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

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.

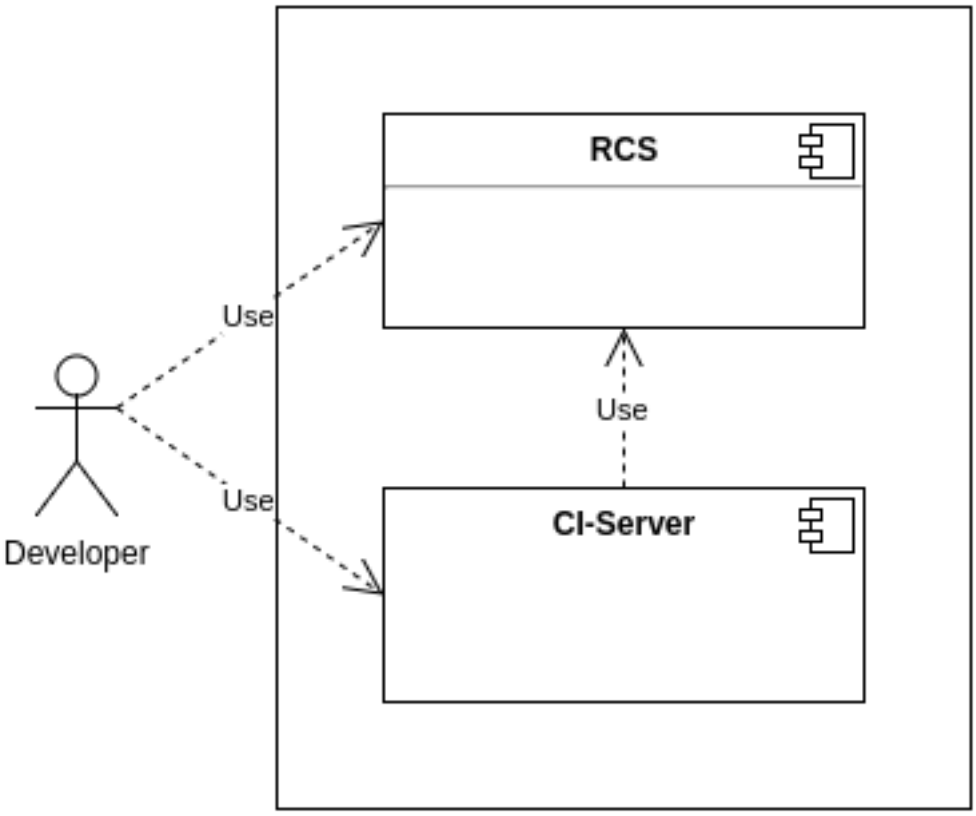

Figure 4.1: Overview Continuous Integration

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 Docker Dashboard

Every project team is equipped with a Docker Dashboard. The dashboard is a web application through which the team can manage running applications on the Docker infrastructure. The dashboard exposes a user interface and a programmable interface for automation purposes. Through the dashboard the team manages:

1. Deployment of CD-pipeline support services
2. Deployment of the application under development

Upon project start a vanilla dashboard is deployed. The team has the freedom to start any combination of Docker containers.

5.5.2 Configuration

Property	Value	Description
jira-reporter-user	reporter	
jira-reporter-password	****	

5.5.3 Nexus

5.5.3.1 Add nexus to Docker dashboard.

Go to the Docker Dashboard user interface, click 'Apps', 'New App'. Enter the following app definition:

```
name: nexus
version: 2.13.0-01
description: Sonatype Nexus repository manager.

#login with admin / admin123

#tags:infra

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

Click 'Save changes' and start the application.

5.5.3.2 Configure security

Go to the Nexus user interface and log in with

	Username	Password
Account	admin	admin123

Go to 'Security', 'Users'. Select user ***anonymous***. Give the user full control over all repositories. Click Add, select Repo: 'All Repositories (Full Control)'. Click 'OK' and 'Save'.

5.5.4 Gitlab

5.5.4.1 Add Gitlab to Docker dashboard

Go to the Docker Dashboard user interface, click 'Apps', 'New App'. Enter the following app definition:

```
name: gitlab
version: 8.6.1

#tags:infra

www:
  image: www.docker-registry.isd.tld:5000/gitlab-ce:8.6.1
  volumes:
```

```
- /etc/gitlab
- /var/log/gitlab
- /var/opt/gitlab
```

Click ‘Save changes’ and start the application.

5.5.4.2 Create root account

Go to the Gitlab user interface. You will be asked to enter a new password for the root user. Enter the password twice and click ‘Change your password’.

5.5.4.3 Create usergroup and user(s)

Go to the Gitlab user interface. Log in with:

	Username	Password
Account	root	***see previous step***

Go to ‘Admin Area’, ‘Groups’, ‘New Group’. Enter:

Group path	Visibility Level
test-group	Private

Click ‘Create Group’.

Go to ‘Admin Area’, ‘Users’, ‘New User’. Enter:

Name	Username	Email

Click ‘Create user’.

Click ‘Edit’ and enter:

Password	Password confirmation
user@123!	user@123!

Click ‘Save Changes’.

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

Repeat for each user in the development team.

5.5.4.4 Create Jenkins user

Jenkins should be able to login to Gitlab in order to be able to checkout copies of the source code. Therefore a dedicated user should be created.

Go to the Gitlab user interface. Log in with:

	Username	Password
Account	root	***see previous step***

Go to ‘Admin Area’, ‘Users’, ‘New User’. Enter:

Name	Username	Email
Jenkins	jenkins	noreply@jenkins.tld

Click ‘Create User’.

Click ‘Edit’ and enter:

Password	Password confirmation
jenkins@123!	jenkins@123!

Click ‘Save Changes’.

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

5.5.4.5 Import Jenkins SSH public key

Go to the Gitlab user interface. Log in with:

	Username	Password
Account	jenkins	jenkins@123!

Go to ‘Profile Settings’, ‘SSH Keys’. Enter the SSH public key:

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCrSFSIYRJjTbWqYuU6cGQ0aNae
wMQ/m0k3m/MA7mNb2LEGNb0CQAoJDwwuPiftfj9cP6UrTlFqKGRKuLlAD7qrf9kQ
OSzgfwDJ7lSYY8QqMP4U1CuL5IuMV/Zwg0npA9SnPpD8KcrxMQgKZ62F12xoR+vX
LMSgMnTwu7olZVQphdMcvu2H5ugV4kBNyyRfKSeDDatsYKnwVirhLBRMtdFTLqo2
wFe8dMM/2mZIiGl5KXg0gCXpD2VEFiVCINARGLsdh9nzn2gxLoagIbXzxWjGRo0t
u69GuS2YqNj7GX5QJMpTP4UAWPvymx1TiJqmWAatejdfhYeJWoTLA6dnxaFV
```

Click ‘Add key’.

5.5.4.6 Create project repository

Go to the Gitlab user interface. Log in with:

	Username	Password
Account	root	***see previous step***

Go to ‘Admin Area’ > ‘Projects’ > ‘New Project’. Enter:

Project Path	Visibility
/test-group/test-project	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.

5.5.5 Docker Registry

5.5.5.1 Add docker-registry to Docker dashboard

Go to the Docker Dashboard user interface, click ‘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' and start the application.

5.5.6 Sonar

5.5.6.1 Add sonar to Docker dashboard

Go to the Docker Dashboard user interface, click 'Apps', 'New App'. Enter the following app definition:

```
name: sonar
version: 4.5.7
description: Manage code quality

#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' and start the application.

5.5.6.2 Install plugins

From a local shell execute:

```
cd /path/to/dir/with/sonar/plugins
scp sonar-checkstyle-plugin-2.4.jar \
    sonar-findbugs-plugin-3.3.jar   \
    sonar-java-plugin-3.14.jar      \
    sonar-pmd-plugin-2.5.jar        \
    sonar-web-plugin-2.4.jar        \
    www.sonar.<your-project>.tld

ssh www.sonar.<your-project>.tld
cd /opt/sonarqube/extensions/plugins
rm * && cp ~/* .
exit
```

Go to the Docker Dashboard user interface and restart the Sonar application.

5.5.6.3 Add quality profiles

Go to the Sonar user interface. Login with:

	Username	Password
Account	admin	admin

Go to 'Quality Profiles', Click 'Restore Profile'. Select 'Development Organization Java profile' to import and click 'Restore'. Click 'Restore Profile'. Select

'Development Organization Web profile' to import and click 'Restore'.

5.5.7 Reporting

5.5.7.1 Create Jira filter

Go to the Jira user interface. Login with:

	Username	Password
Account	jira-reporter-user	jira-reporter-password

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

```
project = <Jira project name> AND type in (Story, "Logical Test Case", Systeemfunctie) ORD
```

Click 'Save as'. Filter name: . Click 'Submit'. Write down the filter-id of the filter (it's displayed in the URL, https://jira.development-organization.nl/jira/issues/?filter=).

5.5.7.2 Add reporting application to Docker Dashboard

Go to the Docker Dashboard user interface, click 'Apps', 'New App'. Enter the following app definition:

```
name: reporting
version: 2.2.2
description: Quality reporting

#tags: autorun

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

db:
    image: docker-registry.isd.tld: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.tld: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’ and start the application.

5.5.8 Selenium

5.5.8.1 Add Selenium to Docker dashboard

Go to the Docker Dashboard user interface, click 'Apps', 'New App'. Enter the following app definition:

```
name: selenium
version: official

#tags: autorun

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

Click 'Save changes' and start the application.

5.5.9 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

5.5.10 Jenkins

5.5.10.1 Add Jenkins to Docker dashboard

Go to the Docker Dashboard user interface, click 'Apps', 'New App'. Enter the following app definition:

```
name: jenkins
version: 1.651.2
description: An extendable open source CI server

#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' and start the application.

5.5.10.2 Install plugins

Go to the Jenkins user interface. Click 'Manage Jenkins', 'Manage Plugins', 'Available'. Tick the box next to 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
- Workspace Cleanup Plugin
- Build Pipeline Plugin

5.5.10.3 Configure system

Go to the Jenkins user interface. Click 'Manage Jenkins', 'Configure System'. Look for the section 'SonarQube servers'. Click 'Click Add SonarQube'. Enter:

Property	Value
Name	SonarQube
Server URL	http://www.sonar..tld:9000
Server version	5.1 or lower
Version of sonar-maven-plugin	3.0.1
Database URL	jdbc:mysql://db.sonar..tld:3306/sonar

Property	Value
Database login	sonar
Database password	sonar

Look for the section ‘Git plugin, Global config’. Enter:

Property	Value
user.name	jenkins
user.email	noreply@jenkins.tld

Look for the section ‘Maven’. Click ‘Add Maven’. Enter:

Property	Value
Name	Maven 3.3.9
Version	3.3.9

Look for the section ‘E-mail Notification’. Enter:

Property	Value
SMT server	smtp.isd.org

Click ‘Save’.

5.5.10.4 Add Gitlab user

Go to the Jenkins user interface. Click ‘Credentials’, ‘Global’, ‘Add credentials’. Enter:

Property	Value
Kind	SSH username with private key
Scope	Global
Username	jenkins
Private key	Enter directly
Description	Gitlab user

Enter the key:

```
-----BEGIN RSA PRIVATE KEY-----
```

```
MIIEowIBAAKCAQEAq0hUiGESY021qmLlOnBkNGjWnsDEP5tJN5vzAO5jW9ixBjW9
AkAKCQ8MLj4n7X4/XD+lKO5RaihkSri5QA+6q3/ZEDks4H8Aye5UmGPEKjD+FNQr
i+SLjFf2cINJ6QPUpz6Q/CnK8TEICmethddsaEfr1yzEoDJO8Lu6JWVUKYXTHL7t
h+boFeJATcskXykngw2rbGCp8FYq4SwUTLXRUy6qNsBXvHTDP9pmSIhpeSl4NIAl
6Q9lRBYlQiDQERi7HYfZ859oMS6GoCG188VoxkaNLbuvRrktmKjY+xl+UCTKUz+F
AFj78psdU4iaplgGrXo3X4WHiVqEywOnZ8WhVQIDAQABAoIBAHSw9W5oe+eNpMut
TrBuq8YM+tLzT4BqIgqxw2+J+cUOLv6lE9z5lhyv1MOYcwlZLn+BmNyVIeBqHlHN
4d+kF7AJjO+BlHJp9DaemaGsrpNOB1ZXakeHcA8wSmRC/dKzWmiKtqolKu8BUZIN
Kmn55xBwl1tkU50OYvkzXFFn5FvYg7Nzvdgajfrywg U6GIm6miGWzkb0F5MRnXtb
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-----
```

Click ‘OK’.

5.5.10.5 Add SSH user to connect to dind-slave

Go to the Jenkins user interface. Click ‘Credentials’, ‘Global’, ‘Add credentials’. Enter:

Property	Value
Kind	SSH username with password
Scope	Global
Username	jenkins
Password	jenkins
Description	SSH user to connect to dind-node

Click ‘OK’

5.5.10.6 Add dind node

Go to the Jenkins user interface. Click 'Manage Jenkins', 'Manage Nodes', 'New Node'. Enter:

Property	Value
Node name	docker-in-docker
Type	Dumb Slave
Number of executors	4
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..tld
Credentials	jenkins (SSH user to connect to dind-node)

Click 'Save'

5.5.10.7 Create Jenkins job for Sonar

Go to the Jenkins user interface. Click 'New item'.

Property	Value
Item name	sonar-app
Type	Maven project

Click 'OK'.

Configure job sonar-app

Go to the Jenkins user interface. Click 'sonar-app', 'Configure'.

	Property	Value
Discard old builds	Max # of builds to keep	5
Git	Repository URL	git@www.gitlab..tld:test-group/test-project.git
	Credentials	jenkins (gitlab user)
Build	Root POM	testapp/pom.xml
	Goals	clean install -DskipTests
Post-build Actions	***select***	SonarQube analysis with Maven

Click 'Save'.

5.5.10.8 Create Jenkins job for OWASP dependency checker

Go to the Jenkins user interface. Click 'New item'.

Property	Value
Item name	dependency-check-app
Type	Maven project

Click 'OK'.

Configure job dependency-check-app

Go to the Jenkins user interface. Click 'dependency-check-app', 'Configure'.

	Property	Value
Discard old builds	Max # of builds to keep	5
Git	Repository URL	git@www.gitlab..tld:test-group/test-project.git
	Credentials	jenkins (gitlab user)
Build	Root POM	testapp/pom.xml
	Goals	clean install -DskipTests
Post Steps	***select***	Run regardless of build resultaat
		Invoke OWASP Dependency-Check analysis
	click	Advanced
	select	Generate optional HTML reports
Post-build Actions	***select***	Publish OWASP Dependency-Check analysis results

Click 'Save'.

5.5.10.9 Create job build-app

Go to the Jenkins user interface. Click 'New item'.

Property	Value
Item name	build-app
Type	Maven project

Click 'OK'.

Configure job build-app Go to the Jenkins user interface. Click 'build-app', 'Configure'.

	Property	Value
Discard old builds	Max # of builds to keep	5
Git	Repository URL	git@www.gitlab..tld:test-group/test-project.git
	Credentials	jenkins (gitlab user)
Additional Behaviours		Check out to specific local branch
	Branch name	master
Build Triggers	***select***	Poll SCM
	Schedule	H/5 * * * *
Build environment	***select***	Installed maven version
Build	Root POM	testapp/pom.xml
	Goals	clean install

Click ‘Save’.

5.5.10.10 Create job build-image

Go to the Jenkins user interface. Click ‘New item’.

Property	Value
Item name	build-image
Type	Freestyle project

Click ‘OK’.

Configure job build-image Go to the Jenkins user interface. Click ‘build-image’, ‘Configure’.

	Property	Value
Discard old builds	Max # of builds to keep	5
Restrict	Label Expression	docker
Git	Repository URL	git@www.gitlab..tld:test-group/test-project.git
	Credentials	jenkins (gitlab user)
Build	***select***	Execute shell
	Command	cd docker && ./build.sh

Click ‘OK’.

5.5.10.11 Create job sonar-art

Go to the Jenkins user interface. Click ‘New item’.

Property	Value
Item name	sonar-art
Type	Maven project

Click ‘OK’.

Configure job sonar-art Go to the Jenkins user interface. Click ‘sonar-art’, ‘Configure’.

	Property	Value
Discard old builds	Max # of builds to keep	5
Git	Repository URL	git@www.gitlab..tld:test-group/test-project.git
	Credentials	jenkins (gitlab user)
Build	Root POM	testART/pom.xml
	Goals	clean install -DskipTests
Post-build Actions	***select***	SonarQube analysis with Maven

Click ‘Save’.

5.5.10.12 Create job build-art

Go to the Jenkins user interface. Click ‘New item’.

Property	Value
Item name	build-art
Type	Maven project

Click ‘OK’.

Configure job build-art Go to the Jenkins user interface. Click ‘build-art’, ‘Configure’.

	Property	Value
Discard old builds	Max # of builds to keep	5
Git	Repository URL	git@www.gitlab..tld:test-group/test-project.git
	Credentials	jenkins (gitlab user)
Additional Behaviours		Check out to specific local branch
	Branch name	master
Build environment	Release Ggoals	-Dresume=false release:prepare release:perform -D
	DryRun goals	-Dresume=false -DdryRun=true release:prepare -D
Build	Root POM	testART/pom.xml

	Property	Value
	Goals	clean install -DskipTests

Click ‘Save’.

5.5.10.13 Create job run-art

Go to the Jenkins user interface. Click ‘New item’.

Property	Value
Item name	run-art
Type	Maven project

Click ‘OK’.

Configure job run-art Go to the Jenkins user interface. Click ‘run-art’, ‘Configure’.

	Property	Value
Discard old builds	Max # of builds to keep	5
Parameters	***select***	This build is parametrized
	select	Add String Parameter
	Name	browserType
	Default Value	FIREFOX
	select	Add String Parameter
	Name	seleniumServerUrl
	Default Value	http://server.selenium..tld:4444/wd/hub
	select	Add String Parameter
	Name	applicationServerUrl
	Default Value	http://www.testapp..tld:8080
Git	Repository URL	git@www.gitlab..tld:test-group/test-project.git
	Credentials	jenkins (gitlab user)
Build	Root POM	testART/pom.xml
	Goals	-DbrowserType=$browserType - DseleniumServerUr$

Add ‘Execute shell’ Post Step. Select ‘Run only if build succeeds’. Enter command:

```
export \
  URL="http://trr.reporting.<your-project>.tld: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}" \
```

5.5.10.14 Create job load-ltcs

Go to the Jenkins user interface. Click 'New item'.

Property	Value
Item name	load-ltcs
Type	Freestyle project

Click 'OK'.

Configure job load-ltcs Go to the Jenkins user interface. Click 'load-ltcs', 'Configure'.

	Property	Value
Discard old builds	Max # of builds to keep	5
Bruild Trigers	***select***	periodically
	Schedule	H 6-20 * * 1-5

Add ‘Execute shell’ Build Step. Enter command:

```
#!/bin/bash -ex

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

if [ "$data" == 'Import completed' ]
then
  exit 0
else
  exit 1
fi
```

5.5.11 Application under development

5.5.11.1 Add application to Docker Dashboard

Go to the Docker Dashboard user interface, click ‘Apps’, ‘New App’. Enter the following app definition:

```
name: testapp
version: latest

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

Click ‘Save changes’ and start the application.

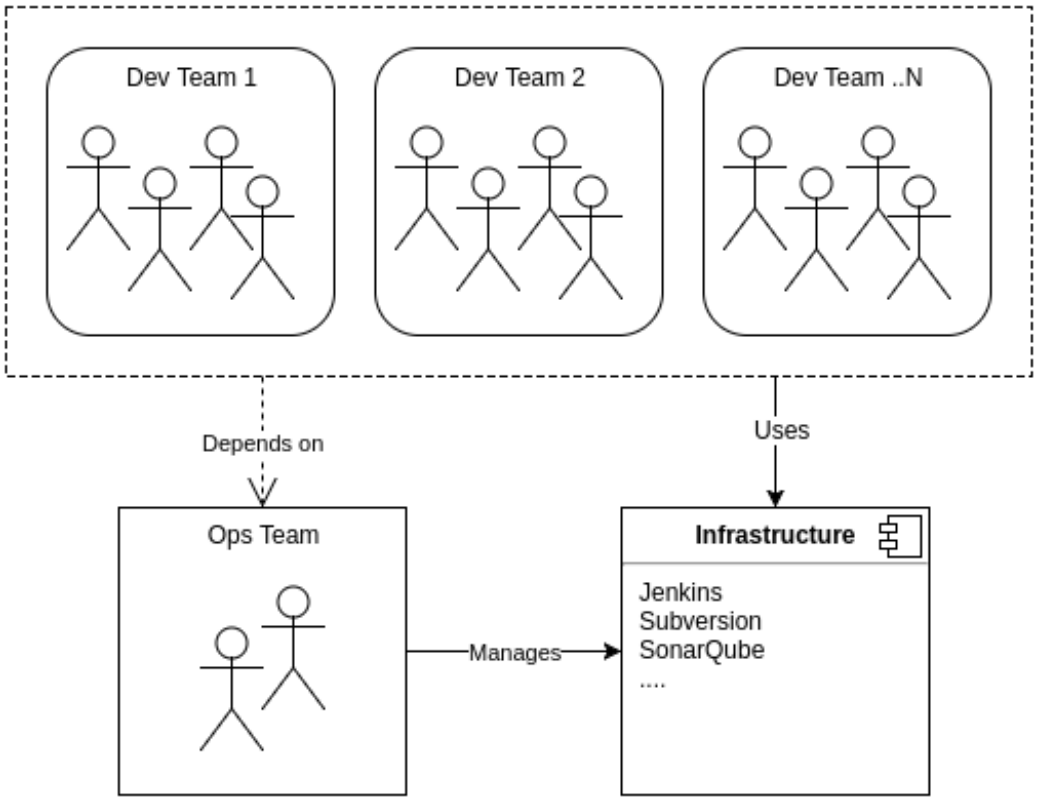

Figure 5.1: Relationship between Ops and Development teams in a shared environment

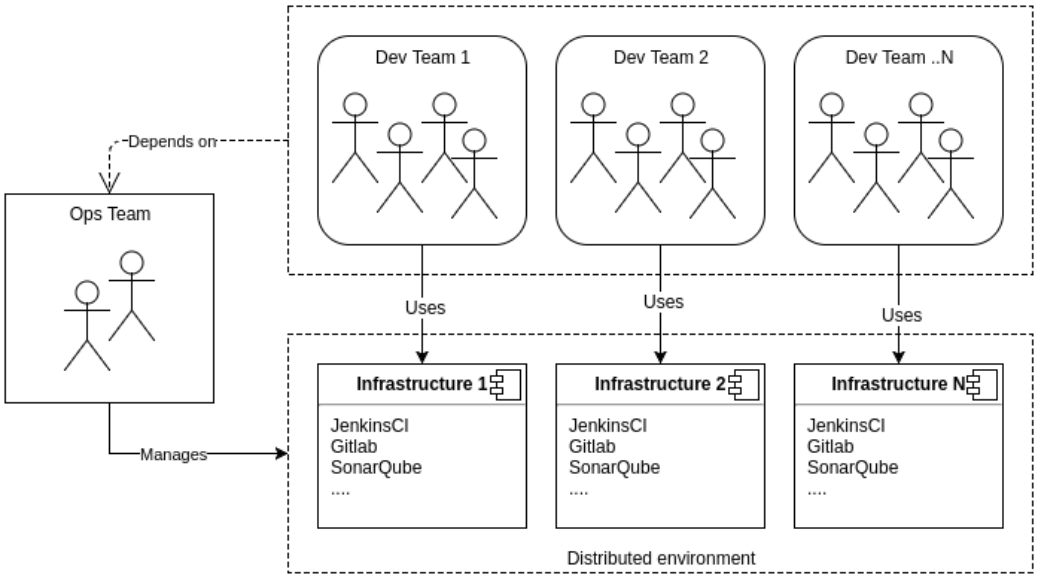

Figure 5.2: Relationship between Ops and Development teams in a distributed environment

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.

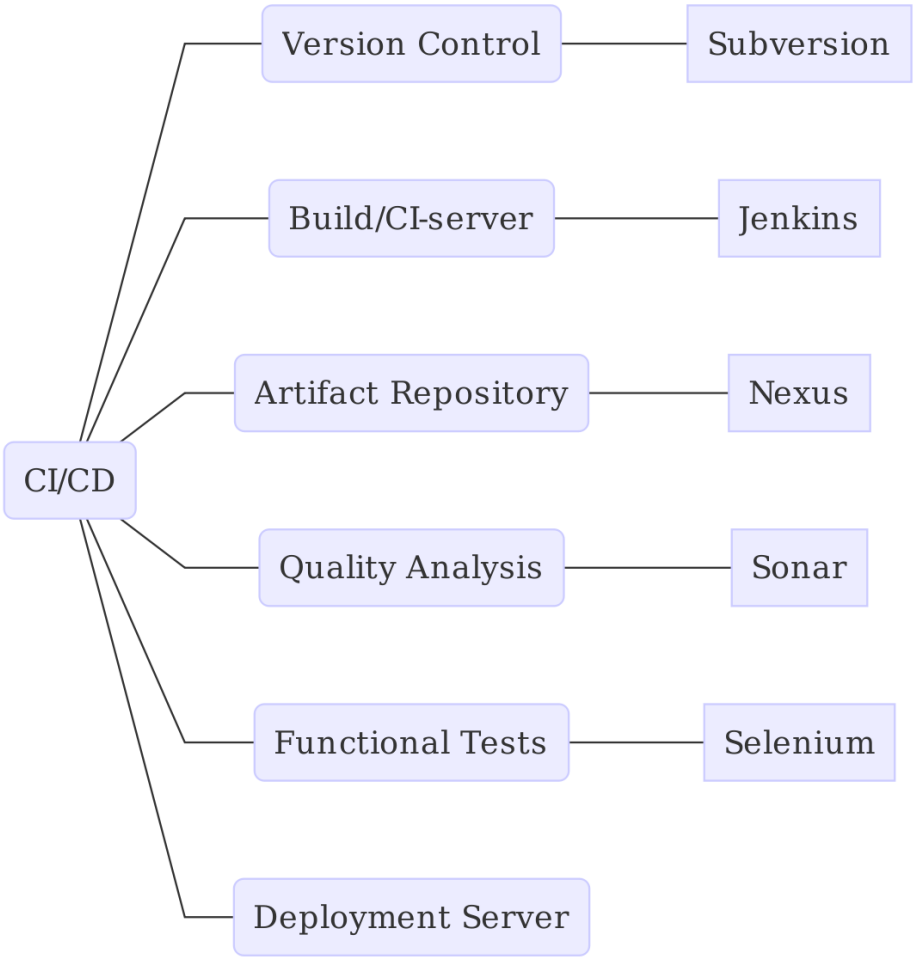

Figure 6.1: CI/CD Schematic Overview

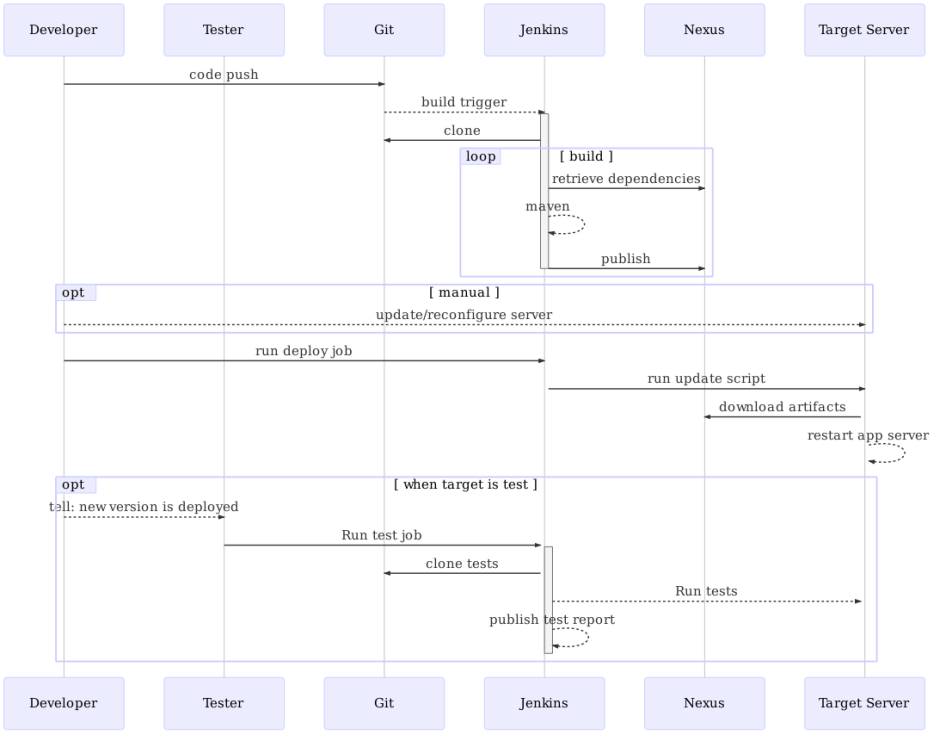

Figure 6.2: Basic CI

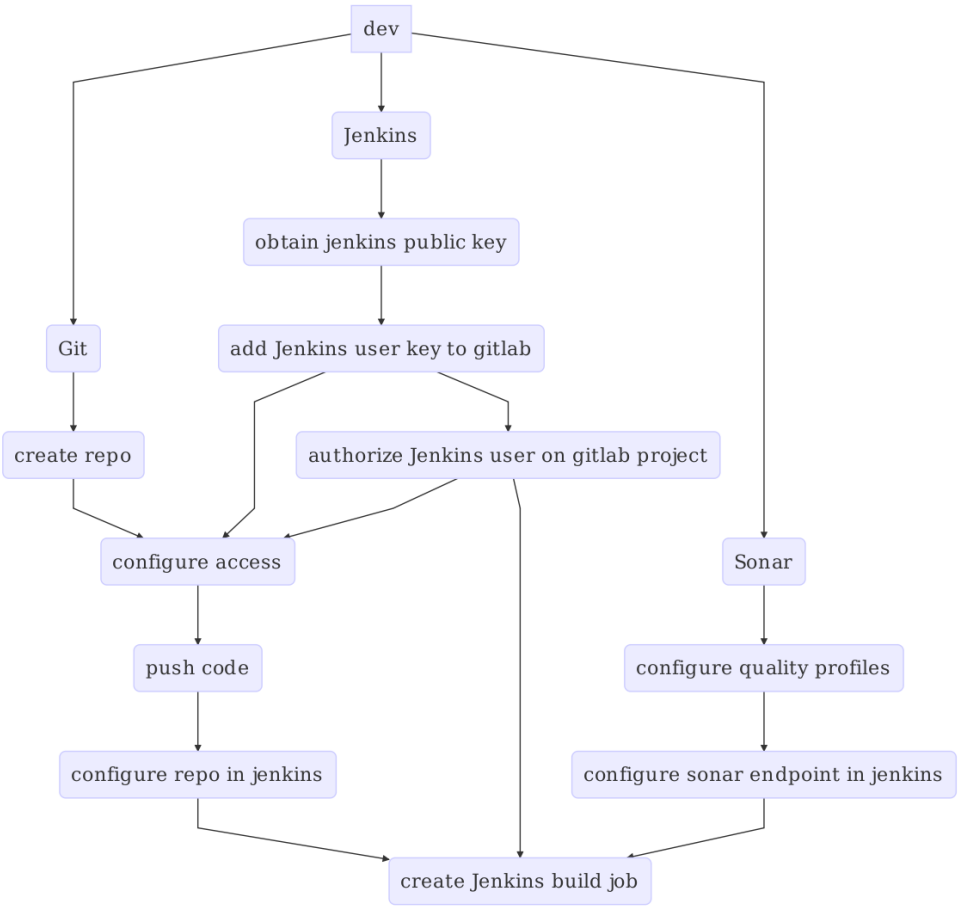

Figure 6.3: Basic CI setup

References