Continuous Integration
CI

15

Continuous Integration

Development practice that requires code be frequently checked into a shared repository.

Each check-in is then verified by an automated build.

- The system is compiled and subjected to an automated test suite, then packaged into a new executable.
- Uses the build script you wrote.

By integrating regularly, developers can detect errors quickly, and locate them more easily.

16

16

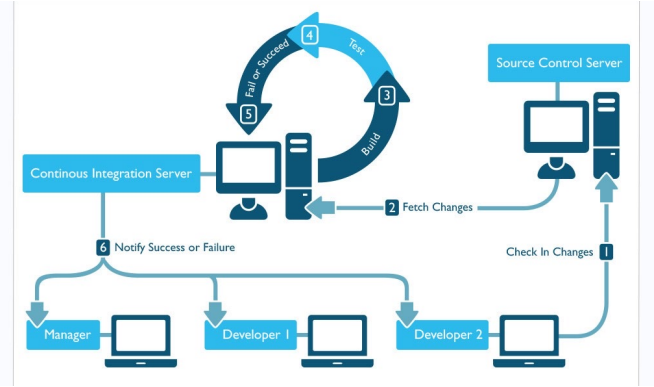
CI Practices

- Maintain a code repository.
- Automate the build.
- Make the build self-testing.
- Every commit should be built.
- Keep the build fast.
- Test in a clone of the production environment.
- Make it easy to get the latest executable.
- Everyone can see build results.
- Automate deployment.

17

17

How Integration is Performed



- Developers check out code to their machine.
- Changes are committed to the repository.
- The CI server:
  - Monitors the repository and checks out changes when they occur.
  - Builds the system and runs unit/integration tests.
  - Releases deployable artefacts for testing.
  - Assigns a build label to the version of the code.
  - Informs the team of the successful build.

18

18

How Integration is Performed

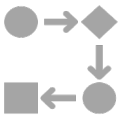
If the build or tests fail, the CI server alerts the team.

The team fixes the issue at the earliest opportunity.

Developers are expected not to check in code they know is broken.

Developers are expected to write and run tests on all code before checking it in.

No one is allowed to check in while a build is broken.



Continue to continually integrate and test throughout the project.

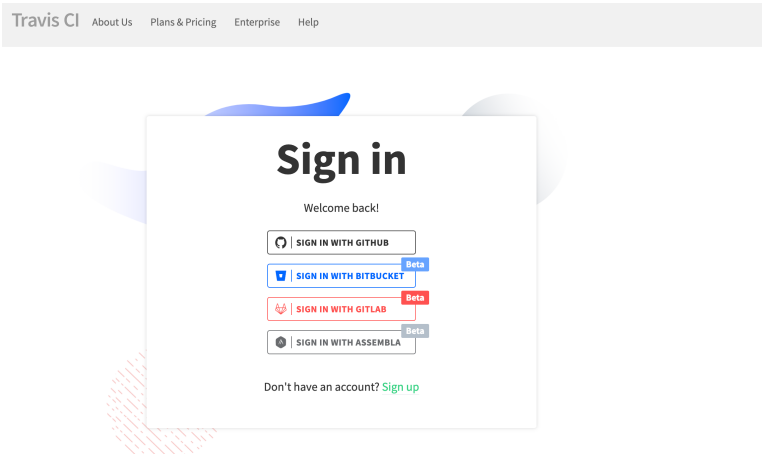
19

19

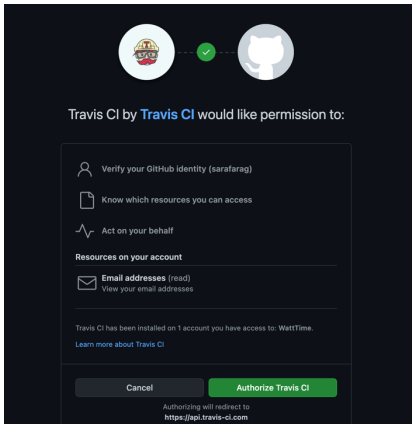
Travis CI

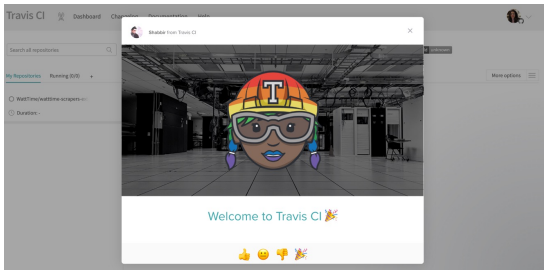
20

Sign In to
https://travis-ci.com



21

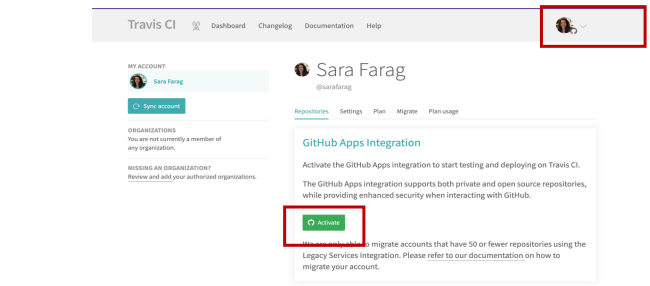


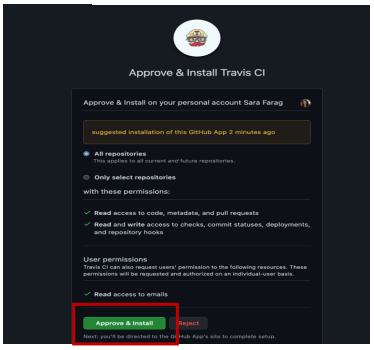


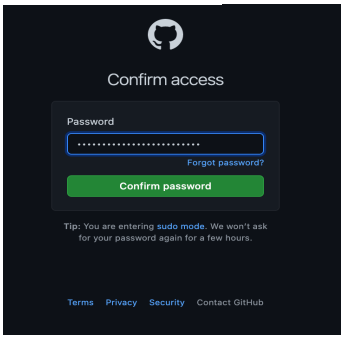
22

1. Click on your Avatar icon to see your profile

2. Choose Activate



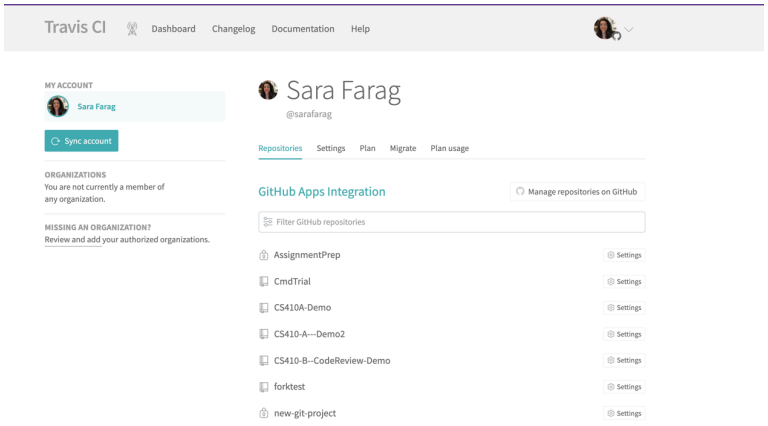




Confirm Password

23

All repositories are showing under your profile



24

TravisCI Process

Code — When code is checked into a repository, TravisCI starts a job.
- An automated process that clones the repository into a virtual environment.
- A job is split into a series of phases.

Install — Installs required dependencies in the virtual environment.

Script — Script: Performs build tasks (compile, test, package, etc.)

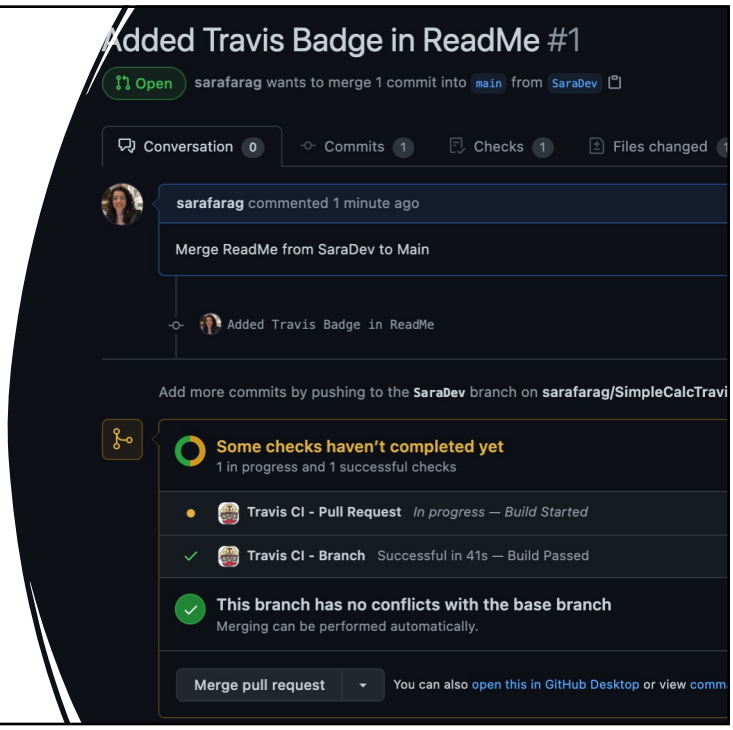
Deploy — Deploy: Deploy code to a production environment (Amazon, Heroku, etc.)

25

25

Travis

- Travis is looking for .travis.yml file on the root directory



26

The TravisCI Configuration File

- Travis uses a config file, **.travis.yml**, to determine how to build the project.
  - **Language** informs TravisCI which language you are developing in.
    - There is a default build process for all supported languages.
  - For Java, the **jdk** field lists the compiler you want to use to build.

```
language: java
jdk: oraclejdk8
install: ...
script: ...
```

28

The TravisCI Configuration File

- Used to determine the OS you want to build on. Supports Linux and MacOS.
- **Addons** are additional programs you need to perform a build.
  - **Apt** is a package manager used in Linux.
  - This example says to install the Maven package before performing the build.

```
os: linux
addons:
  apt:
    packages:
      - maven
```

29

The TravisCI Configuration File

- Env is used to set up environmental variables needed to perform a build.
- Used to perform commands before or after one of the major phases (install, script, deploy).

```
env:
- MY_VAR=EverythingIsAwesome
- NODE_ENV=TEST
```

```
before_install: (after_install, before_script, after_script, etc)
  - ...
```

30

Best Practices

Minimize build time.

- Time spent waiting for results is wasted time.
- Do not make developers wait more than 10 min.
  - If they need to switch tasks, that adds time.
- TravisCI can execute jobs in parallel. Split the test suite into multiple jobs and execute them concurrently in their own virtual environments.

Pull complex logic into shell scripts.

- The configuration file will run any commands you list.
- If your build task is complex, split commands into their own file and call that file.
- Scripts can be run outside of TravisCI too.

32

32

Best Practices

Test multiple language versions for libraries.

Libraries need to operate in multiple version of a language. Make sure you can build in each of them.

You can specify multiple versions in the configuration file (i.e., openjdk8, openjdk9).

- Each will be tried when you build.

Skip unnecessary builds

If you just change documentation or comments, there is no reason to re-test.

Skip commits by adding "[ci skip]" to the commit message.

Can also cancel builds on the TravisCI website.

33

33

34

Jacoco

Code coverage is a software metric used to measure how many lines of our code are executed during automated tests.

JaCoCo mainly provides three important metrics:

- **Lines coverage** reflects the amount of code that has been exercised based on the number of Java byte code instructions called by the tests.
- **Branches coverage** shows the percent of exercised branches in the code – typically related to *if/else* and *switch* statements.
- **Cyclomatic complexity** reflects the complexity of code by giving the number of paths needed to cover all the possible paths in a code through linear combination.

To take a trivial example, if there is no *if* or *switch* statements in the code, the cyclomatic complexity will be 1, as we only need one execution path to cover the entire code.

Generally the cyclomatic complexity reflects the number of test cases we need to implement in order to cover the entire code.

SimpleCalcTravis

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
default		82%		n/a	1	6	2	7	1	6	0	1
Total	4 of 23	82%	0 of 0	n/a	1	6	2	7	1	6	0	1

35

Jacoco

SimpleCalcTravis

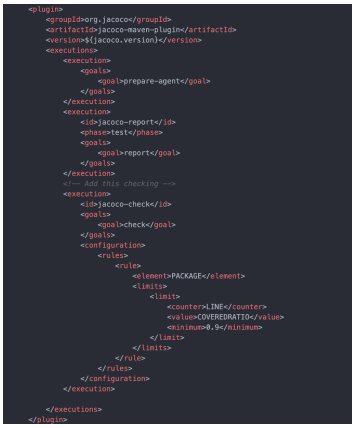
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
default		82%		n/a	1	6	2	7	1	6	0	1
Total	4 of 23	82%	0 of 0	n/a	1	6	2	7	1	6	0	1

36

In the Maven file: Jacoco Config

```xml
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.2</version>
    <executions>
        <execution>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
        <!-- attached to Maven test phase -->
        <execution>
            <id>report</id>
            <phase>test</phase>
            <goals>
                <goal>report</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Declare the following JaCoCo plugin in the pom.xml file.



Make sure lines coverage must meet the minimum 90%.

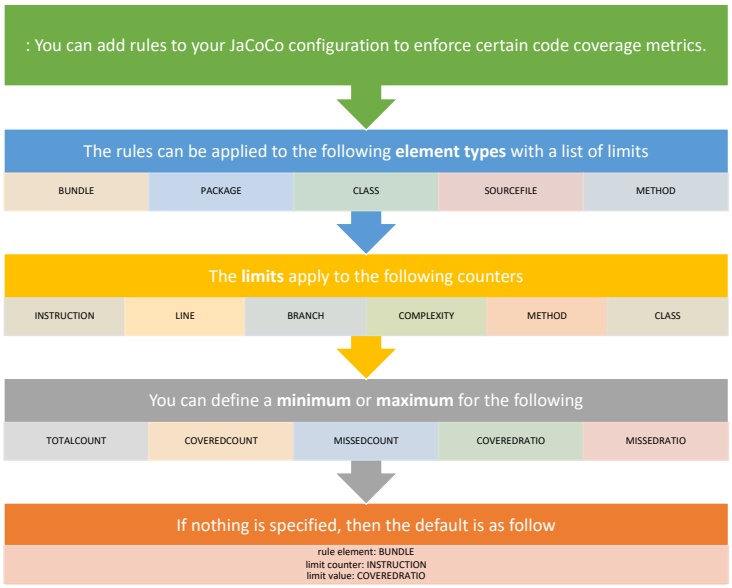
37

How to update the default JaCoCo output folder

```xml
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>${jacoco.version}</version>
    <executions>
        <execution>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
        <execution>
            <id>jacoco-report</id>
            <phase>test</phase>
            <goals>
                <goal>report</goal>
            </goals>
            <!-- default target/jscoco/site/* -->
            <configuration>
                <outputDirectory>target/jacoco-report</outputDirectory>
            </configuration>
        </execution>
    </executions>
</plugin>
```

38

JaCoCo Rules to Enforce code coverage metrics



39

Jacoco Examples

- Here is an example where excludes is used. In this example, a line coverage **minimum of 50%** for every class except test classes is required

```
<rules>
  <rule>
    <element>CLASS</element>
    <excludes>
      <exclude>*Test</exclude>
    </excludes>
    <limits>
      <limit>
        <counter>LINE</counter>
        <value>COVEREDRATIO</value>
        <minimum>50%</minimum>
      </limit>
    </limits>
  </rule>
</rules>
```

```
<rules>
  <rule>
    <element>BUNDLE</element>
    <limits>
      <limit>
        <counter>INSTRUCTION</counter>
        <value>COVEREDRATIO</value>
        <minimum>0.80</minimum>
      </limit>
      <limit>
        <counter>CLASS</counter>
        <value>MISSEDCOUNT</value>
        <maximum>0</maximum>
      </limit>
    </limits>
  </rule>
</rules>
```

- This example requires an overall **instruction coverage** of 80%, and no class must be missed

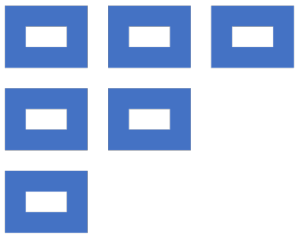
41

RUNNING THE TESTS AND CREATING THE COVERAGE REPORTS

```
mvn clean verify
...
[INFO] --- jacoco-maven-plugin:0.8.3:prepare-agent (coverage-initialize) @ unit-
[INFO] argLine set to -javaagent:/Users/ootero/.m2/repository/org/jacoco/org.jac
...
[INFO] --- maven-surefire-plugin:2.22.1:test (unit-tests) @ unit-integration-tes
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
...
[INFO] --- maven-surefire-plugin:2.22.1:test (integration-tests) @ springboot2-s
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
...
[INFO] --- jacoco-maven-plugin:0.8.3:report (coverage-report) @ unit-integration-
[INFO] Loading execution data file /Users/ootero/Projects/bitbucket.org/unit-int
[INFO] Analyzed bundle 'unit-integration-tests-jacoco-coverage' with 3 classes
...
[INFO] --- jacoco-maven-plugin:0.8.3:check (coverage-check) @ unit-integration-t
[INFO] Loading execution data file /Users/ootero/Projects/bitbucket.org/unit-int
[INFO] Analyzed bundle 'unit-integration-tests-jacoco-coverage' with 3 classes
[INFO] All coverage checks have been met.
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
...
```

42

- While analyzing code coverage reports, the following are some of the benefits a team can reap:
  - Can be used to figure out ways to improve the code or tests for the next phases of development
  - Helps a development team set goals (e.g. getting to a certain amount of coverage, or amount of unit tests that need to be written) and then monitor progress toward those goals
  - Allows your team to evaluate and get a general idea of where potential problem areas or bugs are in your code



43