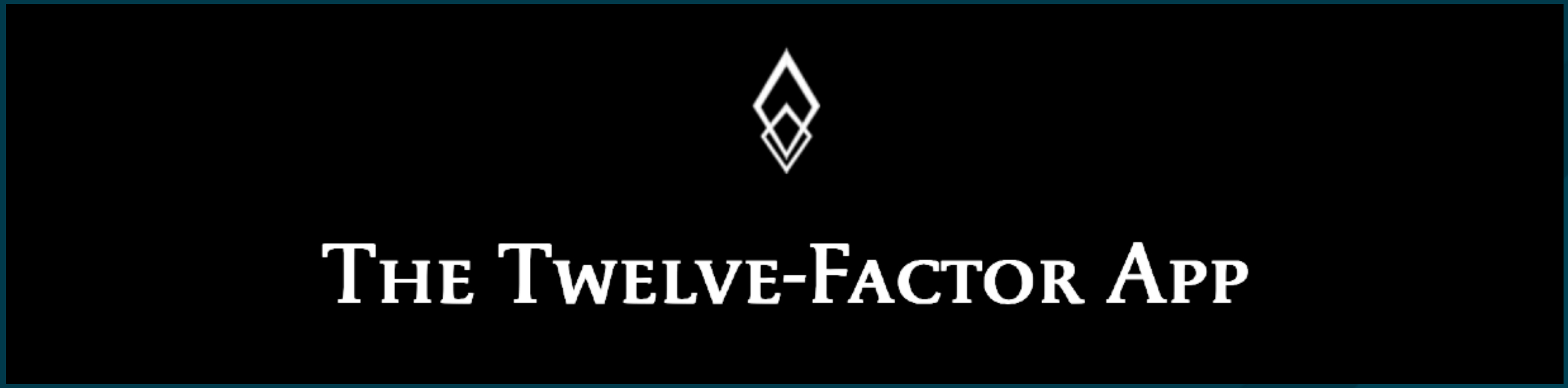
Go back to reproducing with just half of the tests.

Repeat until there are just two.

“The 13th factor: Tests”

Not enough focus on **test** codebase:

* parallelizable
* reliable
* independent of each other



The Twelve-Factor App

dependencies caching

External dependencies

Shave minutes off the build by avoiding to download from the internet

Use transitive dependency locking (Gemfile.lock, package-lock.json, Gopkg.lock, …)

- can be the same across builds
- no point running in “next” build if hasn’t changed from “previous” build
- use some cache

Try to use all CPU cores when installing dependencies.

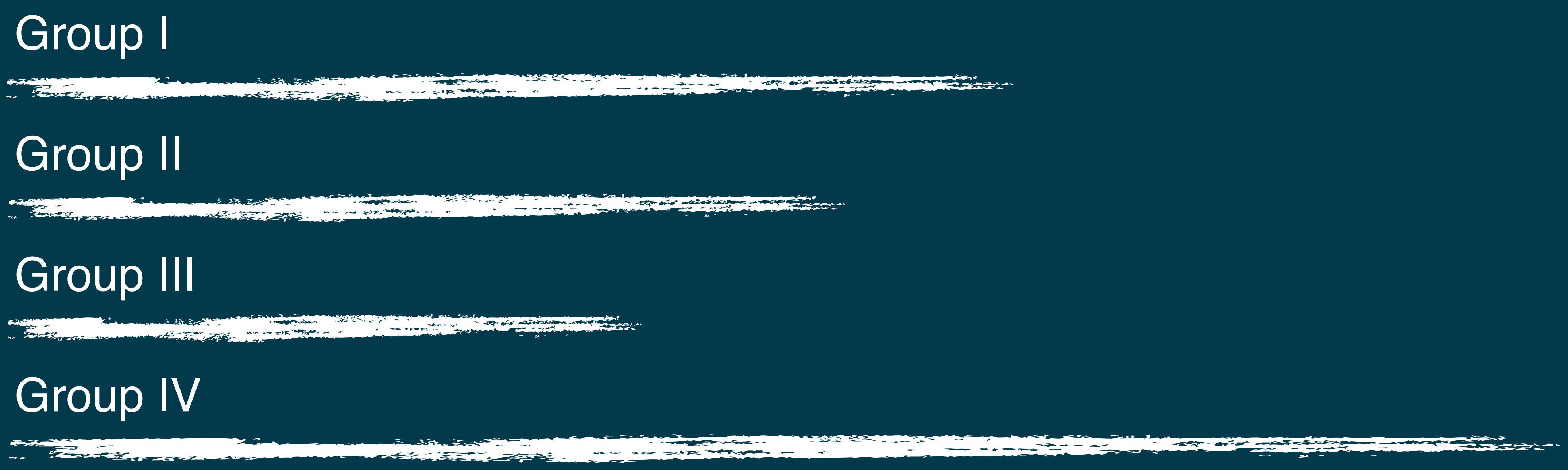
External dependencies

Avoid reinstalling if they didn't change since the last build

Commit
Commit
Commit
Commit
Commit
Commit
Commit

External dependencies

Don’t reinstall for each group. (don’t run `mvn clean verify` in each group…)



Internal dependencies

Artifacts used inside the build

For example transpiled assets, bundling, optimizing images, etc.

The Future of CI~~/CD~~

CI != CD

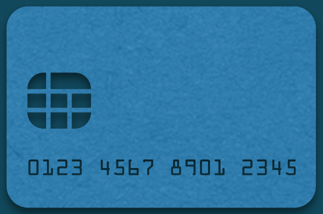
- CI has very different needs than CD
  - Most deployments are usually simple
  - …compared to orchestrating the optimal, parallel execution of a test suite

- CI should only care about executing the tests, as fast as possible

- CI is a production workload with very predictable patterns
  - …unlike other production workloads

- CI should focus on Test, not on Pipeline

Next-gen CI

( + Tests) =

Results

Red Hat

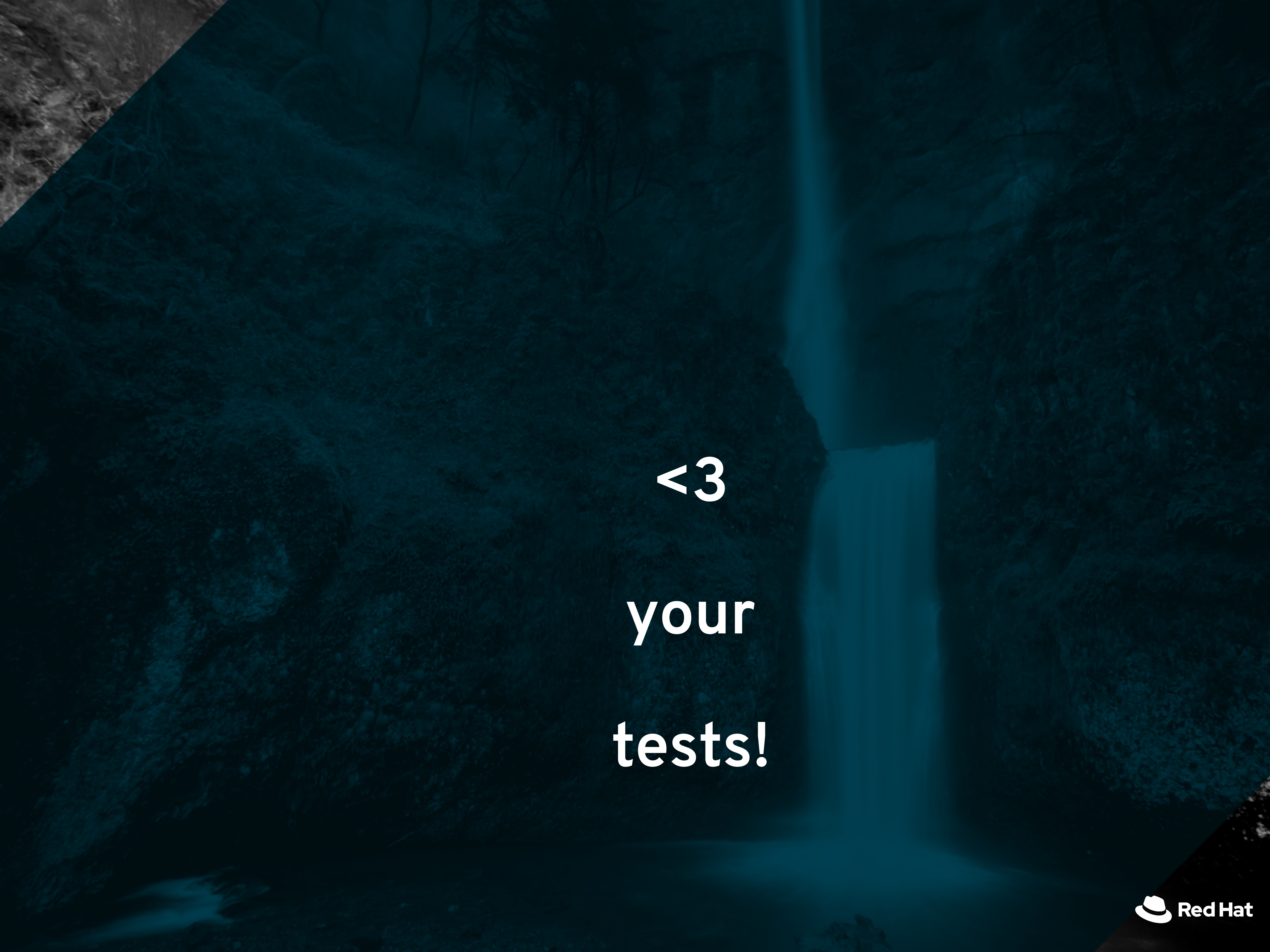
Dynamic test allocation

Optimising test suite parallelisation

Nodes pull more tests to run, when idle

Nodes get pushed a pre-allocated set of tests at start of test run

Versus



<3

your

tests!

Thanks for your attention!

Yorgos Saslis - @gsaslis

Michal Cichra - @mikz

github.com/3scale/porta

3scale BY RED HAT®

Red Hat