Continuous Integration and Delivery with Cider-CI at the ZHdK

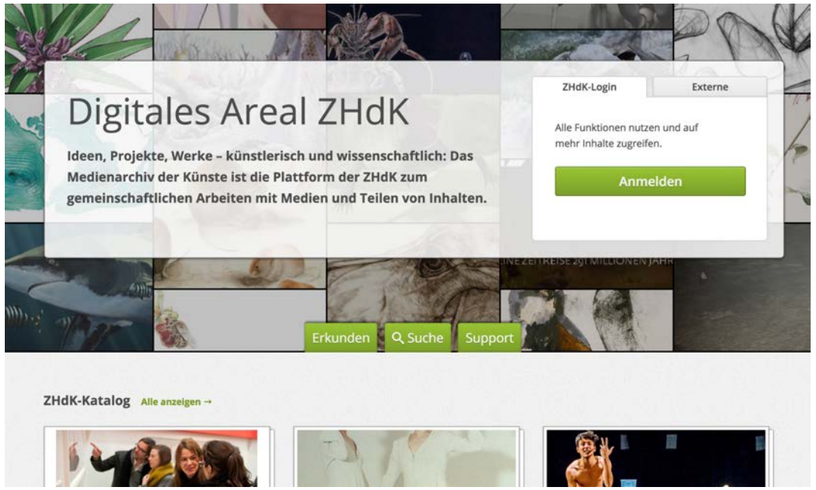
CDDZ Meetup, April 2016


Dr. Thomas Schank

Version 1.0.0

ZHdK → Services → ITZ → Development

→ Leihs / Madek - Team



Web-Applications, HTML, REST, …

Functional Reactive Programming ☺

Ruby on Rails, ClojureScript 👍, Clojure 👍,
React 👍, PostgreSQL 👍, MySQL 👎, …

Continuous Integration / Delivery ☺

Specification by Example , BDD, … ⇒ Integration Testing

Unit tests

- test a function, method, …
- run withing a delimited environment
- consistent, reliable
- fast (seconds … minutes)

Integration

- complete usage cycles
- span over processes and services
- interaction
- hard to set-up and tear-down
- inconsistent and error prone
- slow (… hours)

We will solve the following problems today:

1. **coordinate** test scripts (e.g.: setup database, start services, run tests, shutdown services, clean up),
2. manage false negatives - **resilience**,
3. improve **reproducibility** and **transparency**,
4. build **test** and **deployment chains**, and
5. get the **integration tests** to run fast (≤ 5 Minutes).

Caution

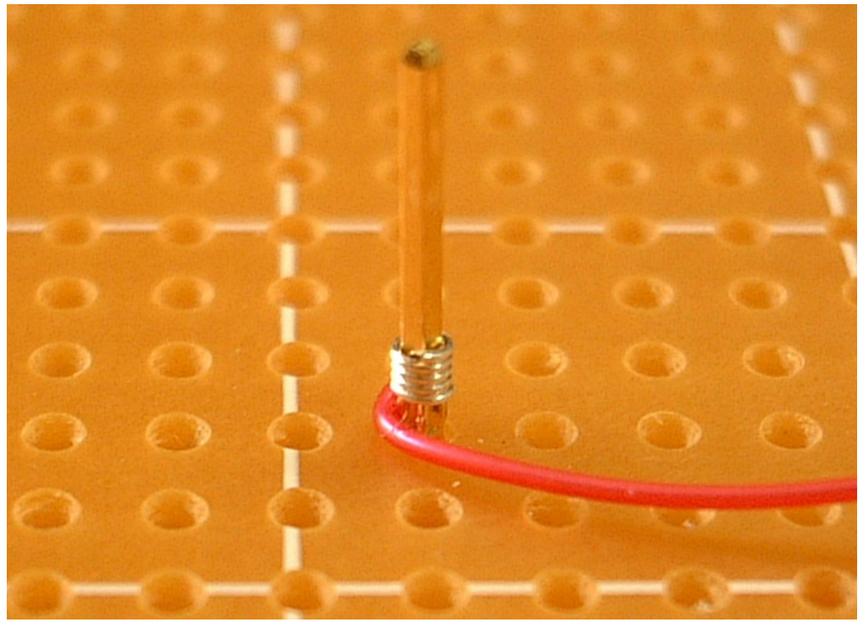
Much of this talk will touch this thing called *Cider-CI*.

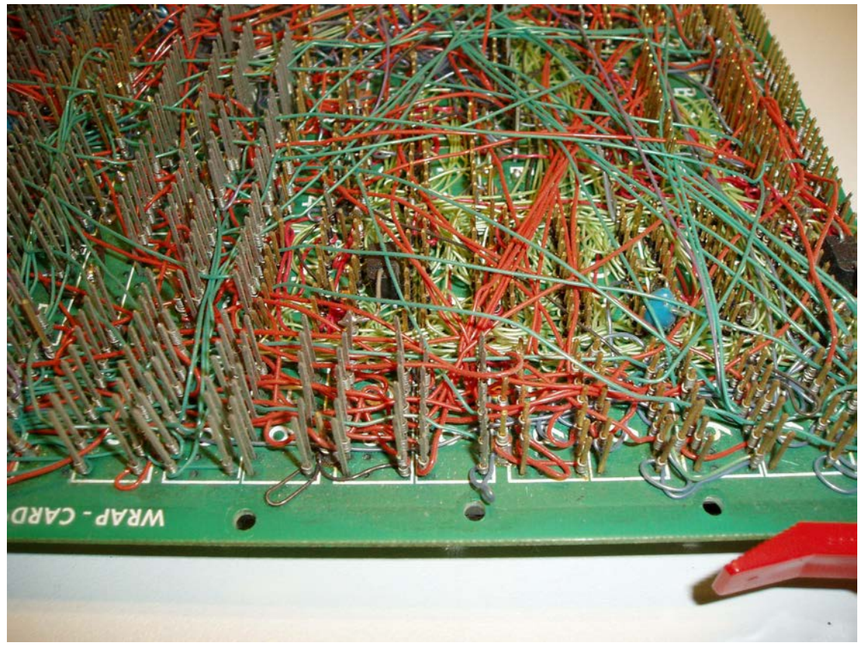
It is an **open source project**. It is **mostly** but not entirely written by myself. There is and was a lot of input by my colleagues and in particular by Max!

I am biased, and I do consult and contract.

> *The primary focus of Cider-CI was, is, and will be to **solve difficult problems** in **testing, continuous integration, and delivery**. It is not about creating commercially successful product for the masses.*

1. Coordinating Test Scripts





Maven test example:

```
mvn test
```

in Cider-CI

```
...
  scripts:
    test:
      name: Run a silly, failing test in bash
      body: |
        #!/usr/bin/env bash
        set -eux
        test a = b
```

"Easy to formulate with any CI."

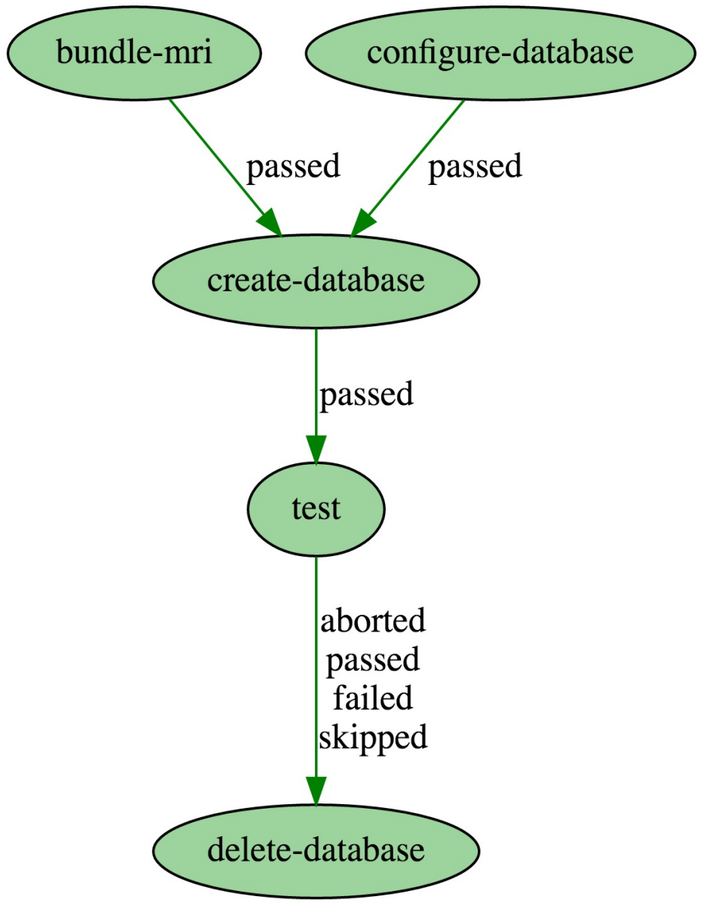
Test Suite

Traditional CI: **one script ≈ test suite**

More modern: **one script** + **before** and **after** scripts

Cider-CI: …

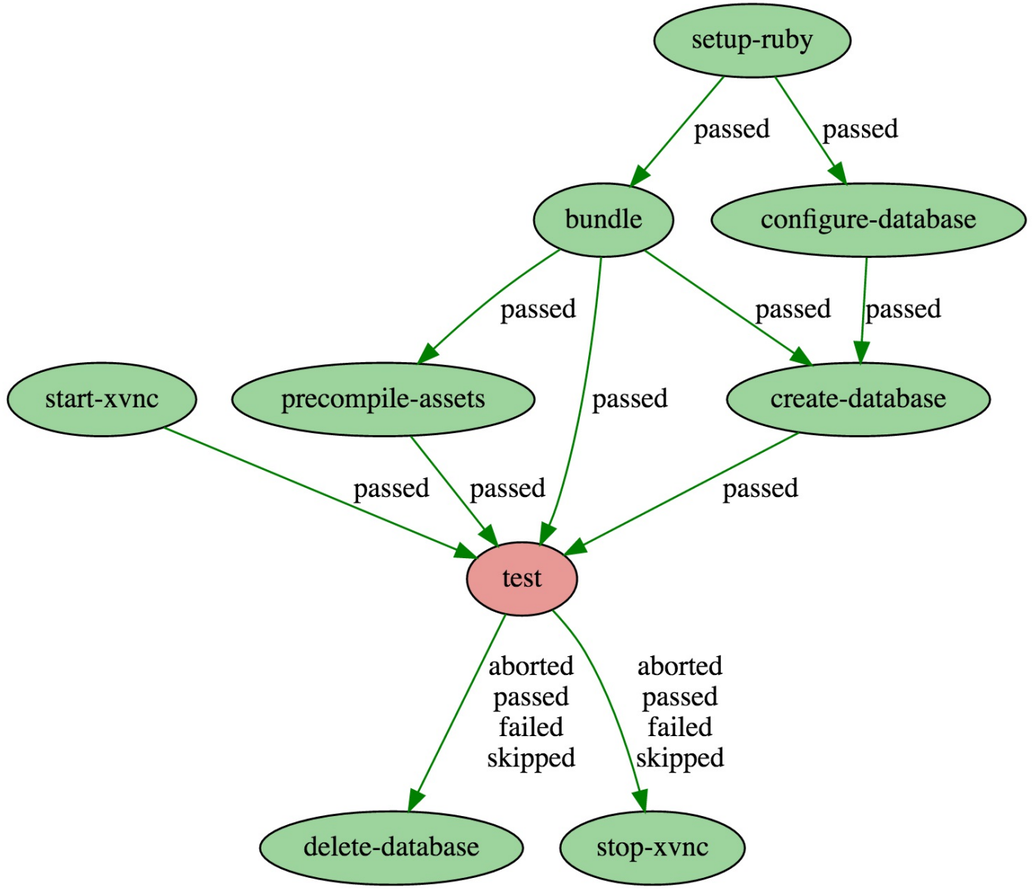
Model Test from Madek-Datalayer




```
scripts:

  configure-database:
    body: bin/configure-database.rb

  bundle-mri:
    exclusive_executor_resource: bundler_2.2
    body: bundle

  create-database:
    body: bundle exec rake db:reset
    start_when:
    - script: bundle-mri
    - script: configure-database

  test:
    body: bundle exec rspec $CIDER_CI_TASK_FILE
    start_when:
    - script: create-database

  delete-database:
    body:  bundle exec rake  db:drop
    ignore_state: true
    start_when:
    - script: test
      states: [aborted, passed, failed, skipped]
```

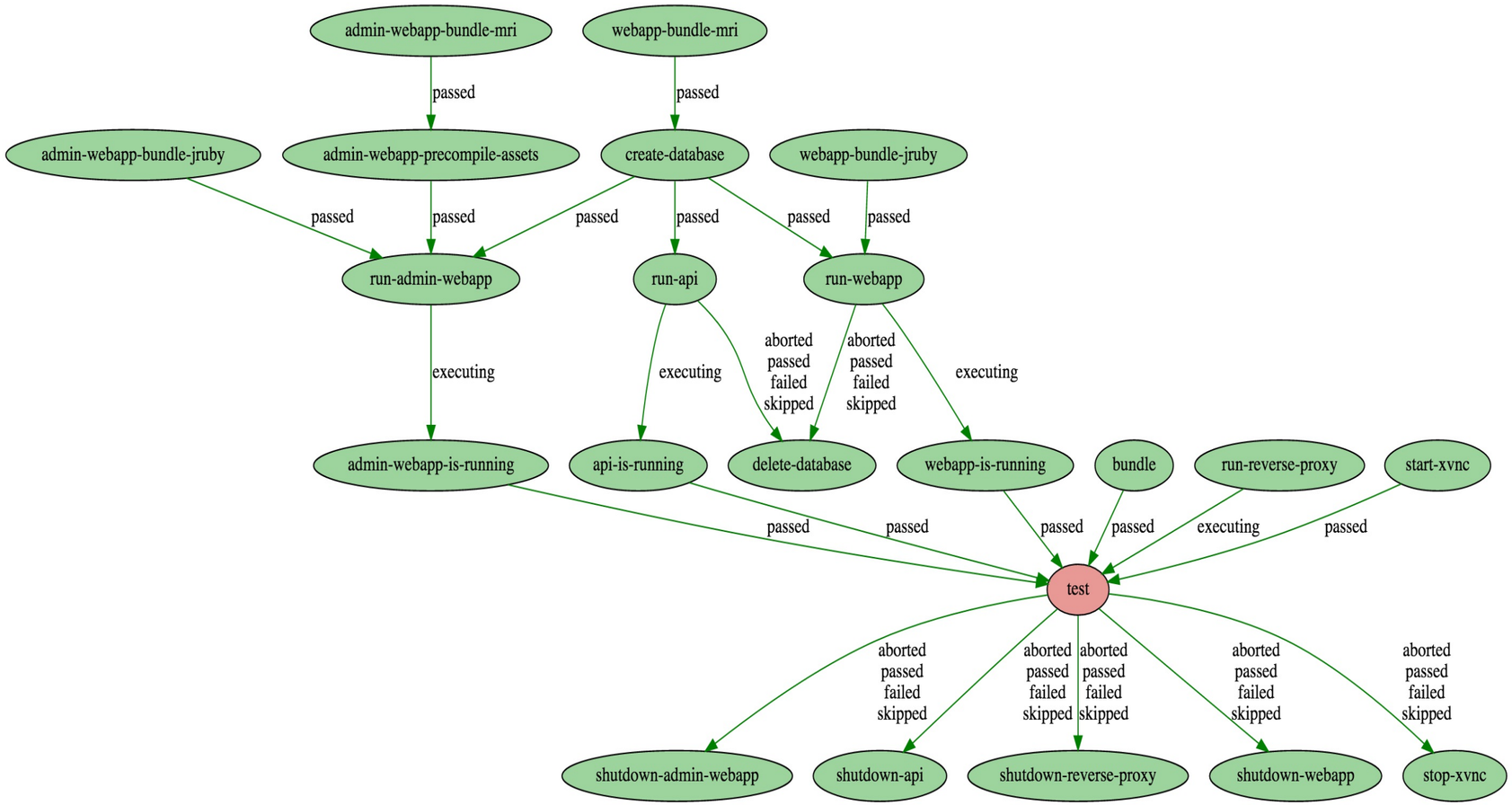
"doable with any CI"

Feature Test from Madek-WebApp



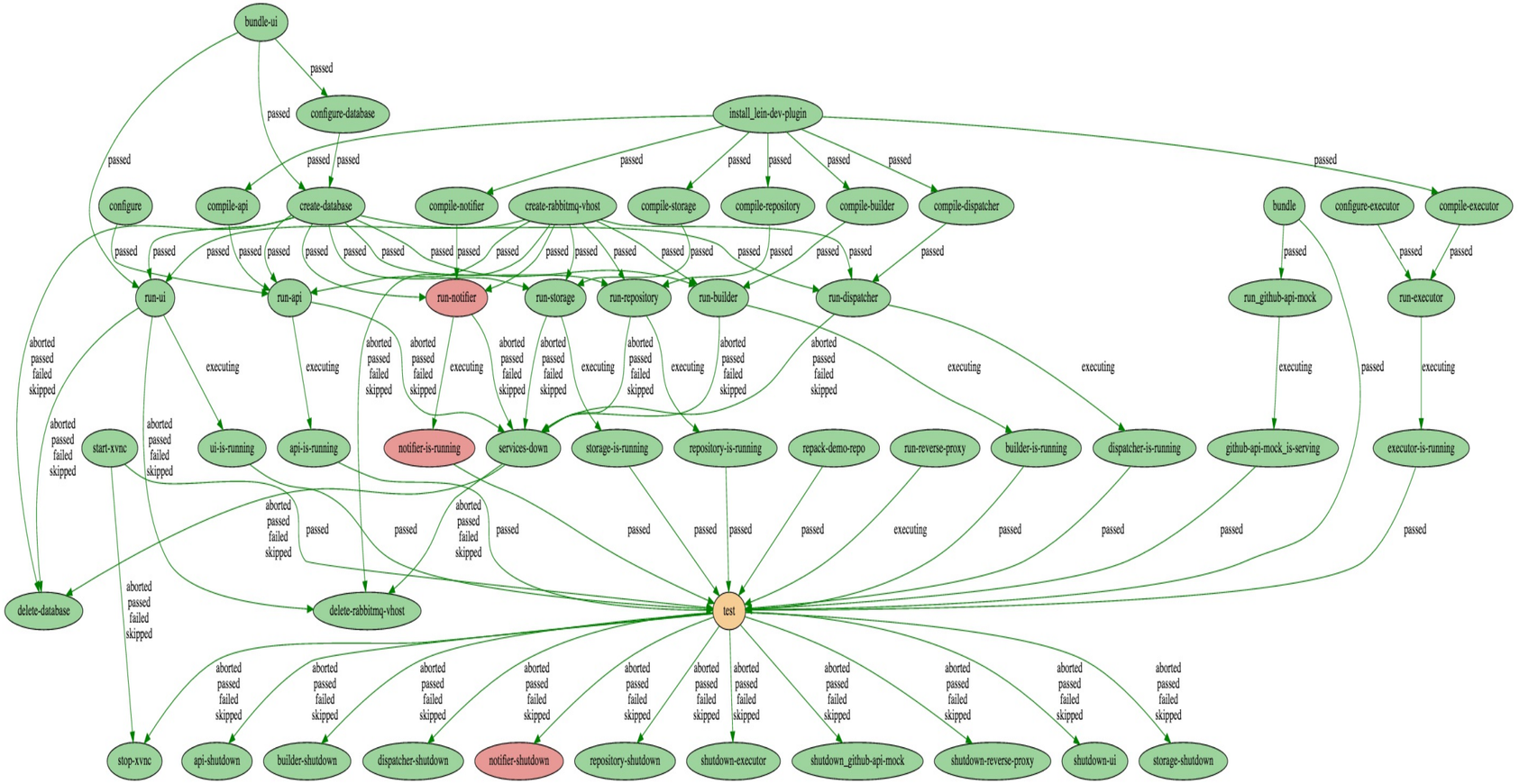
"hard"

Top Level Integration-Tests Madek



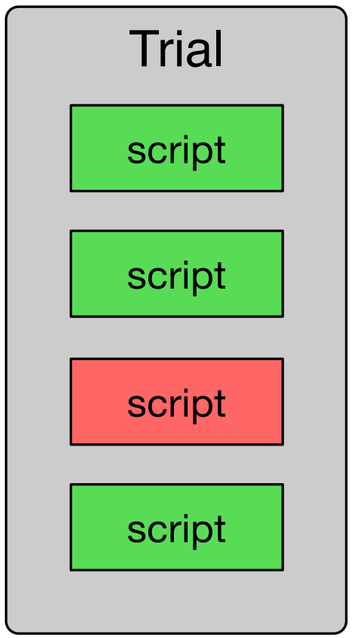
"very hard" - how do you mange change?

Integration-Tests Cider-CI



Demo: http://ci.zhdk.ch/cider-ci/ui/workspace/trials/c6c355cb-d2d8-4ba9-8acb-ccd7803e24bd

"impossible"



Scripts in Cider-CI

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

Coordination - Conclusion

A CI system should provide means to manage complex dependencies and execute them accordingly.

2. False Negatives and Resilience

Madek Project 2012

many new features, many new tests

- testing time 1 1/2 - 2 hours, increasing
- more and more failing tests: **false negatives**
- 1/8 builds pass
- fixes helped only for a short time

⇒ manual retrying of single tests / features

Probability of a False Negative for a whole Test-Suite

	Expression	Example
probability false negative single test	p_f	3%
probability "success"	$p_s = 1 - p_f$	0.97
number of tests	n	100
probability "success" whole suite	$\mathbf{P_s = p_s^n = (1 - p_f)^n}$	$\approx 5\%$

→ only one out of 20 will pass as it should

"succes" = true positive

Why retrying works so well

let k number of independent retries per test

$$P_s(n) = (1 - p_f)^n \Rightarrow \mathbf{P'_s(n, k) = \left(1 - p_f^k\right)^n}$$

Expected successful outcome for $n = 100$ and $p_f = 0.03$

k	P'_s
1	5%
2	91%
3	99.7%

Recipe for Automatic Retries

1. **Split up** your test suite in **tasks**.
2. **Run** the tasks **independently** from each other.
3. **Retry** a task if it fails (e.g. 2 times).
4. **Aggregate** the results.

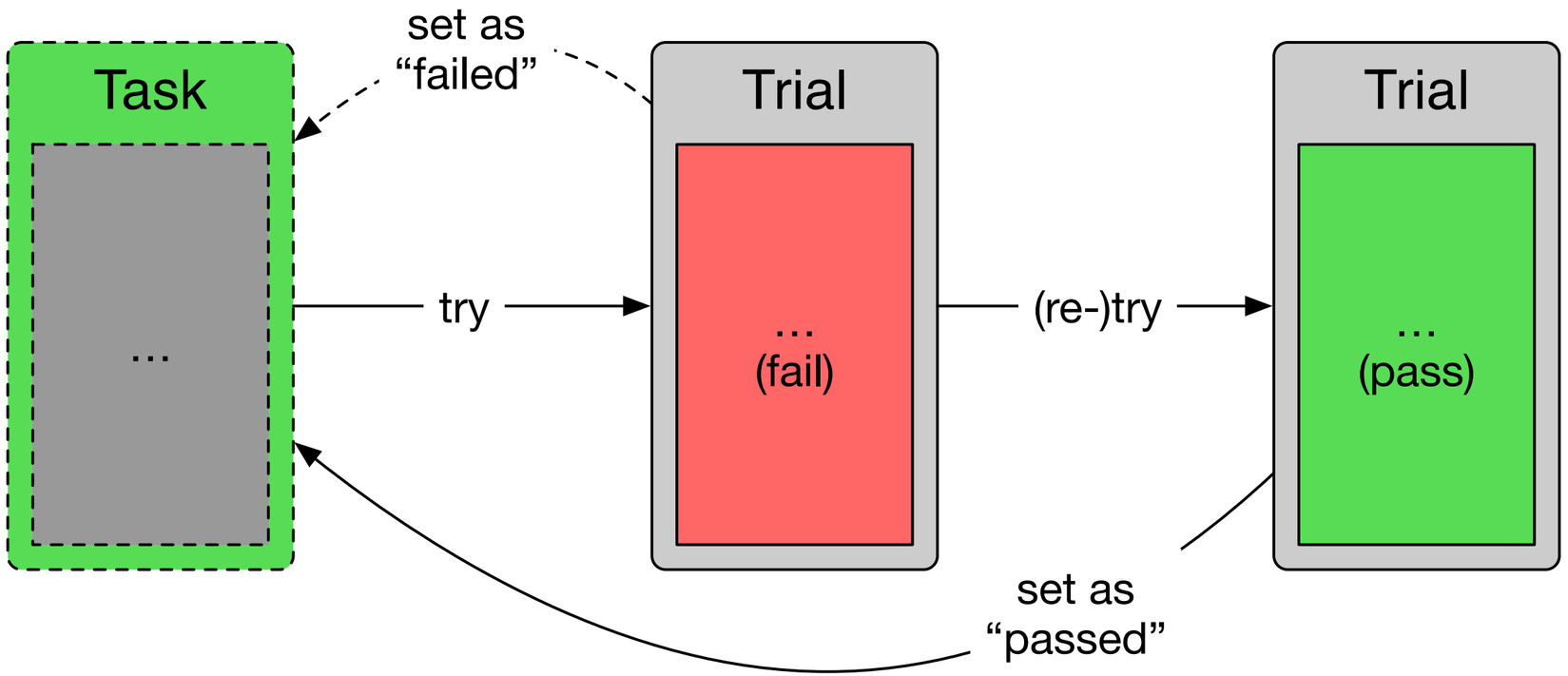
Divide and Conquer

Independent Tasks Benefits

- Executing different project configurations becomes very easy.
- Simulating defects becomes possible.

Independent tasks open an whole area of "things" you can now test easily which were very hard or next to impossible to test before.

Tasks & Trials in Cider-CI



A **task** is much like a **blueprint**. It is a **container** and **state aggregate** for trials. It **describes what** and **how** to be executed. It doesn't embody an execution itself.

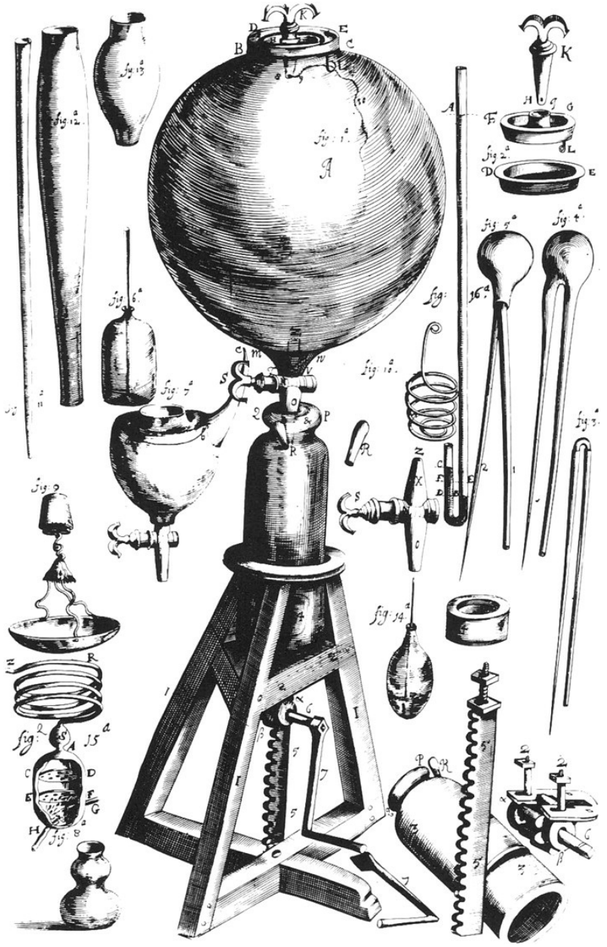
2. Resilience - Conclusion

- more tests → **exponential** increase of likeliness for **false negatives**
- compensate by **retrying** single tests just a **few times**

Retrying tests is not (necessarily) an anti-pattern.

3. Reproducibility

Reproducibility is the ability of an entire experiment or study to be duplicated, either by the same researcher or by someone else working independently. (Wikipedia)



Reproducibility & Project Configuration

We want to be able to reproduce test results at any time (later). The **test configuration** must be **resolvable** from the **source code**!

(source code ↦ test configuration) ⇒ run tests

Simple solution: **put your test configuration together with your source code!**

Project Configuration in Cider-CI

Either top level project file `cider-ci.yml`, `cider-ci.json`, `.cider-ci.yml`, or `.cider-ci.json` will do.

```
jobs:
  intro-demo:
    task: test a = a
```

Cider-CI Compact vs Canonical Notation

```
jobs:
  intro-demo:
    task: test a = a
```

normalization → Job Specification

```
key: intro-demo
name: intro-demo
empty_tasks_warning: true
context:
  tasks:
    '0':
      traits: {}
      scripts:
        main:
          body: test a = a
```

Cider-CI uses the Git and Git Only

You can start any job at any time!

You can retry any task at any time!

- reproducibility
- bisection (aka binary) search for bad commits
- ….

Supporting other SCMs would have compromised many features of Cider-CI.

Totally Opinionated Git Recommendations

- Sign your commits (definitely sign tags / releases)!
- Consider to use something like the git-reflow workflow: squash and rebase! → linear history, traceability.
- Consider to use git in your review process: **author** is the *git author*, the **reviewer** is the *git committer*.
- Consider to use *git submodules*.

Keep Your Project Configuration Manageable with `include`

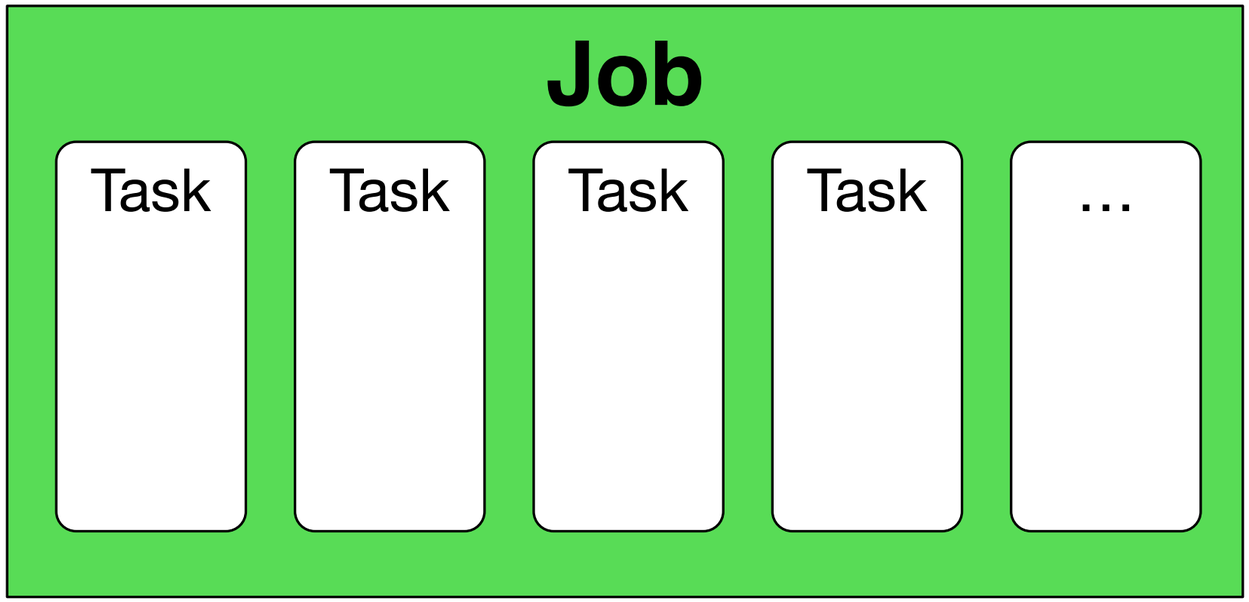
```
include:

  - path: cider-ci/job_meta.yml

  - path: cider-ci/jobs.yml
    submodule: ['deploy']

  - path: cider-ci/job_integration-tests.yml
    submodule: ['integration-tests']

  - path: cider-ci/job_specs-overview.yml
    submodule: ['integration-tests']
```

You can share project configurations (or parts thereof) via git submodules.

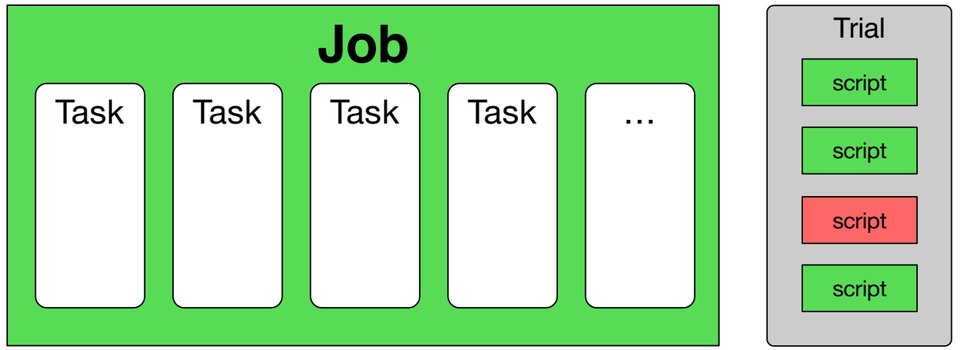
4. Building Test and Deployment Pipelines

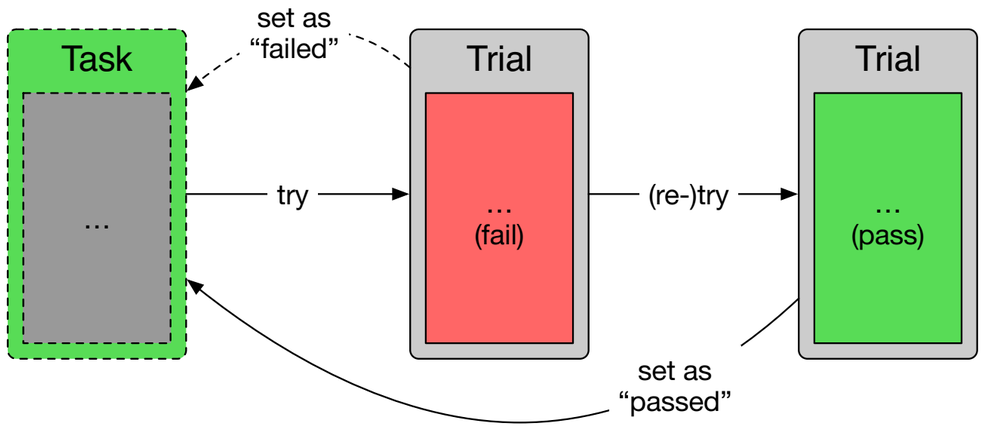
Jobs

A job contains and aggregates tasks.



Overview of the Entities





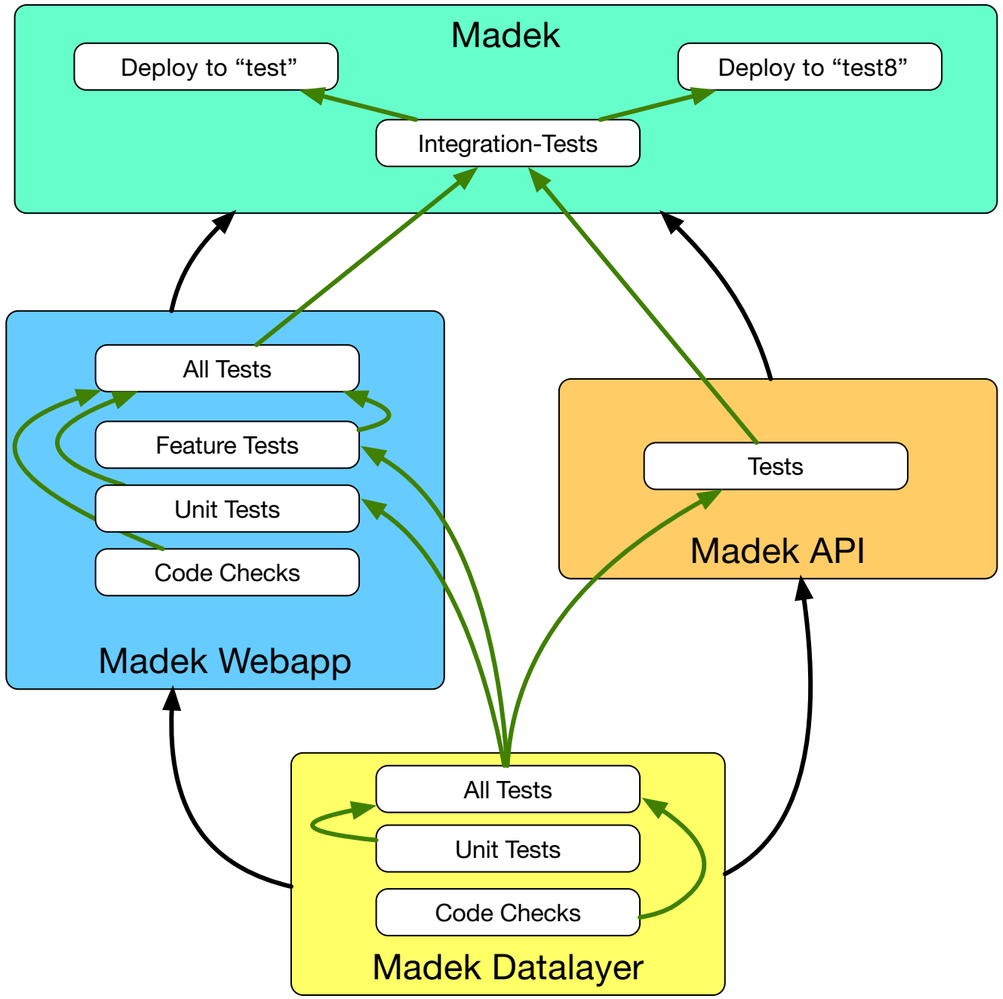
Deploy Job Example

```
jobs:

  deploy-to-test:

    name: Deploy to test.madek

    run_on:
    - type: branch
      include-match: ^master$

    depends_on:
    - type: job
      job: integration-tests
      states: [passed]

    tasks: ...
```

```
...

deploy:
  name: Deploy to test.madek

  traits: [Ansible, MadekTestDeploy]

  max_trials: 1

  aggregate_state: satisfy-last

  scripts:
    deploy:
      timeout: 3 Hours
      body: |
        #!/usr/bin/env bash
        set -eux
        cd deploy
        ansible-playbook \
           -i inventories/zhdk/test \
           play_setup-and-deploy.yml
```

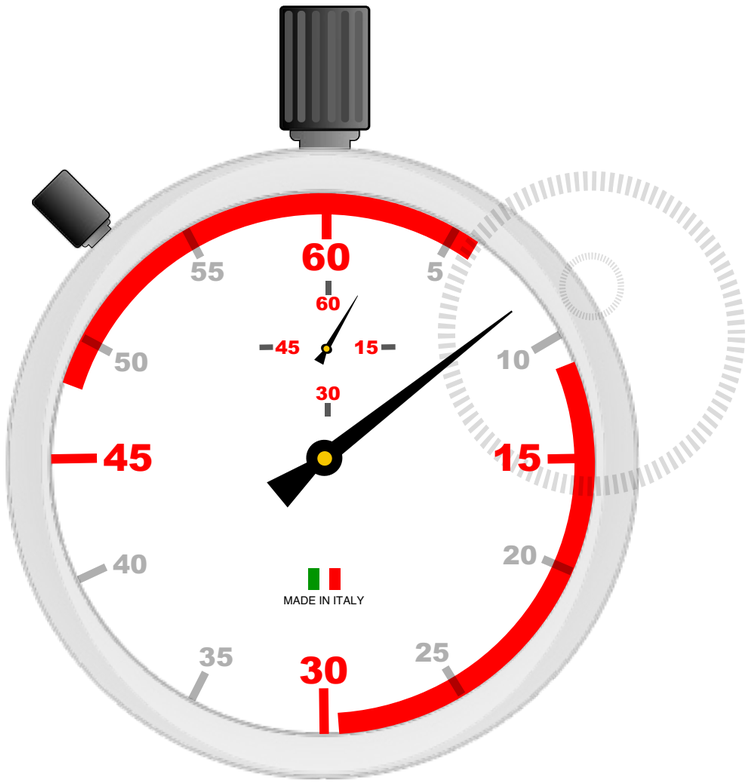
Key properties: `depends_on`, `run_on`

Test and Deployment Chain for Madek



(November 2015 and slightly outdated)

5. Fast Tests



Recall our recipe for retries?

> - ***Run** the tasks **independently** from each other.*

If tasks are **independent** from each other they can be run in **parallel**.

⇒ We are done!

Time for a demo: ci.zhdk.ch

… almost …

Getting Checkouts Fast

- clone/checkout Cider-CI with submodules: 2 - 3 Minutes

→

- keep a **local cache** of the repository
- **no network fetches** if we have **"the tree"** already
- perform **shallow**, **referenced** checkouts
- do some more magic with submodules

⇒ **decentralized** SCM ⇒ **Git** !

Caching

Simple solution: use persistent machines as executors.

⇒ maven, ruby gems, …, gets **cached**

⇒ Trade speed against consistent state! ⇒ security issues!

⇒ We use *blessed* executors for deploys.

Use RAM-Disks

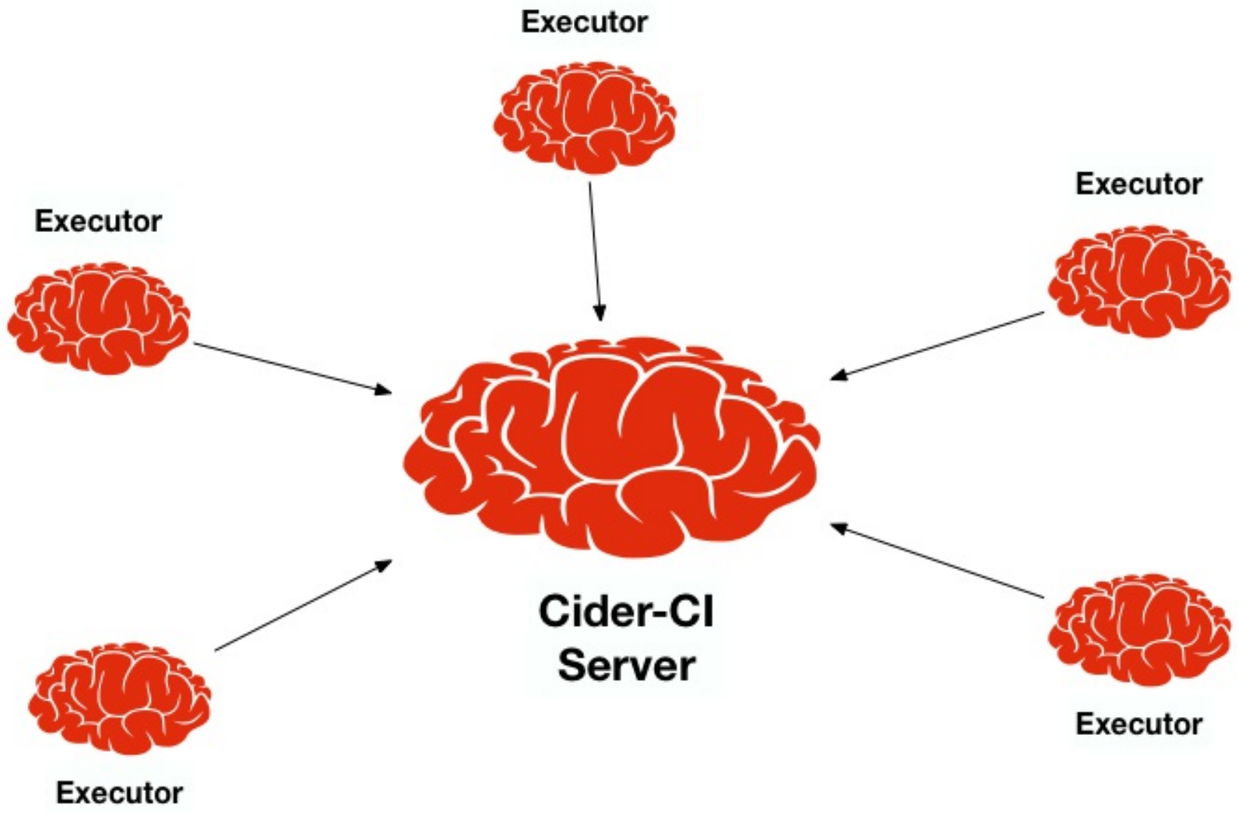
(kudos *Aarno*)

- `/tmp` is a RAM-Disk
- Working Directories are on the RAM-Disk

Cider-CI v4 default *"PostgreSQL Trait"* data-store on RAM-Disk

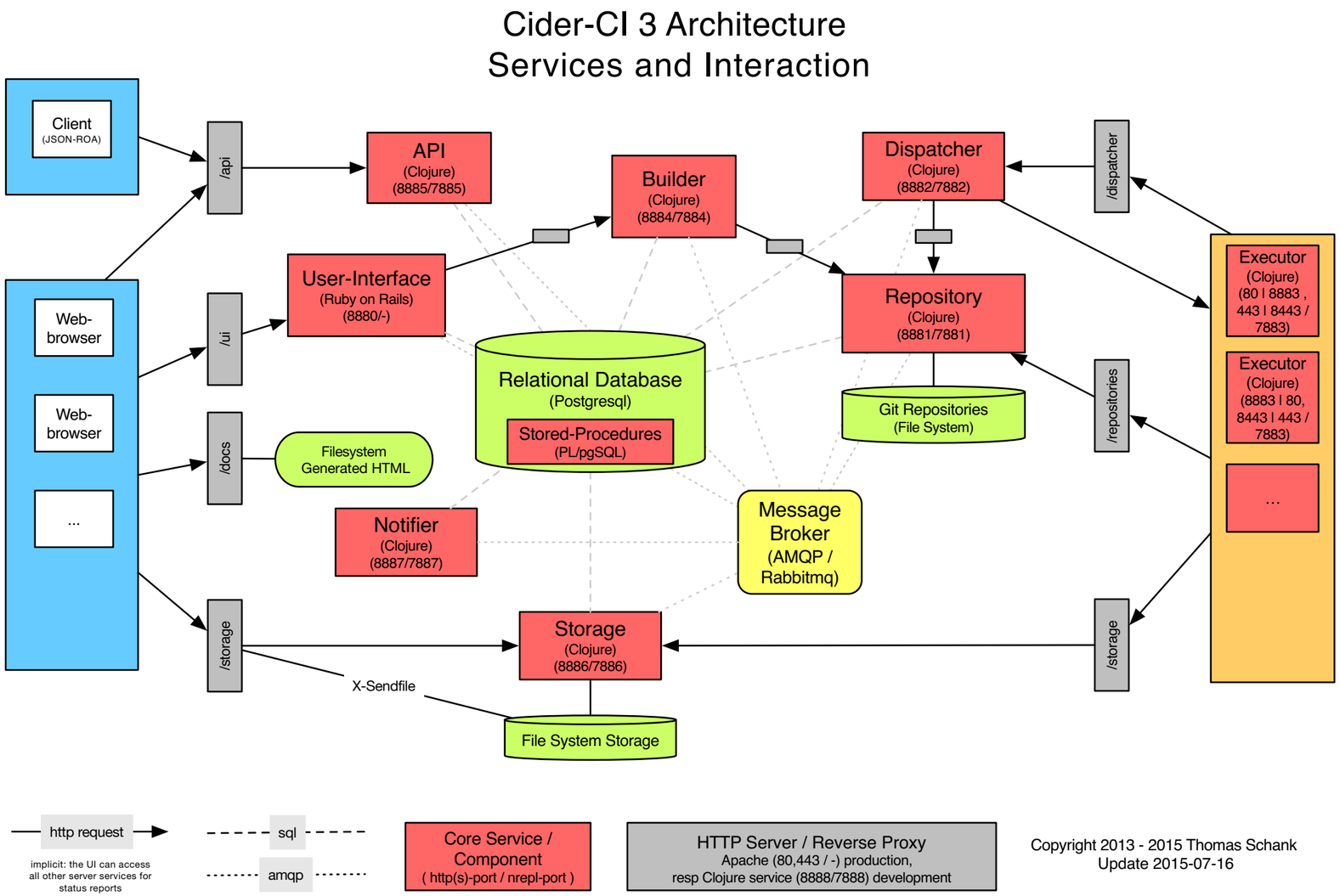
Closure - Cider-CI

Executors & Server



Executors act fairly autonomous.

Architecture



Hardware

ci.zhdk.ch/cider-ci/ui/admin/executors

Cider-CI is an Expert System

it is about **making the hard possible**, and not not about making the simple easy*

- for professionals
- no compromises
- steep learning curve
- high rewards

→ swiss army knife for devops

*see "Simple Made Easy" by Rich Hickey

Cider-CI History & Future

- 2013 v1: Ruby on Rails on Torquebox (JBoss)
- 2014 v2: RoR + Clojure (Torquebox / Immutant)
- 2015 v3: Micro-Service Architecture; GA
- **June 2016 v4**:
  - simplify deployment, fewer system dependencies
    - executors can perform self-upgrades
    - executor: Java8 + Git
    - server: Java8 + PostgreSQL + RabbitMQ + Git
    - Ansible: Debian 8, Ubuntu 16.04
  - additional state: `defective`
  - rename configuration directives, change defaults
  - configuration validator

Want to try Cider-CI?

On Debian jessie:

```
apt-get update && apt-get install curl -y
curl https://raw.githubusercontent.com/cider-ci/cider-ci_deploy/v4/bin/quick-in
stall.sh | bash
```

Resources & documentation and more: cider-ci.info

Consulting, support: Thomas.Schank@AlgoCon.ch.

THANK YOU !