Docker Driven Continuous Delivery

On the gaps between tooling

Jeroen Peeters

A thesis presented for the degree of
Master of Science

Supervised by:
Professor H. Dekkers

The University of Amsterdam
September 2016

I, Jeroen Peeters, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

tbd.

Acknowledgements

tbd

Table of Contents

Abstract **i**

Acknowledgements **ii**

1 Introduction

2 Literature review

2.1 Introduction

2.2 Tools

2.3 Process

2.4 People

2.5 Methodoloy

3 The development organization

4 What is Continuous Delivery

4.1 Continuous Integration

4.1.1 Revision Control System

4.1.2 Continuous Integration Server

4.2 Continuous Delivery

5 Continuous Delivery at the Development Organization

5.1 Scenario's

5.1.1 Scenario 1: A new project

5.1.2 Scenario: Use a new version of Java

5.1.3 Scenario: Upgrade the application server

5.1.4 Scenario: Application instance per tester

5.1.5 Scenario: Parallel frontend test execution

5.1.6 Scenario: Install a plugin in Jenkins CI

5.1.7 Scenario: Upgrade Jenkins CI

5.1.8 Scenario: Switch to another tool

5.2 CI in a shared environment

5.2.1 The situation

5.2.2 Scenario's detailed

5.3 CI in a distributed environment

5.3.1 The situation

5.3.2 Scenario's detailed

5.4 A new project . . . .
5.4.1 Infrastructure setup . . . .
5.4.1.1 Gitlab . . . .
5.4.1.2 Jenkins and build servers . . . .
5.4.1.3 SonarQube . . . .
5.4.1.4 Nexus . . . .
5.4.1.5 Mediawiki . . . .
5.4.1.6 Deployment servers . . . .
5.4.1.7 Jira . . . .
5.4.2 Project setup . . . .
5.4.3 Recurring tasks . . . .
5.5 Configuring the CD-pipeline . . . .
5.5.1 Nexus . . . .
5.5.1.1 Add nexus to Docker dashboard. . . . .
5.5.1.2 Configure security . . . .
5.5.2 Gitlab . . . .
5.5.2.1 Add gitlab to Docker dashboard . . . .
5.5.2.2 Create root account . . . .
5.5.2.3 Create usergroup and user(s) . . . .
5.5.2.4 Create jenkins user . . . .
5.5.2.5 Import jenkins SSH key . . . .
5.5.2.6 Create project repo . . . .
5.5.3 Docker Registry . . . .
5.5.3.1 Add docker-registry to Docker dashboard . . . . .
5.5.4 Sonar . . . .
5.5.4.1 Add sonar to Docker dashboard . . . .
5.5.4.2 Install plugins . . . .
5.5.4.3 Add quality profiles . . . .
5.5.5 Reporting . . . .
5.5.5.1 Create Jira filter . . . .
5.5.5.2 Add reporting to Docker dashboard . . . .
5.5.6 Selenium . . . .
5.5.6.1 Add selenium to Docker dashboard . . . .
5.5.7 Quality-dashboard . . . .
5.5.8 Jenkins . . . .
5.5.8.1 Add jenkins to Docker dashboard . . . .
5.5.8.2 Install plugins . . . .
5.5.8.3 Configure system . . . .
5.5.8.4 Add gitlab user . . . .
5.5.8.5 Add SSH user to connect to dind-slave . . . . . .
5.5.8.6 Add dind-slave . . . .
5.5.8.7 Create job sonar-app . . . .
5.5.8.8 Create job dependency-check-app . . . .
5.5.8.9 Create job build-app . . . .
5.5.8.10 Configure job build-app . . . .
5.5.8.11 Create job build-image . . . .

5.5.8.12 Configure job build-image . . . .
5.5.8.13 Create job sonar-art . . . .
5.5.8.14 Configure job sonar-art . . . .
5.5.8.15 Create job build-art . . . .
5.5.8.16 Configure job build-art . . . .
5.5.8.17 Create job run-art . . . .
5.5.8.18 Configure job run-art . . . .
5.5.8.19 Create job load-ltcs . . . .
5.5.8.20 Configure job load-ltcs . . . .
5.5.9 Application . . . .
5.5.9.1 Add application to Docker dashboard . . . .

6 Stages of Continuous Delivery
6.1 Stage 1: CI in a shared environment . . . .
6.2 Stage 2: Automated CD in a distributed environment . . . .
6.3 Stage 3: — next evolution.. . . . .
6.4 Initial situation . . . .

References

Chapter 1

Introduction

tbd.

Chapter 2

Literature review

2.1 Introduction

tbd.

2.2 Tools

2.3 Process

2.4 People

2.5 Methodoloy

Chapter 3

The development organization

In order to understand the problems at the organization it is important to have a deeper understanding of the development organization's structure. The organization is a semi-governmental IT project organization who's mission is to help other (semi-)governmental organizations with IT project management and the realization of projects. They lead by example and help the customer to shape their project according to agile principles. In this thesis we are only concerned with the department responsible for software project realization. Within the Software Delivery (SD) department project teams build software in an agile way. Because some customers are still used to work according to a waterfall approach the department plays an important role in guiding customers. The SD project team helps the customer getting familiar with Agile/Scrum principles in order for them to steer and make decisions about importance of tasks. Before a project ends up at SD it usually follows a pre-development process in which some architectural decisions are already made. This is mostly because governments have to apply to standards and regulations. Usually the software realization team is not involved in this process since the team is not yet in existence. This procedure as described here may vary per project and customer, but it usually applies. When the realization team is formed most of the fundamental decisions have already been taken.

To be able to quickly react to customer needs the development organization relies heavily on external hiring for the duration of a project. Within SD all project members are externals. This gives the organization the ability to quickly scale up or down depending on the number of active projects. However, it also implies that knowledge is easily lost. The organization tries to move people between projects as much as possible in order to retain them. In order to move people more easily between projects and bring new people up to speed more quickly the development phase is standardized within the department as much as possible. The standardization is targeted at process, tools and development frameworks and languages. This standardization is something that can change over time and is defined by SD itself. It is possible for a single project to differentiate from the standard following the “comply or explain”-principle.

The standardized process is based on Continuous Integration and Delivery (CI/CD) principles. In the next chapter we will take a closer look at the CI/CD process.

Chapter 4

What is Continuous Delivery

In this chapter I will discuss what people generally understand by the term Continuous Delivery.

4.1 Continuous Integration

Continuous Delivery is the natural evolution of Continuous Integration (CI). Practicing Continuous Integration is an absolute necessity before you can start with Continuous Delivery.

CI focuses on integrating different software branches into a main line. This generally occurs when developers make changes to the main line in their development environments.

To do CI one needs at least the following systems:

1. Revision Control System
2. Continuous Integration Server

Figure 4.1 depicts the dependency relationship between the CI systems.

4.1.1 Revision Control System

The RCS covers the integration of code branches into a main line.

4.1.2 Continuous Integration Server

A CI-server, sometimes referred to as the build server, automatically performs the build process when a code branch is integrated into the main line. This ensures that the software in the main line can still be build according to predefined rules. Furthermore it ensures that the change doesn't depend on development specific environments, reducing the 'it builds on my machine'-problem. Preferably the CI-server also executes tests to ensure that previous functionality is still intact.

4.2 Continuous Delivery

Since Continuous Delivery (CD) builds on top of CI it reuses its systems.

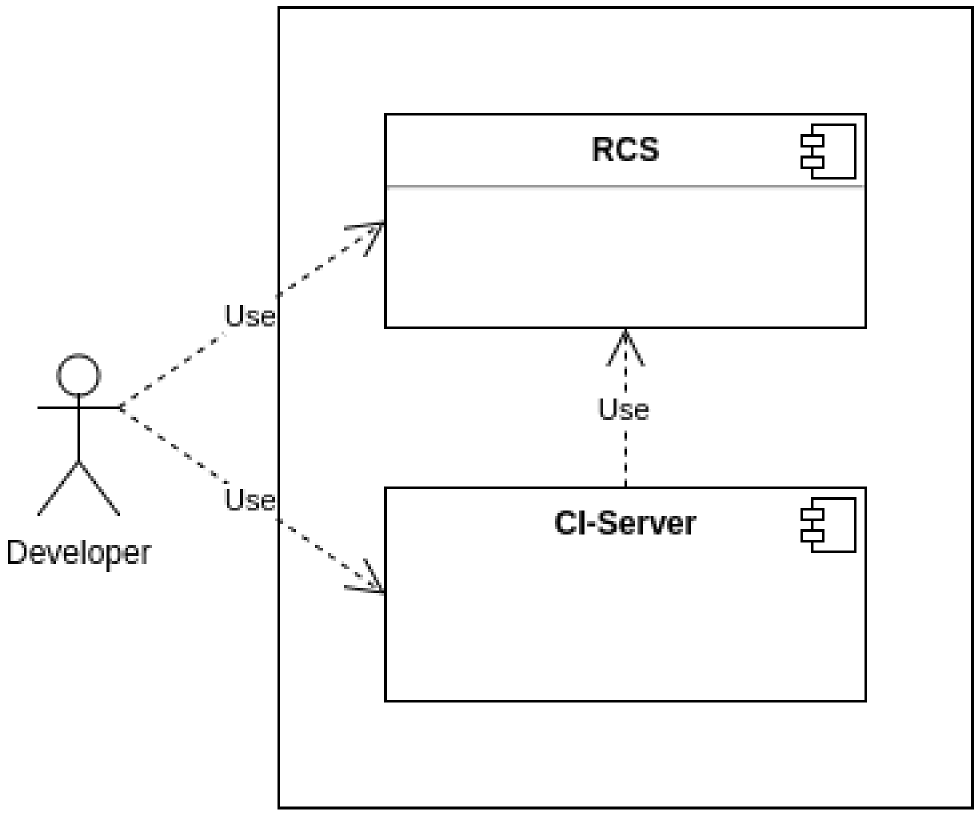

Figure 4.1: Overview Continuous Integration

Chapter 5

Continuous Delivery at the Development Organization

In this chapter I describe the technical setup and infrastructure of the CI/CD environment at the development organization. First I describe the previous situation and then the current situation. I do this by describing a set of common scenario's.

5.1 Scenario's

5.1.1 Scenario 1: A new project

When a new project is taken on by the development organization a technical infrastructure needs to be setup in order to accomodate the development process. It includes the setup of a ci/cd-pipeline, access-management and creation of serveral (virtual) deployment servers.

5.1.2 Scenario: Use a new version of Java

...

5.1.3 Scenario: Upgrade the application server

...

5.1.4 Scenario: Application instance per tester

...

5.1.5 Scenario: Parallel frontend test execution

...

5.1.6 Scenario: Install a plugin in Jenkins CI

...

5.1.7 Scenario: Upgrade Jenkins CI

...

5.1.8 Scenario: Switch to another tool

...

5.2 CI in a shared environment

This paragraph describes the previous CI/CD landscape at the development organization.

5.2.1 The situation

The systems needed for CI/CD are managed by an Ops team. All projects use a set of shared services. Figure 5.1 depicts the relationship between the Ops team and the development teams. The shared services are:

- Subversion
- Jenkins CI
- Jenkins build servers
- SonarQube
- Nexus
- Jira

Besides the shared services each project would be assigned one or more deployment servers. The deployment servers are managed by the Ops team.

The next chapter describes common scenario's that occur in a CI/CD environment on request of the development team. These scenario's describe the impact on the development team.

5.2.2 Scenario's detailed

TODO In this paragraph I will detail the aforementioned scenario's for this type of environment. Which scenario's can be implemented in this environment? Are more troublesome? Cannot be implemented? Require a lot of manual intervention/work/configuration?

5.3 CI in a distributed environment

This paragraph describes the current CI/CD landscape at the development organization.

5.3.1 The situation

Instead of managing the systems needed for CI/CD the Ops team manages a distributed environment in which teams are able to deploy applications at will and on demand.

5.3.2 Scenario's detailed

TODO In this paragraph I will detail the aforementioned scenario's for this type of environment. Which scenario's can be implemented in this environment? Are more troublesome? Cannot be implemented? Require a lot of manual intervention/work/configuration?

5.4 A new project

This paragraph conceptually describes what happens when a new project is embedded within the development organization. Besides organizational arrangements a technical infrastructure is setup to accommodate the development of the software application.

The following systems are employed:

- Gitlab
- Jenkins CI
- Jenkins build servers
- SonarQube
- Nexus
- Mediawiki
- Deployment servers
- Jira
- Releasemanager
- Quality dashboard
- Test reporting

The next paragraphs talk about the tasks that happen initially and tasks that recur more frequently.

5.4.1 Infrastructure setup

Initially every system used needs to be installed onto a target server. Depending on how you choose to do the installation, this might take some time.

5.4.1.1 Gitlab

Gitlab is used as a revision control server.

1. Install Gitlab
2. Configure authentication mechanism (LDAP)
3. Set roles and permissions for users

5.4.1.2 Jenkins and build servers

Depending on the project one or more build servers are needed. A build server has specific tooling on-board to be able to build the application. The following list details the installation steps.

1. Install Jenkins CI
2. Install one or more Jenkins build servers
3. Install specific tooling on the build server
4. Configure the build server in the main Jenkins server
5. Configure authentication mechanism (LDAP)

5.4.1.3 SonarQube

SonarQube is used to continuously monitor the quality of the source code.

1. Install SonarQube
2. Configure authentication mechanism (LDAP)

5.4.1.4 Nexus

Nexus is used to archive and distribute software artifacts.

1. Install Nexus
2. Configure authentication mechanism (LDAP)

5.4.1.5 Mediawiki

Mediawiki is used as a team collaboration tool.

1. Install Mediawiki
2. Configure authentication mechanism (LDAP)

5.4.1.6 Deployment servers

For the purpose of deploying the application in a production like environment a deployment landscape has to be setup. Depending on the application this can be as simple as a single server, or as complex as a clustered setup of a Java application server with a corresponding complex database setup.

5.4.1.7 Jira

Jira is readily available within the organization and doesn't need to be setup. However, it needs to be configured to accommodate the new project.

5.4.2 Project setup

After the infrastructure is setup the project team adds configuration to the tools to be able to build and deploy their application.

1. A user for Jenkins needs to be created in Gitlab and configured in Jenkins so that Jenkins can checkout copies of the source code.

2. Code repositories are created in Gitlab for the corresponding applications.
3. Optional, import existing source code into Gitlab.
4. Configure jobs in Jenkins to build, test and deploy (for every application)
5. Configure the location of the SonarQube API in Jenkins
6.

5.4.3 Recurring tasks

5.5 Configuring the CD-pipeline

5.5.1 Nexus

5.5.1.1 Add nexus to Docker dashboard.

Go to Docker dashboard > Apps > New App. Enter the following app definition:

```
name: nexus
version: 2.13.0-01
description: Sonatype Nexus repository manager.
pic: https://avatars0.githubusercontent.com/u/44938?v=3&s=400

#login with admin / admin123

#tags:infra

www:
 image: sonatype/nexus:2.13.0-01
 user: root
 volumes:
 - /sonatype-work
```

Click Save changes

Start app nexus

5.5.1.2 Configure security

Go to Nexus > Log in with username admin and password admin123

Go to Security > Users Select user anonymous Click Add, select Repo: All Repositories (Full Control), click OK Click Save

5.5.2 Gitlab

5.5.2.1 Add gitlab to Docker dashboard

Go to Docker dashboard > Apps > New App Enter the following app definition:

```
name: gitlab
version: 8.6.1

#tags:infra

www:
  image: www.docker-registry.isd.ictu:5000/gitlab-ce:8.6.1
  volumes:
    - /etc/gitlab
    - /var/log/gitlab
    - /var/opt/gitlab
```

Click Save changes

Start app gitlab

5.5.2.2 Create root account

Go to Gitlab New password: root@123! Confirm new password: root@123! Click Change your password

5.5.2.3 Create usergroup and user(s)

Go to Gitlab > Log in with username root and password root123!

Go to Admin Area > Groups > New Group Group path: test-group Visibility Level: Private Click Create Group

Go to Admin Area > Users > New User Name: Username: Email: Click Create user Click Edit Password: user@123! Password confirmation: user@123! Click Save Changes

Go to Admin Area > Groups > test-group Add user to group with role Developer

5.5.2.4 Create jenkins user

Go to Gitlab > Log in with username root and password root123!

Go to Admin Area > Users > New User Name: Jenkins Username: jenkins Email: noreply@jenkins.ictu Click Create user Click Edit Password: jenkins@123! Password confirmation: jenkins@123! Click Save Changes

Go to Admin Area > Groups > test-group Add user jenkins to group with role Master

5.5.2.5 Import jenkins SSH key

Go to Gitlab > Log in with username jenkins and password jenkins123!

Go to Profile Settings > SSH Keys

Key: ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCrSFSIYRJjTbWqYuU6cGQ0aNaewMQ/
noreply@jenkins.ictu

Click Add key

5.5.2.6 Create project repo

Go to Gitlab > Log in with username root and password root123!

Go to Admin Area > Projects > New Project Project path: http://www.gitlab-test.digilevering.ictu/test-group/test-project Visibility: Private Click Create project

Go to Admin Area > Projects > test-group/test-project > Edit > Protected Branches In table Already Protected select Developers can push for branch master

5.5.3 Docker Registry

5.5.3.1 Add docker-registry to Docker dashboard

Go to Docker dashboard > Apps > New App Enter the following app definition:

```
name: docker-registry
version: 2.1.1

#tags:infra

www:
 image: distribution/registry:2.1.1
 mem_limit: 2048m
 environment:
```

```
 - REGISTRY_VERSION=0.1
 - REGISTRY_LOG_FIELDS_SERVICE=registry
 - REGISTRY_LOG_FIELDS_ENVIRONMENT=production
 - REGISTRY_STORAGE_CACHE_BLOBDESCRIPTOR=inmemory
 - REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY=/docker-registry/
 - 'REGISTRY_HTTP_ADDR=:5000'
 - REGISTRY_HTTP_HEADERS_X-CONTENT-TYPE-OPTIONS=[nosniff]
 - REGISTRY_HEALTH_STORAGEDRIVER_ENABLED=true
 - REGISTRY_HEALTH_STORAGEDRIVER_INTERVAL=10s
 - REGISTRY_HEALTH_STORAGEDRIVER_THRESHOLD=3
 volumes:
 - /docker-registry
```

Click Save changes Start app docker-registry

5.5.4 Sonar

5.5.4.1 Add sonar to Docker dashboard

Go to Docker dashboard > Apps > New App Enter the following app definition:

```
name: sonar
version: 4.5.7
description: Manage code quality
pic: http://docs.sonarqube.org/download/attachments/6951171/SONAR?version=1&modificati

#tags: infra

#login with admin/admin

www:
  image: sonarqube:4.5.7
  environment:
    - SONARQUBE_JDBC_USERNAME=sonar
    - SONARQUBE_JDBC_PASSWORD=sonar
   - "SONARQUBE_JDBC_URL=jdbc:mysql://db/sonar?useUnicode=true&amp;characterEncoding=u
  volumes:
    - /opt/sonarqube/extensions/downloads
    - /opt/sonarqube/extensions/plugins
  links:
    - db
  enable_ssh: true

db:
  image: mysql:5.6
  environment:
    - MYSQL_ROOT_PASSWORD=root
    - MYSQL_DATABASE=sonar
    - MYSQL_USER=sonar
    - MYSQL_PASSWORD=sonar
  volumes:
    - /var/lib/mysql
```

Click Save changes

Start app sonar

5.5.4.2 Install plugins

Open terminal

```
cd /path/to/dir/with/sonar/plugins
scp sonar-checkstyle-plugin-2.4.jar sonar-findbugs-plugin-3.3.jar sonar-java-plugin-3.
ssh -p 22 www.sonar.<your-project>.ictu
cd /opt/sonarqube/extensions/plugins
rm *
cp ~/* .
exit
```

Go to Docker dashboard > Apps Restart sonar app

5.5.4.3 Add quality profiles

Go to Sonar > Login with username admin and password admin

Go to Quality Profiles Click Restore Profile Select ICTU Java profile to import and click Restore Click Restore Profile Select ICTU Web profile to import and click Restore

5.5.5 Reporting

5.5.5.1 Create Jira filter

Go to Jira > Log in with username isf-reporter and password Cf1NzS8INfJUB5wDQHI9

Go to Issues > Search for issues If the basic search is shown instead of the advanced search, click Advanced Enter the following query:

project = AND type in (Story, “Logical Test Case”, Systeemfunctie) ORDER BY type

Click Save as Filter name: Click Submit

Write down the filter-id of the filter (it’s displayed in the URL, https://jira.ictu-sd.nl/jira/issues/?filter=)

5.5.5.2 Add reporting to Docker dashboard

Go to Docker dashboard > Apps > New App Enter the following app definition:

```
name: reporting
version: 2.2.2
description: ISD quality reporting
pic: http://www.timetell.nl/wp-content/uploads/2015/03/logo-ictu.png

#tags: autorun

www:
  image: docker-registry.isd.ictu:5000/birt-reports:2.1.65
  mem_limit: 2g
  environment:
    - REPORT_USER=reporter
    - REPORT_PASSWORD=reporter007
    - 'REPORT_URL=jdbc:postgresql://db:5432/birt'
    - REPORT_USER_RM=reporter
    - REPORT_PASSWORD_RM=reporter007
    - 'REPORT_URL_RM=jdbc:postgresql://db:5432/birt'
  links:
    - db

importer:
  image: docker-registry.isd.ictu:5000/birt-jira-importer:2.4.1
  environment:
    - 'report_jdbc_url=jdbc:postgresql://db:5432/birt'
    - 'jira_filter=filter=<filter-id>'
  links:
    - db

trr:
 image: docker-registry.isd.ictu:5000/birt-test-results-service:2.0.50
  environment:
    - 'report_jdbc_url=jdbc:postgresql://db:5432/birt'
  links:
    - db

db:
    image: docker-registry.isd.ictu:5000/birt-database:2.0.37
    volumes:
        - /var/lib/postgresql/data
    environment:
        - POSTGRES_PASSWORD=my-secret-pw
    mem_limit: 2g

rm:
  image: docker-registry.isd.ictu:5000/releasemanager:1.0.36
  environment:
    - DB_DRIVER=pdo_pgsql
    - DB_HOST=db
    - DB_PORT=5432
    - DB_USER=releasemanager
    - DB_PASSWORD=releasemanager007
    - DB_DATABASE=birt
  volumes:
    - /mnt/publish
  links:
    - db
```

Click Save changes

Start app reporting

5.5.6 Selenium

5.5.6.1 Add selenium to Docker dashboard

Go to Docker dashboard > Apps > New App Enter the following app definition:

```
name: selenium
version: official

#tags: autorun

server:
 image: selenium/standalone-firefox
```

Click Save changes

Start app selenium

5.5.7 Quality-dashboard

TODO! detail how to setup the quality tracking and monitoring system
See http://wiki.isd.org/index.php/HandleidingKwaliteitssysteem > Opzet van een kwaliteitsdashboard

5.5.8 Jenkins

5.5.8.1 Add jenkins to Docker dashboard

Go to Docker dashboard > Apps > New App Enter the following app definition:

```
name: jenkins
version: 1.651.2
description: An extendable open source automation server
pic: http://rabellamy.github.io/DevOps-Sunset/images/jenkins.png


#tags: autorun

www:
  image: jenkins:1.651.2
  volumes:
    - /var/jenkins_home
  environment:
    - "JAVA_OPTS=-Duser.timezone=Europe/Amsterdam"
  links:
    - jnlp
  enable_ssh: true

jnlp:
  image: tehranian/dind-jenkins-slave:latest
  environment:
   - "DOCKER_DAEMON_ARGS=-H unix:///var/run/docker.sock -H tcp://0.0.0.0:2375 --insecur
    - "JAVA_OPTS=-Duser.timezone=Europe/Amsterdam"
  mem_limit: 2g
  privileged: true
  enable_ssh: true
```

Click Save changes Start app jenkins

5.5.8.2 Install plugins

Go to Jenkins > Manage Jenkins > Manage Plugins > Available

Select the following plugins:

Maven Release Plug-in Plug-in (0.14.0) Git plugin (3.0.0) SonarQube Plugin (2.4.4) OWASP Dependency-Check Plugin (1.4.3) Git client plugin (used by Git plugin) SCM API Plugin (used by Git plugin) Conditional BuildStep + Run Condition Plugin? Parameterized Trigger plugin? Static Analysis Utilities (used by OWASP Dependency-Check Plugin) Token Macro Plugin (used by OWASP Dependency-Check Plugin) Workspace Cleanup Plugin? Build Pipeline Plugin?

5.5.8.3 Configure system

Go to Jenkins > Manage Jenkins > Configure System

SonarQube servers Click Add SonarQube Name: SonarQube Server URL: http://www.sonar..ictu:9000 Server version: 5.1 or lower Click Advanced Version of sonar-maven-plugin: 3.0.1 Database URL: jdbc:mysql://db.sonar..ictu:3306/sonar Database login: sonar Database password: sonar

Git plugin Global Config user.name Value: jenkins Global Config user.email Value: noreply@jenkins.ictu

Maven Click Add Maven Name: Maven 3.3.9 Version: 3.3.9

E-mail Notification SMTP server: smtp.isd.org

Click Save

5.5.8.4 Add gitlab user

Go to Jenkins > Credentials > (global) > Add credentials Kind: SSH username with private key Scope: Global Username: jenkins Private key: Enter directly Key:

```
—–BEGIN RSA PRIVATE KEY—– MIIEowIBAAKCAQEAq0hUiGESY021qmLlOnBkNGjWnsDEP5t
AkAKCQ8MLj4n7X4/XD+lK05RaihkSri5QA+6q3/ZEDks4H8Aye5UmGPEKjD+FNQr
i+SLjFf2cINJ6QPUpz6Q/CnK8TEICmethddsaEfr1yzEoDJ08Lu6JWVUKYXTHL7t
h+boFeJATcskXykngw2rbGCp8FYq4SwUTLXRUy6qNsBXvHTDP9pmSIhpeSl4NIAl
6Q9lRBYlQiDQERi7HYfZ859oMS6GoCG188VoxkaNLbuvRrktmKjY+xl+UCTKUz+F
AFj78psdU4iaplgGrXo3X4WHiVqEywOnZ8WhVQIDAQABAoIBAHSw9W5oe+eNpMut
TrBuq8YM+tLzT4BqIgqxw2+J+cU0Lv6lE9z5lhyv1MOYcwlZLn+BmNyVIeBqHlHN
4d+kF7AJjO+BlHJp9DaemaGsrpN0B1ZXakeHcA8wSmRC/dKzWmiKtqolKu8BUZIN
Kmn55xBwl1tkU500YvkzXFFn5FvYg7NzvdgajfrywgU6GIm6miGWzkb0F5MRnXtb
hhx32sSu9H87Fu54DpMIWQzzpqJuPPLL8SxZKheuceYcV/tXT6IG5WnSl3KnNVYX
```

cSLRN4miN6dVlX9GIdwsAxdrqexm/OLw/J5Mgf2SaIKslMNoGciojtaQaLt/ofde
UWa0lWECgYEA3kg9Tv8/iIWIzIYvgC8D/ZyjyExlcX6jVSrbFCjZyYhDqxA2LPSa
SQwcHesLk4c4GS1in47Mo/otxdPYnmu2o0Ob4hoICGLEJiWdiI7ljJ8CWtsjNGts
GhCsv3guTT1WnfOpWbsaKDodinEa2YNekxAjh/9EQZR4x6Y9EQ61C8kCgYEAxUOm
W8orY6M4YYeb1nA9izW1twjrK0jZ6u07n3ooAVJx5WNr69/ATZx/Lw4ou1RGZ3qb
q4/iol5DwenxcaETV5h5q/170pvnhkLoRIWd+Rd+oEu+GtYwVrLd/ybJZswJMYc/
KIRHrw3DpM0yVxrilENe3TQ8k2O3VJRsucmHly0CgYEA2Jq+m5dh2vB9MQhli1zF
X8LfWxUPGXzVPu4HFGsGZzvQ7QZcNIybOCmD0Ke13So8QVSXsXJe+j+VkRxyD1ZZ
YFlGsxq4zysnhyDKlULib5iXm9/FO5Sef/vVyrMbM4tdN4g0c8s+nwqatMio6GL6
qwZkCWd3pQxAchUNluylAfkCgYArkDIH6VDFs0D7QOBobecZfCYCIuUUbQU6/WMC
aA63pAZlGxy1PXeRbDMmKCFUpVra9Ve1fpQVOW4LP+fDKUhFOvX7xoH209lAbDwx
DbUCUm7zZWa5NH3+V4fxFha6LesF1hFbmELgZNDE70/jrptFFM+5WBTck9PjyNdt
/BSGjQKBgDS8bI/B6uMos882AG4eOcUBEVdaTaOIBqJEM0s5u8PPNBaXsx4kfH62
9vnfrX3tf8fj3UgrIsqEg/N2Pze2ktj8ikqz4cIJqX0fHHvEYC+FvqDcdit14Cv9
q0lAlP1AXSP4kry7SguwMTlewfcMUXxwTEIs0PXujqx8uTBLnUBY —–END
RSA PRIVATE KEY—–

Description: gitlab user

Click OK

5.5.8.5 Add SSH user to connect to dind-slave

Go to Jenkins > Credentials > (global) > Add credentials

Kind: Username with password Scope: Global Username: jenkins Password: jenkins Description: SSH user to connect to dind-slave

Click OK

5.5.8.6 Add dind-slave

Go to Jenkins > Manage Jenkins > Manage Nodes > New Node

Node name: dind-slave Select Dumb Slave Click OK

number of executors: 2 Remote root directory: /tmp Labels: docker Usage: Only build jobs with label restrictions matching this node Launch method: Launch slave agents on Unix machines via SSH Host: jnlp.jenkins..ictu Credentials: jenkins (SSH user to connect to dind-slave)

Click Save

5.5.8.7 Create job sonar-app

Go to Jenkins > New item

Item name: sonar-app Select Maven project

Click OK Configure job sonar-app Go to Jenkins > sonar-app > Configure

Select Discard old builds Max # of builds to keep: 5

Source Code Management Select Git Repository URL: git@www.gitlab..ictu:test-group/test-project.git Credentials: jenkins (gitlab user)

Build Triggers Unselect all options

Build Root POM: testapp/pom.xml Goals and options: clean install -DskipTests

Post-build Actions Click Add post-build action, select SonarQube analysis with Maven

Click Save

5.5.8.8 Create job dependency-check-app

Go to Jenkins > New item

Item name: dependency-check-app Select Maven project

Click OK Configure job dependency-check-app Go to Jenkins > dependency-check-app > Configure

Select Discard old builds Max # of builds to keep: 5

Source Code Management Select Git Repository URL: git@www.gitlab..ictu:test-group/test-project.git Credentials: jenkins (gitlab user)

Build Triggers Unselect all options

Build Root POM: testapp/pom.xml Goals and options: clean install -DskipTests

Post Steps Select Run regardless of build resultaat Click Add post-build step, select InvokeOWASP Dependency-Check analysis Click Advanced Select Generate optional HTML reports

Post-build Actions Click Add post-build action, select Publish OWASP Dependency-Check analysis results

Click Save

5.5.8.9 Create job build-app

Go to Jenkins > New item

Item name: build-app Select Maven project

Click OK

5.5.8.10 Configure job build-app

Go to Jenkins > build-app > Configure

Select Discard old builds Max # of builds to keep: 5

Source Code Management Select Git Repository URL: git@www.gitlab..ictu:test-group/test-project.git Credentials: jenkins (gitlab user) Additional Behaviours Click Add, select Check out to specific local branch Branch name: master

Build Triggers Select Poll SCM Schedule: H/5 * * * *

Build environment Select Maven release build Default versioning mode: None

Build Root POM: testapp/pom.xml Goals and options: clean install

Click Save

5.5.8.11 Create job build-image

Go to Jenkins > New item

Item name: build-image Select Freestyle project

Click OK

5.5.8.12 Configure job build-image

Go to Jenkins > build-image > Configure

Select Discard old builds Max # of builds to keep: 5

Select Restrict where this project can be run Label Expression: docker

Source Code Management Select Git Repository URL: git@www.gitlab..ictu:test-group/test-project.git Credentials: jenkins (gitlab user)

Build Click Add build step, select Execute shell Command:

cd docker ./build.sh

5.5.8.13 Create job sonar-art

Go to Jenkins > New item

Item name: sonar-art Select Maven project

Click OK

5.5.8.14 Configure job sonar-art

Go to Jenkins > sonar-art > Configure

Select Discard old builds Max # of builds to keep: 5

Source Code Management Select Git Repository URL: git@www.gitlab..ictu:test-group/test-project.git Credentials: jenkins (gitlab user)

Build Triggers Unselect all options

Build Root POM: testART/pom.xml Goals and options: clean install -DskipTests

Post-build Actions Click Add post-build action, select SonarQube analysis with Maven

Click Save

5.5.8.15 Create job build-art

Go to Jenkins > New item

Item name: build-art Select Maven project

Click OK

5.5.8.16 Configure job build-art

Go to Jenkins > build-art > Configure

Select Discard old builds Max # of builds to keep: 5

Source Code Management Select Git Repository URL: git@www.gitlab..ictu:test-group/test-project.git Credentials: jenkins (gitlab user) Additional Behaviours Click Add, select Check out to specific local branch Branch name: master

Build environment Select Maven release build Release goals and options: -Dresume=false release:prepare release:perform -Darguments="-DskipTests" DryRun goals and options: -Dresume=false -DdryRun=true release:prepare

-Darguments="-DskipTests" Default versioning mode: None

Build Root POM: testART/pom.xml Goals and options: clean install -DskipTests

Click Save

5.5.8.17 Create job run-art

Go to Jenkins > New item

Item name: run-art Select Maven project

Click OK

5.5.8.18 Configure job run-art

Go to Jenkins > run-art > Configure

Select Discard old builds Max # of builds to keep: 5

Select This build is parametrized Click Add Parameter, select String Parameter Name: browserType Default Value: FIREFOX Click Add Parameter, select String Parameter Name: seleniumServerUrl Default Value: http://server.selenium..ictu:4444/wd/hub Click Add Parameter, select String Parameter Name: applicationServerUrl Default Value: http://www.testapp.digilevering.ictu:8080

Source Code Management Select Git Repository URL: git@www.gitlab..ictu:test-group/test-project.git Credentials: jenkins (gitlab user)

Build Root POM: testART/pom.xml Goals and options: -DbrowserType=*browserType−DseleniumServerUrl* =seleniumServerUrl -DapplicationServerUrl=$applicationServerUrl clean test

Post Steps Select Run only if build succeeds Click Add post-build step, select Execute shell Command:

```
export \
  URL="http://trr.reporting.<your-project>.ictu:4567/upload" \
  APP_NAME="Testapp" \
  APP_VERSION="SNAPSHOT" \
  TEST_DESCRIPTION="ART Testapp" \
  TEST_USER="Jenkins" \
  TEST_VERSION="Master" \
  TEST_TARGET="$applicationServerUrl" \
  TEST_PLATFORM="$browserType" \
  TEST_RUN="ART" \
  DIR="testART/target/surefire-reports/junitreports"
```

```
# Parallele upload van resultaat
echo "Sending reports in ${DIR}"
echo "${APP_NAME}"
for file in $DIR/\*.xml; do
  [ -f $file ] || continue
  echo $file
done | xargs -I{} --max-procs 0 bash -c '
  curl ${URL} \
    -s \
    -F "junit=@{}" \
    -F "application_name=${APP_NAME}" \
    -F "application_version=${APP_VERSION}" \
    -F "testrun_description=${TEST_DESCRIPTION}" \
    -F "testrun_user=${TEST_USER}" \
    -F "testrun_version=${TEST_VERSION}" \
    -F "test_target=${TEST_TARGET}" \
    -F "test_platform=${TEST_PLATFORM}" \
    -F "testrun=${TEST_RUN}" \
```

5.5.8.19 Create job load-ltcs

Go to Jenkins > New item

Item name: load-ltcs Select Freestyle project

Click OK

5.5.8.20 Configure job load-ltcs

Go to Jenkins > load-ltcs > Configure

Select Discard old builds Max # of builds to keep: 5

Build Triggers Select Build periodically Schedule: H 6-20 * * 1-5

Build Click Add build-step, select Execute shell Command:

```
#!/bin/bash -ex

# JIRA importer aanroepen
data=$(curl -s http://importer.reporting.<your-project>.ictu:4567/import)

if [ "$data" == 'Import completed' ]
then
```

```
  exit 0
else
  exit 1
fi
```

5.5.9 Application

5.5.9.1 Add application to Docker dashboard

Go to Docker dashboard > Apps > New App Enter the following app definition:

```
name: testapp
version: latest

www:
 image: www.docker-registry.<your-project>.ictu:5000/testapp:latest
  enable_ssh: true
```

Click Save changes

Start app testapp

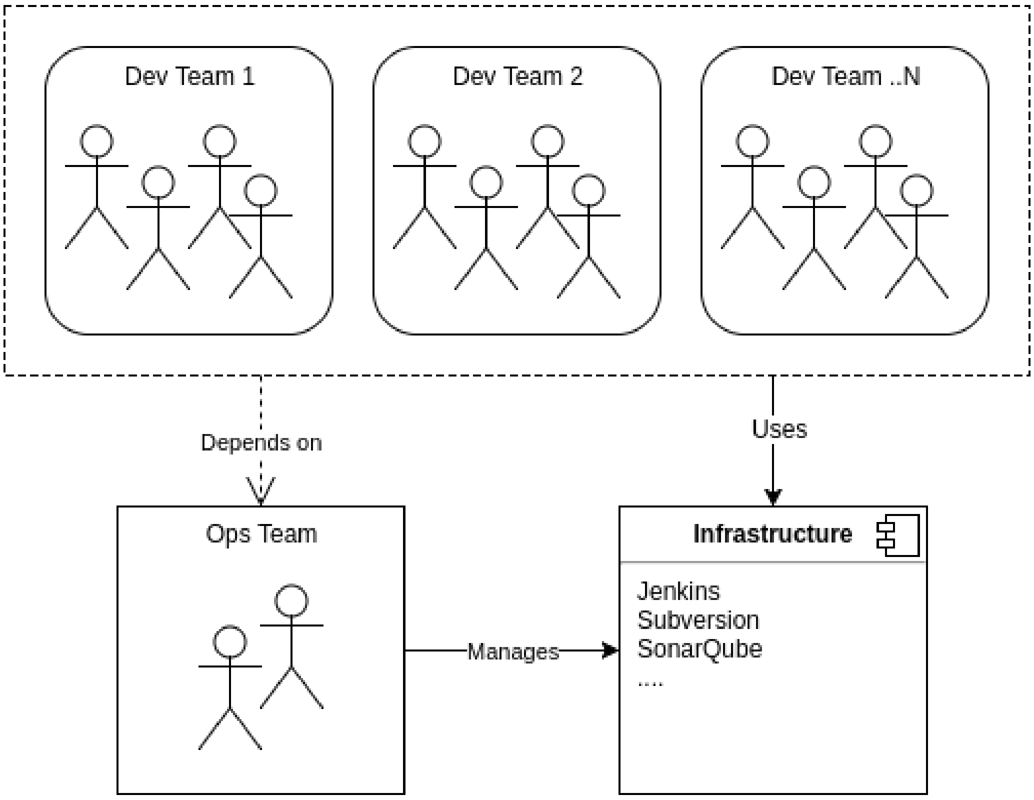

Figure 5.1: Relationship between Ops and Development teams in a shared environment

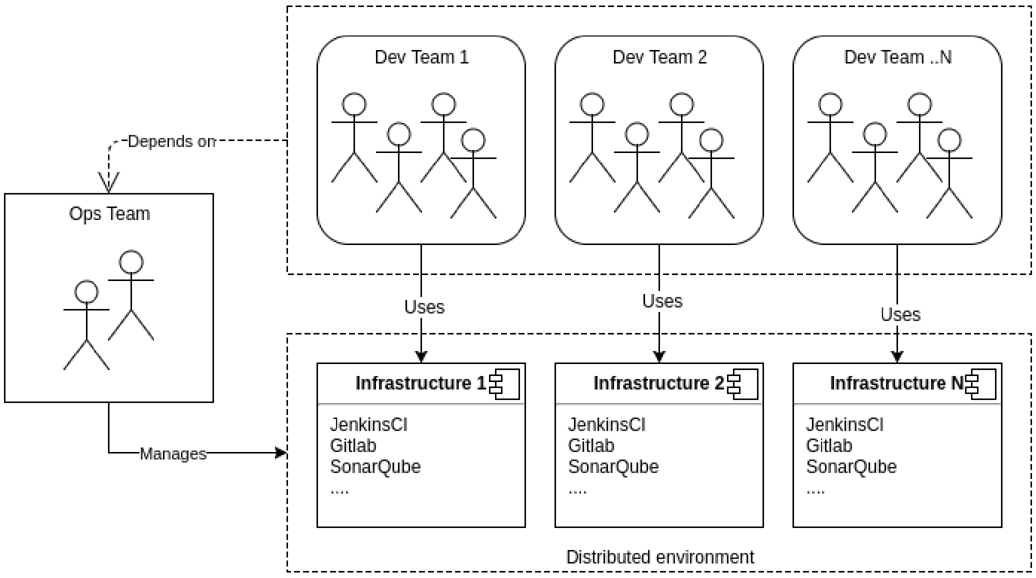

Figure 5.2: Relationship between Ops and Development teams in a distributed environment

Chapter 6

Stages of Continuous Delivery

In this chapter I describe the different stages of continuous delivery that the development organization went through.

Table 6.1: Demonstration of simple table syntax.

Right Le	ft	Center De	fault
12 12		12	12
123 12	3	123	123
1 1		1	1

6.1 Stage 1: CI in a shared environment

Characteristics

.	
CI/CD Environment	Shared
Maintenance	System Administrator
Deployment	Manual
Flexibility	Static

Systems

Server	Type	Depends on
Subversion	Version Control	
Jenkins	Build Server, CI	Nexus, Sonar, Selenium
Nexus	Artifact Repository	
Sonar	Static Code Analysis	

Server	Type	Depends on
Selenium		Deployment Server
Deployment Server		Nexus

Tools

Tool	Type	Used by	Depends on
Maven	Build	Dev, Jenkins	
Java	Language, platform	Dev, Jenkins	
Custom quality reporting	Reporting	Jenkins	Sonar, Selenium

Setup

- Setup is done by system administrators

Manual steps

Task	Depends on	Occurrence
Create build job	Jenkins	every new unit of development
Create deployment job	Jenkins	every new unit of development
Configure quality report	Jenkins, Quality reporting	every new unit of development
Maintain server configuration	Deployment Server	on configuration change
Trigger deployment	Deployment Server, Jenkins	on request of tester or stakeholder
Trigger automated tests	Deployment Server, Jenkins, Selenium	every iteration

Problems

Description	Has negative impact on
Resource sharing between all teams	Scalability
Changes and upgrades affect all teams	Stability
Teams can't change setup or install plugins	Flexibility, Usability
Teams can interfere with each other	Stability
Teams depend on sysadmins	Agility

Description	Has negative impact on
Deployment server changes are difficult to reverse	Flexibility, Scalability
Unable to deploy multiple instances of an application	Agility, Usability

6.2 Stage 2: Automated CD in a distributed environment

Characteristics

.	
CI/CD Environment	Per team
Maintenance	Team
Deployment	Automatic
Flexibility	On-demand

- Each team has his own CI/CD environment
- The team is responsible for the environment (DevOps)
- Dynamic deployment cluster
- Application deployment is scripted
- System administrators maintain the deployment cluster
- Teams decide what their CI/CD landscape looks like

Systems

- Gitlab
- Jenkins
- Nexus
- Sonar
- Selenium
- Deployment server

Tools

- Maven
- Docker
- Custom quality reporting

Manual steps

- s1

Problems

- p1

6.3 Stage 3: — next evolution..

tbd..

6.4 Initial situation

! This needs to be placed elsewhere and rewritten !

Figure 6.2 shows the steps and interactions a developer has with build systems in order to deploy a change in the software to a target server.

Figure 6.3 shows the steps a developer needs to take in order to setup a single source repository and configure the continuous integration pipeline.

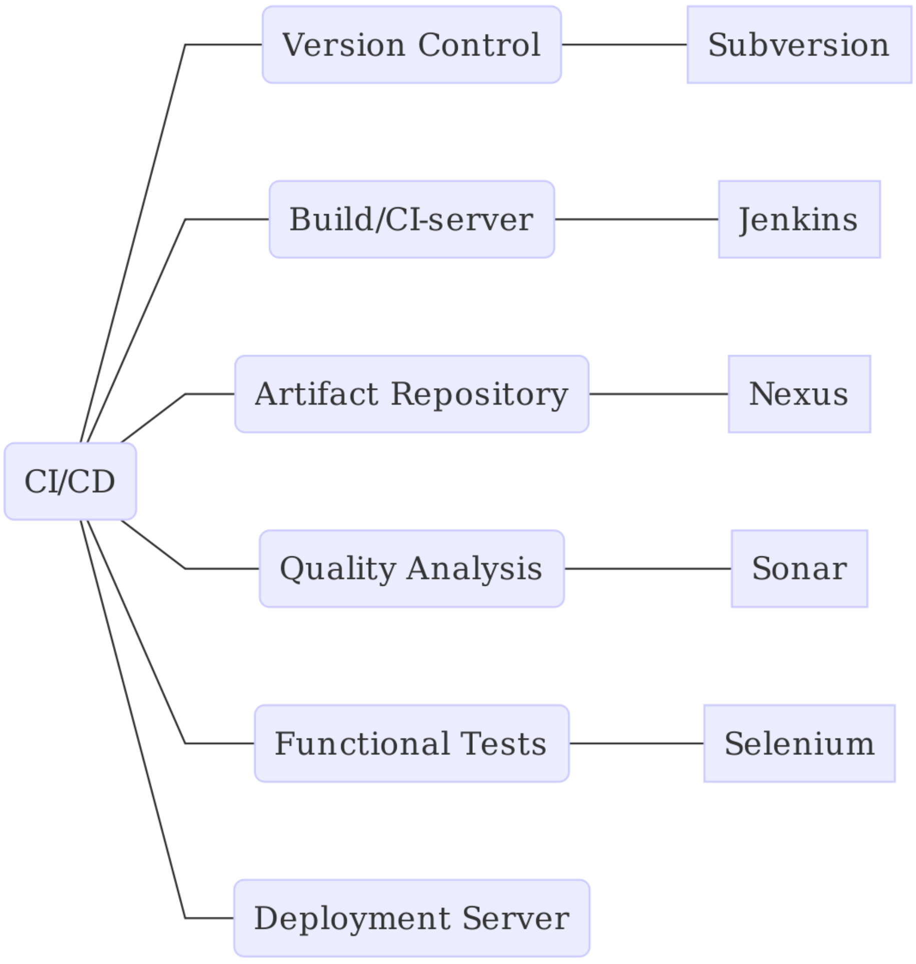

Figure 6.1: CI/CD Schematic Overview

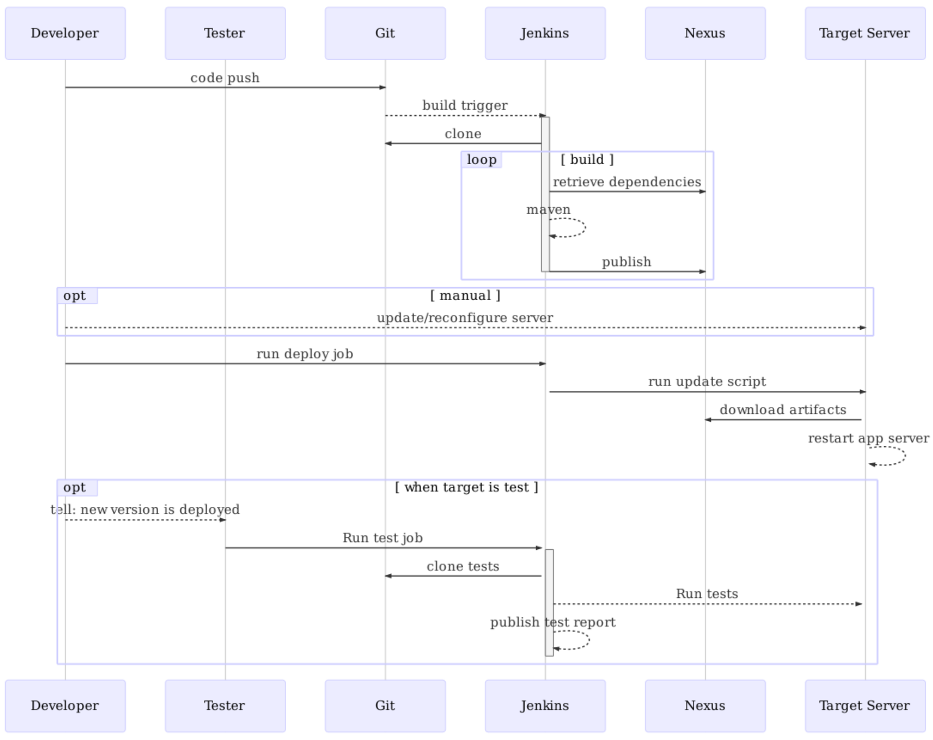

Figure 6.2: Basic CI

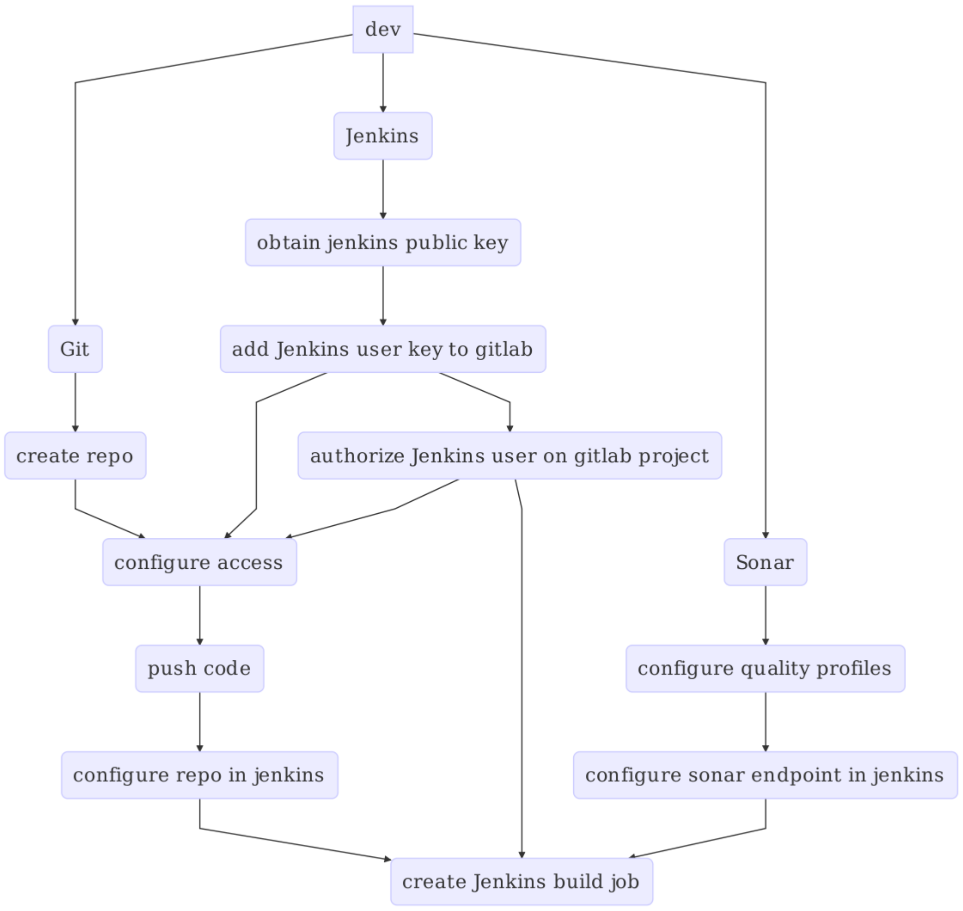
dev
Jenkins
obtain jenkins public key
Git
add Jenkins user key to gitlab
create repo
authorize Jenkins user on gitlab project
configure access
Sonar
push code
configure quality profiles
configure repo in jenkins
configure sonar endpoint in jenkins
create Jenkins build job

Figure 6.3: Basic CI setup

References