DRUPAL CONTINUOUS INTEGRATION

Part I - Introduction

Continuous Integration

"Continuous Integration" is a software development practice where members of a team **integrate work frequently**, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including tests) to **detect integration errors as quickly as possible**.

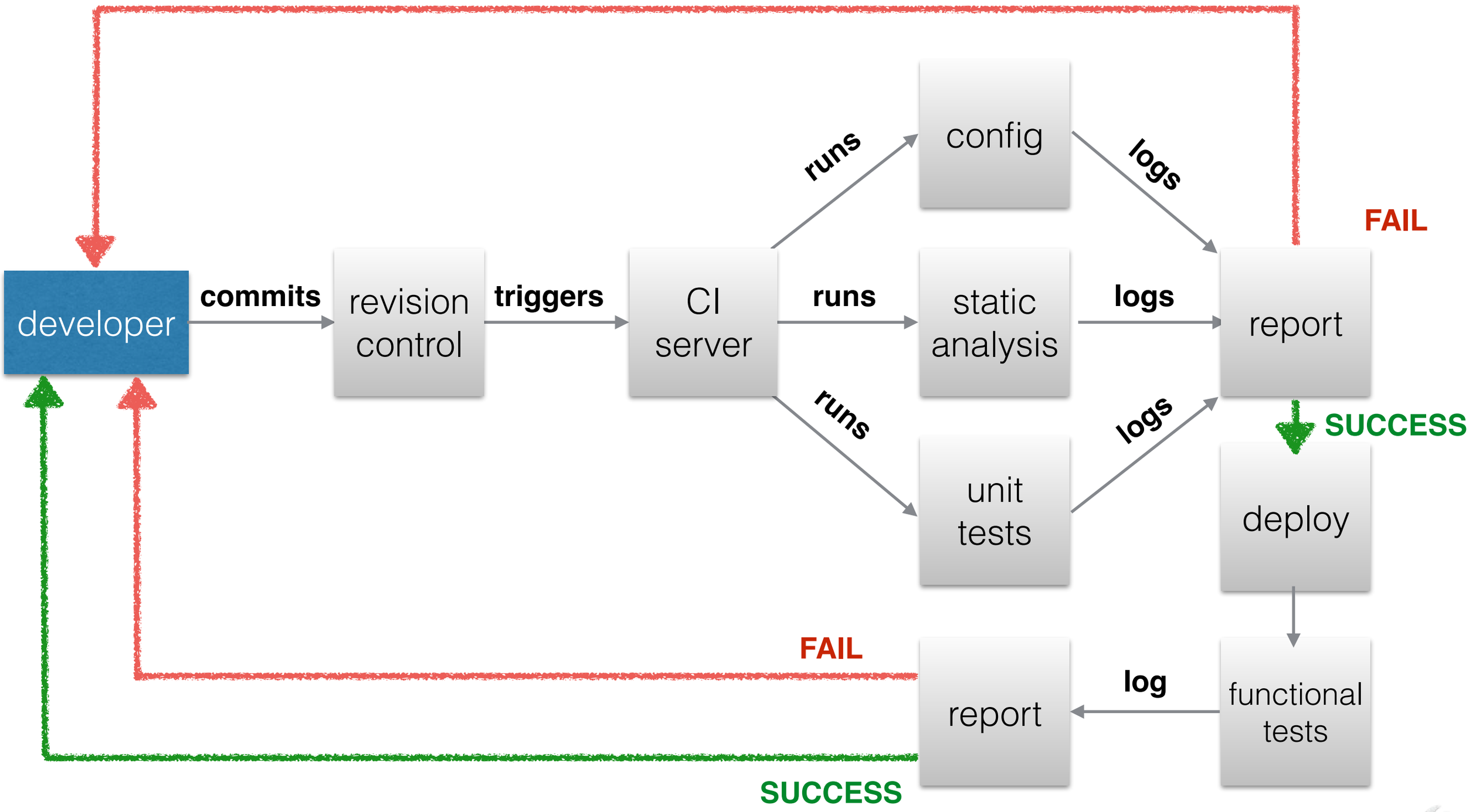
– Martin Fowler

Continuous Integration
ADVANTAGES

- automating repetitive tasks such as build, QA and deploy
- fast and frequent integrations
- quick feedback loop
- ability to deploy quickly and keep track of every released set of code

Continuous Integration

BUILD PROCESS



Continuous Integration
TOOLS

- SCM: git, svn, cvs, …
- CI SERVER: jenkins
- BUILD TOOL: phing
- DRUPAL AUTOMATION: drush
- DEPENDENCIES MANAGEMENT: drush make & composer
- DRUPAL CDD TOOLS: features & migrate
- TESTING: PHPUnit & Behat

Continuous Integration
DRUPAL - I

Drupal, contrary to “classic” frameworks, does not make it easy to setup and maintain a Continuous Integration cycle due to the **high coupling between data and configuration within its database**.

Continuous Integration
DRUPAL - II

To overcome this problem it's necessary to adopt a strictly controlled, disciplined work methodology and to make use of a set of tools that allow us **to transform configuration into code**, **to easily deal with content** and **to incrementally update database's schema**.

Continuous Integration
WORKFLOW - I

A developer's workflow consist not only in the plain old features implementation, but also in the **correct, precise and complete export of all the element of configuration** that are necessary to rebuild the functionalities on several environments.

Key modules: **Features** and **Strongarm**.

Continuous Integration
WORKFLOW - II

It’s often necessary to setup some default content that may vary during development and that might be expanded or updated once the site has been published. It’s therefore necessary **to formalise in code all the data import procedures** in order to better administrate them and to be able to replicate them at any given time.

Key module: **Migrate**.

Continuous Integration
WORKFLOW - III

As soon as a self-contained task such a bug-fix or a new feature implementation has been completed, **it's necessary to commit and push that unity of work to the central repository** in order to integrate, test and deploy it automatically on the development environment.

Key tools: **git/svn**.

Continuous Integration
WORKFLOW - IV

Writing automated tests during development is not only considered a good practice but also a necessity in order to **verify the behaviour** of a given implementation and **to make sure that future changes** -especially when made from other people- **won't break the work** already done, creating regressions e dealing potential damage to users and clients.

Key tools: **PHPUnit**, **Behat**.

Q & A

Continuous Integration
EXERCISES

1. What are the main advantages of a Continuous Integration system?

2. When do Unit Tests are executed during the build? How about functional? Why are they executed at different times?

3. Why is it important to write tests?

4. Which benefits come from using a Source Control Management system?

5. Why is Drupal so hard -compared to other tools such as “classic” frameworks- to use in Continuous Integration environments?

6. Which tools allow Drupal to be successfully used in a Continuous Integration development cycle?