Continuous Deployment to the Cloud with Spring Cloud Pipelines

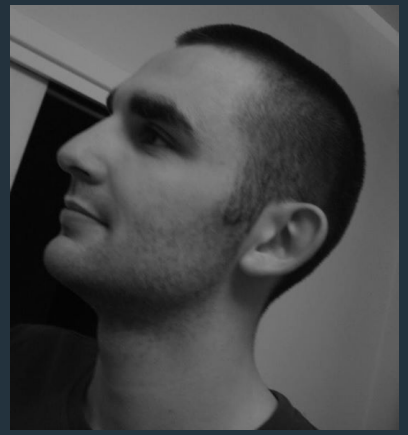
Marcin Grzejszczak, @mgrzejszczak

About me

Spring Cloud developer at Pivotal

Working mostly on

- Spring Cloud Sleuth
- Spring Cloud Contract
- Spring Cloud Pipelines



Twitter: @mgrzejszczak

Blog: http://toomuchcoding.com

Pivotal

Agenda

Why do we deploy software?

Opinionated pipeline presentation with demo

Agenda

Why do we deploy software?

Opinionated pipeline presentation with demo

What are the developers paid for?

What are the devs paid for?

- business doesn’t care about the used technologies
- they don’t care what Java / PHP / Clojure is - they care about money
- and IT
  - costs a LOT of money
  - as an industry we’re not good at meeting deadlines
  - tends to be a big risk

IT projects are much more riskier than we think

Study by Flyvbjerg and Budzier in the Harvard Business Review (2011) [1]

- 1400+ projects surveyed
- ~27% were over budget
- 16%
  - had cost over 200% of they were supposed to
  - have passed due dates by almost 70%
  - were a complete disaster and strained company budgets dramatically

[1] - https://hbr.org/2011/09/why-your-it-project-may-be-riskier-than-you-think

IT projects are much more riskier than we think

http://giphy.com/gifs/college-field-study-YJjvTqoRFgZaM

spring

Why typically business is unhappy with the developers?

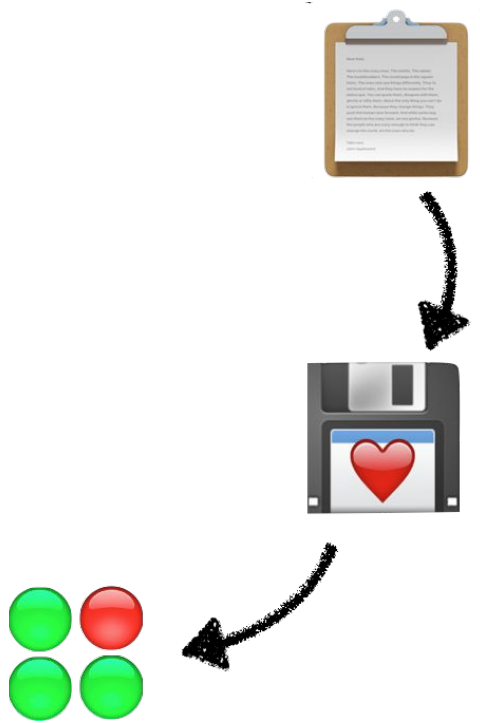
Responsibility and late feedback

Limited dev feedback a traditional development process

Developers stop being responsible for anything besides pushing code

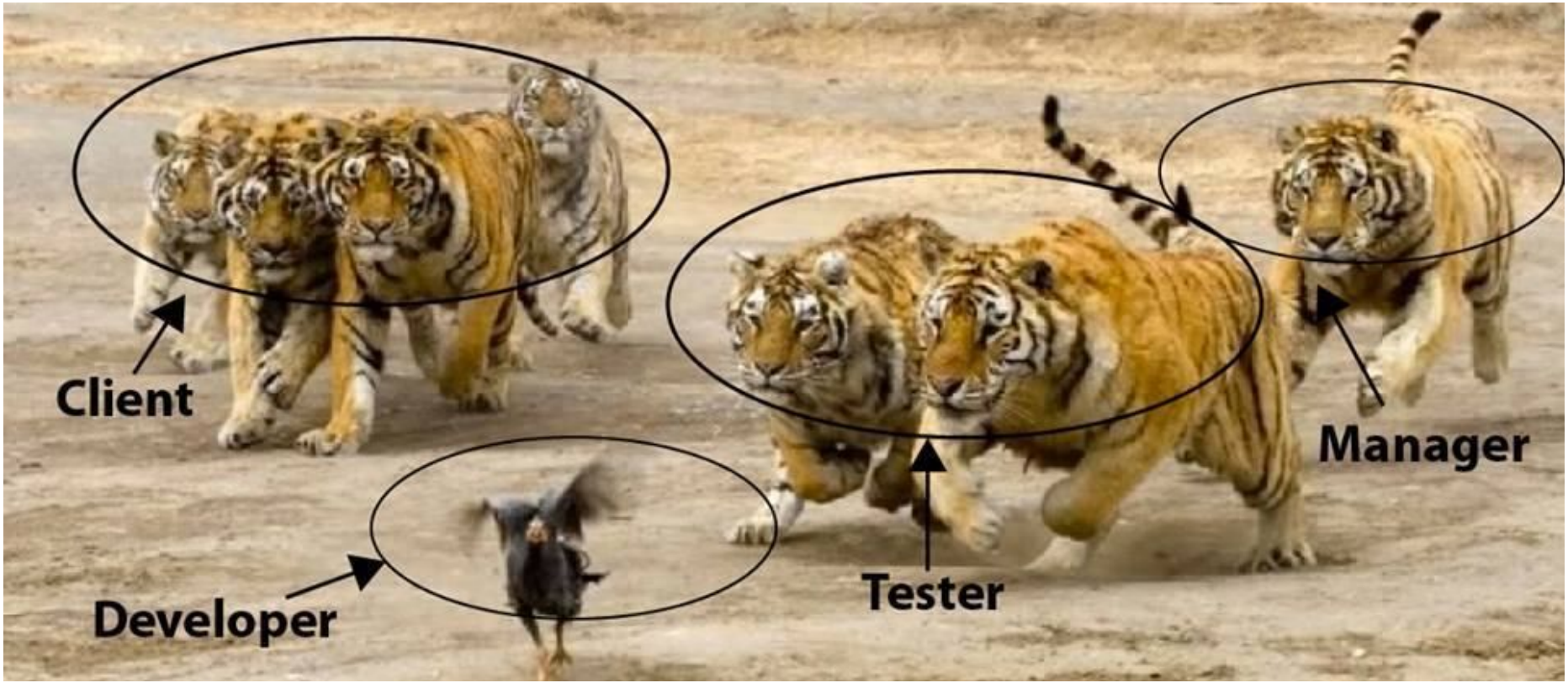
https://raygun.com/blog/2016/07/software-testing/

spring



https://www.youtube.com/watch?v=7APZD0me1nU by Onsi Fakhouri

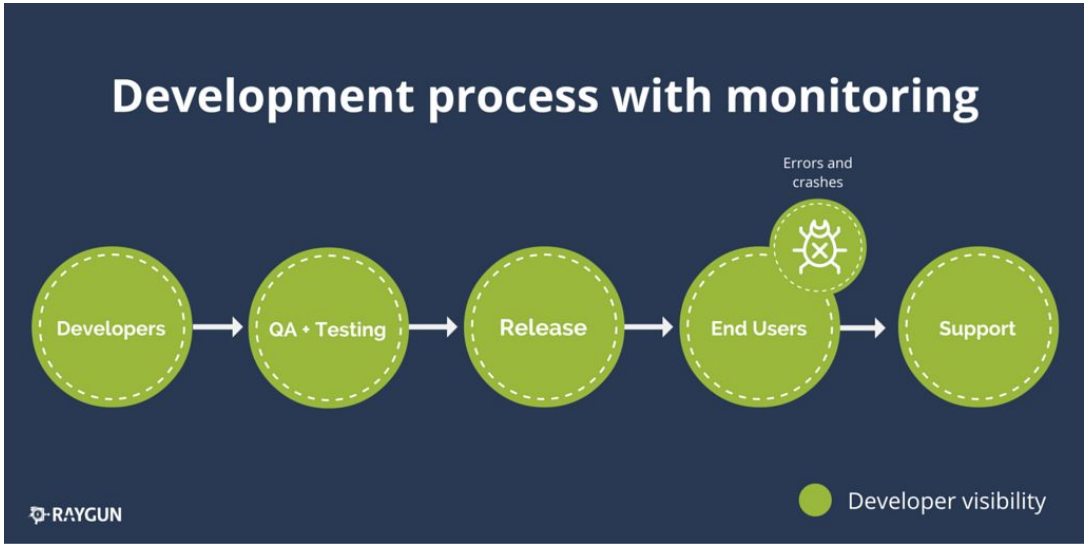
Limited dev feedback a traditional development process



spring

It’s not always dev’s fault...

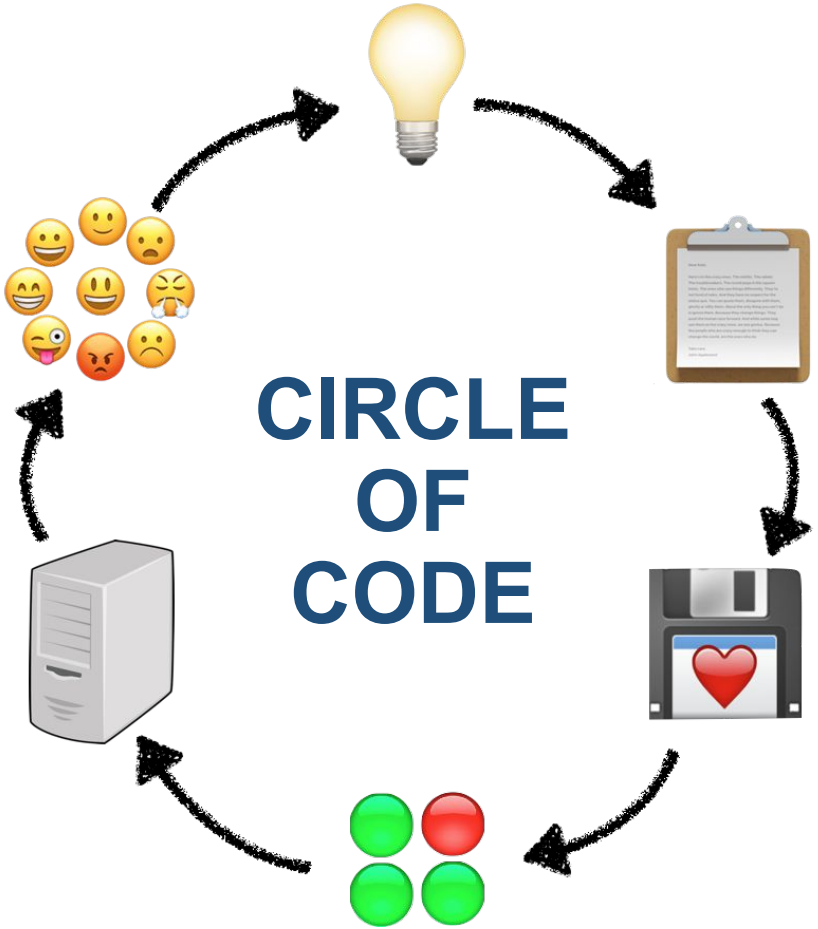
- Developers love their software
- They don't run away from responsibility
- They often don't have the possibility to take it



https://raygun.com/blog/2016/07/software-testing/

spring

Developers want to apply XP and agile approaches



https://www.youtube.com/watch?v=7APZD0me1nU by Onsi Fakhouri

spring

Potential solution to making business happy?

Deliver small chunks of features iteratively instead of doing rare gigantic deployments

But actually… when is a feature delivered?

When is a feature delivered?

- when it’s written?
- when it’s written, unit and integration tested?
- when it’s written, unit and integration tested and passed QA tests?
- none of the above?

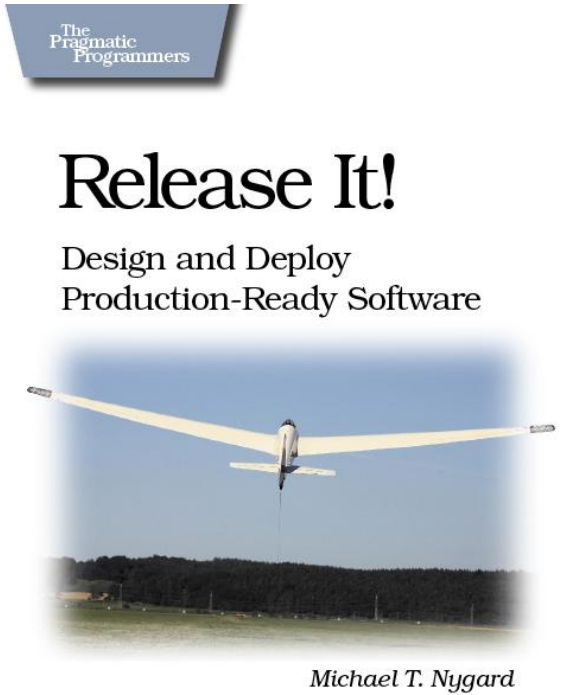
The feature is delivered when it’s deployed to production!

Speaking of deployment...

- Who is sending a deployment package via email?
- Who is passing a deployment package via USB stick?
- Who is deploying via scripts and SCP?
- Who is using some tool like Ansible / Puppet / Chef?

For those who deploy manually...

Release It!



Release It!: Design and Deploy Production-Ready Software (Pragmatic Programmers) 1st Edition by Michael T. Nygard
https://www.amazon.com/Release-Production-Ready-Software-Pragmatic-Programmers/dp/0978739213

spring

Why is saving money on automation a terrible idea?

If you can spend $5,000 on an automated build and release system that avoids downtime during releases, the company will avoid $200,000. I think that most CFOs would not mind authorizing an expenditure that returns 4,000% ROI. - Michael T. Nygard, Release It!

This assumes $10,000 per release (labor plus cost of planned downtime), four releases per year, and a five-year horizon. Most companies would like to do more than four releases per year, but I'm being conservative.

Release It!: Design and Deploy Production-Ready Software (Pragmatic Programmers) 1st Edition by Michael T. Nygard
https://www.amazon.com/Release-Production-Ready-Software-Pragmatic-Programmers/dp/0978739213

Manual steps == potential mistake

- Have you ever removed sth by accident on production?
- Have you had to write your app from scratch (not stored in SCM)?
- Have you ever had to rollback anything from a backup?

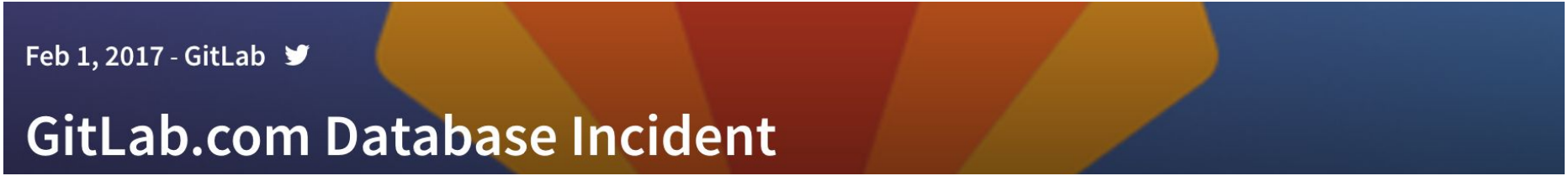
Manual steps == potential mistake

Have you heard about the Gitlab.com database issue [1] ?

[1] - https://about.gitlab.com/2017/02/01/gitlab-dot-com-database-incident/

spring

Gitlab.com database issue



Trying to restore the replication process, an engineer proceeds to wipe the PostgreSQL database directory, errantly thinking they were doing so on the secondary. Unfortunately this process was executed on the primary instead. The engineer terminated the process a second or two after noticing their mistake, but at this point around 300 GB of data had already been removed.

https://about.gitlab.com/2017/02/01/gitlab-dot-com-database-incident/

spring

Why shouldn’t you even need to ssh to a machine?

Every single time a human touches a server is an opportunity for unforced errors. It’s best to keep people off the production systems to the greatest extent possible. - Michael T. Nygard, Release It!

Release It!: Design and Deploy Production-Ready Software (Pragmatic Programmers) 1st Edition by Michael T. Nygard
https://www.amazon.com/Release-Production-Ready-Software-Pragmatic-Programmers/dp/0978739213

Ok, but we have backups...

Gitlab.com database issue

Problems Encountered

- LVM snapshots are by default only taken once every 24 hours. *Team-member-1* happened to run one manually about six hours prior to the outage because he was working in load balancing for the database.
- Regular backups seem to also only be taken once per 24 hours, though *team-member-1* has not yet been able to figure out where they are stored. According to *team-member-2* these don't appear to be working, producing files only a few bytes in size.
- *Team-member-3*: It looks like `pg_dump` may be failing because PostgreSQL 9.2 binaries are being run instead of 9.6 binaries. This happens because omnibus only uses Pg 9.6 if data/PG_VERSION is set to 9.6, but on workers this file does not exist. As a result it defaults to 9.2, failing silently. No SQL dumps were made as a result. Fog gem may have cleaned out older backups.
- Disk snapshots in Azure are enabled for the NFS server, but not for the DB servers.
- The synchronisation process removes webhooks once it has synchronised data to staging. Unless we can pull these from a regular backup from the past 24 hours they will be lost
- The replication procedure is super fragile, prone to error, relies on a handful of random shell scripts, and is badly documented
- Our backups to S3 apparently don't work either: the bucket is empty
- So in other words, out of five backup/replication techniques deployed none are working reliably or set up in the first place. We ended up restoring a six-hour-old backup.
- pg_basebackup will silently wait for a master to initiate the replication progress, according to another production engineer this can take up to 10 minutes. This can lead to one thinking the process is stuck somehow. Running the process using "strace" provided no useful information about what might be going on.

https://about.gitlab.com/2017/02/01/gitlab-dot-com-database-incident/

Gitlab.com database issue

Problems Encountered

- LVM snapshots are by default only taken once every 24 hours. *Team-member-1* happened to run one manually about six hours prior to the outage because he was working in load balancing for the database.
- Regular backups seem to also only be taken once per 24 hours, though *team-member-1* has not yet been able to figure out where they are stored. According to *team-member-2* these don't appear to be working, producing files only a few bytes in size.
- *Team-member-3*: It looks like `pg_dump` may be failing because PostgreSQL 9.2 binaries are being run instead of 9.6 binaries. This happens because omnibus only uses Pg 9.6 if data/PG_VERSION is set to 9.6, but on workers this file does not exist. As a result it defaults to 9.2, failing silently. No SQL dumps were made as a result. Fog gem may have cleaned out older backups.
- Disk snapshots in Azure are enabled for the NFS server, but not for the DB servers.
- The synchronisation process removes webhooks once it has synchronised data to staging. Unless we can pull these from a regular backup from the past 24 hours they will be lost
- The replication procedure is super fragile, prone to error, relies on a handful of random shell scripts, and is badly documented
- Our backups to S3 apparently don't work either: the bucket is empty
- So in other words, out of five backup/replication techniques deployed none are working reliably or set up in the first place. We ended up restoring a six-hour-old backup.
- `pg_basebackup` will silently wait for a master to initiate the replication process, according to another production engineer this can take up to 10 minutes. This can lead to one thinking the process is stuck somehow. Running the process using "strace" provided no useful information about what might be going on.

https://about.gitlab.com/2017/02/01/gitlab-dot-com-database-incident/

Gitlab.com database issue

- So in other words, out of five backup/replication techniques deployed none are working reliably or set up in the first place. We ended up restoring a six-hour-old backup.

https://about.gitlab.com/2017/02/01/gitlab-dot-com-database-incident/

spring

Conclusion?

Deployment pipelines should automatically deploy software and test rollback possibilities

What other problems stop us from deploying to production?

Deployment problems

- Non-working hours deployments
  - there's nobody there to really test your application
  - **SOLUTION: deploy during working hours**
- Rare deployments
  - low chances of rolling back due to DB changes
  - statistically more bugs comparing with more frequent deploys
  - **SOLUTION: deploy more often**
- Being scared of failure
  - embrace failure - there will be bugs / sth will go wrong
  - you can spend 10 months on UATs, still things will blow up on production
  - you can use feature toggles to enable features
  - **SOLUTION: deploy more often**

Scared of failure? Zero downtime deployment / AB testing could help you!

Benefits of zero downtime deployment

For many of my clients, the direct cost of downtime exceeds $100,000 per hour. In one year the difference between 98% uptime and 99.99% uptime adds up to more than $17 million. Imagine adding $17 million to the bottom line just through better design! - Michael T. Nygard, Release It [1]

[1] - Release It!: Design and Deploy Production-Ready Software (Pragmatic Programmers) 1st Edition by Michael T. Nygard
https://www.amazon.com/Release-Production-Ready-Software-Pragmatic-Programmers/dp/0978739213

Downtime costs

Availability	98%	99.99%
Downtime Min/Month	864	4
Downtime $/Month	$21,600 ($1,500 per h)	$108
Added Cost (5 years)	$0	$98,700
Net Savings (5 years)	$0	$1,289,520

Release It!: Design and Deploy Production-Ready Software (Pragmatic Programmers) 1st Edition by Michael T. Nygard
https://www.amazon.com/Release-Production-Ready-Software-Pragmatic-Programmers/dp/0978739213

You could always also roll back to a previous version if something goes wrong!

Why do we deploy software?

- We're paid for delivering business value
- Features are done when they are deployed to production
- It's better to deploy frequently for faster feedback
  - You can apply XP patterns in deployment
- It's better to fail the deployment pipeline as fast as possible
- Deployments should take place in a standardised, automated fashion
- Your deployment pipeline should test rollback
  - That way you can perform A/B or zero downtime deployment to production

Agenda

Why do we deploy software?

Opinionated pipeline presentation with demo

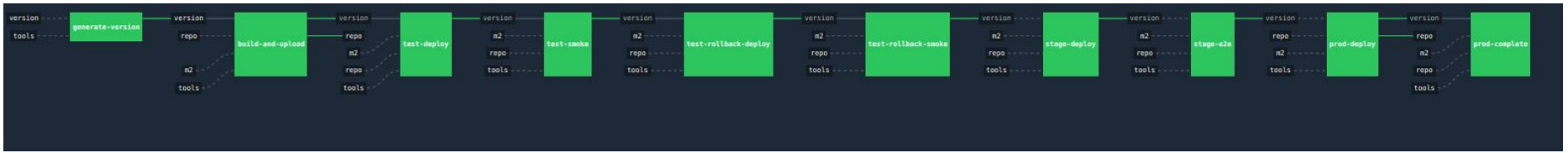
Spring Cloud Pipelines

- opinionated template of a deployment pipeline
- based on good practices from different projects
- we believe in the “Infrastructure as Code” approach
  - live documentation of the infrastructure
  - in case of automation server going down you can recover everything in no time
  - the automation server setup should be automated too!
  - you can even write tests for your pipelines
- no more clicking of jobs manually!

Spring Cloud Pipelines

- we support following automation servers out of the box
  - Concourse
  - Jenkins using Jenkins Job DSL plugin
  - Jenkins using Jenkinsfile with Blue Ocean
- logic of all steps done via Bash scripts
  - you can convert the internals to suit your needs
  - you can use whatever automation server you want
  - supports Maven & Gradle

Spring Cloud Pipelines - Concourse

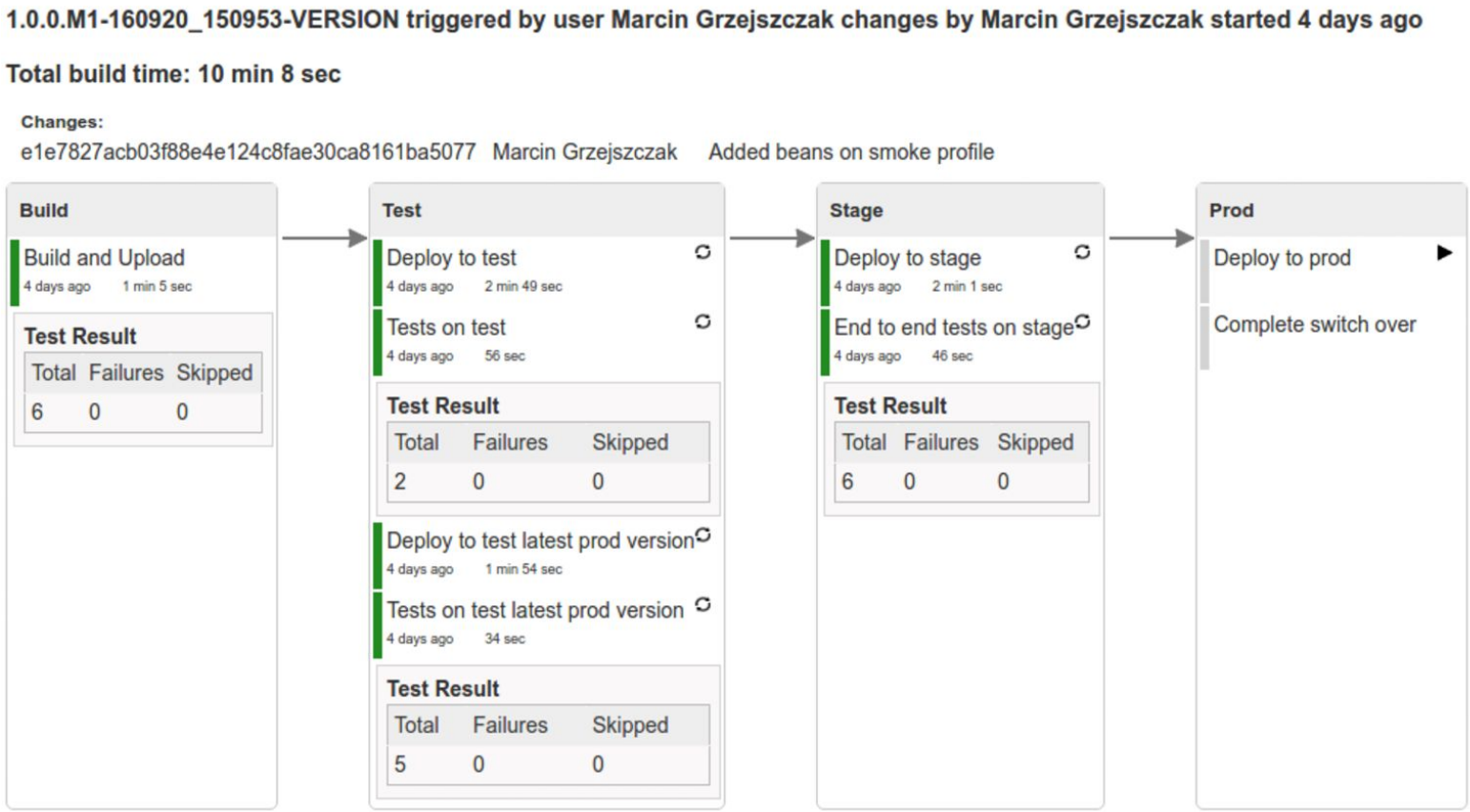


spring

Concourse

```
- name: build-and-upload
  serial: true
  public: false
  plan:
    - aggregate:
      - get: tools
      - get: repo
      - get: m2
      - get: version
        resource: version
        passed: [ generate-version ]
        trigger: true
    - task: build-and-upload
      file: tools/concourse/tasks/build-and-upload.yml
      params:
        - _JAVA_OPTIONS: -Djava.security.egd=file:/dev/./urandom
        - GIT_EMAIL: {{git-email}}
        - GIT_NAME: {{git-name}}
        - REDOWNLOAD_INFRA: {{redownload-infra}}
        - REDEPLOY_INFRA: {{redeploy-infra}}
        - M2_REPO: {{maven-local-dir}}
    - put: repo
      params:
        repository: out
        tag: out/tag
        only_tag: true
```

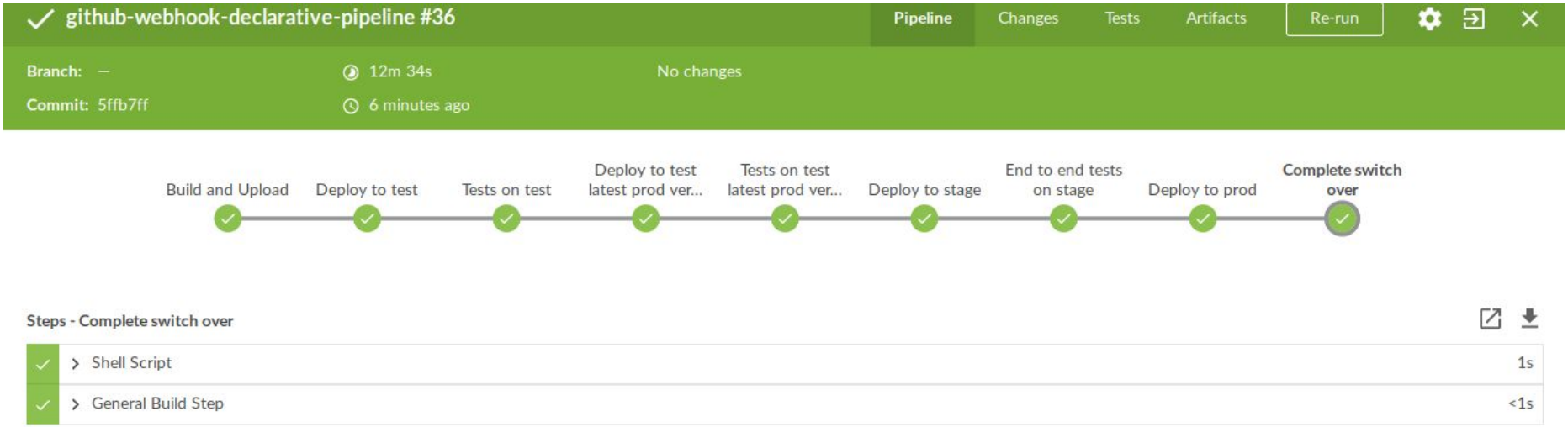
Spring Cloud Pipelines - Jenkins Job DSL



Jenkins Job DSL

```
dsl.job("${projectName}-build") {
        deliveryPipelineConfiguration('Build', 'Build and Upload')
        triggers {
                cron(cronValue)
                githubPush()
        }
        wrappers {
                deliveryPipelineVersion(pipelineVersion, true)
                environmentVariables {
                        environmentVariables(defaults.defaultEnvVars)
                        groovy(PipelineDefaults.groovyEnvScript)
                }
                parameters(PipelineDefaults.defaultParams())
                timestamps()
                colorizeOutput()
                maskPasswords()
                timeout {
                        noActivity(300)
                        failBuild()
                        writeDescription('Build failed due to timeout after {0} minutes of inactivity')
                }
        }
        jdk(jdkVersion)
```

Spring Cloud Pipelines - Jenkinsfile & Blue Ocean



Jenkinsfile

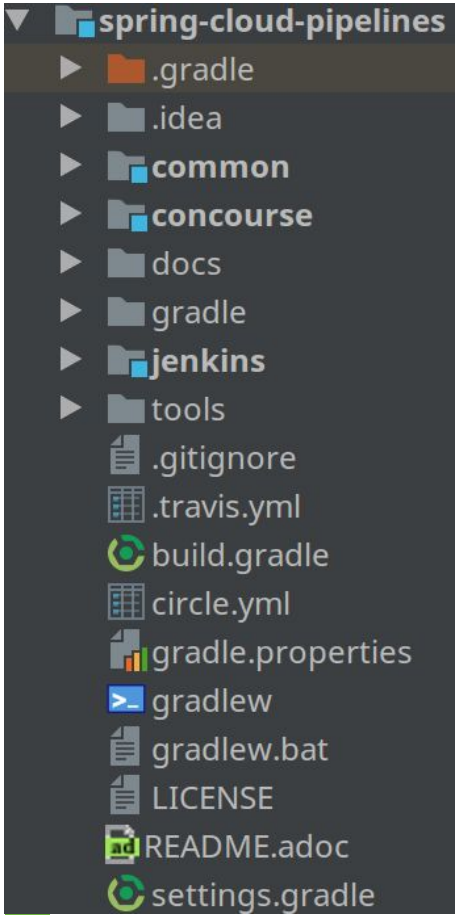
```
stage("Deploy to test") {
        steps {
                sh '''#!/bin/bash
                ${WORKSPACE}@tools/common/src/main/bash/test_deploy.sh
                '''
        }
}

stage("Tests on test") {
        steps {
                sh '''#!/bin/bash
                ${WORKSPACE}@tools/common/src/main/bash/test_smoke.sh
                '''
        }
}

stage("Deploy to test latest prod version") {
        steps {
                sh '''#!/bin/bash
                ${WORKSPACE}@tools/common/src/main/bash/test_rollback_deploy.sh
                '''
        }
}
```

SPRING CLOUD PIPELINES REPOSITORY DEMO

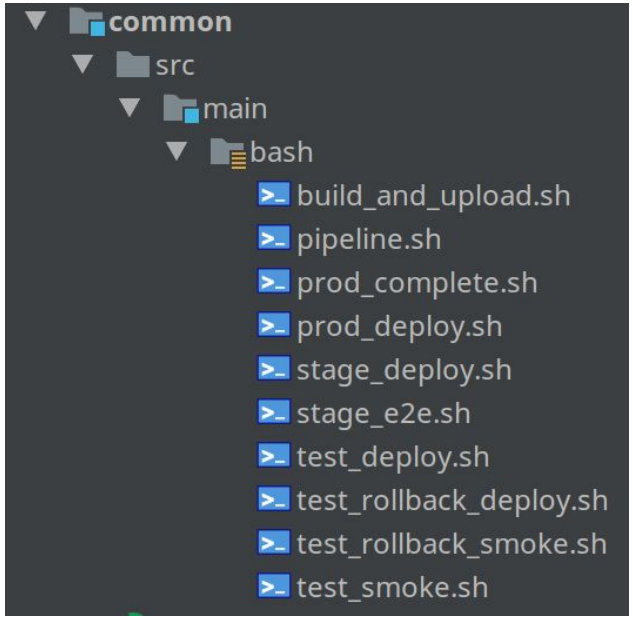
Spring Cloud Pipelines - repo



- **common** - contains all logic in bash scripts
- **concourse** - tasks for Concourse
- **jenkins** - seed jobs and job definitions for Jenkins
- **tools** - helpful scripts for publishing of infrastructure artifacts and PCF Dev manipulation

spring

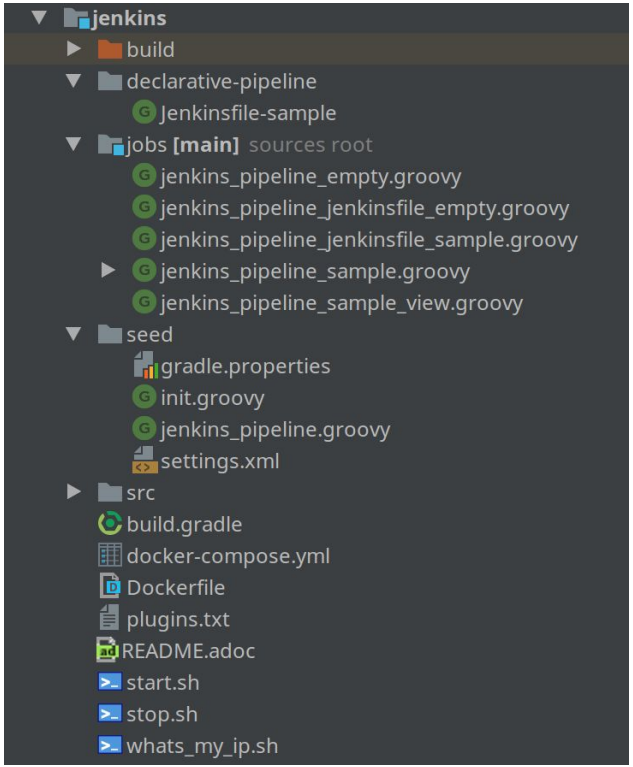
Spring Cloud Pipelines - repo



- **pipeline.sh** - contains all functions in Bash
- **all the other scripts** - represent a single step inside the pipeline. It's enough to pass environment variables to make the script work

spring

Spring Cloud Pipelines - repo

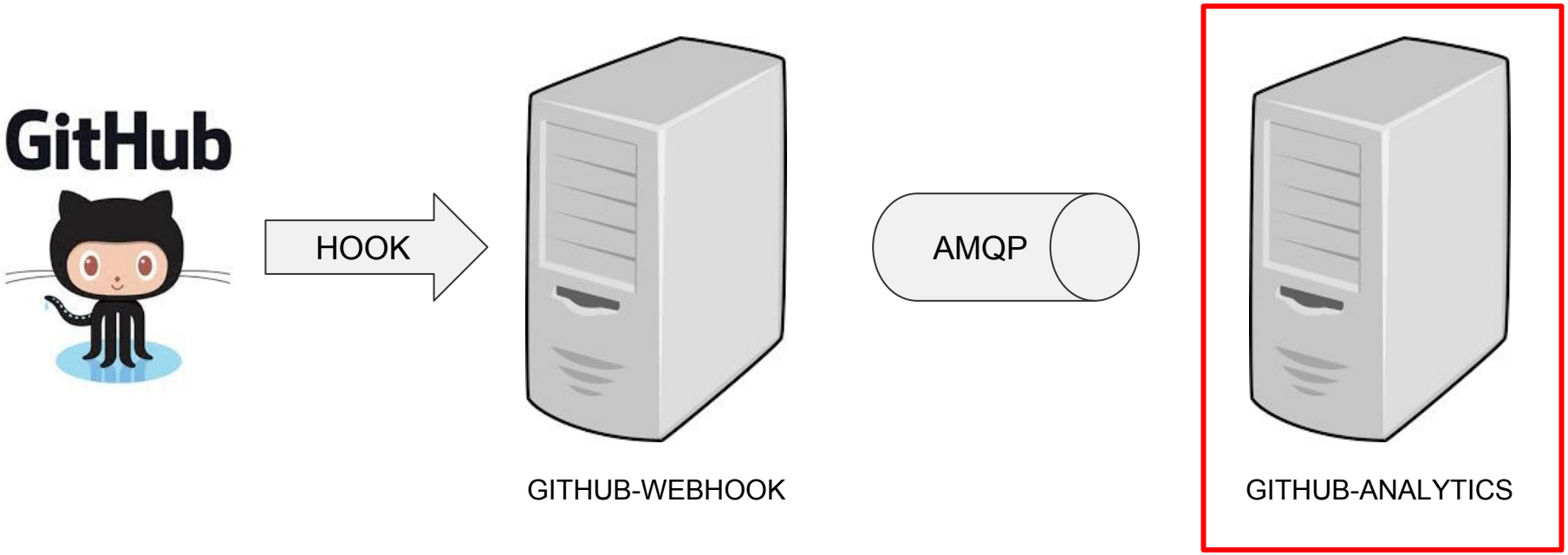


- **jenkins_pipeline.groovy** - seed job that generates other jobs and views
- **Jenkinsfile-sample** - contains the pipeline that is using the new Jenkinsfile approach
- **jenkins_pipeline_sample*.groovy** - views and job defitinitions for Jenkins Job DSL plugin based deployment pipeline

spring

WHAT ARE WE GOING TO WORK WITH?

Demo - flow



spring

Definitions

- **Build** environment is a machine where the building of the application takes place. It's a CI / CD tool worker
- **Test** is an environment where you can deploy an application to test it. It doesn't resemble production
- **Stage** is an environment where you can deploy an application to test it. It does resemble production. Typically shared by multiple teams
- **Prod** is a production environment where we want our tested applications to be deployed for our customers

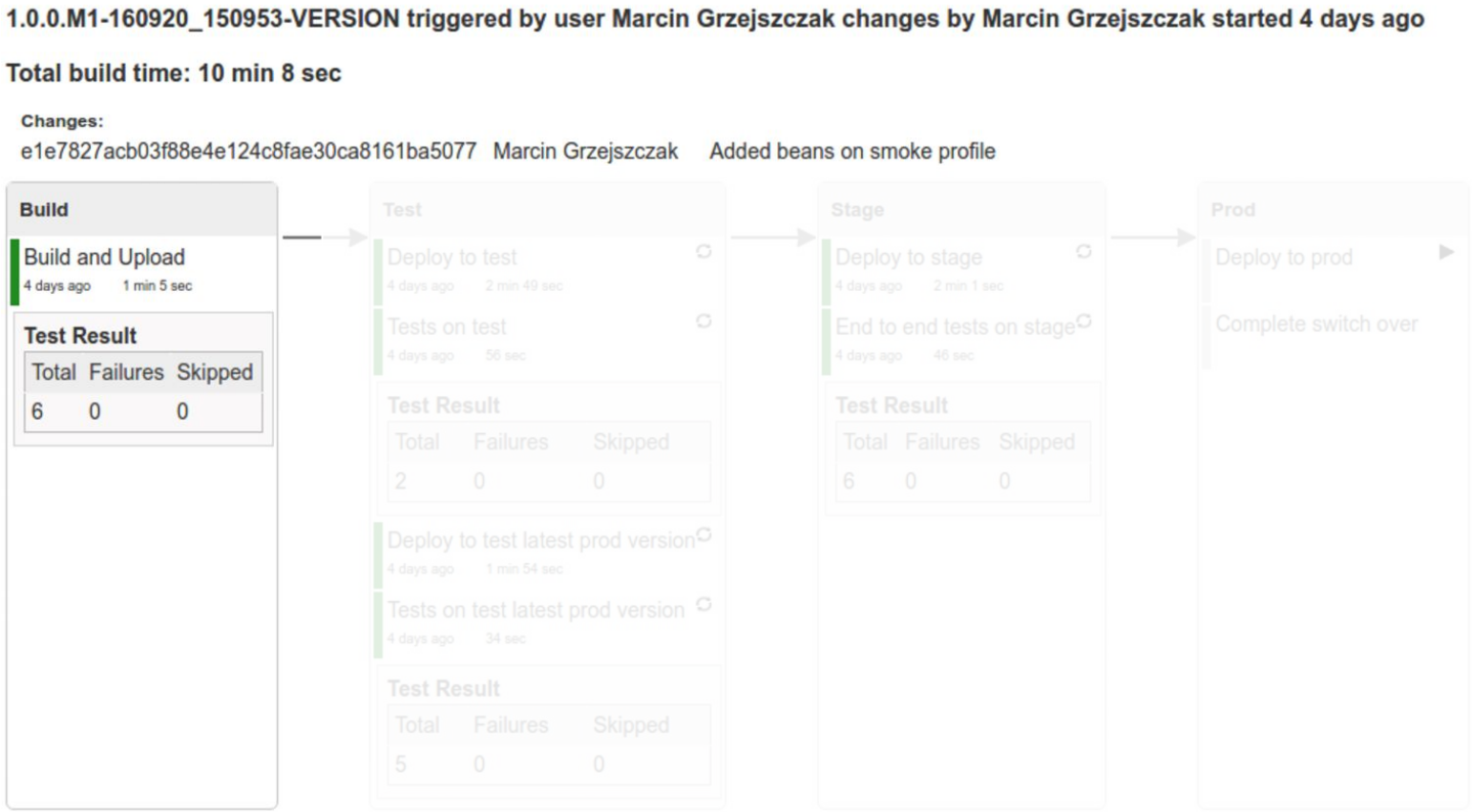
Spring Cloud Pipelines

- We’re paid for delivering business value
- Features are done when they are deployed to production
- It’s better to deploy frequently for faster feedback
  - You can apply XP patterns in deployment
- It’s better to fail the deployment pipeline as fast as possible
- Deployments should take place in a standardised, automated fashion
- Your deployment pipeline should test rollback
  - That way you can perform A/B or zero downtime deployment to production

Spring Cloud Pipelines

- We're paid for delivering business value
- Features are done when they are deployed to production
- It's better to deploy frequently for faster feedback
  - You can apply XP patterns in deployment
- It's better to fail the deployment pipeline as fast as possible
- Deployments should take place in a standardised, automated fashion
- Your deployment pipeline should test rollback
  - That way you can perform A/B or zero downtime deployment to production

Spring Cloud Pipelines



Problem - slow feedback

- Nobody wants to wait until the end of the pipeline to see that something is not working fine
- We want to fail fast - have feedback as fast as possible
- If integration is faulty we want to fail even during build time
- There should be no need to wait until end to end tests are executed

Solution - fast feedback

- We’re running unit and integration tests during build time
- To test faulty integration we use Spring Cloud Contract for HTTP / messaging
  - Producer defines a contract
  - From the contract
    - tests are generated to see if the producer is not lying
    - stubs are generated for consumers to use
  - Consumer can reuse it without running the producer
  - Fail at build time if integration fails (fail fast!)
  - All stubs reside in Nexus / Artifactory (thus can be reused)

Build - producer

```
org.springframework.cloud.contract.spec.Contract.make {
        // Human readable description
        description 'Some description'
        // Label by means of which the output message can be triggered
        label 'issue_created_v2'
        // input to the contract
        input {
                // the contract will be triggered by a method
                triggeredBy('createIssueV2()')
        }
        // output message of the contract
        outputMessage {
                // destination to which the output message will be sent
                sentTo 'messages'
                // the body of the output message
                body('''{"username":"smithapitla","repository":"spring-cloud/spring-cloud-netflix","type":"issue","action":"created"}''')
                // the headers of the output message
                headers {
                        header('version', 'v2')
                }
        }
}
```


spring

Build - consumer

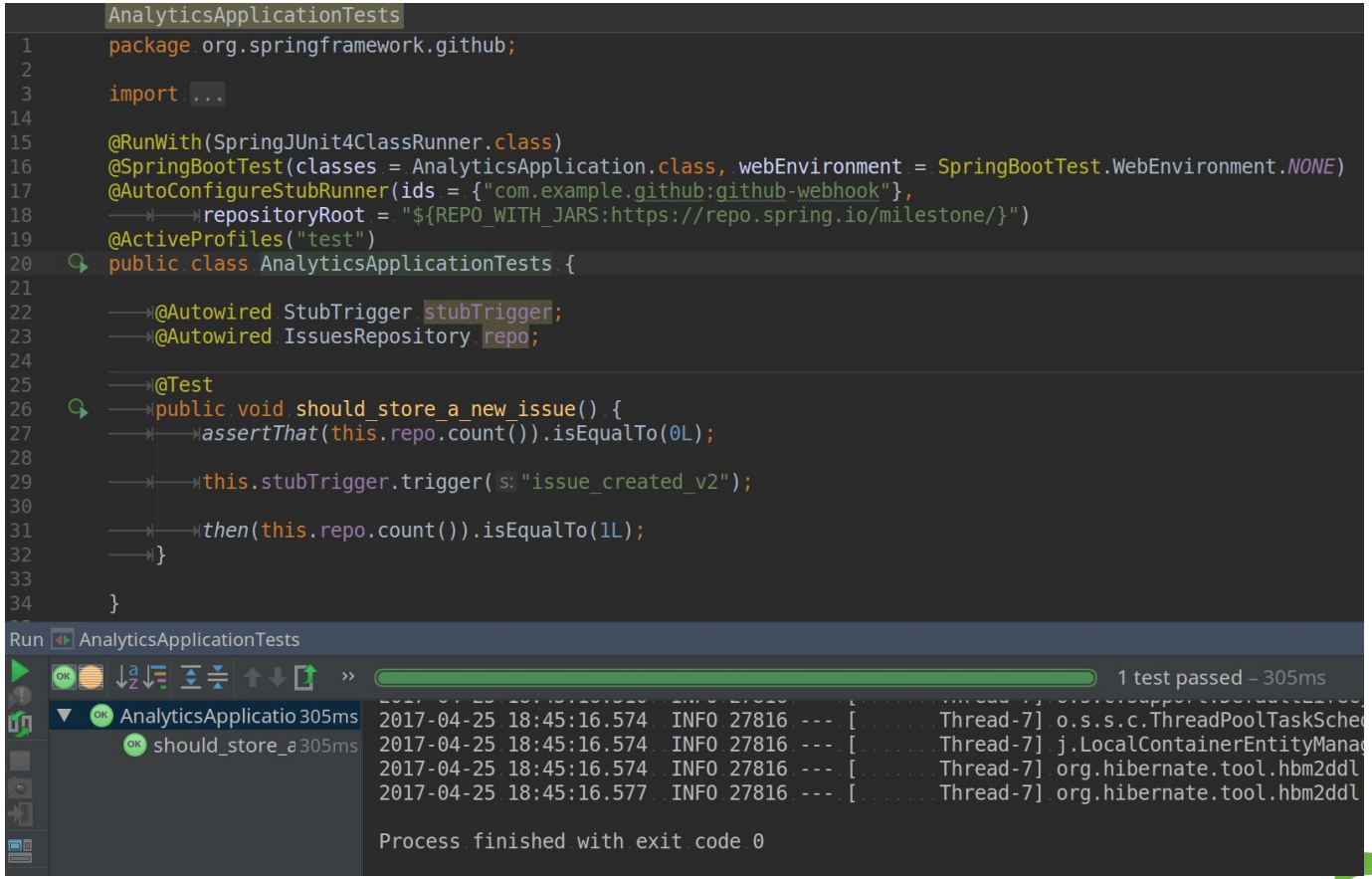
```
23  @RunWith(SpringJUnit4ClassRunner.class)
24  @SpringBootTest(classes = AnalyticsApplication.class)
26  @AutoConfigureStubRunner(
27                  repositoryRoot = "${REPO_WITH_JARS:http://localhost:8081/artifactory/libs-release-local}",
28                  ids = {"com.example.github:github-webhook"})
30  public class AnalyticsApplicationTests {
31
35          @Autowired StubTrigger stubTrigger;
36          @Autowired GithubDataListener githubDataListener;
37
64          @Test
65          public void testWithV2StubData() {
66                  int initialSize = this.githubDataListener.stats.get();
67
68                  this.stubTrigger.trigger("issue_created_v2");
69
70                  then(this.githubDataListener.counter).isNotEmpty();
71                  then(this.githubDataListener.stats.get()).isGreaterThan(initialSize);
72          }
83
84  }
```

CONTRACT TESTS DEMO

Expected model (consumer)

```
public class GithubDatum {
    private String username;
    private String repository;
    private String type = "unknown";
    private String action = "unknown";
```

Passing contract test (consumer)



spring

Breaking contract (producer)

```
1  package contracts.messaging
2
3  org.springframework.cloud.contract.spec.Contract.make {
4      // Human readable description
5      description 'Some description'
6      // Label by means of which the output message can be triggered
7      label 'hook_created_v2'
8      // input to the contract
9      input {
10         // the contract will be triggered by a method
11         triggeredBy('createHookV2()')
12     }
13     // output message of the contract
14     outputMessage {
15         // destination to which the output message will be sent
16         sentTo 'messages'
17         // the body of the output message
18         body('''{"user":"dsyer","repo":"spring-cloud-samples","type":"hook","action":"updated"}''')
19         // the headers of the output message
20         headers {
21             header('versi                    ')
22             header('co                    pplicationJson())
23         }
24     }
25 }
26
```

Was **username** and **repository** and we've made a breaking change by converting those to **user** and **repo**

Installing broken stubs locally (producer)

```
→  github-webhook git:(broken_contract) ✗ ./mvnw clean install
[INFO]
[INFO] --- spring-cloud-contract-maven-plugin:1.0.4.RELEASE:generateStubs (default-generateStubs) @ github-webhook ---
[INFO] Files matching this pattern will be excluded from stubs generation [**/*.groovy]
[INFO] Building jar: /home/marcin/repo/github-webhook/target/github-webhook-0.0.1-SNAPSHOT-stubs.jar
[INFO]
[INFO] --- maven-jar-plugin:2.6:jar (default-jar) @ github-webhook ---
[INFO] Building jar: /home/marcin/repo/github-webhook/target/github-webhook-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:1.5.2.RELEASE:repackage (default) @ github-webhook ---
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ github-webhook ---
[INFO] Installing /home/marcin/repo/github-webhook/target/github-webhook-0.0.1-SNAPSHOT.jar to /home/marcin/.m2/repository/com/example/
github/github-webhook/0.0.1-SNAPSHOT/github-webhook-0.0.1-SNAPSHOT.jar
[INFO] Installing /home/marcin/repo/github-webhook/pom.xml to /home/marcin/.m2/repository/com/example/github/github-webhook/0.0.1-SNAPS
HOT/github-webhook-0.0.1-SNAPSHOT.pom
[INFO] Installing /home/marcin/repo/github-webhook/target/github-webhook-0.0.1-SNAPSHOT-stubs.jar to /home/marcin/.m2/repository/com/ex
ample/github/github-webhook/0.0.1-SNAPSHOT/github-webhook-0.0.1-SNAPSHOT-stubs.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 18.268 s
[INFO] Finished at: 2017-04-25T18:44:31+02:00
[INFO] Final Memory: 62M/623M
[INFO] ------------------------------------------------------------------------
```

New stubs with backward incompatible changes installed in Maven local

spring

Broken contract test locally (consumer)

Expected **repository** and **username** but got **repo** and **user**

spring

Spring Cloud Pipelines

- We’re paid for delivering business value
- Features are done when they are deployed to production
- It’s better to deploy frequently for faster feedback
  - You can apply XP patterns in deployment
- It’s better to fail the deployment pipeline as fast as possible
- Deployments should take place in a standardised, automated fashion
- Your deployment pipeline should test rollback
  - That way you can perform A/B or zero downtime deployment to production

Spring Cloud Pipelines

- We're paid for delivering business value
- Features are done when they are deployed to production
- It's better to deploy frequently for faster feedback
  - You can apply XP patterns in deployment
- It's better to fail the deployment pipeline as fast as possible
- Deployments should take place in a standardised, automated fashion
- Your deployment pipeline should test rollback
  - That way you can perform A/B or zero downtime deployment to production

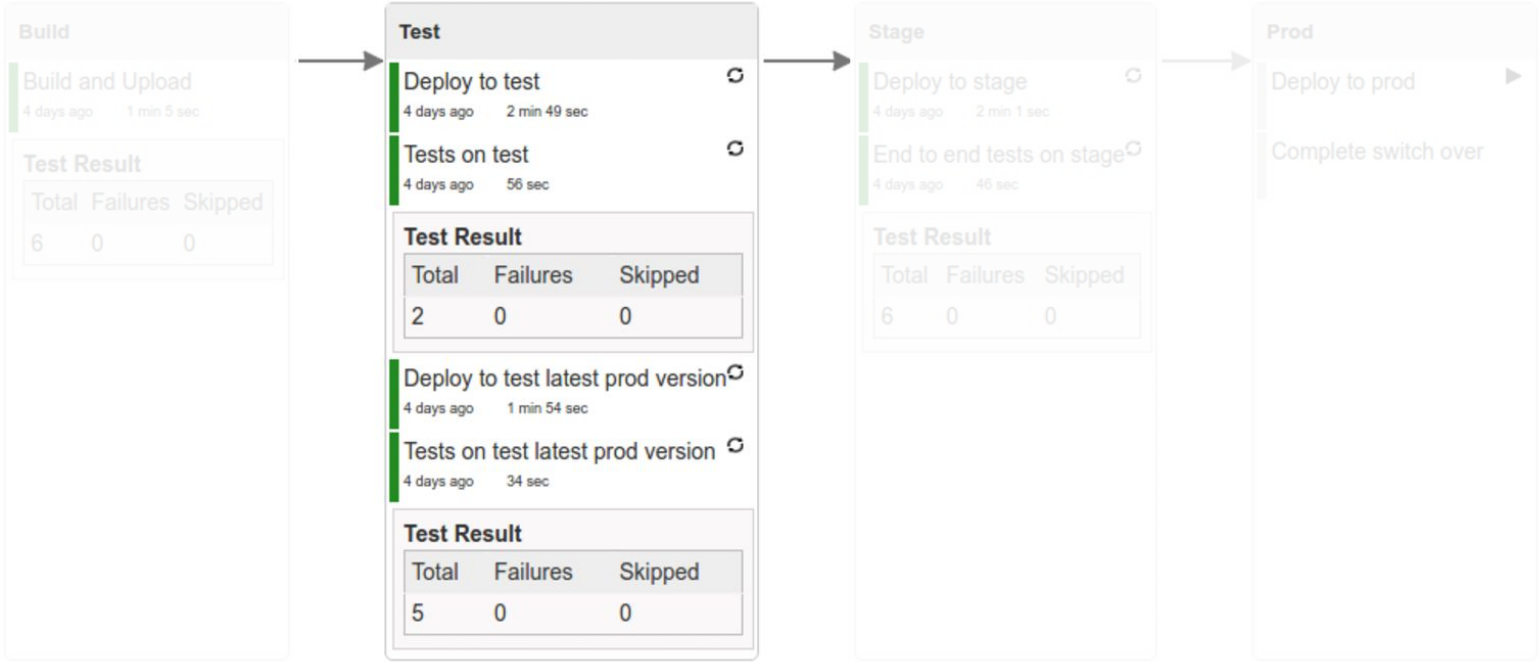
Spring Cloud Pipelines

1.0.0.M1-160920_150953-VERSION triggered by user Marcin Grzejszczak changes by Marcin Grzejszczak started 4 days ago

Total build time: 10 min 8 sec

Changes:

e1e7827acb03f88e4e124c8fae30ca8161ba5077 Marcin Grzejszczak Added beans on smoke profile



Problem - standardisation

- Nobody wants to support X different ways
  - applications are ran (e.g. `run.sh`, `start.sh`, `java -jar` etc.)
  - of deploying an artifact (e.g. bash scripts, ansible, puppet, chef etc.)
  - of verifying application health (e.g. `/ping`, `/health` etc.)
  - of upgrading a DB schema (e.g. separate jobs, manual steps etc.)
- Standardisation of approaches gives lower support costs and more chances of people rotation

Solution - standardisation

- Use a PaaS to standardise the way you deploy and monitor your software
- Spring Cloud Pipelines uses Cloud Foundry [1] as a PaaS implementation
  - For the demo purposes we're using PCF Dev [2]
- Cloud Foundry abstracts the application governance from underlying infrastructure
  - you can deploy, scale, manage applications in the same way if CF is running on your laptop, Amazon, Google, Azure etc.
- Database schema upgrade is done via tools like Flyway [3] or Liquibase [4]

[1] https://www.cloudfoundry.org
[2] https://pivotal.io/pcf-dev
[3] https://flywaydb.org/
[4] http://www.liquibase.org/

Deploying apps with CF

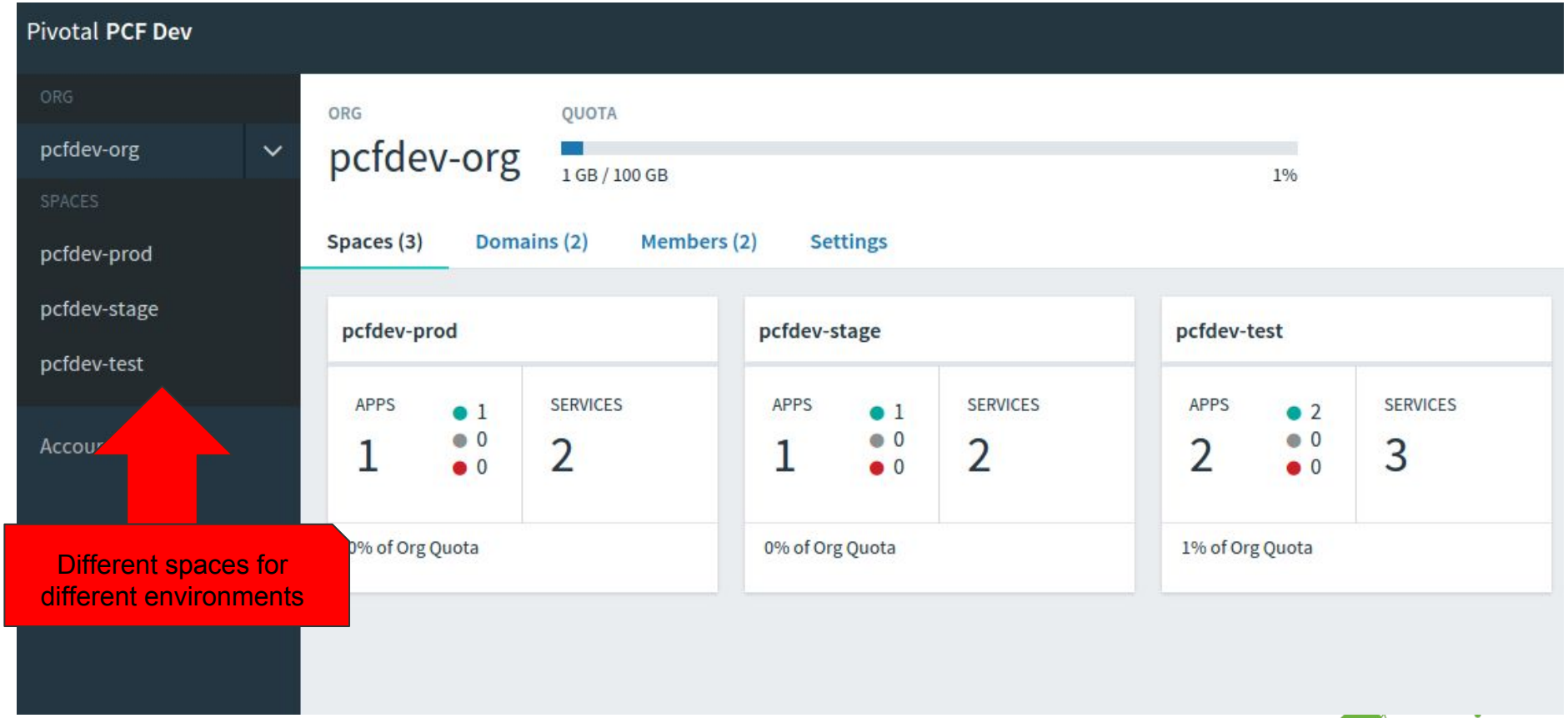
```
$ cf push
```


https://www.cloudfoundry.org
https://pivotal.io/pcf-dev

spring

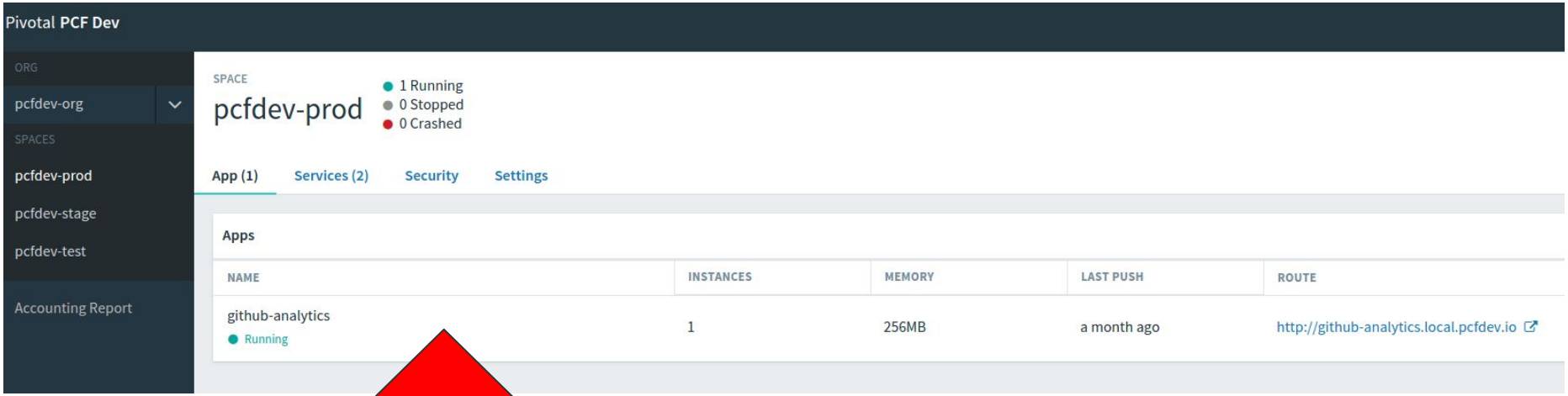
CF DEMO

Pivotal Cloud Foundry Apps Manager



spring

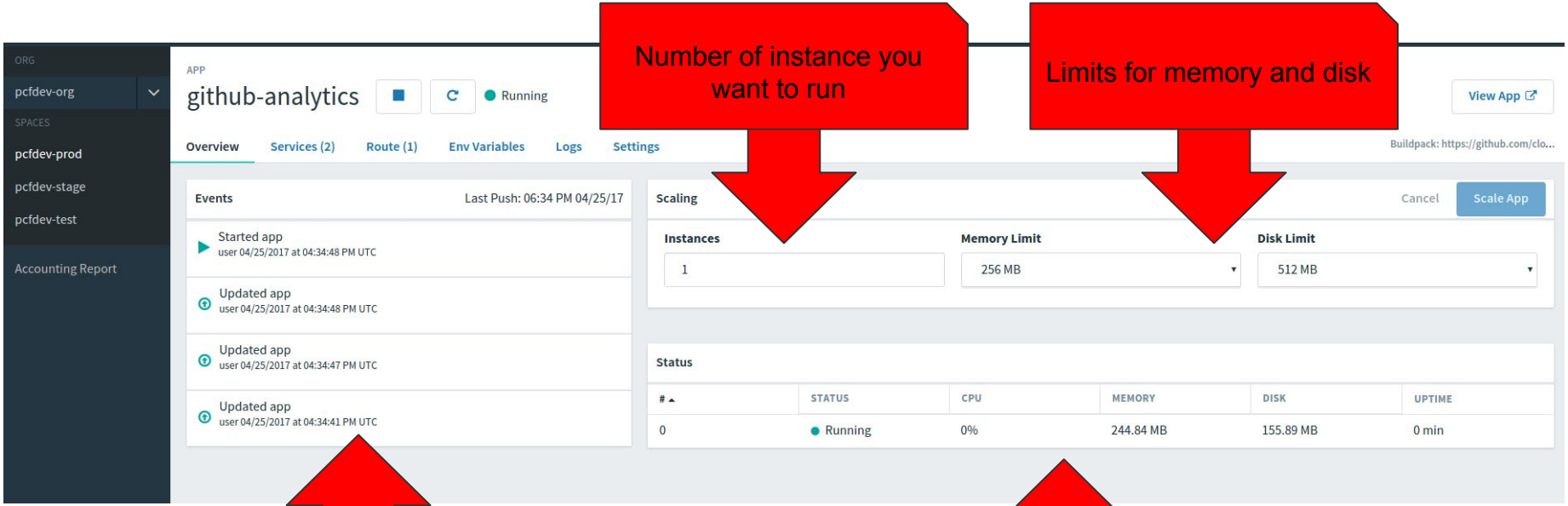
Pivotal Cloud Foundry Apps Manager



Running application in production space

spring

Pivotal Cloud Foundry Apps Manager



Number of instance you want to run

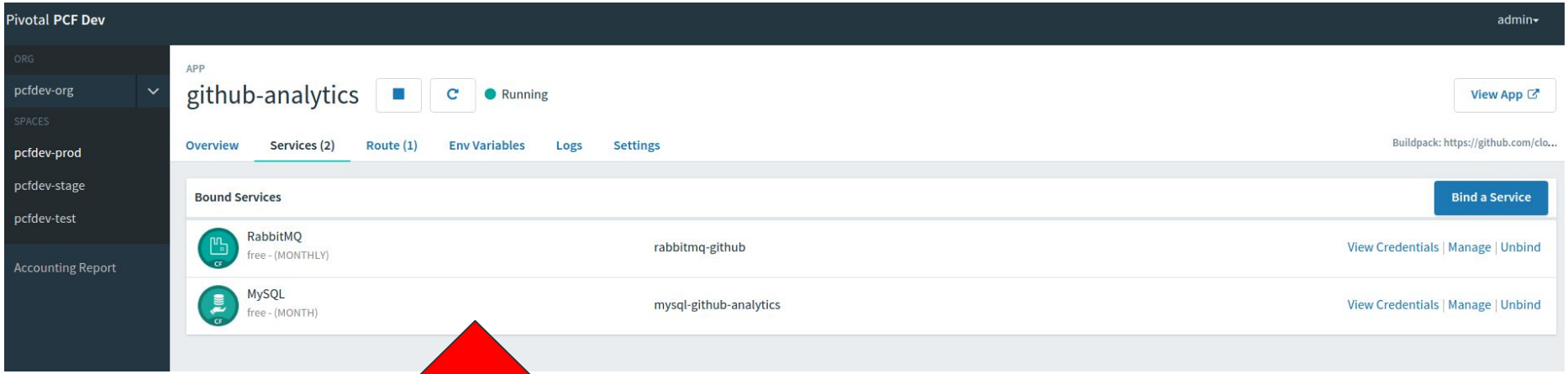
Limits for memory and disk

Audit events of what happened with your instances

State of your instances

spring

Pivotal Cloud Foundry Apps Manager



Bound services to the application. Credentials get injected automatically

spring

Solution - schema upgrade standardisation (Flyway example)

spring-boot / spring-boot-samples / spring-boot-sample-flyway / src / main / resources / db / migration / **V1__init.sql**

dsyer Use Flyway to bind flyway.* e118515 on 20 May 2014

1 contributor

7 lines (6 sloc) | 206 Bytes Raw Blame History

```
1  CREATE TABLE PERSON (
2          id BIGINT GENERATED BY DEFAULT AS IDENTITY,
3          first_name varchar(255) not null,
4          last_name varchar(255) not null
5  );
6
7  insert into PERSON (first_name, last_name) values ('Dave', 'Syer');
```


spring

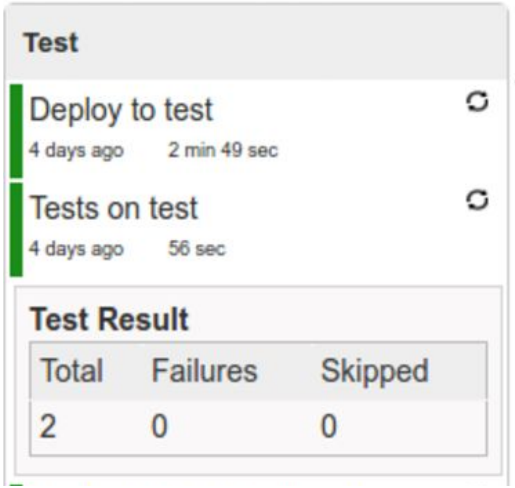
Problem - slow feedback

- It happens that QA departments write very expensive tests for every single feature
  - typically end to end or UI
  - “we are paid to write tests” approach
- Tests take ages to finish
- They are extremely brittle

Spring Cloud Pipelines

After the application got deployed to test environment

- The database schema gets updated upon application startup
- We run a handful of smoke tests to see if crucial parts of our application are working fine
- We want to test if the app is properly packaged
- The application is surrounded by stubs - no real integrations take place
- Spring Cloud Contract Stub Runner Boot app is responsible for serving stubs



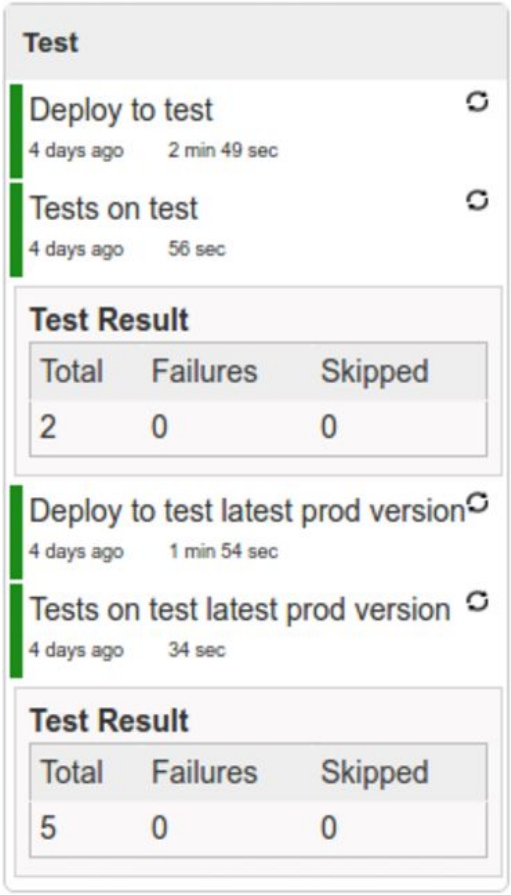
spring

Problem - rollback DB

- Deployment pipelines should test whether the application can be rolled back
- Rolling back database changes is extremely difficult
  - Especially if you deploy once every 6 months (the number of changes is gigantic)
  - How can you roll back a deletion of a column?
- Maintaining rollback scripts is not the most pleasant task

Solution - application rollback

- The easiest solution is… NOT TO DB ROLLBACK
- Perform only backward compatible changes (always add data)
  - Or perform backward incompatible changes in a backward compatible way [1]
- Roll back the application only (the JAR)
- The application and DB changes need to be forward / backward compatible
- That way you can ensure that two applications (old / new versions) can run simultaneously at the same time



[1] https://spring.io/blog/2016/05/31/zero-downtime-deployment-with-a-database

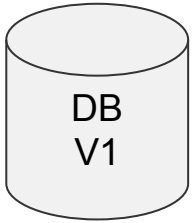
spring

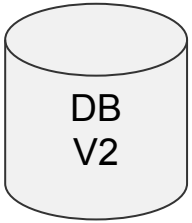
Demo - backward incompatible DB change

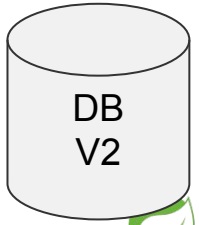
GITHUB-ANALYTICS V1

GITHUB-ANALYTICS V2

GITHUB-ANALYTICS V1

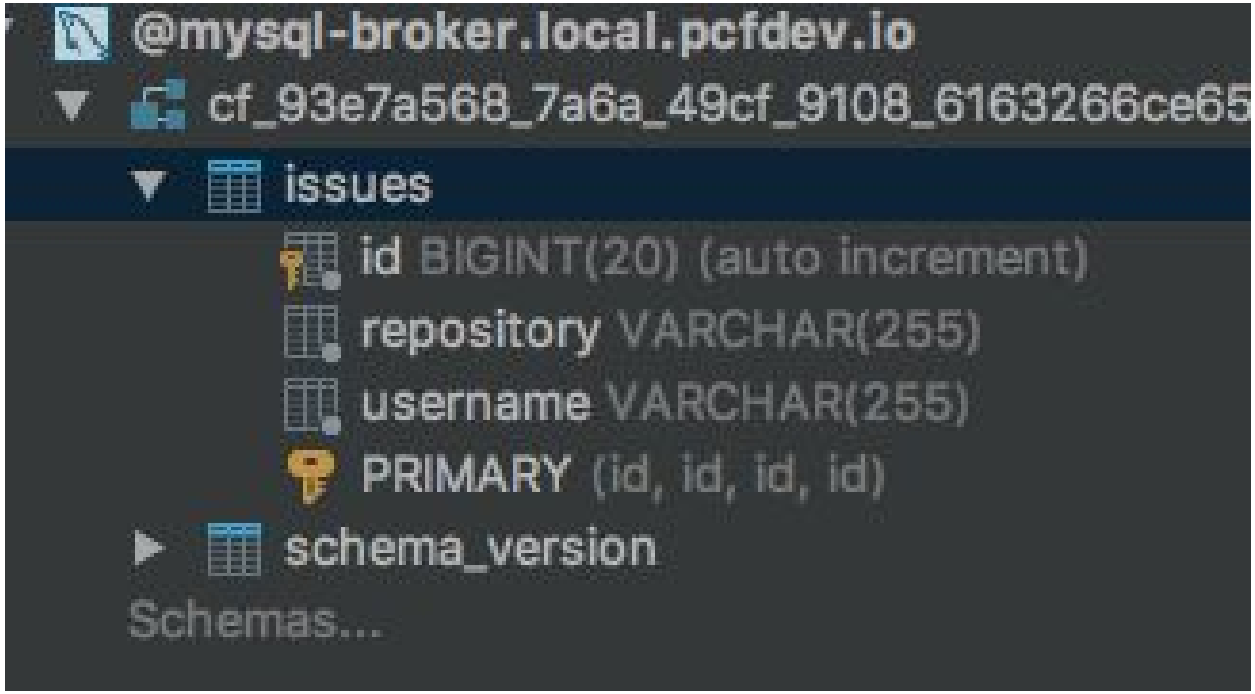






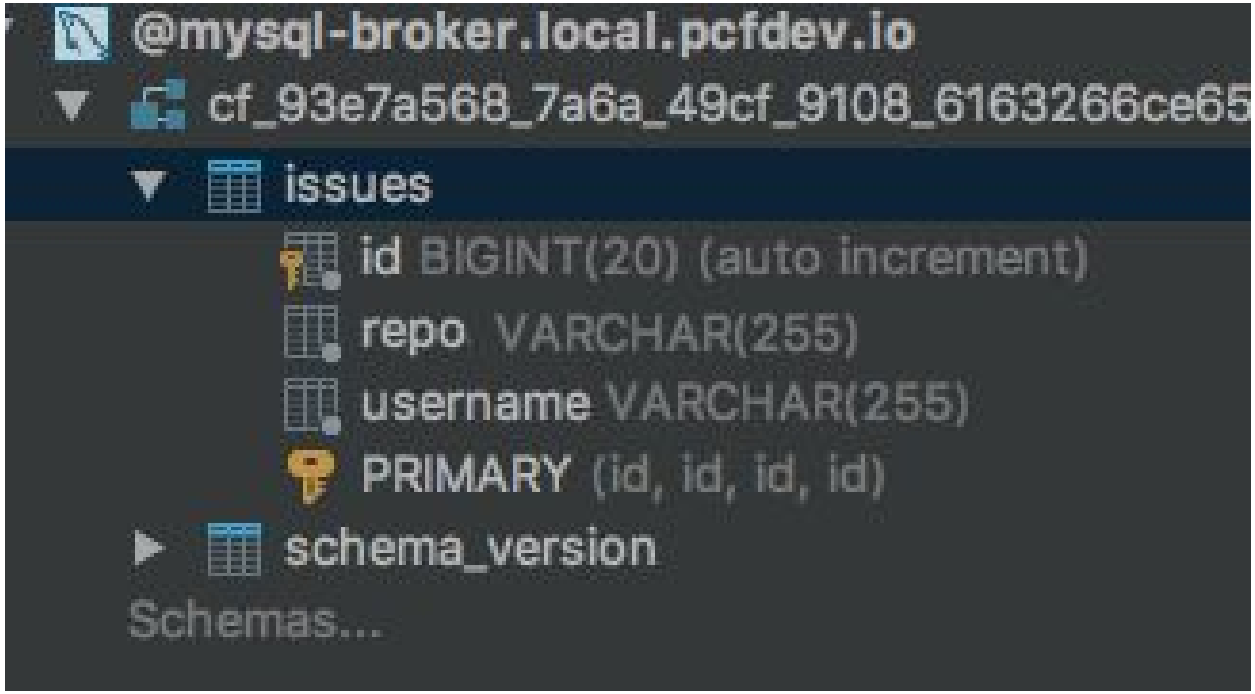
spring

Demo - initial state



spring

Demo - backward incompatible DB change



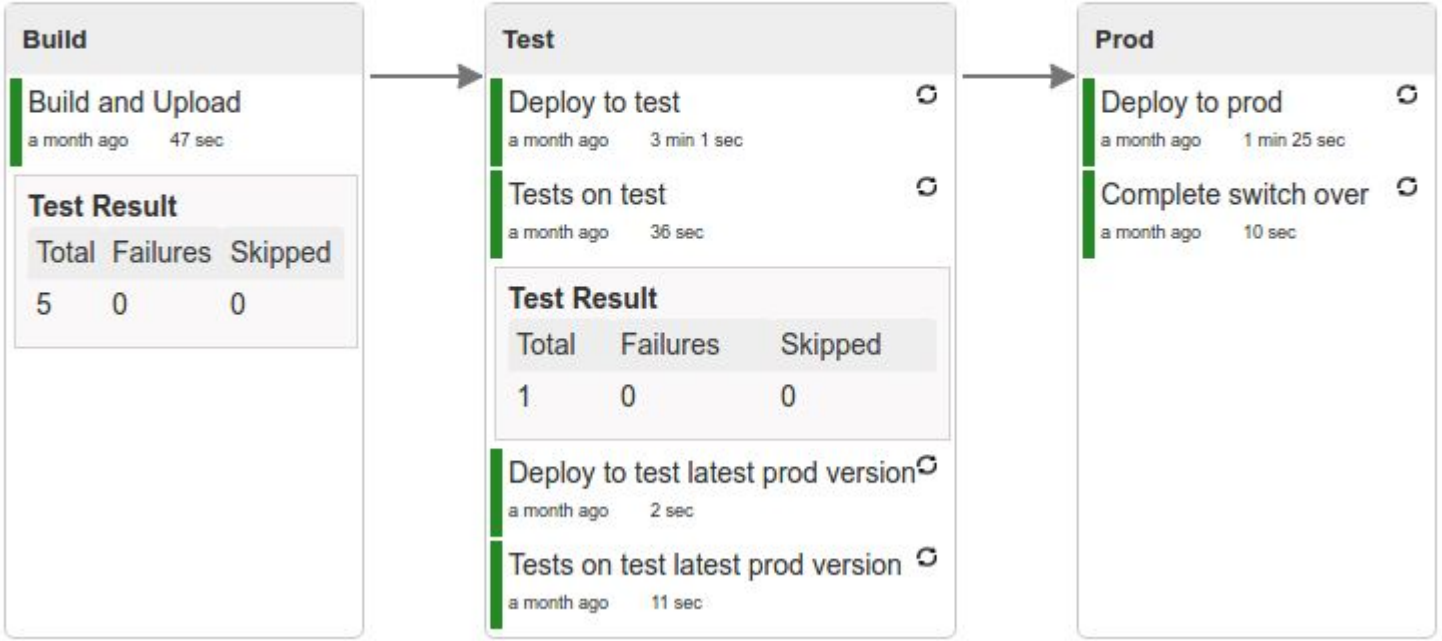
spring

ROLLBACK DEMO

Demo - first run

1.0.0.M1-170322_214634-VERSION triggered by user anonymous started a month ago

Total build time: 6 min 15 sec



spring

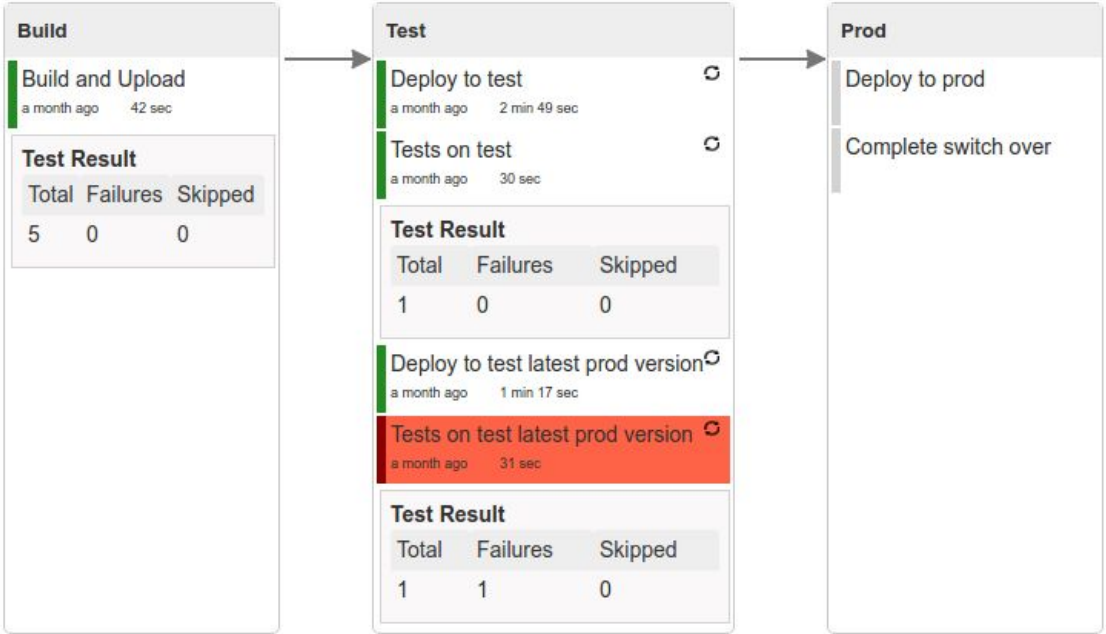
Demo - deploying backward incompatible DB change

1.0.0.M1-170322_220209-VERSION triggered by user anonymous changes by mgrzejszczak started a month ago

Total build time: 5 min 51 sec

Changes:

e2a3cfe6232c62afc1eeee75dd2242c156693362 mgrzejszczak Backward incompatible change



spring

Demo - errors in the app logs


```
OUT    ... 67 common frames omitted
OUT Caused by: com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Unknown column 'repository' in 'field list'
OUT    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method) ~[na:1.8.0_91]
OUT    at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62) ~[na:1.8.0_91]
OUT    at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45) ~[na:1.8.0_91]
OUT    at java.lang.reflect.Constructor.newInstance(Constructor.java:423) ~[na:1.8.0_91]
OUT    at com.mysql.jdbc.Util.handleNewInstance(Util.java:425) ~[mysql-connector-java-5.1.41.jar!/:5.1.41]
OUT    at com.mysql.jdbc.Util.getInstance(Util.java:408) ~[mysql-connector-java-5.1.41.jar!/:5.1.41]
OUT    at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:943) ~[mysql-connector-java-5.1.41.jar!/:5.1.41]
OUT    at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:3973) ~[mysql-connector-java-5.1.41.jar!/:5.1.41]
OUT    at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:3909) ~[mysql-connector-java-5.1.41.jar!/:5.1.41]
OUT    at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:2527) ~[mysql-connector-java-5.1.41.jar!/:5.1.41]
OUT    at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:2680) ~[mysql-connector-java-5.1.41.jar!/:5.1.41]
OUT    at com.mysql.jdbc.ConnectionImpl.execSQL(ConnectionImpl.java:2501) ~[mysql-connector-java-5.1.41.jar!/:5.1.41]
OUT    at com.mysql.jdbc.PreparedStatement.executeInternal(PreparedStatement.java:1858) ~[mysql-connector-java-5.1.41.jar!/:5.1.41]
```


spring

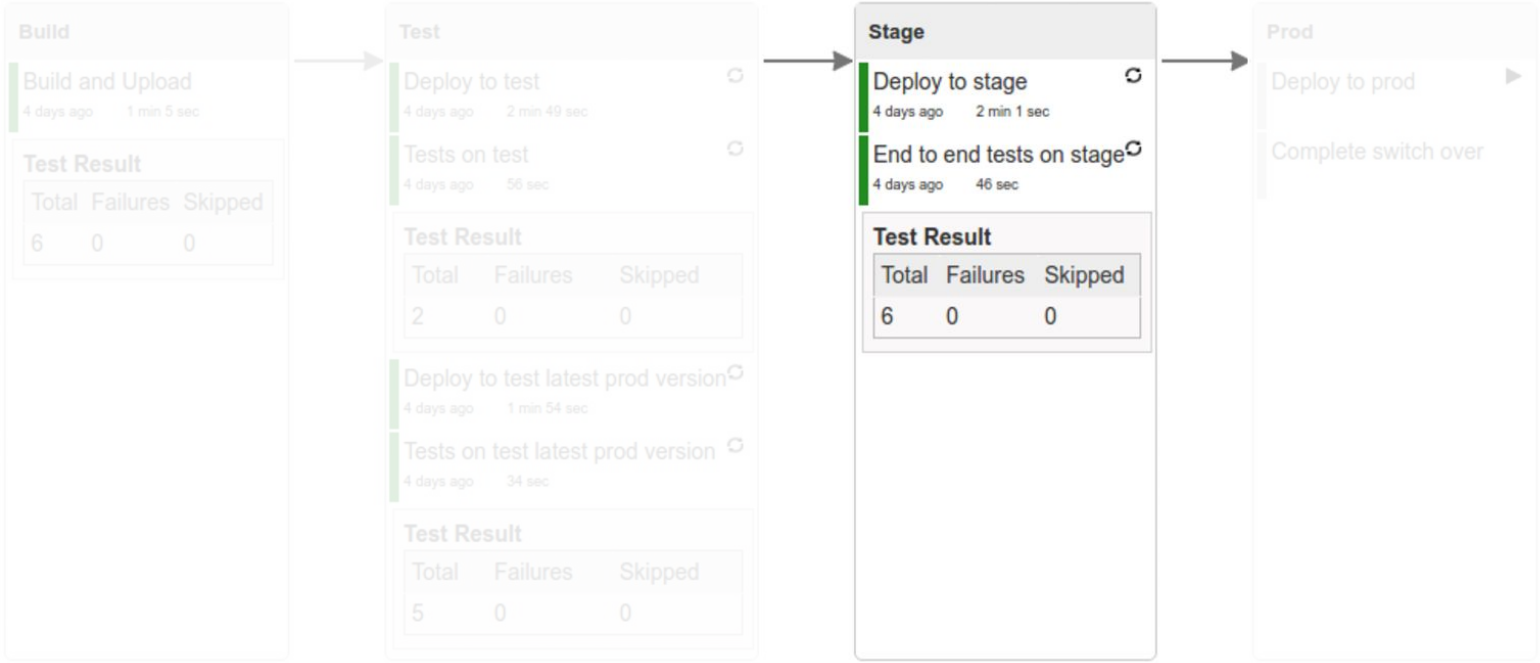
Spring Cloud Pipelines

1.0.0.M1-160920_150953-VERSION triggered by user Marcin Grzejszczak changes by Marcin Grzejszczak started 4 days ago

Total build time: 10 min 8 sec

Changes:

e1e7827acb03f88e4e124c8fae30ca8161ba5077 Marcin Grzejszczak Added beans on smoke profile



spring

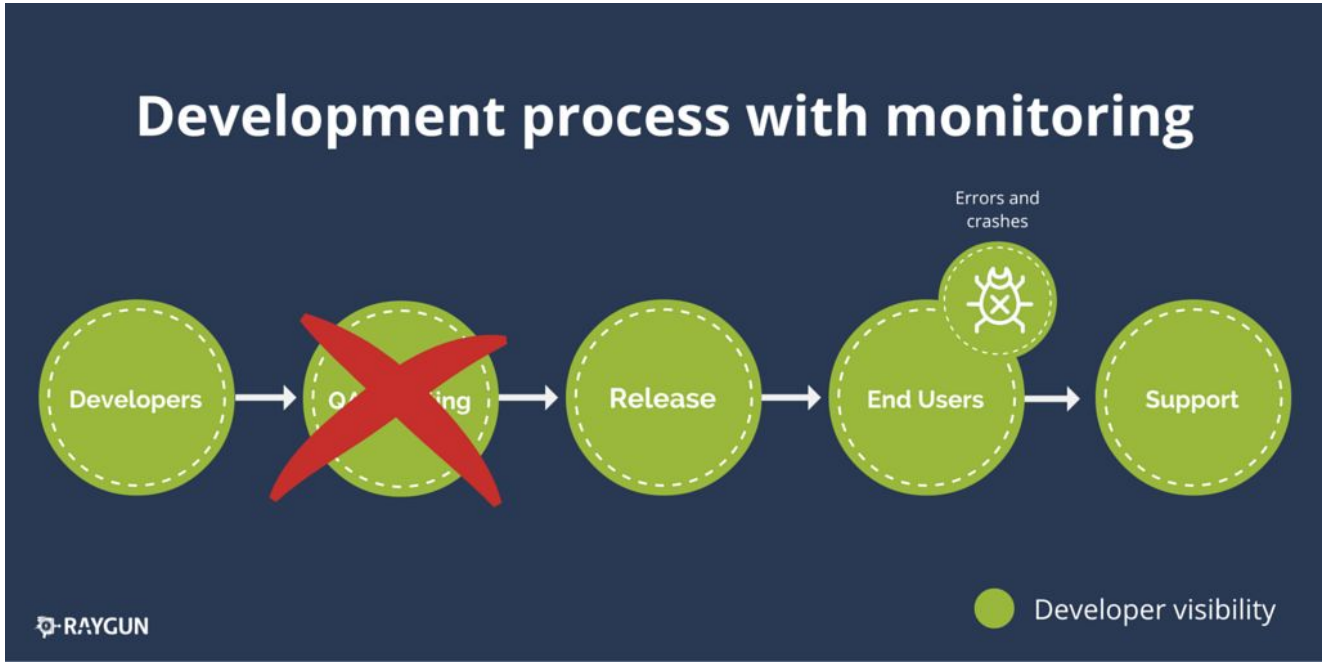
Problem - end to end tests

- It takes ages to run end to end tests
- They are slow and brittle
- QA department writes an E2E for every feature we have
- E2E environment setup
  - one environment shared between all applications?
  - one environment per application?
- Surrounding apps should be deployed in
  - production versions?
  - development versions?

Solution - don’t do E2E?

- Regardless of the time spent on QA / UAT you can still have bugs on production
- Assuming that you ...
  - embrace failure
  - introduce monitoring of business KPIs
  - introduce alerting over the metrics
  - ensure that you can rollback on production
- … you could stop doing any end to end tests

Solution - don’t do QA ?

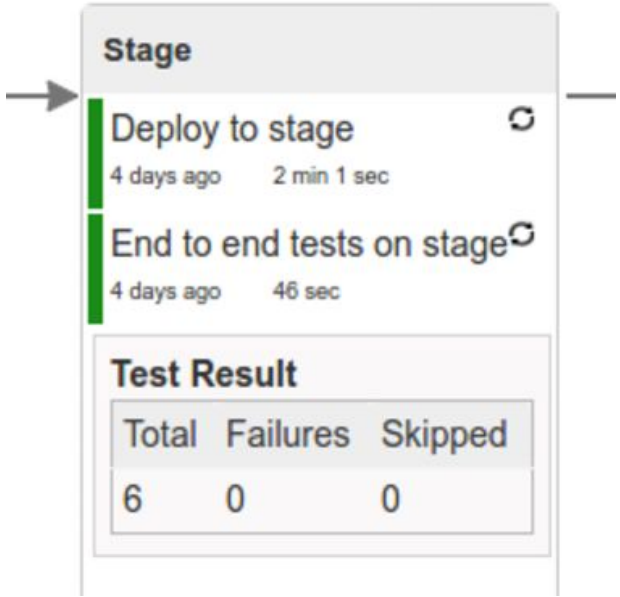


https://raygun.com/blog/2016/07/software-testing/

spring

Spring Cloud Pipelines

- The whole step is optional
  - it’s left there cause the “no e2e tests” approach is controversial
- Deploy to stage and running e2e tests are manual steps
  - you have to wait for your turn for the env
  - some manual work has to be done to purge stale data etc.

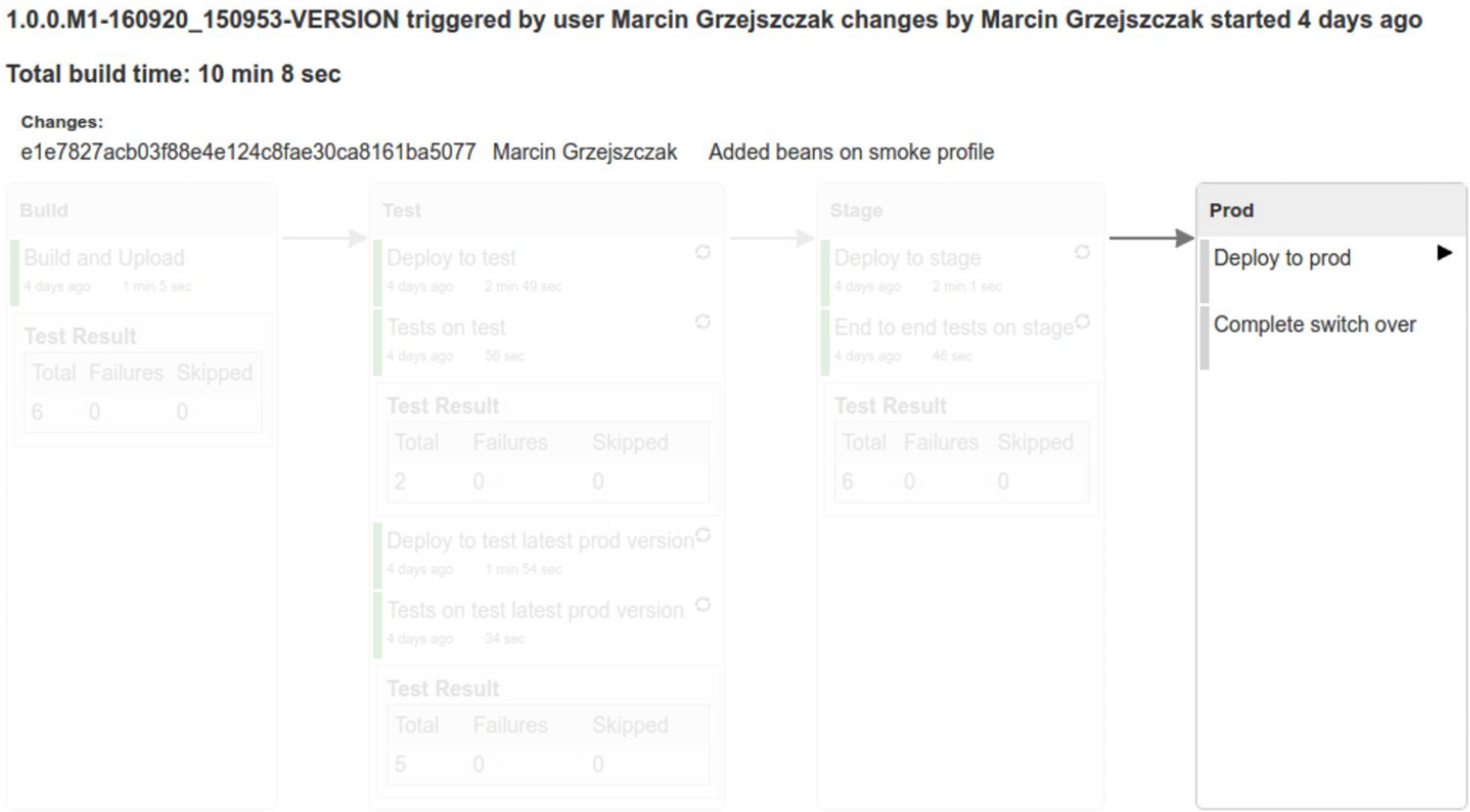


spring

Spring Cloud Pipelines

- We're paid for delivering business value
- Features are done when they are deployed to production
- It's better to deploy frequently for faster feedback
  - You can apply XP patterns in deployment
- It's better to fail the deployment pipeline as fast as possible
- Deployments should take place in a standardised, automated fashion
- Your deployment pipeline should test rollback
  - That way you can perform A/B or zero downtime deployment to production

Spring Cloud Pipelines



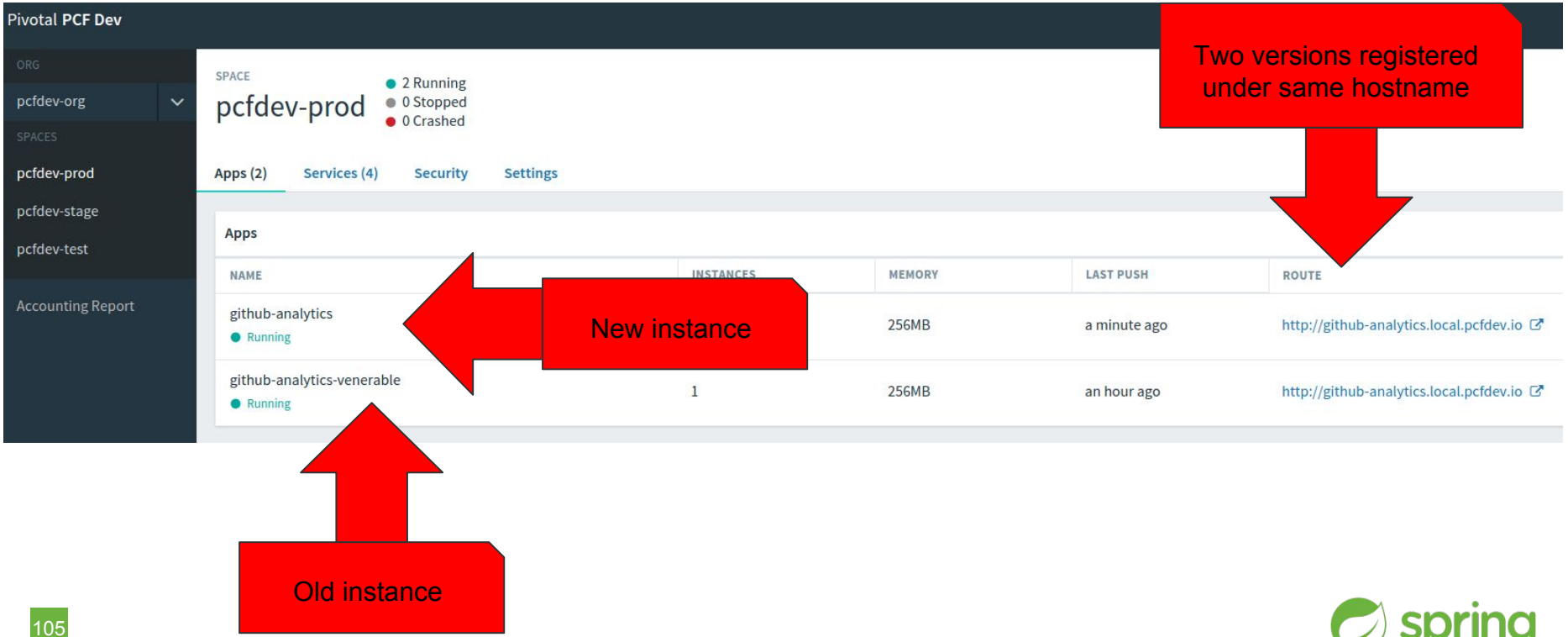
Problem - deploy to production

- We don’t want to deploy the application to production at night
- We want to treat a production deployment like any other deployment
- We’d like to be able to perform A/B testing and zero downtime deployment
- We’d like to easily rollback when sth goes wrong

Solution - CF + SC Pipelines

- Our application has KPI monitoring in place
- Alerting are set for KPIs
- It has been tested if it can be rolled back so if anything goes wrong we can easily roll back
- We’re deploying a new version of the application registered under the same hostname
  - That way we have both old and new version present on production
  - Cloud Foundry performs automatic load balancing of the instances
  - We can easily stop the new version if it turns out to be faulty

Two versions running at the same time



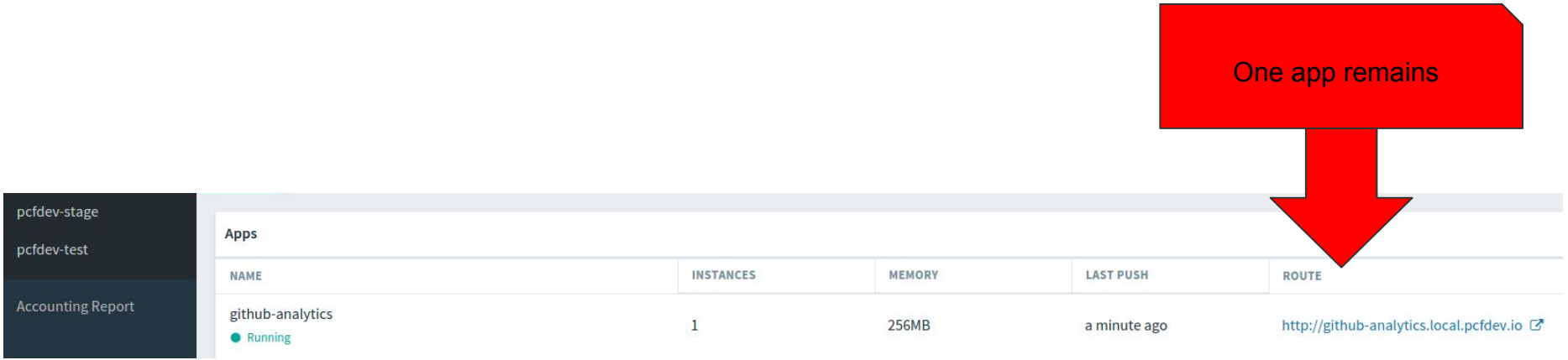
spring

Spring Cloud Pipelines

- **Deploy to prod** deploys the pipeline version of the app to production **next to** the current production version
- **Complete switch over** allows to delete the old instance and leave only the new one
- Once deployed we tag the repo with `prod/VERSION_NUMBER`

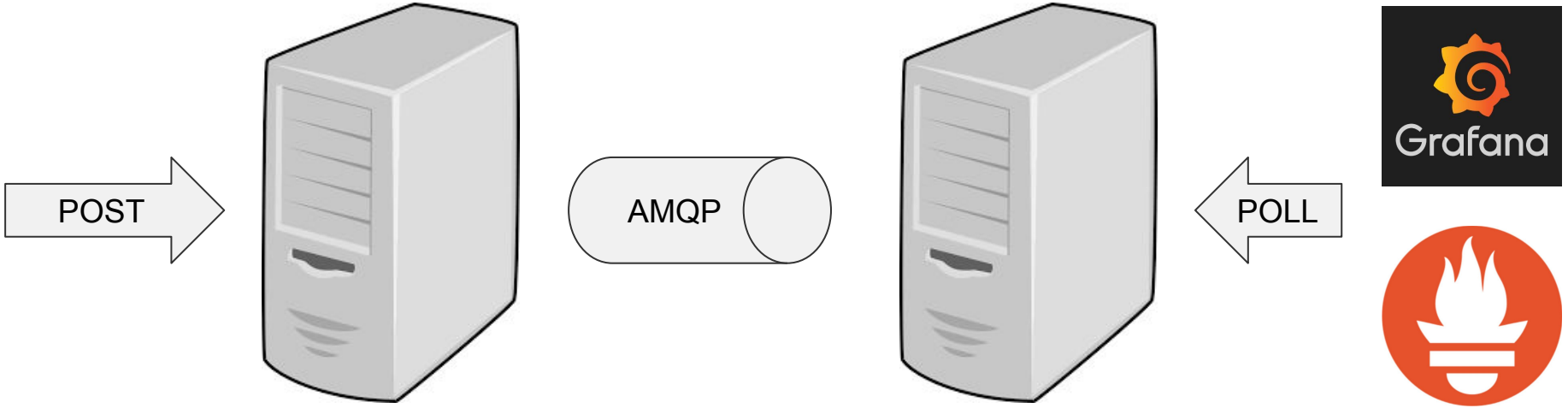
spring

After complete switch over



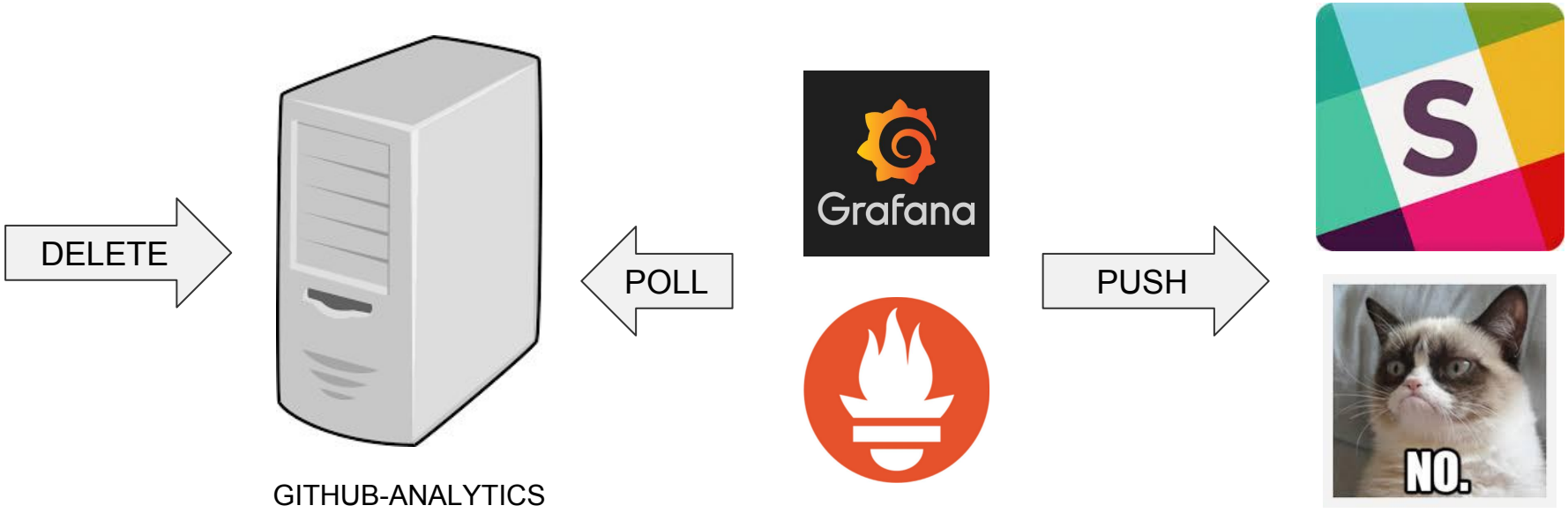
spring

Demo - alerts



spring

Demo - alerts



spring

METRICS & ALERTS DEMO

Metric definition in the code

```
@Service
class IssuesService {
    private final IssuesRepository repository;
    private final GaugeService gaugeService;

    IssuesService(IssuesRepository repository, GaugeService gaugeService) {
        this.repository = repository;
        this.gaugeService = gaugeService;
    }

    void save(String user, String repo) {
        this.repository.save(new Issues(user, repo));
        submitCountMetric();
    }

    private void submitCountMetric() { submitCountMetric(count()); }

    private void submitCountMetric(long numbers) { this.gaugeService.submit( s: "issues.count", numbers); }

    List<IssueDto> allIssues() {
        List<IssueDto> dtos = new ArrayList<>();
        this.repository.findAll().forEach(i -> dtos.add(new IssueDto(i.getUsername(), i.getRepository())));
        return dtos;
    }

    long numberOfIssues() {
        long count = count();
        submitCountMetric(count);
        return count;
    }

    private long count() { return this.repository.count(); }
```

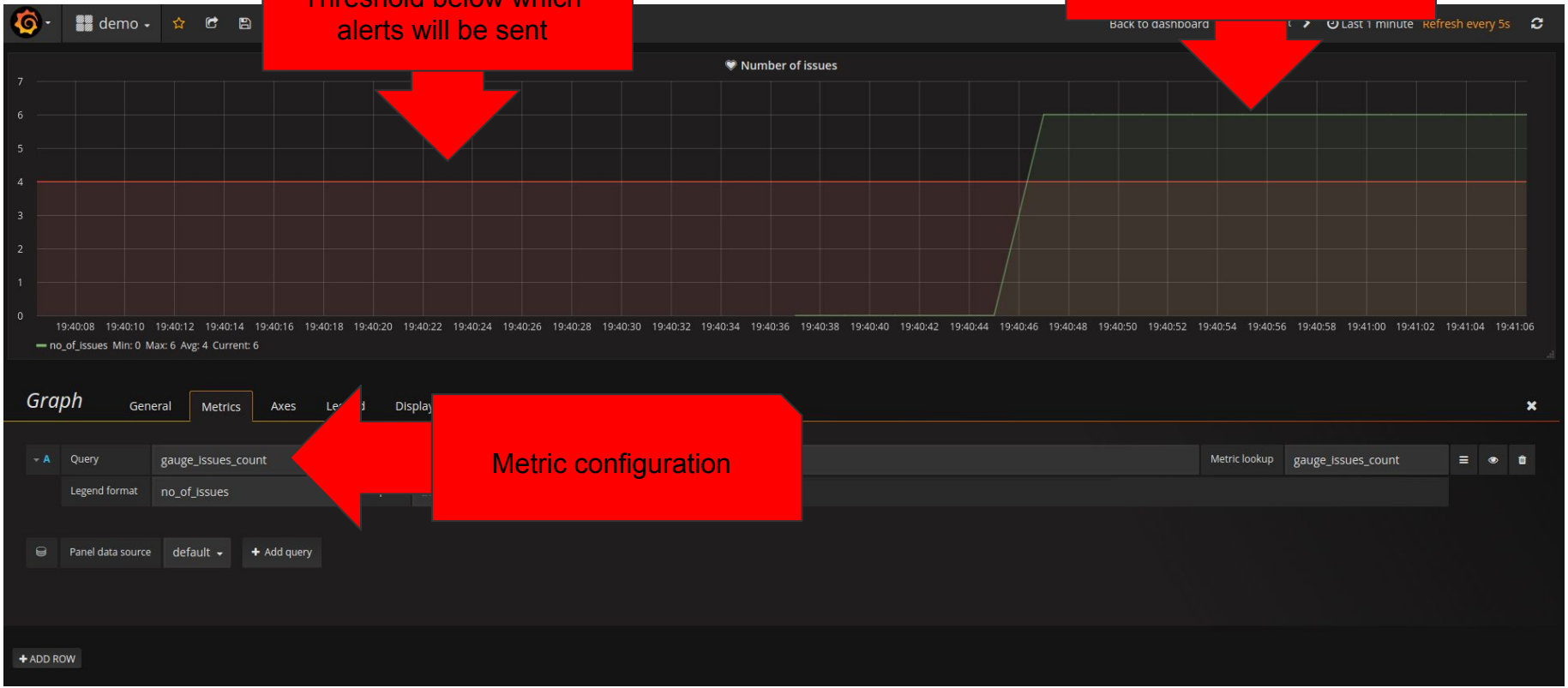
Will be resolved to **gauge_issues_count** i Prometheus

spring

Insert some data to the service

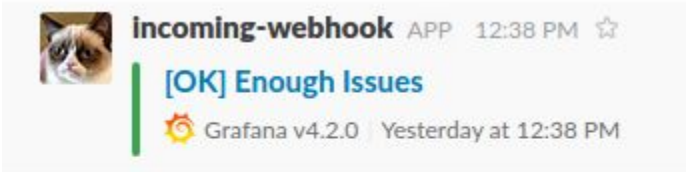
```
github-analytics-demo git:(master) ./scripts/curlWebhook.sh
github-analytics-demo git:(master) ./scripts/curlWebhook.sh
github-analytics-demo git:(master) ./scripts/curlWebhook.sh
github-analytics-demo git:(master) ./scripts/curlWebhook.sh
github-analytics-demo git:(master) ./scripts/curlWebhook.sh
github-analytics-demo git:(master) ./scripts/curlWebhook.sh
```

Grafana



spring

Slack notifications



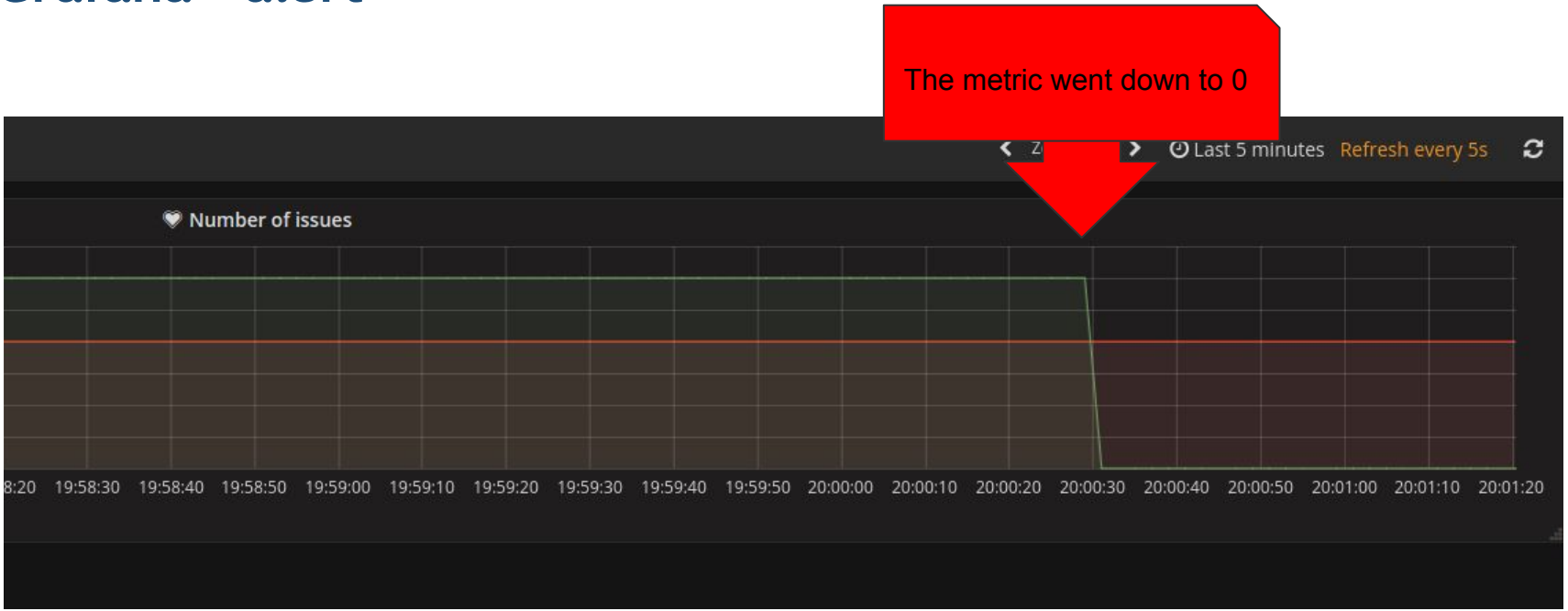
spring

Delete all issues

```
github-analytics-demo git:(master) curl -X DELETE http://github-analytics.local.pcfdev.io/issues/
```


spring

Grafana - alert

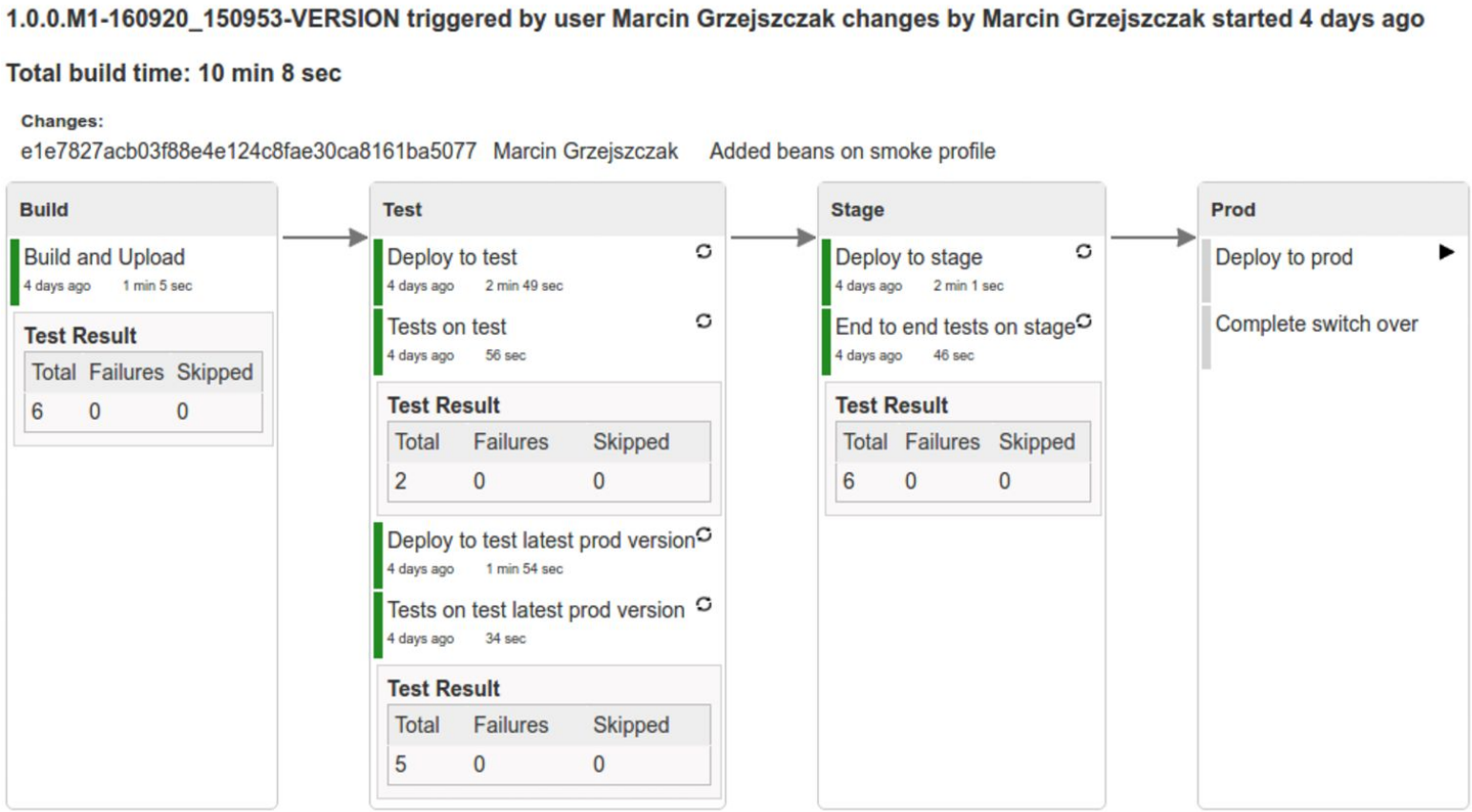


spring

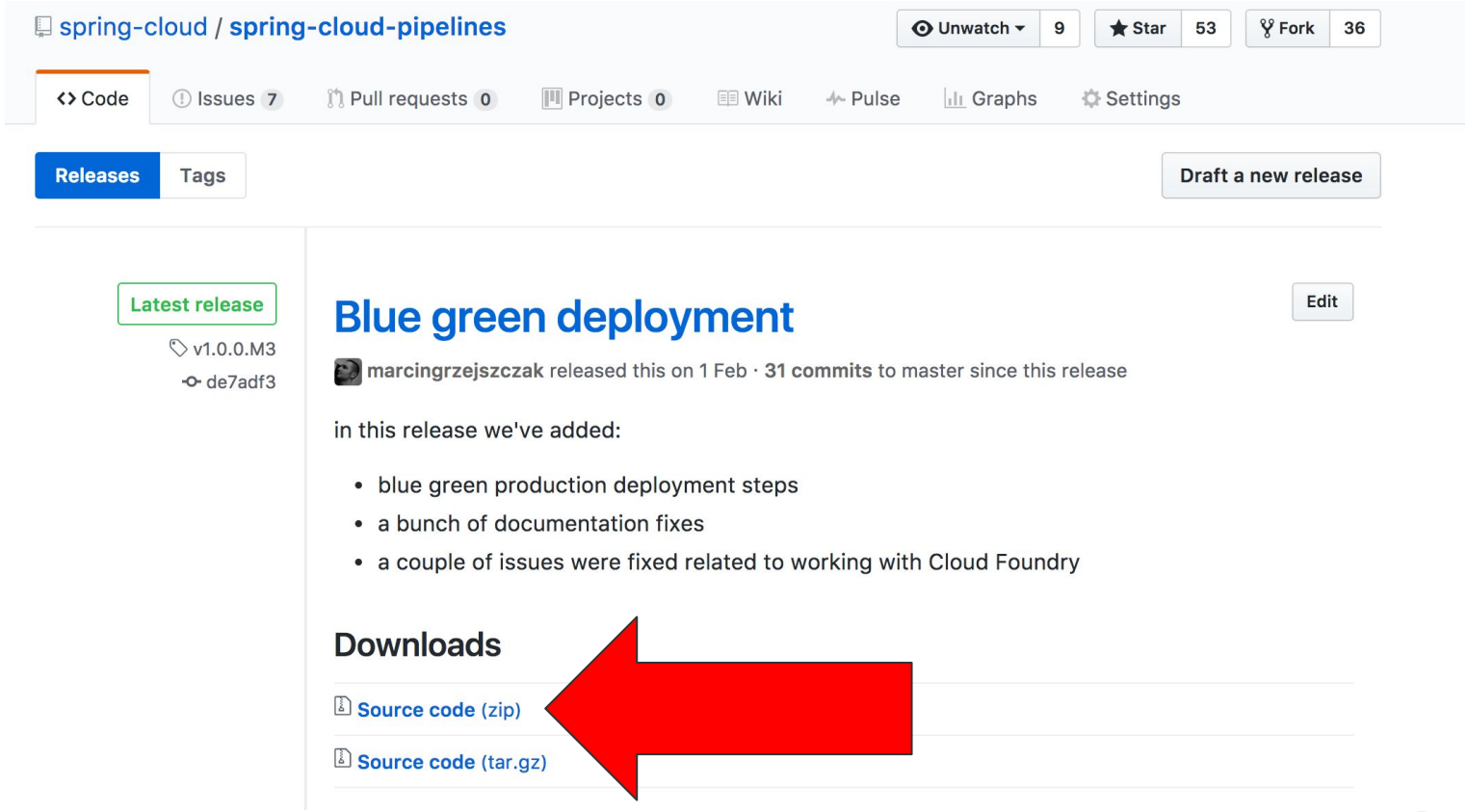
Slack notifications

spring

Spring Cloud Pipelines



Customizing Spring Cloud Pipelines



spring

Customizing Spring Cloud Pipelines

```
$ curl -LOk https://github.com/spring-cloud/spring-cloud-pipelines/archive/v1.0.0.M4.zip
$ unzip v1.0.0.M4.zip
$ cd spring-cloud-pipelines-v1.0.0.M4
$ git init
$ # modify the pipelines to suit your needs
$ git add .
$ git commit -m "Initial commit"
$ git remote add origin ${YOUR_REPOSITORY_URL}
$ git push origin master
```

Possible customizations

- Add email / Slack / HipChat notifications on build results
- Change the defaults to the ones acceptable in your company
- Add steps related to performance testing
- For Jenkins Job DSL you can scan your organization for repos and pass them to the seed job
  - that way you'll build pipelines for all your repos in no time!

Summary

- Continuous Deployment allows you to continuously deliver business value
- Spring Cloud Pipelines gives you OOB tooling to test your software via
  - unit and integration testing
  - contract testing
  - rollback testing
- Spring Cloud Pipelines allows you to easily adjust the deployment pipeline to suit your company's needs
- Thanks to Cloud Foundry you can easily do A/B & zero downtime deployment

spring

Links

- Code for this presentation (together with Prometheus submodule)
  https://github.com/marcingrzejszczak/github-analytics-demo
- Github Webhook: https://github.com/spring-cloud-samples/github-webhook
- SC-Pipelines documentation: https://cloud.spring.io/spring-cloud-pipelines/
- No end to end tests
  - https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html
  - http://www.alwaysagileconsulting.com/articles/end-to-end-testing-considered-harmful/
- Prometheus on CF https://github.com/mkuratczyk/prometheus-on-PCF
- Prometheus for PCF Dev (Docker compose) https://github.com/vegasbrianc/prometheus
- Pivotal Web Services trial : https://run.pivotal.io/
- PCF Dev (CF on your laptop) : https://docs.pivotal.io/pcf-dev/

spring

Learn More. Stay Connected.

- **Read the docs**

  http://cloud.spring.io/spring-cloud-pipelines/

- **Talk to us on Gitter**

  https://gitter.im/spring-cloud/spring-cloud-pipelines

Twitter: twitter.com/springcentral

YouTube: spring.io/video

LinkedIn: spring.io/linkedin

Google Plus: spring.io/gplus

spring

Pivotal

Transforming How The World Builds Software

mgrzejszczak