About Me

Dragan Rakas

Senior Software Engineer in Test

M. Sc. Computer Science
B. Eng. Computer Engineering

https://www.linkedin.com/in/draganrakas/

I love to watch and play chess!
About Flipp

1. Helps shoppers find the best local flyer deals
2. Extensive back-end API that serves mobile app
3. Highly agile environment (many deploys per week)
Agenda

- Continuous Integration Goals
- Continuous Integration Essentials
- First approach: Downstream Jobs
- Architecture
- Second approach: Declarative Pipeline
- Code Coverage
- Pipeline Optimization
Continuous Integration Goals

1. Early problem detection mitigates risk
2. Encourage frequent code check-ins
3. Providing development feedback as fast as possible
Continuous Delivery Essentials

Before Deployment

- Source Control Integration
- Unit Tests (+Coverage Report)
- Static Analysis and Linting
- Generate API Documentation
- Automated Deploy to Staging
Continuous Delivery Essentials

After Deployment

- Regression Tests (+Coverage Report)
- API Schema / Contract Tests
- Promote to Production
- Health Check: Regression Tests
Branching Strategy

```
master

<table>
<thead>
<tr>
<th>git rebase</th>
<th>git merge</th>
</tr>
</thead>
</table>

development

Pull Request

FB-01

FB-02

master should be locked!
```
GitHub Feedback

- **Review requested**: Review has been requested on this pull request. It is not required to merge. Learn more.

- **Some checks were not successful**: 1 failing and 4 successful checks

  - **ci/circleciEnterprise: regression_test**: Your tests failed on CircleCI Enterprise
  - **ci/circleciEnterprise: build_container**: Your tests passed on CircleCI Enterprise
  - **ci/circleciEnterprise: build_jar**: Your tests passed on CircleCI Enterprise
  - **ci/circleciEnterprise: deploy_to_staging**: Your tests passed on CircleCI Enterprise
  - **ci/circleciEnterprise: unit_test**: Your tests passed on CircleCI Enterprise

- **This branch has no conflicts with the base branch**: Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.
First approach: Downstream Jobs
<table>
<thead>
<tr>
<th>S</th>
<th>W</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>01 Pull Search-API Branch</td>
</tr>
<tr>
<td></td>
<td>☀️</td>
<td>02 Run MW Unit Tests</td>
</tr>
<tr>
<td></td>
<td>☁️</td>
<td>03 Build MW Jar Files</td>
</tr>
<tr>
<td></td>
<td>☀️</td>
<td>04 Deploy Instrumented MW Jar</td>
</tr>
<tr>
<td></td>
<td>☀️</td>
<td>05 Run MW Regression Tests</td>
</tr>
<tr>
<td></td>
<td>☀️</td>
<td>06 Deploy Normal MW Jar</td>
</tr>
<tr>
<td></td>
<td>☀️</td>
<td>07 Run MW Schema Tests</td>
</tr>
</tbody>
</table>
Architecture

GitHub

Web hook

Jenkins

Delegate/Plugin

Slave

docker

pytest

allure

git commit

git push

Pull Request Builder Plugin
Second approach: Declarative Pipeline
Declarative Pipeline

Average stage times:
(Average full run time: ~11min 8s)

Jun 13
11:34
No Changes

Start
Pull Branch
Run Unit Tests
Build Jar Files
Deploy Instrumented Jars
Run Regression Tests
Deploy Normal Jars
Run Schema Tests
End

11m 8s
No changes
4 minutes ago
Started by user Dragan Rakas
Pipeline Script

1. Define each stage programmatically
2. Belongs in repository as ‘Jenkinsfile’
3. Can work on any Jenkins instance
4. More scalable than downstream jobs
Regression Test Coverage Strategy

Compile:
- Instrument codebase
- Create instrumented binary

Run Tests:
- Instrumented binary writes stats to file

Deploy:
- Instrumented binary

Coverage Report:
- Same as unit test coverage reports
Coverage Example

```c
int fibcache[1000]; //initially 0s

int fib(int i) //fast Fibonacci
{
    int t;

    switch(i)
    {
        case 0:
        case 1: return 1;
        default:
            if (fibcache(i))
            {
                return fibcache(i);
            }
            else
            {
                t = fib(i - 1);
                fibcache(i) = t + fib(i - 2);
                return fibcache(i);
            }
    }
};
```
Instrumentation

```c
int fibcache[1000]; //initially 0s

int fib(int i) //fast Fibonacci
{
    int t;
    visited[1] = 1;
    switch(i)
    {
        case 0: visited[2] = 1;
        case 1: visited[3] = 1; return 1;
        default:
            visited[4] = 1;
        if (fibcache(1))
        {
            visited[5] = 1;
        return fibcache(i);
        }
        else
        {
            visited[6] = 1;
        t = fib(i - 1)
        fibcache(i) = t + fib(i - 2)
        return fibcache(i);
        }
    }
    visited[7] = 1;
}
```
Integration Test Coverage Strategy

Compile:
- Instrument codebase
- Create instrumented binary

Deploy:
- Instrumented binary

Run Tests:
- Instrumented binary writes stats to file

Coverage Report:
- Same as unit test coverage reports
Pipeline Optimization
“If your build is reproducible, the outputs from one machine can be safely reused on another machine, which can make builds significantly faster.” - Bazel Documentation

Example:
- Application has 30 .jar file dependencies
- A code change is pushed to 1 dependent module

Solution:
- Load cached 30 .jar files from data store (e.g. from S3)
- Have a script to detect which module’s code changed
- Build and replace only the 1 modified dependency
Which is longer? Which one should go first?
- Running 50,000 unit tests
- Running static analysis
- Running mutation tests
- Building a .jar file

It shouldn’t matter!
- All of the above are independent and can often be done at the same time!
Some companies have huge test suites (100,000+ tests)

Categorize tests by priority and impact (sanity, smoke, regression, etc.)

Run critical path tests for immediate feedback

Schedule periodic builds for full regression test runs

@ pytest.mark.sanity
def test.middleware_is_running_correct_version(environment_host):
Questions?