Building an Effective CI Pipeline

Dragan Rakas
June 15, 2018

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

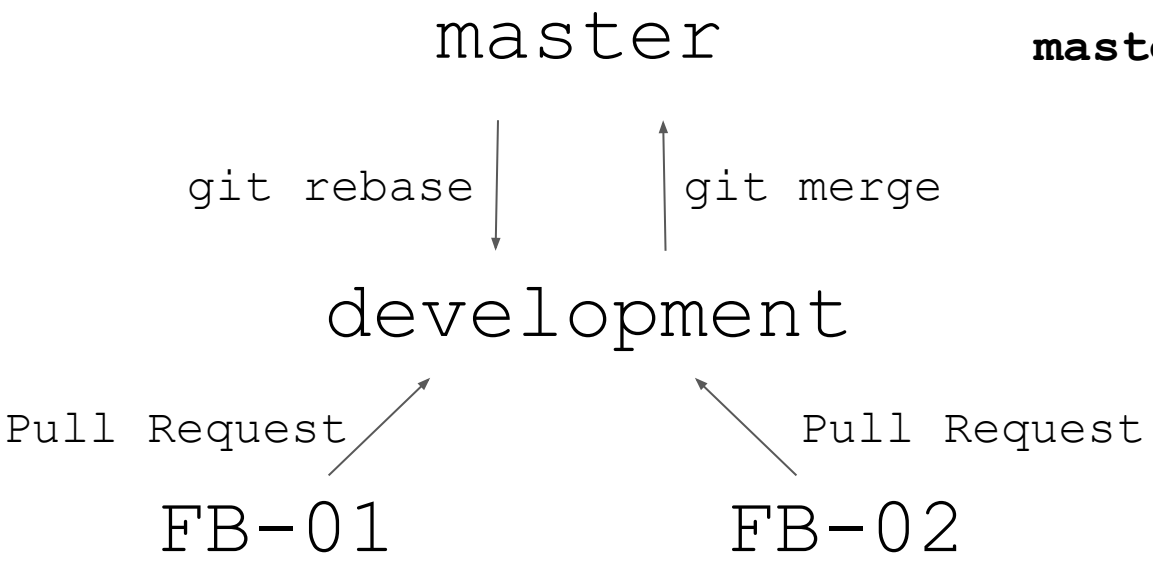

- Regression Tests (+Coverage Report)


- API Schema / Contract Tests


- Promote to Production

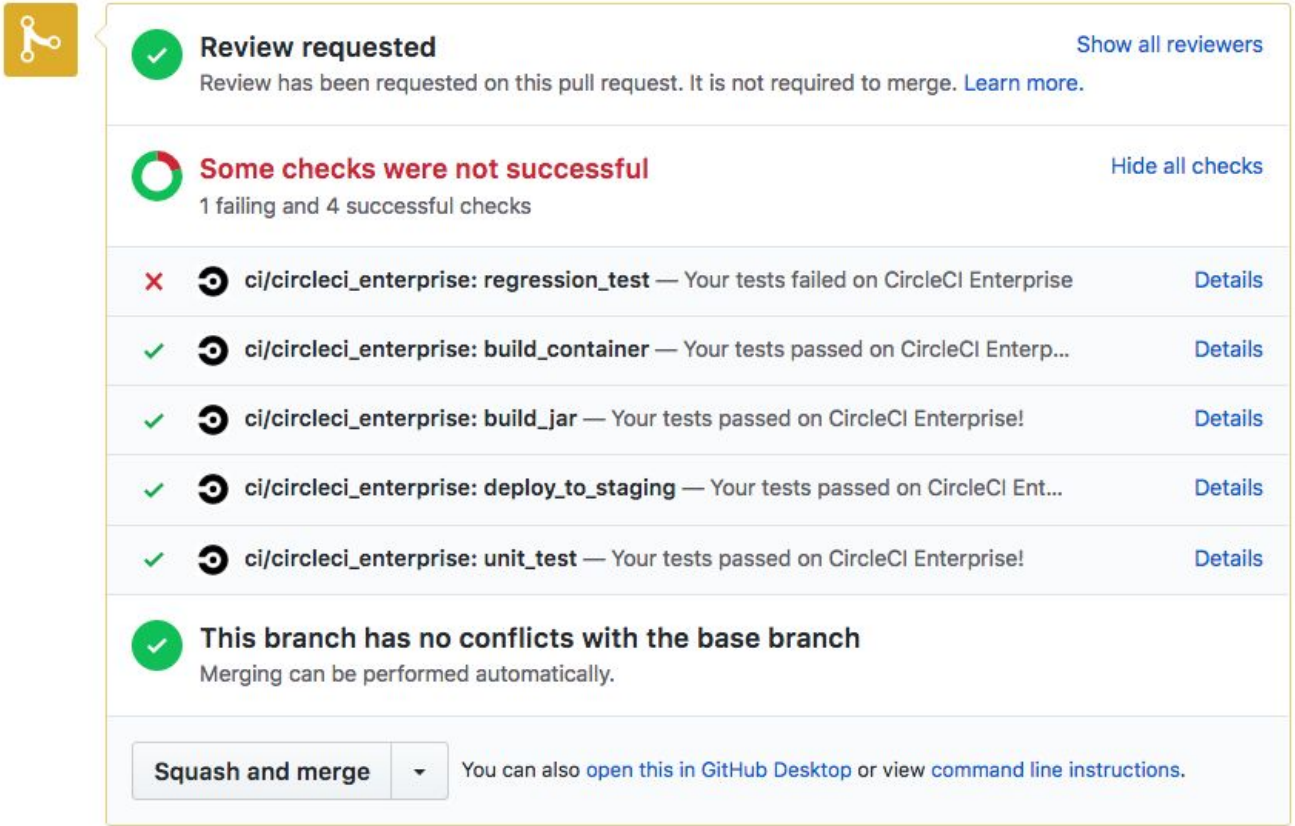

- **Health Check**: Regression Tests

Branching Strategy



`master` should be locked!

GitHub Feedback

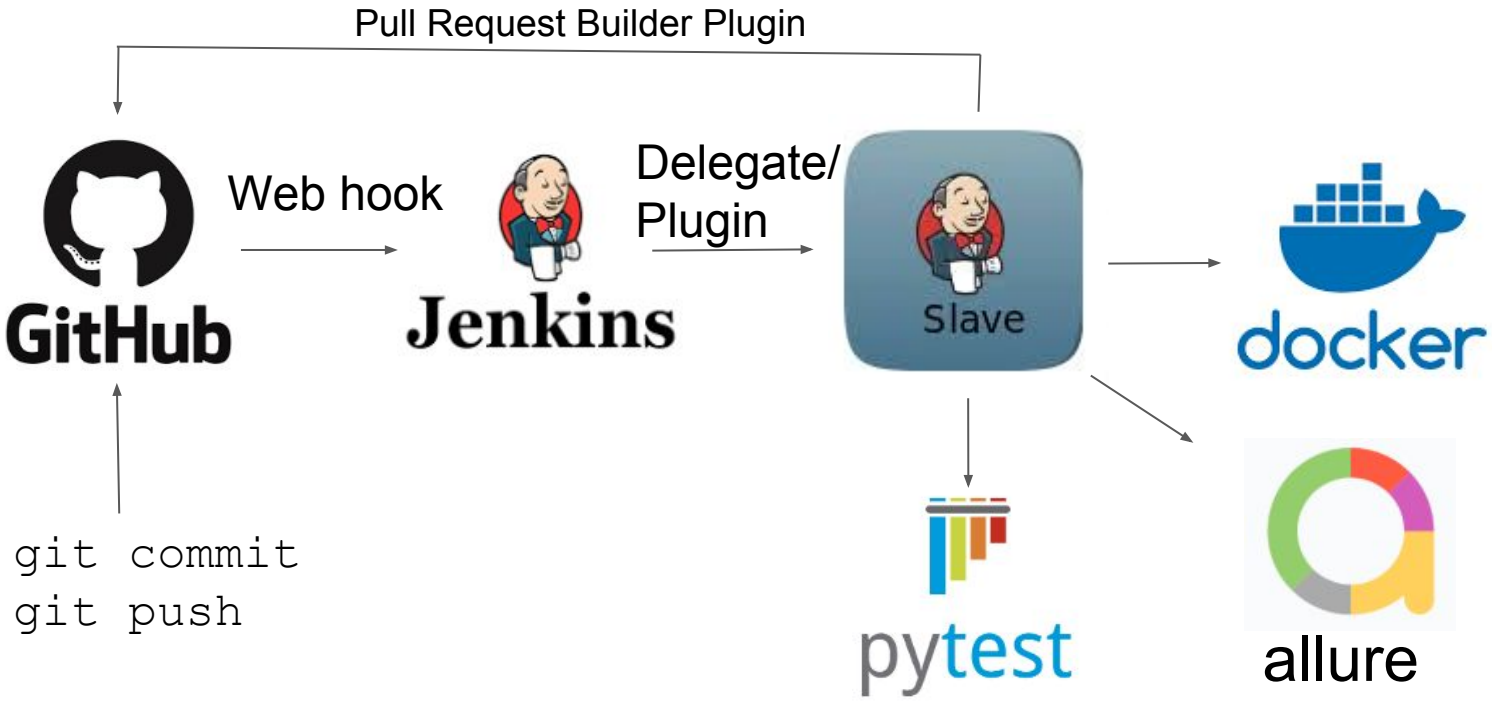


First approach: Downstream Jobs

All +

S	W	Name ↓
		01 Pull Search-API Branch
		02 Run MW Unit Tests
		03 Build MW Jar Files
		04 Deploy Instrumented MW Jar
		05 Run MW Regression Tests
		06 Deploy Normal MW Jar
		07 Run MW Schema Tests

Architecture



Second approach: Declarative Pipeline

Declarative Pipeline

	Pull Branch	Run Unit Tests	Build Jar Files	Deploy Instrumented Jars	Run Regression Tests	Deploy Normal Jars	Run Schema Tests
Average stage times: (Average full run time: ~11min 8s)	2s	58s	1min 52s	25s	2min 7s	22s	5min 21s
#3 Jun 13 11:34 — No Changes	2s	58s	1min 52s	25s	2min 7s	22s	5min 21s

11m 8s — No changes

4 minutes ago — Started by user Dragan Rakas

Pipeline Script

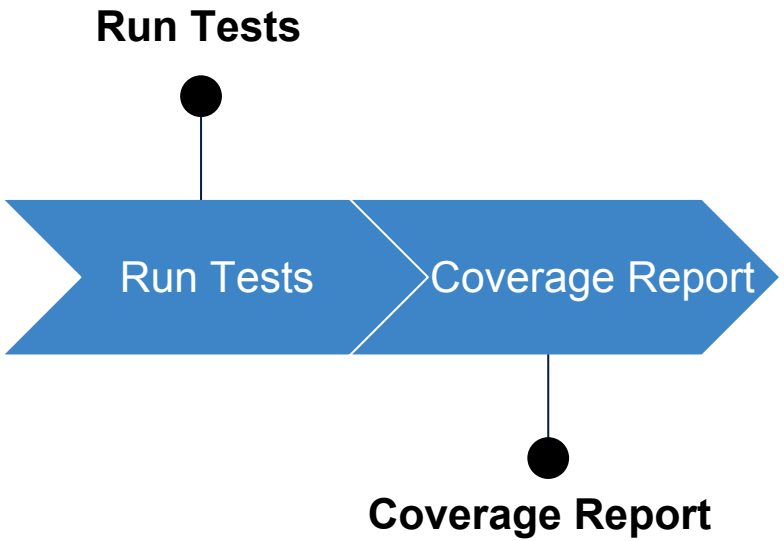
1. Define each stage programmatically
2. Belongs in repository as ‘Jenkinsfile’
3. Can work on any Jenkins instance
4. More scalable than downstream jobs

24 lines (20 sloc) | 368 Bytes

```
pipeline {
    agent { label 'Jenkins Slave' }

    stages {
        stage('Pull Branch'){
            steps {
                script {
                # .. Shell Script ..
                }
            }
        }

        stage('Run Unit Tests') {
            steps {
                script {
                # .. Shell Script ..
                }
            }
        }

        # .. Remaining Pipeline Stages ..
    }
}
```


flipp
Code Coverage

Unit Test Coverage Strategy



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

flipp

Coverage Example

```
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

```
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(i))
               {
                    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

Run Tests:
- Instrumented binary writes stats to file

Deploy:
- Instrumented binary

Coverage Report:
- Same as unit test coverage reports

Pipeline Optimization

Caching

- “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

flipp

Parallelizing

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!

flipp

Test Categorization

- 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?