Cider-CI

Multi-service integration tests

Traditional CI:

Continous *builds*

single shell script

hooks *around* that script (`before`, `after`, …)

Problems:

hard to make faster/parallelize

hard to set up/manage complex setups

Cider-CI approach

- complex, but **explicit configuration**
- **declarative dependencies**
- very little assumptions about your workflow
- *pro:* hackability
- *con:* simples cases are relatively verbose

Cider-CI overview

Projects have *1* **(git) Repo**

Repo contains configuration for *1*+ **Jobs**

Jobs run *1*+ **Tasks** *in parallel*

Tasks run *1*+ (shell) **Scripts** *in order*

Jobs can be **triggered** from branches and **depend** on each other

Tasks can be re-tried

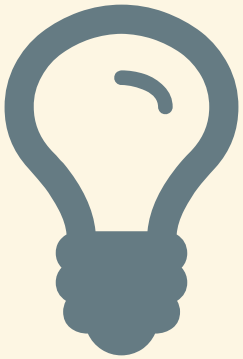
Scripts can **depend** on each other

Details, quick walkthrough

(Excerpt from much longer Talk about Cider-CI)

- HTML: **http://drtom.ch/talks/2015/CL/**
- PDF: **http://drtom.ch/talks/2015/CL/slides.pdf**

4. CONCEPTS IN CONTEXT

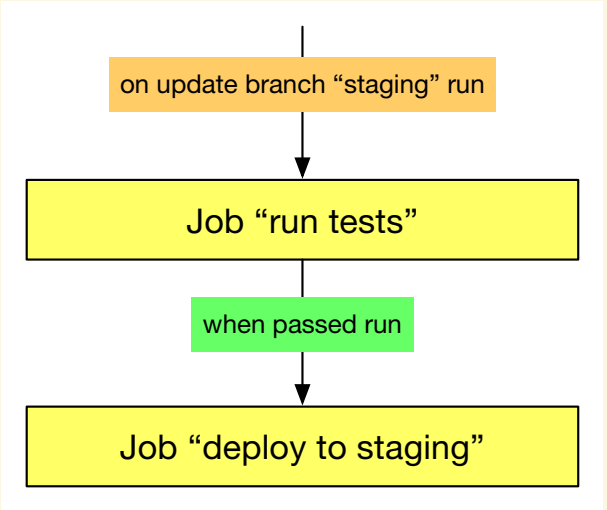


JOBS

EXAMPLES

- run test-suite
- perform static code checks
- build
- deploy

jobs can be **triggered** and can **depend on each other**



PROJECT CONFIGURATION

`cider-ci.yml` file in the project

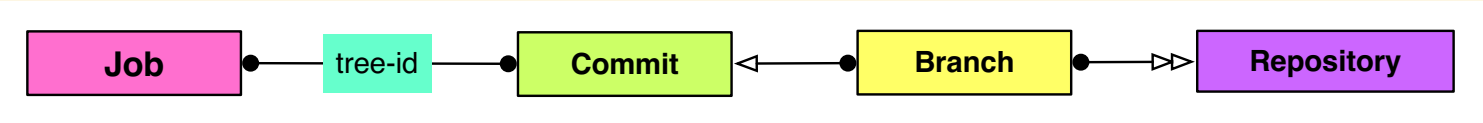
```
jobs:
  deploy_test:
    name: Deploy to test

    depends-on:
    - type: job
      job: integration-tests
      states: [passed]

    run-on:
    - type: branch
      include-match: ^master$

    # specify tasks etc
```

The source is the truth.

configuration: reproducible, reviews, audits ???

CIDER-CI AND THE SOURCE CODE

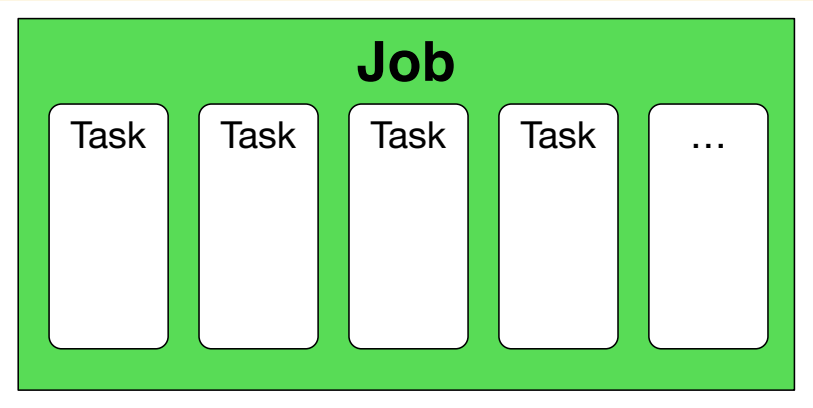
Cider-CI "knows" about commits, branches, submodules,…



`tree-id`: fingerprint of your source code

- **reproducibility**
- jobs can be **run at any time** (later)
- **binary search** for "bad" commits
- commit amends, squashing: **existing job remains valid**

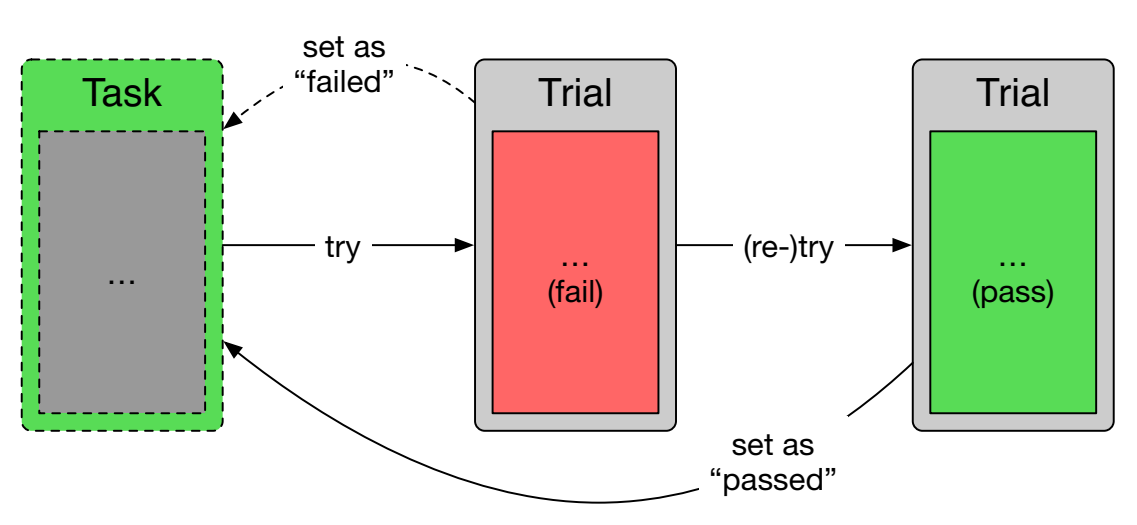
JOBS & TASKS



job: container and state aggregate for tasks

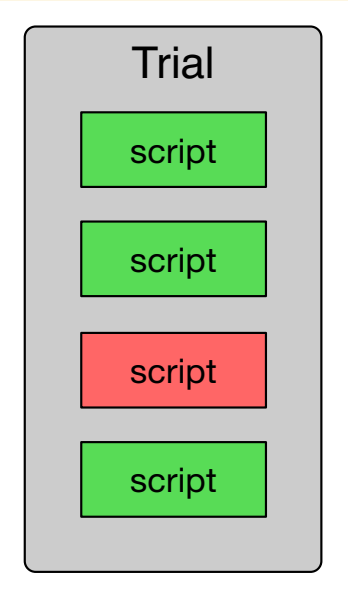
→ **parallelization**

TASKS & TRIALS



- **blueprint**
- container and state aggregate for trials

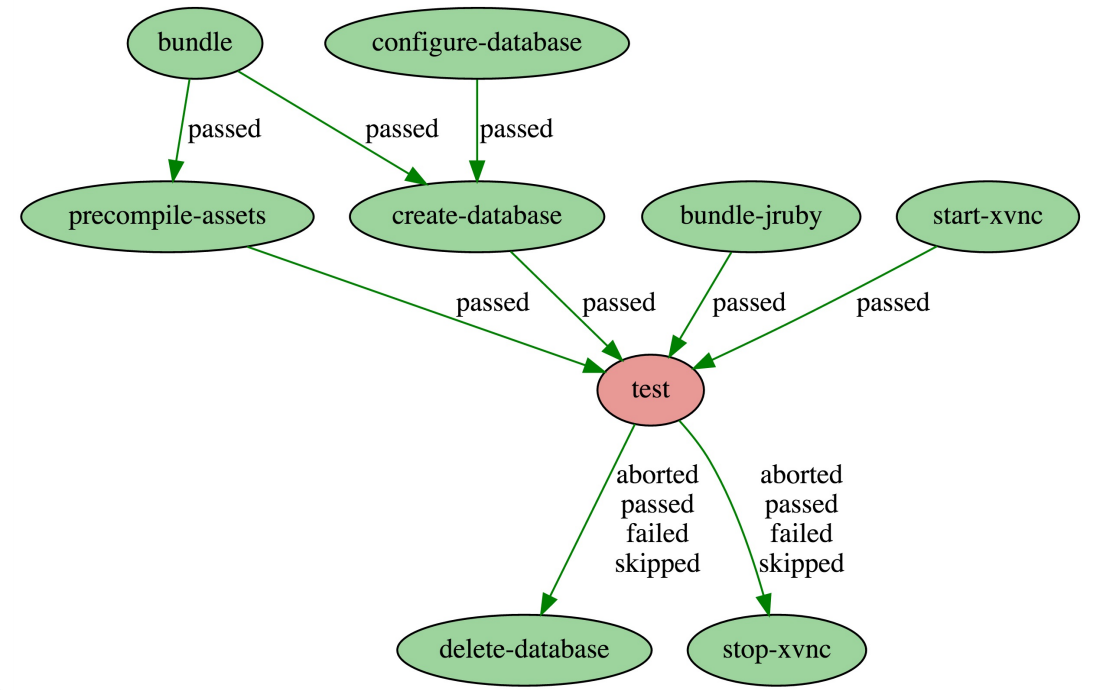
→ resilience



TRIAL & SCRIPTS

- actual unit of execution
- executed in the **same context**
- **depend** on each other

SCRIPT DEPENDENCIES



- traditional CI: one "build" ⇔ one script
- more modern: one main script + before and after "hooks"
- Cider-CI: **scripts with dependencies**

What is it good at?

Speed:

run lots of tasks **in parallel & retry** them

instead of `sh tests/*`

Continous Integration/Delivery/…

trigger and run different kinds of jobs

instead of `sh tests/* && ./build && ./deploy`

Flexibility

1 instance for your organization *or* 1 per project
(re-)use existing infrastructure *or* "in the cloud"

What does it not do?

Access management

always trusts the repository
control (push) access there

Secrets management

set up your own infrastructure and/or excutors,

executors can be locked to "accepted repos"

Examples

- Jobs: "Test" *triggers* "Build" → "Release" → "Deploy-Docs"
- Job: **"Good To Merge"**, *depends on* "Lint", "Unit Tests", "Feature Tests"

Complex example (CI-inception)

Trial for the task *spec/features/attachments_spec.rb* in job *Integration-Tests* of *Cider-CI*

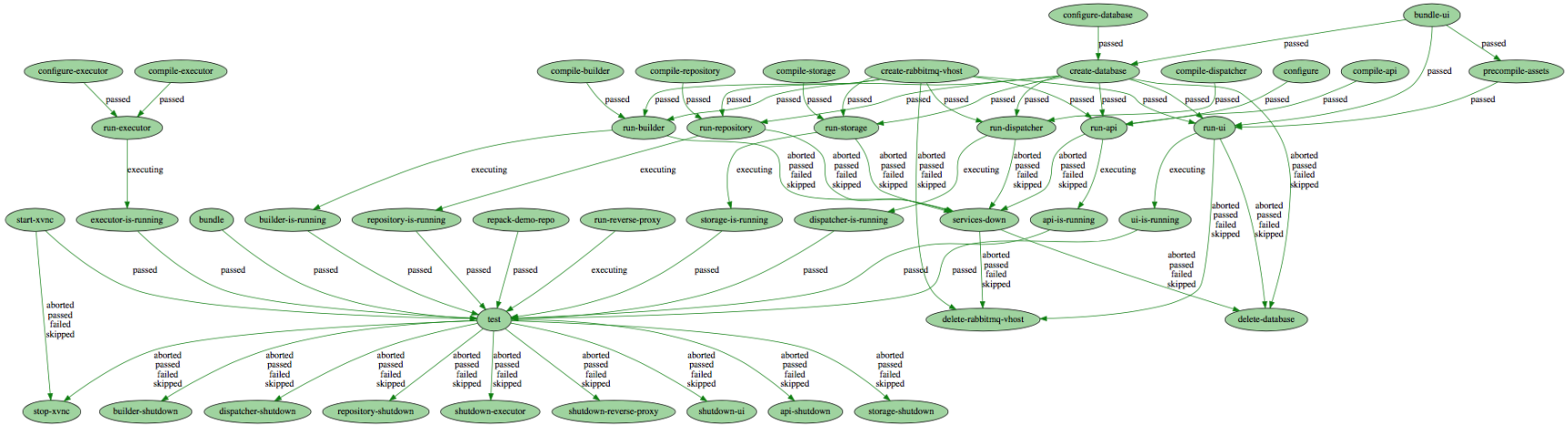
Retry

passed created **14 days ago on Sunday, 29th November** started **14 days ago on Sunday, 29th November** executed in **3 minutes** (167.83 seconds) on **cislave5**

Scripts Overview

configure run-reverse-proxy configure-database start-xvnc create-rabbitmq-vhost configure-executor repack-demo-repo compile-storage compile-builder compile-api
compile-dispatcher compile-executor compile-repository run-executor executor-is-running bundle bundle-ui create-database precompile-assets run-ui run-api
run-storage ui-is-running api-is-running run-repository run-dispatcher run-builder storage-is-running builder-is-running repository-is-running dispatcher-is-running
test shutdown-ui builder-shutdown shutdown-reverse-proxy dispatcher-shutdown stop-xvnc storage-shutdown shutdown-executor api-shutdown repository-shutdown
services-down delete-database delete-rabbitmq-vhost

Start-Dependencies



Terminate-Dependencies

more features

- REST-ful **API** to implement any workflow you want
  - "nightly" builds and deploys
  - integrate with external services
- **attachments**
  - per *Trial*: for debugging (logs)
  - per *git-tree*: for **build artefacts** (binaries)
- good support for `git submodules`
- some github support: listen to update hook, set repo status

Try it out

free software, installs with two commands (ansible)

`docs.cider-ci.info/`

or read the sources: `github.com/cider-ci/cider-ci`

If you want to try something out here at the 32C3, contact me: `1@178.is`

THX!