Continuous Integration/Delivery

Idaho State University | Computer Science

Isaac Griffith

CS 2263
Department of Informatics and Computer Science
Idaho State University

ROAR

Outcomes

After today’s lecture you will be able to:

- Describe the basic practices of CI
- Understand how integration is performed in CI
- Use Travis CI to perform CI

Inspiration

"There should be two tasks for a human being to perform to deploy software into a development, test, or production environment: to pick the version and environment and to press the "deploy" button." – David Farley

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.

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

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.

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.

TravisCI

- CI service that is free of open-source developers, hooked into GitHub
- Connects to a GitHub repository and performs the CI process at specified time.
  - When code is pushed to a repository
  - When a pull request is created
- Adds a “badge” to the GitHub project page displaying the current build status

TravisCI Process

When code is checked into a repository, TravisCI starts a **job**.

- An automated process that clones the repository into a virtual environment.
  - An isolated environment with a clean OS install.
- A job is split into a series of **phases**
  - Sequential steps of a job.
  - Three core phases in TravisCI:
    - **Install:** Installs required dependencies in the virtual environment.
    - **Script:** Performs build tasks (compile, test, package, etc.)
    - **Deploy:** Deploy code to a production environment (Amaxon, Heroku, etc.)

The TravisCI Configuration File

- Travis uses a config file, **.travis.yml**, to determine how to build the project.

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

- **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.

The TravisCI Configuration File

```
os: linux
```

- Used to determine the OS you want to build on. Supports Linux and MacOS.

```
addons:
  apt:
    packages:
      - maven
```

- **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

The TravisCI Configuration Files

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

- Env is used to set up environmental variables needed to perform a build.

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

- Used to perform commands before or after one of the major phases (install, script, deploy).

Install, Script, Deploy

- Major phases specified by listing a set of commands to run.
- If you have a build file, you do not need to explicitly specify commands
  - TravisCI can detect Ant, Maven, and Gradle (among others) build files and has default targets it will run.
  - By default, the script phase will execute "**ant test**"
    - By convention, this will compile and test the project.
    - If you want to execute different targets instead, you can specify this in the configuration file.

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 you build task is complex, split commands into their own file and call that file.
  - Scripts can be run outside of TravisCI too.

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 version 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.

Are there any questions?