Table of Contents

Intros

1. Common Remarks
   1.1. Questions and Answers
   1.2. Continuous Development
   1.3. Line Breaks in Command Lines
2. Setup
   2.1. Install Git
   2.2. GitLab Account
   2.3. Java
   2.4. Maven
   2.5. IDE

Exercises

1. Git
   1.1. Inhalt von Git-Objekten
   1.2. Merge mit Konflikten
   1.3. Rebasing mit Konflikten
   1.4. Push und Pull durchführen
   1.5. Projekt anlegen
   1.6. Merge-Request
2. Build-Automatisierung mit Maven
   2.1. Projekt productmodel bauen
   2.2. Abhängiges Projekt productmodeluser bauen
   2.3. <dependencies> verstehen
   2.4. Aggregator-Projekt
   2.5. Parent-POMs
3. Continuous-Integration mit GitLab
   3.1. Ergebnisse sichern
   3.2. productmodel herunterladen
   3.3. productmodel unter Versionskontrolle stellen
   3.4. Projekt productmodel in GitLab anlegen
   3.5. Runner beobachten
   3.6. Job für Javadoc-Erzeugung
4. Qualitätssicherung
   4.1. Klassifikation von Anforderungen
   4.2. Einfache Bedingungsüberdeckung
   4.3. JUnit-Tests für isValidCombination
   4.4. Verbesserung der Code-Coverage
   4.5. Mockito
   4.6. Selenium
   4.7. JMeter
   4.8. SonarQube
5. Container Virtualization with Docker
   5.1. Docker on a GCP VM
   5.2. Ubuntu Linux in a Container
   5.3. Images
   5.4. Networking
   5.5. Mounts
   5.6. Resource Limitation

5.7. Out-of-Memory-Killer
6. Google Cloud Platform
6.1. Projects
6.2. Cloud SQL
6.3. Buckets
6.4. Service Accounts and Credentials
6.5. Virtual Machines
6.6. Scaling
6.7. SQL Failover Standby Instances and Read Replica
6.8. Load-Balancing
6.9. Domain Names
6.10. SSL Certificate
6.11. Observability
7. Kubernetes
7.1. GCP Project
7.2. Install kubectl
7.3. Kubernetes Cluster
7.4. Redis Leader Deployment
7.5. Redis Leader Service
7.6. Redis Follower Deployment
7.7. Redis Follower Service
7.8. Frontend Deployment
7.9. Frontend Service
7.10. Inspect Pods and Services
7.11. Frontend Service with Load Balancer
7.12. Test in Web Browser
7.13. Login into a Container
7.14. DNS
7.15. Manual Scaling
7.16. Rolling Updates
7.17. Rollback
7.18. Clean up
Appendix A: Appendix
A.1. Additional Exercise Resources
Labs
1. Praktikumsaufabe zu Git und Maven
1.1. Lokales Repository erzeugen
1.2. Working-Tree
1.3. Initiale Implementierung von Toolbox.java
1.4. Exkurs: Markdown
1.5. Staging
1.6. Commit
1.7. Changelog in README.md
1.8. Javadoc-Kommentar für main-Methode
1.9. Änderungen committen
1.10. Tagging
1.11. Remote-Repository auf GitLab anlegen (Teammitglied A)
1.12. Remote-Repository clonen (Teammitglied B)
1.13. Feature-Branch command-reverse (Teammitglied B)
1.14. Feature-Branch "markdown-remark" (Teammitglied A)

1.15. unproblematischer Merge-Request (Teammitglied A)
1.16. problematischer Merge-Request (Teammitglied B)
1.17. Mavenizing
2. Praktikumsaufgabe zur Qualitätssicherung
2.1. Parcer im Überblick
2.2. Parcer herunterladen
2.3. Parcer unter Versionskontrolle stellen
2.4. Parcer starten
2.5. Requests absetzen
2.6. H2-Console verwenden
2.7. Messung der Code-Abdeckung
2.8. HTML-Report
2.9. JaCoCo-Dokumentation
2.10. Mindestabdeckung
2.11. Erweiterung der Code-Abdeckung
2.12. JMeter
2.13. Sonarqube (optionale Aufgabe)
2.14. Zerlegung in Maven-Module (optionale Aufgabe)
3. Praktikumsaufgabe zur Google Cloud und Docker-Containern
3.1. Google Cloud Projekt aufsetzen
3.2. Firewall öffnen
3.3. Parcer.Customer direkt ausführen
3.4. Docker installieren
3.5. Dockerfiles
3.6. Addressvalidation-Container starten
3.7. Docker-Compose
3.8. Weiterentwicklung

Intros

The following exercises accompany the lecture "Software-Management (SM)". Participants of SM are encouraged to work on these exercises **before** the respective lecture. Please consult the schedule (located in the respective Software-Management Ilias course) to learn about which exercise corresponds with which lecture.

1. Common Remarks

Most of the tasks are tutorials. If possible, work on the tasks alone on your own machine.

Concerns the seminar room in the IWZ: Your laptop should be 100% charged before the lecture / exercise. Socket strips are connected in the rows of tables, but only to a limited extent due to the sparse electrification in the seminar room.

In case of course DevOps, we use another GitLab instance. In the following, occurrences of

- `gitlab.nt.fh-koeln.de` have to be replaced by `git.coco.study` and
- `gitlab.nt.fh-koeln.de:10022` by `git.coco.study:22`

1.1. Questions and Answers

Some exercises include test questions with an IMPORTANT admonition like this

IMPORTANT | What is the meaning of life?

The error messages like

Unresolved directive in intros/generic_intro_en.asciidoc - include::solutionfile.asciidoc[]

can be ignored as there are placeholder for answers to the test questions which will be included on the `solution` branch one after another.

TIP | If you read this text in GitLab: There is a dropdown list above that allows for switching branches. Those solutions that have already been published can be easyly be switched on and off with that.

1.2. Continuous Development

Please note that the exercises are still under development. Exercises with higher section numbers than the current exercise (according to the schedule) might be extended or rewritten without further notice. Everyone is highly welcome to improve the exercises. If something is misleading, important informations are missing or you stumble upon typos, please clone the exercises, create a branch with name `asgm-XXX` (where `XXX` is a short but speaking name of your fix), push that branch and create a merge request assigned to me (`rwoerzbe`).

1.3. Line Breaks in Command Lines

Please note that some single command lines include line breaks which have to be escaped with a `\`.

Example:

```
echo "Ich bin ein ziemlich langer Parameter, der aber in einer einzigen \
Kommandozeile verwendet wird, die daher mittels Backslash \
umgebrochen wird und das sogar über mehrere Zeilen."
```

---

2. Setup

2.1. Install Git

Install [the Git command line client](). Keep the default settings during the installation. After the installation, windows users are now supposed to have the `Git bash` in their start menu, which emulates Linux' bash. MacOS should also have something like a bash (ZSH works as well).

In your bash, execute

```
git --version
```

which should something like

```
git version 2.16.2.windows.1
```

Set your username and email address globally in Git

CAUTION | As your `user.name` you MUST use your lowercase campus ID in this course (e.g., mine is `rwoerzbe`). For `user.email`, you MUST use your smail address.

```
git config --global user.name "campusid"
git config --global user.email "firstnam.lastname@smail.th-koeln.de"
```

Check via

```
git config --list
```

if these properties have been set properly.

2.2. GitLab Account

2.2.1. Register

Open https://gitlab.nt.fh-koeln.de in your browser. Please note that you have to be connected to the VPN in order to do that.

Please register using your campus id in lowercase as your username and your smail email address.

2.2.2. Create SSH Keys

Check via

```
ls -l ~/.ssh
```

if you already got a key pair. If so, the output should include something like

```
~/.ssh/id_rsa
~/.ssh/id_rsa.pub
```

Otherwise, execute

```
ssh-keygen
```

which creates the abovementioned files. The passphrase can be left empty.

The key pair `id_rsa` (private key) and `id_rsa.pub` (public key) are now used to authenticate your local user at a SSH server. In our case, the GitLab server is such a SSH server if we use the `ssh://` scheme.

WARNING The private key `id_rsa` MUST not be made public in any case. It should never be copied or moved nor sent via email etc. Otherwise, it should be considered compromised and unsecure.

2.2.3. Paste Public Key into GitLab Settings

In GitLab, open `User Settings / SSH Keys` and paste the contents of your local `~/.ssh/id_rsa.pub`.

Using

```
cat ~/.ssh/id_rsa.pub
```

you can output these contents. Now your local user can authenticate to the GitLab server.

2.2.4. Add gitlab to `known_hosts`

Conversely, in order to let your local user trust the GitLab server SSH credentials (i.e. the SSH server must also authenticate to your local user), add its hostname like so:

```
ssh-keygen -R gitlab.nt.fh-koeln.de
ssh-keyscan -t rsa gitlab.nt.fh-koeln.de >> ~/.ssh/known_hosts
```

Now, you should be able to clone arbitrary repositories.

2.3. Java

Install the latest longterm support version (LTS) of Java JDK. Please ensure, that the executables `java` and `javac` are in the search `PATH` for executables and the `JAVA_HOME` environment variable is set to the installation directory of the JDK.

2.4. Maven

Download and install Apache Maven. Again please ensure that the executable `mvn` is in the search `PATH`.

2.5. IDE

As IDEs / source code editors, we mainy support IntellJ IDEA and Visual Studio Code. (Currently, the exercises are adapted for the former but should still work with the latter).

Students can obtain free licenses for IntelliJ here: https://www.jetbrains.com/shop/eform/students

CAUTION	Please carefully read the license agreement. Accepting these agreements and possible violations are completely in your responsibility.

Exercises

1. Git

Führen Sie den Beispiel-Ablauf aus der Vorlesung in einer Bash aus.

```
mkdir proj
cd proj
git init
echo "A" > file.txt
git add file.txt
git commit -m "Added file.txt with content A"
echo "B" > file.txt
git add file.txt
echo "C" > file.txt
git commit -m "Changed A->B in file.txt"
echo "D" > file.txt
git commit -am "Changed B->D in file.txt"
git checkout HEAD~
git checkout master
echo "E" > file.txt
git commit -am "Changed D->E in file.txt"
echo "F" > file.txt
git add file.txt
git reset -- file.txt
git checkout -- file.txt
```

Führen Sie anschließend

```
git log
```

aus.

IMPORTANT	Was können Sie dem Ergebnis entnehmen?

Bisherige Commit-Historie in umgekehrter Chronologie im aktuellen Branch mit jeweiligem Autor, Commit-Zeitpunkt und -Message

Bisherige Commit-Historie in umgekehrter Chronologie im aktuellen Branch mit jeweiligem Autor, Commit-Zeitpunkt und -Message

Führen Sie außerdem

```
git log --oneline --graph --all --decorate
```

aus.

IMPORTANT | Was bewirken die einzelnen Parameter?

1. `online`: Eine Zeile pro Commit
2. `graph`: links die Commit-Struktur als Graph
3. `all`: alle Branches
4. `decorate`:

1. `online`: Eine Zeile pro Commit
2. `graph`: links die Commit-Struktur als Graph
3. `all`: alle Branches
4. `decorate`:

Behalten Sie das `proj`-Verzeichnis so für die Übung in der nächsten Woche.

1.1. Inhalt von Git-Objekten

Geben Sie den Inhalt von `file.txt` im letzten Commit, nämlich "E", durch Verwendung von

```
git log
```

und sukzessive Verwendung von

```
git cat-file -p
```

aus.

Überzeugen Sie sich, dass der Hash-Wert für `file.txt` im Tree-Objekt korrekt ist, indem Sie ihn für das identische `file.txt` im Working-Tree (mit Inhalt `"E"`) auf der Kommandozeile berechnen.

```
git hash-object --no-filters file.txt
```

1.2. Merge mit Konflikten

Wechseln Sie wieder (hoch) in ein nicht-versioniertes Verzeichnis

Führen Sie in einer Bash

```
curl -skL https://gitlab.com/rwoerzbe/public/-/raw/main/gitmerge01.sh | bash -s -- mergeDemo
```

aus. Dies erzeugt ein lokales Repository in `mergeDemo`.

Verschaffen Sie sich zunächst einen Überblick über den aktuellen Stand mittels

```
git log --oneline --graph --all --decorate
```

Der aktuelle Branch ist also `master`.

Führen Sie einen Merge des Branches `req01` auf den Branch `master` durch wie in der Vorlesung gezeigt.

IMPORTANT | Welche Git-Kommandozeilen sind hierfür nötig?

```
git merge req01 master
git mergetool doc/README.txt
git commit -a
```


```
git merge req01 master
git mergetool doc/README.txt
git commit -a
```


1.3. Rebasing mit Konflikten

Wechseln Sie wieder (hoch) in ein nicht-versioniertes Verzeichnis

Führen Sie in einer Bash

```
curl -skL https://gitlab.com/rwoerzbe/public/-/raw/main/gitmerge01.sh  | bash -s -- rebaseDemo
```

aus. Dies erzeugt das gleiche Repository wie in der vorherigen Aufgabe, jedoch in `rebaseDemo`.

Führen Sie ein Rebasing derart aus, dass die Commits von `req01` auf den Änderungen von `master` basieren. D.h. es soll so getan werden, als hätten die Commits von `master` zeitlich vor den von `req01` stattgefunden.

IMPORTANT | Welche Git-Kommandozeilen sind hierfür nötig?

```
git rebase master req01
git mergetool doc/README.txt
git commit -a
git rebase --continue
```

```
git rebase master req01
git mergetool doc/README.txt
git commit -a
git rebase --continue
```

Führen Sie

```
git log --oneline --graph --all --decorate
```

dann einmal in `mergeDemo` und einmal in `rebaseDemo` aus.

IMPORTANT | Welche Unterschiede sehen Sie?

```
Lineare Historie in rebaseDemo
Dedizierter Merge-Commit in mergeDemo (insg. ein Commit mehr), da kein Fast-Forward-Merge möglich
```


```
Lineare Historie in rebaseDemo
Dedizierter Merge-Commit in mergeDemo (insg. ein Commit mehr), da kein Fast-Forward-Merge möglich
```


1.4. Push und Pull durchführen

Führen Sie in einer Bash

```
cd ~/sm
curl -skL https://gitlab.com/rwoerzbe/public/-/raw/main/gitremote01.sh  | bash
```

aus. Es werden dabei die drei Repositories

```
pushpull/remoteRepo/.git
pushpull/localRepo1/.git
pushpull/localRepo2/.git
```

erzeugt, die normalerweise auf verschiedene Rechner verteilt sind und nicht wie hier der Einfachheit halber in einem Verzeichnis `pushpull` liegen.

1.4.1. Pushen des req02-Branches von localRepo1

Wechseln Sie nun in `pushpull/localRepo1` und erzeugen Sie einen Branch `req02`, den Sie direkt auschecken.

IMPORTANT | Welche Kommandozeilen haben Sie hierfür eingegeben?

```
cd pushpull/localRepo1
git checkout -b req02
```

Ändern Sie `HelloWorld.java` so, dass ein Abschiedsgruß unter dem Kasten ausgegeben wird.

Sie können das in einem Editor Ihrer Wahl tun oder (etwas kryptisch) auf der Kommandozeile

```
sed -i '7i \ \ \ \ System.out.println("Auf Wiedersehen\\n");' HelloWorld.java
```

Committen Sie die Änderungen (in `pushpull/localRepo1/.git`).

IMPORTANT | Welche Kommandozeile ist hierfür nötig?

```
git commit -am "Abschiedgruß eingefügt"
```

Pushen Sie jetzt nur den Branch `req02`.

IMPORTANT | Welche Kommandozeile ist hierfür nötig?

```
git push origin req02
```

1.4.2. Pullen des req02-Branches in localRepo2 (master-Branch)

Wechseln Sie in `pushpull/localRepo2` und pullen Sie den neuen Branch `req02`, d.h. der Commit in `req02` soll in den master von `localRepo2` gemergt werden.

IMPORTANT | Welche Kommandozeile ist hierfür nötig?

```
git pull origin req02
```

1.4.3. Pushen des master-Branches von localRepo2

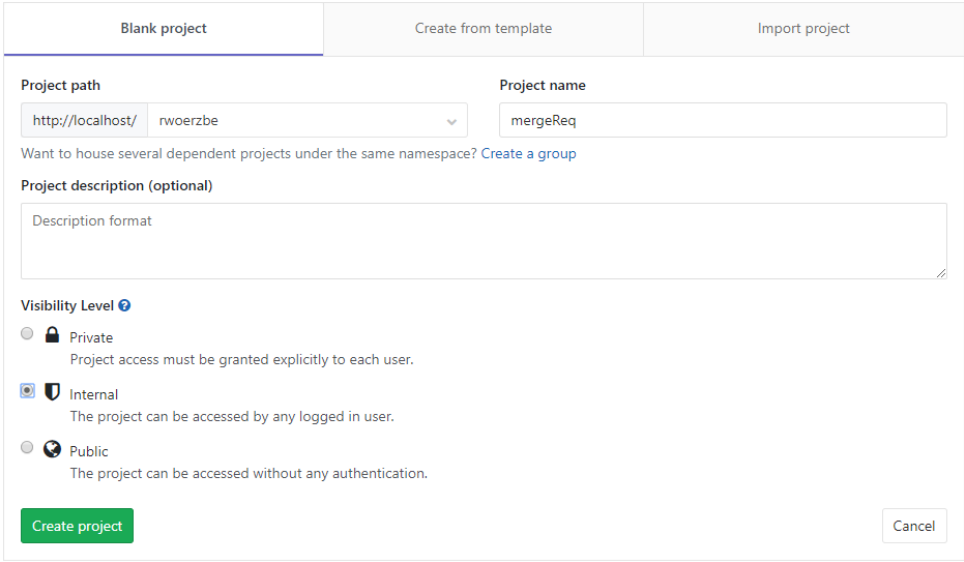
Pushen Sie anschließend das Merge-Commit-Objekt auf dem `master`.

IMPORTANT | Welche Kommandozeile ist hierfür nötig?

```
git push origin master
```

1.5. Projekt anlegen

Legen Sie in GitLab im Namespace Ihres Users (CampusID) ein Projekt namens `mergeReq` an mit Visibility-Level "Internal". Entfernen Sie den Haken in der Checkbox "Initialize repository with a README".



Kopieren Sie die URL zum erzeugten Remote-Repository in die Zwischenablage (clipboard)

1.6. Merge-Request

Führen Sie anschließend folgenden Befehl aus, wobei Sie die kopierte URL anstelle von `<remote_repo_url>` einsetzen.

```
cd ~/sm
curl -skL https://gitlab.com/rwoerzbe/public/-/raw/main/gitremote02.sh  | bash -s mergeReq <remote_repo_url>
```

Beispiel:

```
curl -skL https://gitlab.com/rwoerzbe/public/-/raw/main/gitremote02.sh  | bash -s mergeReq \
ssh://git@gitlab.nt.fh-koeln.de:10022/rwoerzbe/mergeReq.git
```

Es werden nun nur "lokale Repositories" erzeugt, d.h.

```
mergeReq/localRepo1
mergeReq/localRepo2
```

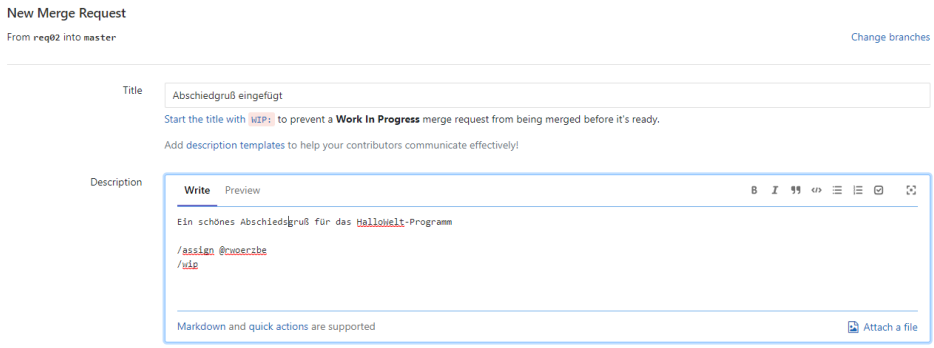
Führen Sie Aufgabe Pushen des req02-Branches von localRepo1 erneut durch.

1.6.1. Merge-Request erzeugen

Beim Pushen nach `origin` wird eine Ausgabe der Art

```
remote: To create a merge request for req02, visit:
remote: https://gitlab.nt.fh-koeln.de:10022/rwoerzbe/mergeReq/merge_requests/new?merge_request%5Bsource_branch%5D=req02
```

erzeugt. Öffnen Sie die URL im Browser.



Befüllen Sie die Description wie angezeigt, wobei Sie anstelle von `rwoerzbe` Ihren eigenen Username eintragen.

CAUTION Typischerweise ordnet man sich Merge-Request gerade nicht selbst zu, sondern einem anderen Projekt-Mitglied, damit mindestens noch ein anderes Mitglied auf die Änderungen schaut, bevor diese in den z.B. `master`-Branch gemerged werden.

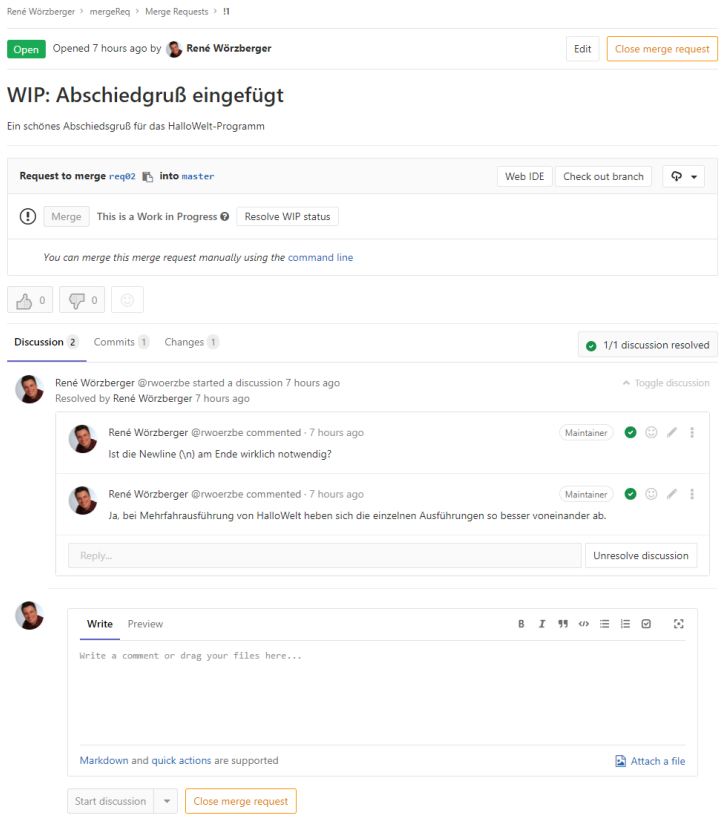
NOTE Was bewirken die letzten beiden Zeilen mit `assign` und `wip`?

```
assign = Zuweisung an rwoerzbe
wip = Markierung als Work In Progress
```

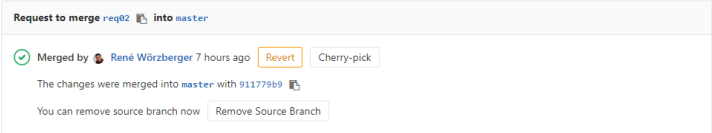
Erzeugen Sie den Merge-Request durch `Submit merge request`.

1.6.2. Merge-Request durchführen

Unter Merge-Requests sehen sie Sie nun den erzeugten Merge-Request. Diesen können Sie öffnen und bei Bedarf hierüber Diskussionen entspinnen oder einzelne Kommentare hinterlassen.

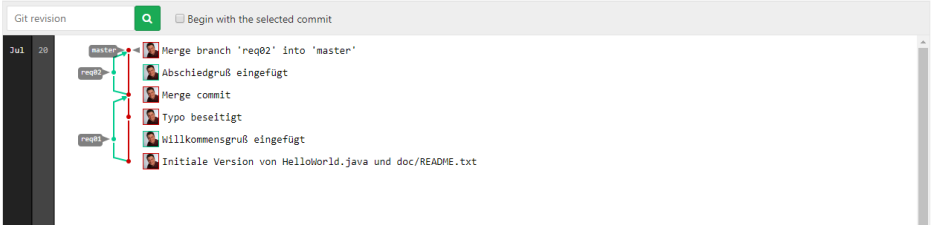


Führen Sie den Merge aber letztlich durch!



Anschließend schließen Sie den Merge-Request über `Close merge request`.

Überprüfen Sie mittels `Repository / Graph`, ob die Commit-Historie nun Ihren Erwartungen entspricht.



---

2. Build-Automatisierung mit Maven

In dieser Übung soll der Umgang mit Maven geübt werden.

2.1. Projekt productmodel bauen

Öffnen Sie eine Bash und führen Sie

```
mkdir -p ~/sm/productmodel
curl -skL https://gitlab.com/rwoerzbe/public/-/raw/main/productmodel.tar.gz  | tar xvfz - -C ~/sm/productmodel
```

aus. Das `productmodel`-Verzeichnis können Sie frei wählen.

Wechseln Sie in das `productmodel`-Verzeichnis

```
cd ~/sm/productmodel
```

und führen Sie dort

```
mvn install -Dmaven.test.failure.ignore=true
```

aus.

2.2. Abhängiges Projekt productmodeluser bauen

Öffnen Sie eine Bash und führen Sie

```
mkdir -p ~/sm/productmodeluser
curl -skL https://gitlab.com/rwoerzbe/public/-/raw/main/productmodeluser.tar.gz| tar xvfz - -C ~/sm/productmodeluser
```

aus.

Wechseln Sie in das `productmodeluser`-Verzeichnis und führen Sie

```
mvn install
```

aus. Vergewissern Sie sich anschließend, dass das betreffende Artefakt sowohl

- im `target`-Verzeichnis als auch
- im richtigen Verzeichnis in Ihrem lokalen Repository

liegt.

Führen Sie die `main`-Methode über Maven aus, indem Sie das Plugin `exec` mit dem Goal `java` verwenden:

```
mvn exec:java -Dexec.mainClass="de.hsd.shipme.productmodeluser.Main"
```

(Die Programmausgabe muss man zwischen den Ausgaben von Maven etwas suchen.)

2.3. <dependencies> verstehen

2.3.1. Projekt productmodel ändern

Öffnen sie `productmodel` in IntelliJ.

Ändern Sie die Implementierung von `productmodel` dahingehend, dass Sie einen neuen Service `ShipmentService.NOTIFY` einführen, der mit allen anderen Services kombiniert werden kann, d.h. dessen Enthaltensein in `services` beim Aufruf von

```
public boolean isValidCombination(Set<ShipmentService> services)
```

nicht ursächlich dafür sein darf, dass `false` statt `true` retourniert wird. Erweitern Sie hierzu einfach die Enumeration `de.hsd.shipme.productmodel.api.ShipmentService`.

Ändern Sie die `<version>` in `productmodel/pom.xml` auf `1.2.4` und führen Sie dann

```
mvn clean install -Dmaven.test.failure.ignore=true
```

aus.

2.3.2. Implementierung von productmodeluser nachziehen

Erweitern Sie in der `main`-Methode `de.hsd.shipment.productmodeluser.Main.main(String[])` das `testSet` um den neuen Service `ShipmentService.NOTIFY`.

Bauen Sie `productmodeluser` wiederum mit

```
mvn clean install
```

IMPORTANT Warum erhalten Sie nun einen Kompliationsfehler?

In `productmodeluser/pom.xml` ist die `dependency` noch auf `1.2.3` gerichtet.

2.3.3. Dependency nachziehen

Ändern Sie in `productmodeluser/pom.xml` die `<dependency>` zu `productmodel` so, dass die Kompliation dann funktioniert.

2.3.4. Dependencies darstellen

Führen Sie in `productmodel` und in `productmodeluser` jeweils

```
mvn dependency:tree
```

aus.

2.4. Aggregator-Projekt

Erzeugen Sie ein Verzeichnis `~/sm/productmodelaggregator`

Verschieben Sie die Verzeichnisse `productmodel` und `productmodeluser` dort hinein.

Erzeugen Sie in `productmodelaggregator` eine `pom.xml` mit folgendem Inhalt:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

   <modelVersion>4.0.0</modelVersion>
   <groupId>de.hsd.swe.shipme</groupId>
   <artifactId>productmodelaggregator</artifactId>
   <version>1.2.4</version>
   <packaging>pom</packaging>

   <modules>
      <module>productmodeluser</module>
      <module>productmodel</module>
   </modules>
</project>
```

Führen Sie in `productmodelaggregator` nun

```
mvn clean install -Dmaven.test.failure.ignore=true
```

aus.

IMPORTANT Wie erklären Sie sich die Build-Reihenfolge, wie sie schließlich in `Reactor Summary` angegeben wird und die nicht der Reihenfolge in `<modules>` folgt?

Die Build-Reihenfolge folgt dem azyklischen Graph der Abhängigkeiten

2.5. Parent-POMs

2.5.1. productmodeluser mit Eingabe

Ändern Sie in `de.hsd.shipment.productmodeluser.Main.main(String[])` die Initialisierung des `testSet` dahingehen, dass in `args` Services übergeben werden können. Das geschieht z.B. folgendermaßen:

```
Set<ShipmentService> testSet = Stream.of(args[0].split(",")).map(s ->
ShipmentService.valueOf(s)).collect(Collectors.toCollection(HashSet::new));
```

Bauen Sie `productmodeluser` erneut mit

```
mvn clean install -Dmaven.test.failure.ignore=true
```

und führen Sie die `main`-Methode aus (eine Zeile):

```
mvn exec:java -Dexec.mainClass="de.hsd.shipme.productmodeluser.Main" \
-Dexec.args="COD,NEIGHBOUR"
```

2.5.2. Tests für productmodeluser

Legen Sie folgendes Verzeichnis unterhalb von `productmodeluser` an:

```
src/test/java/de/hsd/shipme/productmodeluser
```

HINT: In einer Bash geht das mit einem Befehl

```
mkdir -p src/test/java/de/hsd/shipme/productmodeluser
```

Legen Sie dort die Klasse `MainTest.java` an:

```
package de.hsd.shipme.productmodeluser;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.io.ByteArrayOutputStream;
import java.io.PrintStream;

public class MainTest {

    private final ByteArrayOutputStream outContent = new ByteArrayOutputStream();
    private final PrintStream originalOut = System.out;

    @BeforeEach
    public void setUpStreams() {
        System.setOut(new PrintStream(outContent));
    }

    @AfterEach
        public void restoreStreams() {
        System.setOut(originalOut);
    }

    @Test
    void isValidCombination_COD_NEIGHBOUR_false() {
        Main.main(new String[] {"COD,NEIGHBOUR"});
        assert(outContent.toString().trim().endsWith("false"));
    }
}
```

Führen Sie

```
mvn clean install -Dmaven.test.failure.ignore=true
```

aus.

IMPORTANT | Wie erklären Sie sich die Kompilationsfehler?

> Dependency zu JUnit fehlt.

Informieren Sie sich auf https://junit.org/junit5/docs/current/user-guide/#running-tests-build-maven , wie Sie den Fehler beheben können.

2.5.3. productmodelparent erzeugen

Erzeugen Sie ein Verzeichnis

```
~/sm/productmodelaggregator/productmodelparent
```

und legen Sie dort ein `pom.xml` mit folgendem Inhalt an:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>de.hsd.swe.shipme</groupId>
    <artifactId>productmodelparent</artifactId>
    <version>1.2.4</version>
    <packaging>pom</packaging>

    <build>
    ...
    </build>

    <reporting>
    ...
    </reporting>

    <dependencyManagement>
        <dependencies>
        ...
        </dependencies>
    </dependencyManagement>

</project>
```

2.5.4. productmodel und productmodeluser anpassen

Bitte **verschieben** Sie aus der `productmodel/pom.xml` die Abschnitte `<build>`, `<reporting>` und **kopieren** Sie `<dependencies>` an die entsprechenden Stellen in der `productmodelparent/pom.xml`.

Fügen Sie in `productmodel/pom.xml` und `productmodeluser/pom.xml` einen Verweis auf die Parent-POM ein:

```
<parent>
    <groupId>de.hsd.swe.shipme</groupId>
    <artifactId>productmodelparent</artifactId>
    <version>1.2.4</version>
    <relativePath>../productmodelparent/pom.xml</relativePath>
</parent>
```

Entfernen Sie in diesen beiden `pom.xml` die `<version>` und `<scope>` aus den `<dependency>`.

Fügen Sie in der `productmodelaggregator/pom.xml` ein Modul für `productmodelparent` ein. Bauen Sie `productmodelaggregator` nun mit

```
mvn clean install -Dmaven.test.failure.ignore=true
```

2.5.5. Effektive-POM

Wechseln Sie in `productmodel` und führen Sie

```
mvn help:effective-pom
```

aus.

IMPORTANT | Wie deuten Sie die Ausgabe?

> Es wird die "effektive" POM dargestellt, also die `pom.xml`, die auch von Parent-POM geerbten, ggf. redefinierten Eigenschaften beinhaltet.

---

3. Continuous-Integration mit GitLab

In dieser Übung soll der Umgang mit Arbeitsabläufe des Continuous-Integration geübt werden.

Im Folgenden wird davon ausgegangen, dass Sie im Verzeichnis

```
~/sm/productmodel
```

arbeiten. Andernfalls müssen Sie Vorkommnisse dieses Pfades im Folgenden anpassen.

3.1. Ergebnisse sichern

Wenn Sie unter

```
~/sm/productmodel
```

noch Ergebnisse der letzten Übung haben, sichern Sie diese und löschen das Verzeichnis anschließend.

3.2. productmodel herunterladen

Laden Sie `productmodel` erneut (in weiterentwickelter Version) herunter. Führen Sie in einer Bash Folgendes aus:

```
mkdir -p ~/sm/productmodel
curl -skL https://gitlab.com/rwoerzbe/public/-/raw/main/productmodel_cicd.tar.gz | tar xvfz - -C ~/sm/productmodel
```

3.3. productmodel unter Versionskontrolle stellen

Sorgen Sie dafür, dass das `productmodel`-Verzeichnis (nicht `productmodel/productmodel`) unter (zunächst lokale) Git-Versionskontrolle gestellt wird.

IMPORTANT | Mit welchen (Git-)Kommandozeilen bewerkstelligen Sie das?

```
cd ~/sm/productmodel
git init .
git add .
git commit -m "Aufgabenstellung"
```

3.4. Projekt productmodel in GitLab anlegen

Loggen Sie sich in https://gitlab.nt.fh-koeln.de ein und legen Sie ein Projekt namens `productmodel` an.

Kopieren Sie die URL des erzeugten Remote-Repositories, die folgende Form haben sollte

```
ssh://git@gitlab.nt.fh-koeln.de:10022/campusid/productmodel.git
```

Setzen Sie die URL für den Shortname `origin` auf den Ihres Repositories.

```
git remote add origin ssh://git@gitlab.nt.fh-koeln.de:10022/campusid/productmodel.git
```

wobei Sie `campusid` natürlich durch Ihren eigentlichen Username (CampusID) ersetzen müssen.

Pushen Sie den `master`-Branch nun.

3.5. Runner beobachten

Durch den Push müsste nun eine Pipeline angestoßen worden sein, die aus zwei Jobs `compile` und `jacoco` besteht. Sie können

die Konsolenausgaben durch Navigation zum jeweiligen Job verfolgen.

Eventuell vergeht einige Zeit, bis Ihrem Job ein verfügbarer Runner zugeteilt wird.

Vergewissern Sie sich, dass nach Beendigung des Jobs die erzeugten Artefakte verblieben sind. Laden Sie sie herunter und öffnen Sie die `/productmodel/target/site/jacoco/index.html`.

IMPORTANT Welche Code-Überdeckung wurde in `de.hsd.swe.shipme.productmodel.impl.PMManager.isValidCombination` gemessen?

73%

3.6. Job für Javadoc-Erzeugung

Erweitern Sie die `productmodel/.gitlab-ci.yml` und einen weitere Job namens `javadoc`. Dieser soll

- Der Stage `build` zugeordnet werden
- Das Script `mvn javadoc:javadoc` ausführen
- Artefakte jeweils unterhalb von `target/site/apidocs` exportieren.

IMPORTANT Welche Zeilen müssen Sie ergänzen?

```
javadoc:
    stage: build
    script: mvn javadoc:javadoc
    artifacts:
        paths:
        - productmodel/target/site/apidocs
        - productmodeluser/target/site/apidocs
```

Committen und pushen Sie die Änderung.

IMPORTANT Weshalb schlägt die Ausführung fehl?

Beim Ausführen des Goals `productmodeluser` fehlt dann eine Depenency im lokalen Maven-Repository, da dieses parallel vom Job `compile` erst befüllt wird

```
{empty}[ERROR] Failed to execute goal on project productmodeluser: Could not resolve dependencies for project
de.hsd.swe.shipme:productmodeluser:jar:1.2.3: Could not find artifact de.hsd.swe.shipme:productmodel:jar:1.2.4 in
central (https://repo.maven.apache.org/maven2) -> [Help 1]
```

IMPORTANT Wie können Sie das Problem lösen?

`compile`-Job bis zu Phase `install` laufen lassen **UND** `javadoc`-Job in nachgelagerter Phase `doc` laufen lassen

4. Qualitätssicherung

In dieser Übung soll die Automatisierung von Unit-Tests mit JUnit geübt werden. Die Güte einer vorhandenen Testautomatisierung soll mit dem Code-Coverage-Tool JaCoCo ermittelt werden können.

Wenn Sie die Ergebnisse der letzten Übung nicht mehr vorliegen haben, können Sie sich die Ausgangsstellung mittels

```
rm -rf ~/sm/productmodel
mkdir -p ~/sm/productmodel
curl -skL https://gitlab.com/rwoerzbe/public/-/raw/main/productmodel_cicd.tar.gz| tar xvfz - -C ~/sm/productmodel
```

erneut herstellen.

4.1. Klassifikation von Anforderungen

"Parcel Services" (kurz: Parcer) ist ein System zur Erfassung von Paket-Versandaufträgen. Es unterstützt die Eingabe und Validierung von Sendungsdaten (Absender, Empfänger, Sendungsgewicht und evtl. Services wie Wunschort-Zustellung) und erzeugt daraus Versandetiketten.

IMPORTANT Klassifizieren Sie folgende Anforderungen, indem Sie die rechte Spalte folgender Tabelle befüllen

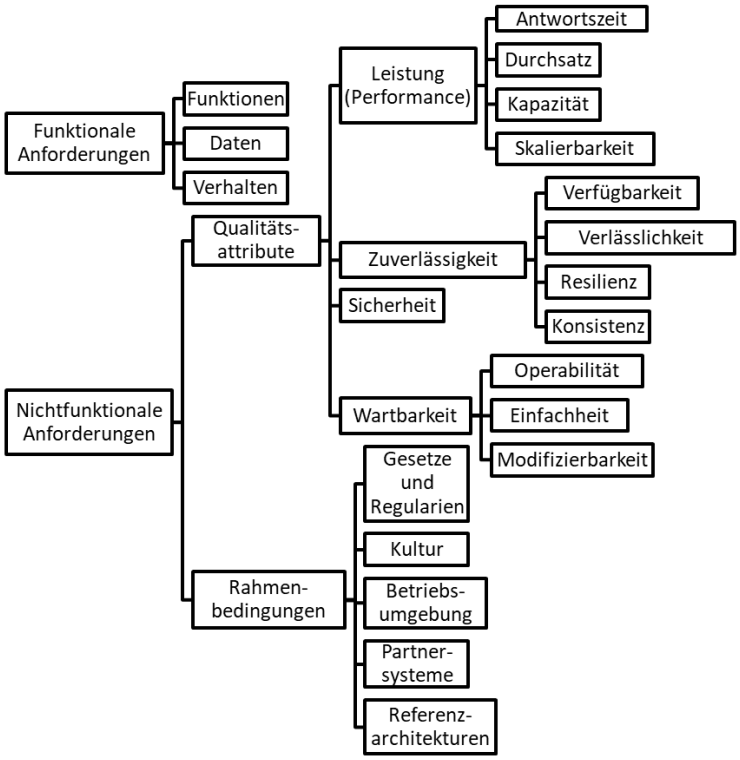
Anforderung	**Klasse**
Parcer muss eine Maske zur Erfassung einzelner Sendungen zur Verfügung stellen.	
Ein Sendungsauftrag besteht aus Absender- und Empfängeradresse sowie gewählten Services inkl. eventueller Serviceattribute	
Nach Übergabe von Sendungsauftragsdaten soll innerhalb von 2 Sekunden ein Label bereitgestellt werden.	
Parcer muss den Versuch, einen abgeschlossenen Sendungsauftrag zu löschen mit einer Fehlermeldung quittieren.	
Parcer soll bis zu 100 Sendungsaufträge / Sekunde ohne Einfluss auf die Antwortszeit verarbeiten können.	
Das Sendungsauftragsarchiv soll bis zu 1 Milliarde Sendungen speichern können.	
Die Änderung von Bankverbindungen erfordert einen 2-Factor-Login.	
Sendungsaufträge sollen aus E-Bay importiert werden können.	
Änderungen an Services sollen konfigurativ, ohne eigenes Release und ohne temporäre Downtime möglich sein.	
Parcer muss zu 99,95% verfügbar sein.	
Parcer muss den Architekturvorgaben XY genügen.	
Administrative Funktionen dürfen in Parcer nur nach einer 2-Factor-Authentifizierung durchgeführt werden.	

Anforderung	Klasse
Parcer muss von deutsch- oder englischsprachigen Benutzer bedient werden können.	
Parcer muss die Anforderungen an die Informationssicherheit gemäß ISO 27000 erfüllen.	
Parcer muss in der Standard-Betriebsumgebung X betrieben werden können (s. Betriebliche Anforderung)	

Anforderung	Klasse
Parcer muss eine Maske zur Erfassung einzelner Sendungen zur Verfügung stellen.	Funktionen
Ein Sendungsauftrag besteht aus Absender- und Empfängeradresse sowie gewählten Services inkl. eventueller Serviceattribute	Daten
Nach Übergabe von Sendungsauftragsdaten soll innerhalb von 2 Sekunden ein Label bereitgestellt werden.	Antwortzeit
Parcer muss den Versuch, einen abgeschlossenen Sendungsauftrag zu löschen mit einer Fehlermeldung quittieren.	Verhalten
Parcer soll bis zu 100 Sendungsaufträge / Sekunde ohne Einfluss auf die Antwortszeit verarbeiten können.	Durchsatz
Das Sendungsauftragsarchiv soll bis zu 1 Milliarde Sendungen speichern können.	Kapazität
Die Änderung von Bankverbindungen erfordert einen 2-Factor-Login.	Sicherheit
Sendungsaufträge sollen aus E-Bay importiert werden können.	Systeme im Kontext
Änderungen an Services sollen konfigurativ, ohne eigenes Release und ohne temporäre Downtime möglich sein.	Modifizierbarkeit
Parcer muss zu 99,95% verfügbar sein.	Verfügbarkeit
Parcer muss den Architekturvorgaben XY genügen.	Referenzarchitekturen
Administrative Funktionen dürfen in Parcer nur nach einer 2-Factor-Authentifizierung durchgeführt werden.	Sicherheit
Parcer muss von deutsch- oder englischsprachigen Benutzer bedient werden können.	Kulturell (sprachlich)
Parcer muss die Anforderungen an die Informationssicherheit gemäß ISO 27000 erfüllen.	Sicherheit / Regulatorisch/Rechtlich
Parcer muss in der Standard-Betriebsumgebung X betrieben werden können (s. Betriebliche Anforderung)	Betriebssystemumgebung

Orientieren Sie sich an den Klassen aus [Anforderungenklassen].

Anforderungsklassen



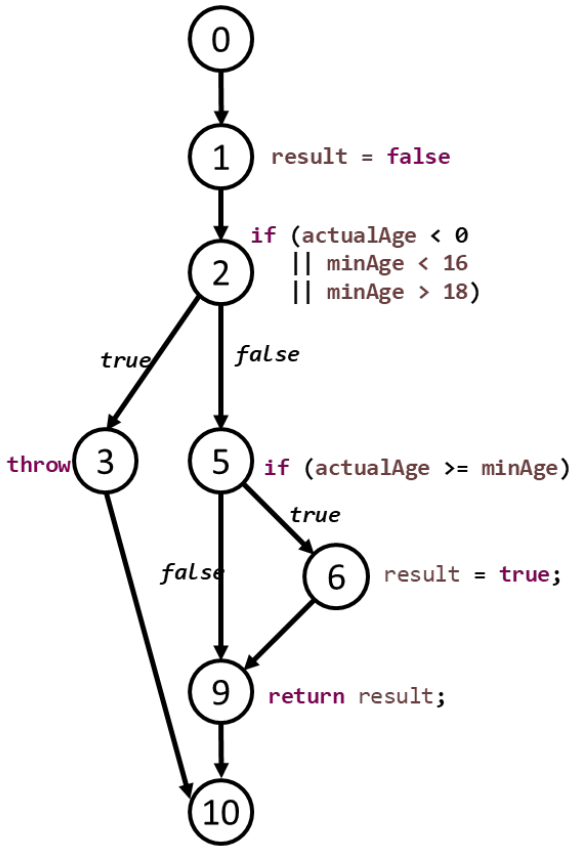
4.2. Einfache Bedingungsüberdeckung

Welche minimale Anzahl von Aufrufen führt zu einer einfachen Bedingungsüberdeckung der Funktion `checkAge`?

checkAge Quelltext

```
 /**
  * Checks, if an actual age matches a minimal age
  * @param actualAge a positive actual age <= 100
  * @param minAge minimal age in the interval [16,18]
  * @return true iff <code>acutalAge >= minAge</code>
  */
00  public boolean checkAge(int actualAge, int minAge) {
01    boolean result = false;
02    if (actualAge < 0 || minAge < 16 || minAge > 18) {
03        throw new IllegalArgumentException("Parameters not within bounds.");
04    } else {
05        if (actualAge >= minAge) {
06            result = true;
07        }
08    }
09    return result;
10  }
```

checkAge-Kontrollflussgraph



IMPORTANT

Vervollständigen Sie hierfür folgende Tabelle, d.h. die Lücken in den Spalten- und Zeilenüberschriften sowie die einzelnen Zellen, die die Werte `true`, `false` und `nicht eval.` haben können.

	actualAge < 0			
checkAge(20,18)	false			
checkAge(5,19)				
checkAge(5,15)				
checkAge(				
checkAge(				

	actualAge < 0	**minAge < 16**	**minAge > 18**	**actualAge >= minAge**
checkAge(20,18)	false	false	false	true
checkAge(5,19)	false	false	true	nicht eval.
checkAge(5,15)	false	true	nicht eval.	nicht eval.
checkAge(-3,99)	true	nicht eval.	nicht eval.	nicht eval.
checkAge(10,17)	false	false	false	false

4.3. JUnit-Tests für isValidCombination

Öffnen Sie das Submodul `productmodel` in IntelliJ oder Visual Studio Code.

TIP **IntelliJ** Wenn Sie sich in der Bash im Verzeichnis `~/sm/productmodel/productmodel` befinden, können Sie IntelliJ mit `idea64 .` so starten, dass das Submodul direkt geöffnet wird. Voraussetzung ist allerdings, dass das `bin`-Verzeichnis der IntelliJ-Installation im Suchpfad (Umgebungsvariable `PATH`) liegt. IntelliJ fragt bei der Installation, ob es den Suchpfad entsprechend anpassen soll.

TIP **Visual Studio Code** Wenn Sie sich in der Bash im Verzeichnis `~/sm/productmodel/productmodel` befinden, können Sie IntelliJ mit `code .` so starten, dass das Submodul direkt geöffnet wird. Voraussetzung ist allerdings, dass das `bin/code`-Verzeichnis der Visual-Studio-Code-Installation im Suchpfad (Umgebungsvariable `PATH`) liegt. Visual Studio Code passt diesen Suchpfad bei der Installation normalerweise selbst an.

In der letzten Übung haben Sie das Projekt `productmodel` geklont. Führen Sie alle Tests der Test-Klasse

```
de.hsd.swe.shipme.productmodel.impl.PMManagerImplTest
```

aus. Neben den erfolgreichen Tests gibt es auch `isValidCombination_singleService_true` im Zustand `fail` und `isValidCombination_noService_true` im Zustand `error`.

TIP In **IntelliJ** müssen Sie im Project-Explorer auf die Klasse rechtsklicken und im Kontextmenü `Run 'PMManagerImplTest'` auswählen. Alternativ können Sie die Play-Buttons auf der linken Seite im Code-Editor anklicken, sobald Sie die o. a. `PMManagerImplTest` geöffnet haben. Sollten diese nicht erscheinen oder Kompilationsfehler moniert werden, sollten Sie das Projekt neu bauen (`Strg+F9`).

TIP In **Visual Studio Code** müssen Sie zunächst das Extension Pack für Java und den Test Runner for Java installieren. Erkennt Visual Studio Code ein Java-Projekt, so wird in der Activity Bar ein weiterer Menüpunkt `Testing` dargestellt, über den Test-Methoden einzeln oder alle einer Klasse, eines Packages oder Projekts ausgeführt werden können.

Ändern Sie die Implementierung des Testsubjekts `PMManagerImpl` derart, dass beide Tests erfolgreich sind.

IMPORTANT Welche Änderungen müssen Sie in welcher (einzigen) Zeile durchführen?

```
if (services == null || services.size() <= 1)
```

4.4. Verbesserung der Code-Coverage

Führen Sie die Tests in

```
de.hsd.swe.shipme.productmodel.impl.PMManagerImplTest
```

mit Code-Coverage aus.

In **IntelliJ**: Öffnen Sie nur das Maven-Modul `productmodel/productmodel`. (IntelliJ hat Schwierigkeiten,

TIP

JUnit-Tests als solche zu erkennen, wenn das Aggregator-Modul `productmodel` geöffnet wird.) Führen Sie in IntelliJ `PMManagerImplTest` mindestens einmal aus. Öffnen Sie `Run / Edit Configurations` zunächst `JUnit / PMManagerImplTest` und stellen Sie (rechts) unter `Code Coverage` dann `Choose coverage runner: JaCoCo` ein. Schließen Sie den Dialog und rechtsklicken Sie im Project-Explorer auf `PMManagerImplTest` und wählen dann `Run 'PMManagerImplTest' with Coverage` aus.

TIP

In **Visual Studio Code**: Installieren Sie das Plugin Coverage Gutters. Diese zeigt die Code-Coverage an, sobald das in der `productmodelparent/pom.xml` konfigurierte JaCoCo-Maven-Plugin via Maven mittels `mvn clean install -Dmaven.test.failure.ignore=true` einmal ausgeführt wurde und daher eine Datei `target/jacoco.exec` existiert, die gemessene Code-Abdeckung beinhaltet. Sie müssen nach dem Maven-Lauf das Coverage-Gutters-Plugin noch aktivieren, indem Sie unten links in der (meistens blauen) Side-Bar von Visual Studio Code auf `Watch` klicken.

IMPORTANT

Erweitern Sie anschließend `PMManagerImplTest`, dass Sie eine 100%ige einfache Bedingungsüberdeckung erhalten.

```
import java.util.Arrays;

...

@Test
void isValidCombination_twoCompatibleServices() {
    ShipmentService[] servArr = {ShipmentService.COD, ShipmentService.NOTIFY};
    Set<ShipmentService> servSet = new HashSet<>(Arrays.asList(servArr));
    assertTrue(new PMManagerImpl().isValidCombination(servSet));
}

@Test
void isValidCombination_twoIncompatibleServices() {
    ShipmentService[] servArr = {ShipmentService.COD, ShipmentService.NEIGHBOUR};
    Set<ShipmentService> servSet = new HashSet<>(Arrays.asList(servArr));
    assertFalse(new PMManagerImpl().isValidCombination(servSet));
}

@Test
void isValidCombination_twoIncompatibleServices_2() {
    ShipmentService[] servArr = {ShipmentService.COD, ShipmentService.LOCATION};
    Set<ShipmentService> servSet = new HashSet<>(Arrays.asList(servArr));
    assertFalse(new PMManagerImpl().isValidCombination(servSet));
}
```

Bauen Sie das Projekt anschließend mit Maven

```
mvn clean install
```

Prüfen Sie, ob das mitausgeführte JaCoCo-Plugin ebenfalls eine 100%ige einfache Bedingungeüberdeckung misst. Sie finden den erzeuten Report in `target/site/jacoco/index.html`.

4.5. Mockito

In den folgenden Übungen beschäftigen wir uns mit der Web-Applikation TodoList. Diese wurde in Java auf Basis des Spring Frameworks implementiert. Der Verständlichkeit des Code wurde hierbei besonderer Vorrang eingeräumt. Insbesondere wurde auf ein ansprechendes Styling des Web-Applikation verzichtet.

Über TodoList können Beschreibungen und Prioritäten von Todos tabellarisch angezeigt und hinzugefügt werden. Oberhalb der Todo-Liste wird ein Chuck-Norris-Zitat dargestellt, das von einer externen Web-API bezogen wird. Insbesondere diese Web-API soll nun mittels Mockito gemockt werden.

4.5.1. TodoList herunterladen

Führen Sie Folgendes in einer Bash aus

```
mkdir -p ~/sm/todolist
curl -skL https://gitlab.com/rwoerzbe/public/-/raw/main/todolist-no-solutions.tar.gz | tar --strip-components=1 -C
~/sm/todolist -xz todolist-no-solutions
```

Führen Sie in der Kommandozeile

```
cd ~/sm/todolist
mvn spring-boot:run
```

aus und vergewissern Sie sich, dass Sie `http://localhost:8080/todolist` im Browser öffnen können.

4.5.2. Mockito-Test starten

Öffnen Sie das Verzeichnis `todolist` in IntelliJ oder Visual Studio Code.

Öffnen Sie die Datei `src\test\java\de\thk\rwoerzbe\todolist\TodoListControllerMockitoTest.java` und starten Sie die Tests.

IMPORTANT Die Test-Methode `testGetTodosContainsQuote()` ist noch unvollständig. Vervollständigen Sie sie so, dass getestet wird, ob eine Dummy-Quote wirklich im erzeugten HTML zu finden ist. Orientieren Sie sich an den `TODO`-Kommentaren.

```
@Test
public void testGetTodosContainsQuote() throws Exception {
    // TODO 1 fixture setup: prepare RandomServiceMock s. t. it always returns "Dummy Quote"
    final String DUMMY_QUOTE = "Dummy Quote";
    Mockito.when(randomQuoteService.getRandomQuote()).thenReturn(DUMMY_QUOTE);

    // 2 exercise subject under test (SUT)
    // We do not call TodoListController.get(Model) directly but simulate
    // an incoming HTTP GET request to /todolist
    ResultActions resultActions = mockMvc.perform(get("/todolist"));

    // 3 result verification
    // 3.1 Is HTTP status code equal to 200 (OK)?
    resultActions.andExpect(status().isOk());
    // TODO 3.2.1 Is the text under html/body/span really "Dummy Quote"?
    // Hint: The xpath expression is "/html/body/span/text()"
    resultActions.andExpect(xpath("/html/body/span/text()").string(DUMMY_QUOTE));
    // TODO 3.2.2 Check if the RandomQuoteService has actually been called once
    Mockito.verify(randomQuoteService, times(1)).getRandomQuote();
}
```

4.6. Selenium

JUnit ist im Wesentlichen eine Bibliothek, über die JUnit-Testmethoden ausgeführt werden, die die Testsubjekte "in-process" aufrufen. Der Nachteil hierbei ist, dass ein Browser nicht simuliert wird. Es kann nur das geprüft werden, was server-seitig erzeugt wird, also beispielsweise die Struktur von HTML-Dokumenten.

JavaScripts, die im Browser ausgeführt werden und die Darstellung einer Web-Seite ebenfalls beeinflussen können, können so nicht getestet werden.

Selenium ist eine Bibliothek, über die ein richtiger Browser programmatisch gesteuert und (nach Ausführung von JavaScripts) abgefragt werden kann. Dies kann insbesondere in JUnit-Testmethoden geschehen.

4.6.1. WebDriver installieren

Selenium benötigt zu Steuerung eines Browsers einen sogenannten WebDriver, der spezifisch für das Betriebssystem und den jeweiligen Browser ist. Im Folgenden wird davon ausgegangen, dass Sie einen Chrome installiert haben. Im Root-Verzeichnis von `todolist` befinden bereits Treiber `chromedriver.exe` für Chrome unter Windows. Falls Sie Linux oder MacOS benutzen, finden Sie passende WebDriver unter https://sites.google.com/chromium.org/driver/.

4.6.2. Selenium-Test starten

Öffnen Sie das Verzeichnis `todolist` in IntelliJ. Vergewissern Sie sich, dass die TodoList-Applikation aktuell läuft. Öffnen Sie die Datei

```
src/test/java/de/thk/rwoerzbe/todolist/TodoListControllerSeleniumTest.java
```

Passen Sie ggf. die Konfiguration in der Methode `configureSelenium()` an. Führen Sie dann die Testfälle aus.

Sie sollten sehen, dass sich zwischenzeitlich ein Browser-Fenster öffnet und schließt.

IMPORTANT

Implementieren Sie die Test-Methode `testTodoNumbers()`. Diese prüft, ob die Summierung der Todos richtig dargestellt wird. Diese wird im Browser über Javascript ausgeführt, wäre also ohne Selenium nicht prüfbar.

```
@Test
public void testTodoNumbers() {
    // TODO 2 Exercise SUT: Just open the web page again
    driver.get(TODOLIST_URL);

    // TODO 3 Result Verification: Check if the todo count is greater than 0
    // Tip: You can find the element with the respective text (String) under its id
    // "todonumbers"
    // You can convert a String to an Integer via Integer.parseInt(String)
    WebElement todoNumbers = new WebDriverWait(driver,
Duration.ofSeconds(5)).until(ExpectedConditions.elementToBeClickable(By.id("todoNumbers")));
    assertTrue(Integer.parseInt(todoNumbers.getText()) > 0);
}
```

4.7. JMeter

In der `todolist/pom.xml` ist bereits ein JMeter-Plugin konfiguriert. Führen Sie im `todolist`-Verzeichnis

```
mvn jmeter:configure jmeter:gui
```

aus. Wahrscheinlich werden bei Ihnen nun zunächst viele fehlende Artefakte installiert. Anschließend sollte die JMeter-GUI aber erscheinen.

Öffnen Sie in der JMeter-GUI über `Datei / Öffnen` die Datei `todolist/src/test/resources/jmeter.jmx`. Vergewissern Sie sich, dass die Applikation auf http://localhost:8080/todolist läuft und starten Sie den Testplan über den grünen Play-Button.

Öffnen Sie `http://localhost:8080/todolist` im Browser und vergewissern Sie sich, dass neue Todos angelegt wurden.

Öffnen Sie in JMeter `Testplan / Thread-Gruppe / Graph`, aktivieren Sie die Checkboxen neben `Columns to display` und klicken dann auf `Graphen anzeigen`.

IMPORTANT

Fügen Sie unterhalb von `Thread-Gruppe` im Kontextmenü via `Hinzufügen / Sampler / HTTP-Request` einen HTTP-GET-Request hinzu, der `http://localhost:8080/todolist` abruft. Starten Sie den Testplan dann erneut und schauen sich den Graphen an.

4.8. SonarQube

Diese Aufgabe setzt voraus, dass Docker installiert ist.

Starten Sie eine Instanz von SonarQube über Docker

```
docker run -d --name sonarqube -p 9000:9000 -p 9092:9092 sonarqube
```

Unter Linux muss docker als `root` also vorangestellten `sudo` ausgeführt werden.

Verfolgen Sie die Log-Ausgabe mit

```
docker logs -f sonarqube
```

Sonarqube sollte bereit sein, sobald ein Log-Eintrag wie

```
2022.08.22 06:57:02 INFO app[][o.s.a.SchedulerImpl] SonarQube is up
```

erscheint.

Vergewissern Sie sich, dass SonarQube wirklich läuft, in dem Sie http://localhost:9000 im Browser öffnen. Per Default ist der Username `admin` und das Passwort auch `admin`. Sie werden aufgefordert, dieses zu ändern. Ändern Sie das Passwort in `admin123`

CAUTION Der Start von SonarQube kann mehrere Minuten dauern.

Führen Sie im Verzeichnis `~/sm/todolist`

```
mvn \
org.jacoco:jacoco-maven-plugin:0.8.8:prepare-agent \
test \
org.jacoco:jacoco-maven-plugin:0.8.8:report \
org.sonarsource.scanner.maven:sonar-maven-plugin:3.9.1.2184:sonar \
-Dmaven.test.failure.ignore=true -Dsonar.login=admin -Dsonar.password=admin123
```

aus.

NOTE Die längliche Befehlszeile erspart einem im Gegenzug, die beiden Plugins in der `pom.xml` zu konfigurieren.

Öffnen Sie die Links im Browser, die zum Schluss des Maven-Outputs stehen.

Schauen Sie sich an, wie die Code-Abdeckung dargestellt wird und was die gefundenen Issues zu bedeuten haben.

---

5. Container Virtualization with Docker

In this exercise, we practise how to use Docker.

5.1. Docker on a GCP VM

Although Docker is meant to run on production systems as well as on development machines like your laptop, we use GCP virtual machines in order to have the same prerequisites.

So, open a cloud shell and set-up a new project `docker-${CAMPUS_ID}-01` like in google_cloud_en.html. Do not forget to

google_cloud_en.html.

Use the region `us-east1`, as we get the smallest virtual machine of type `f1-micro` for free if it is non-preemptible.

Create a small virtual machine with low costs

```
gcloud compute instances create vminstance-01 \
--machine-type=f1-micro \
--image-project=ubuntu-os-cloud \
--image-family=ubuntu-2004-lts \
--metadata=startup-script='#! /bin/bash
sudo apt update -y
sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common -y
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt-get update -y
sudo apt-get install docker-ce docker-ce-cli containerd.io -y
sudo apt install docker-compose -y
sudo usermod -aG docker '"${USER}"
```

If this leads to an error message like

```
ERROR: (gcloud.compute.instances.create) Could not fetch resource:
- The zone 'projects/docker-rwoerzbe-01/zones/us-east1-b' does not have enough resources available to fulfill the
request. Try a different zone, or try again later.
```

please try another zone (c or d) in the same region.

Thanks to the startup script, Docker and Docker Compose are preinstalled on `vminstance-01`.

SSH into the `vminstance-01`

```
gcloud compute ssh vminstance-01
```

If you put your current user in the `docker` group, you do not have to run `docker` with `sudo`.

```
sudo usermod -aG docker $USER
```

You have to logout via

```
exit
```

and relogin to the `vminstance-01` to make this change effective.

Test your installation with

```
docker container run hello-world
```

which should yield an output on the console containing

```
Hello from Docker!
```

This message shows that your installation appears to be working correctly.

5.2. Ubuntu Linux in a Container

In the `vminstance-01`: Run

```
docker container run --name myubuntu --interactive --tty ubuntu:20.04 bash
```

IMPORTANT | What happens during execution of this command and what is its result?

Es werden Images heruntergeladen. Man ist als Root in einem Ubuntu eingeloggt

5.3. Images

5.3.1. Install ping

In the still running container `myubuntu`: Execute

```
ping google.de
```

As you can see, `ping` is not installed. Please do so via

```
apt-get update -y
apt-get install -y iputils-ping
```

IMPORTANT | Which is the drawback for doing so?

Es handelt sich letztlich wieder um Dateiänderungen im Thin-RW-Layer. Beim Neuerzeugen des Containers müsste man die Installation wiederholen.

5.3.2. Dockerfile

In the Cloud Shell: Create a `Dockerfile`

```
cat <<EOT > Dockerfile
FROM ubuntu:20.04
RUN apt-get update -y; exit 0
RUN apt-get install -y iputils-ping
EOT
```

5.3.3. Build Image

In the Cloud Shell: Run

```
docker image build --tag ubuntu-ping:latest .
```

in order to build an image with the tag `ubuntu-ping:latest`.

Convince yourself that this image is locally available via

```
docker image ls
```

Inspect the image via

```
docker image inspect ubuntu-ping:latest
```

5.3.4. Start Container

In the Cloud Shell: Instantiate a container from the new image

```
docker container run --interactive --tty ubuntu-ping:latest bash
```

and convince yourself that `ping` is installed.

5.3.5. Tagging

In the Cloud Shell: Now push the image to the `gcr.io` Docker registry in order to copy it to the `vminstance-01`. In order to do so, you have to tag it properly

```
docker image tag ubuntu-ping:latest gcr.io/${DEVSHELL_PROJECT_ID}/ubuntu-ping:1.0
```

Convince yourself that the tag has been properly set.

```
docker image ls --all
```

5.3.6. Authenticate at gcr.io

In the Cloud Shell: Ensure that the Cloud Shell is authenticated at `gcr.io/docker-${CAMPUS_ID}-01`.

```
gcloud auth configure-docker
gcloud auth print-access-token | docker login -u oauth2accesstoken --password-stdin https://gcr.io
```

5.3.7. Enable Container Registry API

In the Cloud Shell: You have to enable the Container Registry API

```
gcloud services enable containerregistry.googleapis.com
```

5.3.8. Push

In the Cloud Shell: Push that image

```
docker image push gcr.io/${DEVSHELL_PROJECT_ID}/ubuntu-ping:1.0
```

5.3.9. Pull

In the `vminstance-01`: Repeat Authenticate at gcr.io in the `vminstance-01`.

Now pull the image from the repository.

```
docker image pull gcr.io/docker-${CAMPUS_ID}-01/ubuntu-ping:1.0
```

5.3.10. Instantiate

In the `vminstance-01`: Instantiate a container from exactly this pulled image:

```
docker container run --interactive --tty gcr.io/docker-${CAMPUS_ID}-01/ubuntu-ping:1.0 bash
```

5.4. Networking

In the `vminstance-01`: Create and start a docker container

```
docker container create --name echo-server -p 1234:1234 jdkelley/busybox-echo-server:latest
```

```
docker container start echo-server
```

In the host (`vminstance-01`) run

```
nc localhost 1234
```

and type an arbitrary text followed by a return. You can stop the process via `Ctrl+C`.

This is the Dockerfile of `jdkelly/busybox-echo-server`:

```
FROM busybox
EXPOSE 1234
CMD ["nc","-ll","-p","1234","-e","/bin/cat"]
```

IMPORTANT | What do the instructions `EXPOSE` and `CMD` do?

> Basierend auf dem Image busybox wird der Port `1234` von einem instanziierenden Container exponiert. Im Container läuft ein `netcat`-Prozess, der auf Port `1234` lauscht und eingehende Daten as-is retourniert.

Stop and remove the container

```
docker container stop echo-server
docker container rm echo-server
```

5.5. Mounts

5.5.1. Create Volume

In the `vminstance-01`: Create a new volume with name `my-ubuntu-vol`

```
docker volume create my-ubuntu-vol
```

Convince yourself that it has been actually created

```
docker volume ls
```

5.5.2. Mount

The Docker command `run` is a combination of the commands `create`, `start`, and `exec`. Therefore, each run creates a new container.

Run another container named `mounttest` based on the image `ubuntu:20.04`. However, this time we make use of different storage options:

```
docker container run --interactive --tty --name mounttest \
--mount type=volume,source=my-ubuntu-vol,target=/my-ubuntu-vol \
--mount type=bind,source=/etc,target=/host-etc \
--mount type=tmpfs,target=/host-tmpfs \
ubuntu:20.04 bash
```

Now, that you are logged into the container: Create empty files with `touch`

```
touch /myFile
touch /my-ubuntu-vol/myFile
touch /host-etc/myFile
touch /host-tmpfs/myFile
```

Exit the container

```
exit
```

Restart the container

```
docker container start mounttest
```

Open a bash in the running container

```
docker container exec -it mounttest bash
```

IMPORTANT | Which files are still in place and why?

`/host-tmpfs/myFile` ist verschwunden, weil sie im tmpfs-Mount war

5.6. Resource Limitation

In the `vminstance-01`:

By means of Linux' `cgroups`, Docker is capable of limiting (virtualized) hardware resources per container.

Memory limits can be set like this

```
docker run --detach --publish 80:80 --name nginx --memory=6m nginx:latest
```

As you can see by

```
docker container exec -it nginx bash
```

you cannot login into the container because it failed to start before. Delete the container and re-run it with more memory

```
docker container stop nginx
docker container rm nginx
docker run --detach --publish 80:80 --name nginx --memory=10m nginx:latest
```

Figure out the `EXTERNAL_IP` of `vminstance-01`. Then you should be able to open

```
http://<EXTERNAL_IP>
```

in your browser. (Before, please add a firewall rule as desribed in [the exercise for the Google Cloud]()

Convince yourself that the memory limit is properly set within the host's filesystem beneath `/sys/fs/cgroup`.

```
cat /sys/fs/cgroup/memory/docker/$(docker ps --no-trunc -aqf "name=nginx")/memory.limit_in_bytes
```

Login into the nginx container

```
docker exec -it nginx bash
```

and convince yourself that the `memory.limit_in_bytes` is now in the memory root directory

```
cat /sys/fs/cgroup/memory/memory.limit_in_bytes
```

5.7. Out-of-Memory-Killer

Being still logged in the nginx container, execute

```
yes | tr \\n x | head -c $((1024*1024*10)) | grep n
```

As you might notice, the process group exceeds the container's memory limits. However, the container is still up. This is because the processes of the group have been created as children of the bash process and could be sacrificed as such.

If you stop, remove and run the container `nginx` with `10m` but with the additional process group

```
docker run --detach --publish 80:80 --name nginx --memory=10m \
nginx:latest 'yes | tr \\n x | head -c $((1024*1024*10)) | grep n'
```

again, the container will not start-up as you can see by

```
docker container ls --all
```

---

6. Google Cloud Platform

In this assignment, we practise how to use the Google Cloud Platform. We will create a cluster that could indeed be used for a application in production. For exemplification, we indeed deploy a small Spring Boot based application called "parcer".

Wichtige Frage zum Übung Cloud-Computing:

- Es wird auf unserem System ein Extern-IP eingebaut, die unsere Anwendung im Browser erreicht.
- Wir beschäftigen wir uns von Backend:
• Wir haben einen Virtual Maschine (vminstancegroup-01) installiert, die mit unsere Daten Bank redet.
• Die Virtual Maschine exportiert einen port 8080, die mit dem Extern IP kommuniziert.

- Unsere Project zum Cloud-Computing ist so geteilt:
• Wir haben den Project name mit dieser Syntax: Parcer-campusid-01(muss aber eindeutig sein).
• Unser Projekt soll mit einem Billing-Account verbinden (Es ist 50$ initial). Die Ressource, die wir benutzen, werden die Dollars in die Zeit verbrauchen.
• Wir bauen einen Daten Bank. Dazu brauchen wir einen sqlinstance-01.
• Dann werden unsere sqlinstance-01 mit unsere virtual-maschine(vmistance-01) verbinden. Dann läuft unsere vmistance-01, die unsere parcer.jar Anwendung ausführt.
• Die Jar-Dateien werden in Bucket gespeichert. Darin braucht man eine Crediantial-Datei(parcer.jar), die man in Daten Bank zugreifen kann.
• Diese wird über eine port 8080 exponiert, Dann erreicht man den Browser.

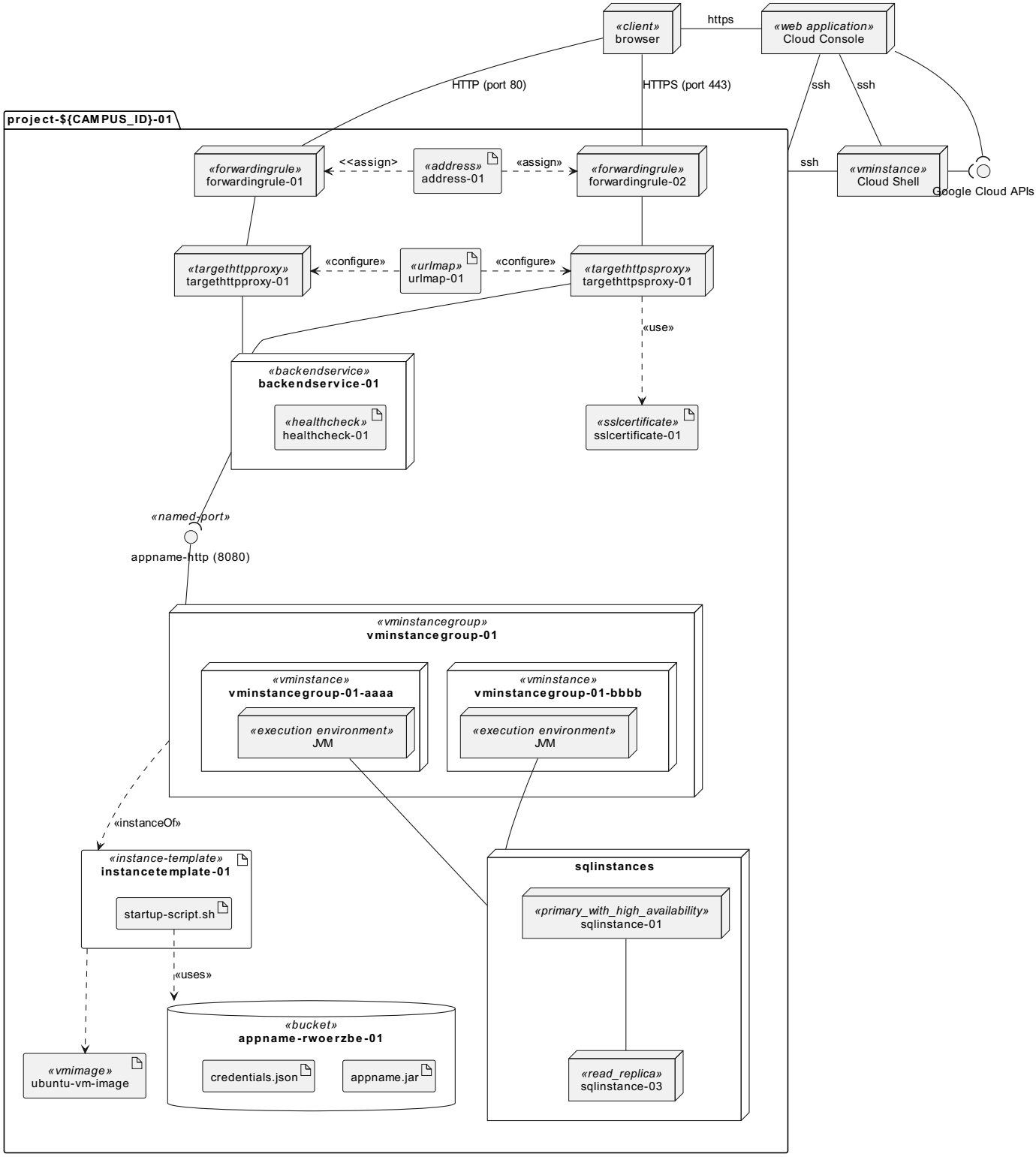


Figure 1. Target Deployment Architecture

The figure above depicts how the cluster with all its resources will look like after completion of the whole exercise.

6.1. Projects

Building an managing a cluster in Google Cloud is all about creating and manipulating so-called **resources** that represent some part of the cluster like a virtual machine. Every resource in Google Cloud must be accomodated in a project. In this task, we will create one with the Project ID `parcer-${CAMPUS_ID}-01`, in which `${CAMPUS_ID}` is just a placeholder for your Campus ID (like `rwoerzbe`).

6.1.1. Billing

Your usage of Google Cloud in this course is financed by free coupons for educational purposes. Be aware that Google also offers free 300$ trials for everybody, which ask for payment information during registration.

WARNING | **Please do not provide any payment information in the context of this course!**

In order to obtain a coupon and charge your Google Cloud billing account with it, follow the following steps:

1. Open the respective coupon retrival page. The actual URL is specific to your course an will be revealed in the respective Ilias course folder.
2. Enter your data. Enter your @smail.th-koeln.de email address as only these email addresses are eligible for receiving coupons.
3. Poll your @smail emails
4. Verify your @smail email address
5. Again, poll your @smail emails
6. Copy the coupon code
7. (If you have no Google account yet, please create one and log in.)
8. Call https://console.cloud.google.com/education and paste the coupon code there.

Open the billing in the Cloud Console. There is supposed to be a billing account charged with $50.

CAUTION | Keep in mind that most Google Cloud resources drain your billing account after creation even if they are not actively used. Therefore, delete Google Cloud projects you do not longer need or at least disconnect them from billing.

6.1.2. Cloud Shell

gcloud ist ein Kommandozeile für die Cloud-Computing Befehl

There are several way, i.e., clients that allow for managing resources in the Google Cloud. In this exercise, we mainly use the command line tool `gcloud` in the Cloud Shell instead of the Web UI of the Cloud Console. The Cloud Shell is a virtual machine with preinstalled command line tools like `gcloud`, `docker` and `kubectl` which is used by means of a terminal that runs in the browser.

In order to open the Cloud Shell Log in into Cloud Console. Then, open a Cloud Shell using the button for `Active Cloud Shell` on the top right corner. After a while a terminal should pop up at the bottom of the browser tab. Please check, if you can invoke gcloud

```
gcloud --version
```

IMPORTANT | Unless otherwise stated, all the commands in this exercise have to be executed in the Cloud Shell. Please note that the Cloud Shell disconnects after some time of inactivity and requires to have the region and zone to be set again (cf. [gcloud configuration])

- You could also install `gcloud` locally in your machine and use it there instead of in the Cloud Shell. For sake of comparability, we all use the Cloud Shell in this assignment.
- Pasting text sometimes works with `Shift+Insert` and sometime with `Ctrl+V` depending on the client's OS and browser.
- APIs in the Google Cloud must be activated per project at first use. In such cases please just follow the URL in the terminal output and activate the API.

- Most of the following steps can also be done via the web UI of the Cloud Console. However, it is up to you to transfer the Cloud Shell commands into actions within the web UI.
- The commands in this exercise are sometimes quite long and exceed one line. If you copy & paste these commands, please take care that you do not copy additional line feeds. Some commands are deliberately broken up into several lines. In these cases, the linefeeds at the end of each line are masked with a `\`. In these cases, please make sure that the line feed is indeed pasted, i.e. that no `\` is pasted somewhere in the middle of a line.

6.1.3. Set the CAMPUS_ID

tag::campusid[]

In order to avoid naming collisions between projects and resources of students, we use our Campus ID for discriminators in names. Wherever you encounter `${CAMPUS_ID}` in the following, please replace it with **your** Campus ID (mine is `rwoerzbe`). Those occurences of `${CAMPUS_ID}` which are in Bash Shell commands can be left in place since they are replaced automatically if you execute beforehand:

```
echo "export CAMPUS_ID=<yourCampusID>" >> ~/.bashrc
source ~/.bashrc
```

WARNING Please replace `<yourCampusID>` with your Campus ID here!

This has to be done for every Bash we will use, so in the Cloud Shell and for each user on each virtual machine (at the beginning of the very first SSH session). Please reassure yourself that `CAMPUS_ID` is actually set by executing

```
echo ${CAMPUS_ID}
```

Occurences of `${CAMPUS_ID}` which are not in Bash shell commands but in, e.g., URLs have to be manually replaced with your Campus ID.

Some seem to prefer a Z shell (`zsh`) before a Bourne Again Shell (`bash`). This is maybe due to the fact that MacOS uses zsh as default. In this case, you have to

```
echo "export CAMPUS_ID=<yourCampusID>" >> ~/.zshrc
source ~/.zshrc
```

end::campusid[]

6.1.4. Create a Project

Create a project via

```
gcloud projects create parcer-${CAMPUS_ID}-01
```

CAUTION Please do so via this command line but not via the Cloud Console. Creating a project via the Cloud Console omits the automatic creation of a default service account which is is needed later on in Service Accounts and Credentials. A posterioi addition of a default service account with an ID in the expected format is not possible after project creation.

Convince yourself that the creation was successful via

```
gcloud projects list
```

Even more details about the project can be obtained via

```
gcloud projects describe parcer-${CAMPUS_ID}-01
```

or alternatively by navigating to

https://console.cloud.google.com/iam-admin/settings/project?project=parcer-${CAMPUS_ID}-01

- Our Project ID scheme consists of three segements: `system-${CAMPUS_ID}-attempt`. Example: `parcer-rwoerzbe-01`. Here
  - `system` is the name of an application like parcer
  - `${CAMPUS_ID}` your Campus ID which avoids naming collisions with other students.
  - `attempt` a two-digit number that allows for repeating this whole exercise just by increasing this number.
- IDs of resources are prefixed by a name for the resource type and suffixed by again a two-digit number. Resources ids must be unique just within their project. Example: `vminstance-01`

6.1.5. gcloud Configuration

Some settings don't have to be set in every Cloud Shell session. `gcloud` takes those from the current configuration. List those values with

```
gcloud config list
```

and if there is no line containing

```
project = parcer-${CAMPUS_ID}-01
```

you have to set this by hand in the active terminal with

```
gcloud config set project parcer-${CAMPUS_ID}-01
```

After doing so, the project ID is stored in the environment variable `DEVSHELL_PROJECT_ID` of the Cloud Shell.

6.1.6. Project Billing

You have to link your project with your billing account so that chargeable resources can be allocated. First determine the AccountID of your billing account which you created in Billing

```
gcloud beta billing accounts list
```

NOTE Some `gcloud` functions are in alpha or beta status, therefore the preceding beta. Google can change functions in Alpha- or Beta-Status without providing donward compaitiblity.

Link your project to your billing account with:

```
gcloud beta billing projects link ${DEVSHELL_PROJECT_ID} --billing-account=<AccountID>
```

The moment you link your billing account with your project you start to spend your credit. **Please note that resources like virtual machines consume these credits after creation whether you use them or not!**

To make sure that you don't waste your credit over night, you should delete projects after you stop working on them like parcer-${CAMPUS_ID}-01 **after** completing this exercise with:

```
gcloud projects delete ${DEVSHELL_PROJECT_ID}
```

Alternatively, you could delete the link between the project and billing account. In this way, credit consuming resources are deleted or deactivated automaticely.

```
gcloud beta billing projects unlink ${DEVSHELL_PROJECT_ID}
```

6.1.7. Enable APIs

Google Cloud is managed through a set of APIs, which have identifiers like `compute.googleapis.com`. You can print the whole set via

```
gcloud services list --available
```

APIs have to be enabled in order to create and manage resources of certain types in your project. For example, creating virtual machines require the `compute.googleapis.com` to be enabled. It is not advisable to enable all available APIs at once as some require or create certain cost-incurring resources.

`gcloud` in the Cloud Shell as well as the Cloud Console proactively ask for activating a respective API on the first command that requires that API. However, at least in Cloud Shell it seems that this on-demand activation is not reliable. You should enable at the following API explicitely:

```
gcloud services enable compute.googleapis.com
gcloud services enable sqladmin.googleapis.com
```

Wir setzen die Region und zones ,damit der Google cloud beim jeden Befehle weißt, worauf es sich bezieht, um diese Befehle auf den bestimmten Zone auszuführen

6.1.8. Set Default Compute Region and Zone

Whenever you create a resource, you have to specify the zone or at least the region, where the resource is supposed to be located. You can do this with the command command line parameters `--region` and `--zone` in every `gcloud` command. Alternatively, you can set them as project properties like so

```
gcloud config set compute/region europe-west3
gcloud config set compute/zone europe-west3-c
```

TIP

These `region` and `zone` settings are only valid for your current Cloud Shell session. Enter them again when you start a new one.

6.1.9. Add Users

I might ask you to add me to your Google Cloud project. To do so, use the Cloud Console, click `IAM & admin → IAM → ADD → New members` and add `rene.woerzberger@gmail.com` with the role `Basic / Editor`.

6.2. Cloud SQL

In the following, we will create a production ready cluster for our application parcer. We start from the very botton, i.e. backend which is the database. In this exercise, you will create a create a Postgres database and later connect it with the application program. There are three options to create a Postgres instance within Google Cloud:

- Install Postgres manually in a VM instance
- Postgres as part of a Kubernetes cluster

- Cloud SQL as managed SQL service

In this exercise pursuits the latter option.

6.2.1. Create Cloud-SQL-database

Create a new Cloud SQL instance named `sqlinstance-01` in the Cloud Shell.

```
gcloud sql instances create sqlinstance-01 --cpu=1 --memory=3840MiB --database-version=POSTGRES_13 --region=europe-west3
```

- Der Connection-Name ist wichtig, damit der parcer-Anwendung das im Virtualle Maschine läuft, die Datenbank findet.

(This may take a while)

> **WARNING** Even after deleting a Cloud SQL instance you can not use the same name of this instance again in the same project and zone!

6.2.2. Set Password for Postgres User

Set the password for the default user `postgres` in `sqlinstance-01`.

```
gcloud sql users set-password postgres --instance=sqlinstance-01 --password=x79bi1vzU77
```

> **WARNING** Of course, it is more secure if you choose an individual password. If you choose a different password, you must alter the value for `spring.datasource.password` in Run Postgres-Client, Create Virtual Machine Instance, and Instance Templates.

6.2.3. Run Postgres-Client

Test the database using the Postgres client and some bare SQL statements.

Open a session in the Postgres client

```
gcloud sql connect sqlinstance-01 --user=postgres --quiet
```

When asked for a password, use the one you have set before (default: `x79bi1vzU77`). Please note that the password is not revealed in the console if you type it in or paste it. The prompt now is supposed to change to `postgres`⇒

Create the table `todolist`:

```
CREATE TABLE todolist (id SERIAL, name VARCHAR(255) NOT NULL);
```

Add a row to `todolist`:

```
INSERT INTO todolist (name) VALUES ('Kochen'), ('Putzen');
```

Print the content of `todolist`:

```
SELECT * FROM todolist;
```

Now leave the Postgres command line client

```
quit;
```

6.3. Buckets

Buckets are another way to persist and retrive data. Buckets are different from databases like Cloud SQL insofar that they rather work like a file system that is accessible remotely via a web API. The standard command line client for the Cloud Bucket API is `gsutil`, which can particularly be used to copy files between your local file system or the file system of the Cloud Shell or some virtual machine and a bucket.

In the following, we use a bucket particularly as a central store for the `parcer-with-postgres.jar` file which contains the most recent version of our parcer Java program. In theory, we can update this `parcer-with-postgres.jar` in the bucket. The startup scripts of the virtual machines will then pull the current version of `parcer-with-postgres.jar` form our bucket during each boot.

6.3.1. Get Application JAR

In this exercise, we do want to temper with parcer's source code nor build it with Apache Maven. Therefore, you can download a pre-built `parcer-with-postgres.jar` to the Cloud Shell's local file system:

```
curl -o ~/parcer-with-postgres.jar -L https://gitlab.com/rwoerzbe/public/-/raw/main/parcer_with_postgres.jar
```

Please check in the Cloud Shell if a `ls` particularly lists `parcer-with-postgres.jar` and it is roughly 50 MB big.

`parcer-with-postgres.jar` is already configured in a way that it requires a Postgres database to run at all. The only property that needs to be configured is the specific connection name to the Postgres database. We will see how that can be done in Create Virtual Machine Instance.

6.3.2. Create Bucket

Now we have to create a Bucket

```
gsutil mb gs://${DEVSHELL_PROJECT_ID}
```

Please note that such buckets can be exposed to resources of other Google Cloud projects and the internet and thus need a name that is unique within the whole Google Cloud. Here, we just take the already unique name of our Google Cloud project, which is stored in the environment variable `DEVSHELL_PROJECT_ID`.

6.3.3. Upload parcer-with-postgres.jar to Bucket

Now we can upload our `parcer-with-postgres.jar` to this bucket

```
gsutil cp ~/parcer-with-postgres.jar gs://${DEVSHELL_PROJECT_ID}
```

Convince yourself that the file has been uploaded to the bucket

```
gsutil ls gs://${DEVSHELL_PROJECT_ID}
```

6.4. Service Accounts and Credentials

Aus Sicherheitsgründen können beliebige Programme nicht aus der Ferne mit sqlinstance-01 verbinden, sondern müssen ordnungsgemäß authentifiziert und autorisiert werden

In Virtual Machines we will connect our program parcer to our `sqlinstance-01`. Due to security reasons, arbitrary programs cannot connect remotely to `sqlinstance-01` but need to be authenticated and authorized properly. Like humans users authenticate to services via user accounts, programs use so called service accounts in order to authenticate and get authorized.

In each project, there is already a default service account with a name like `$PROJECT_NUMBER-compute@developer.gserviceaccount.com` (e.g. `458953960021-compute@developer.gserviceaccount.com`), which we can use.

> This default service account by default has the role `roles/editor` which encompasses much more

WARNING permissions than needed to access `sqlinstance-01`. This is a violations of the least privilege principle but spares us to create a dedicated service account.

Clients that run on dedicated virtual machines (like parcer) will need a file `credentials.json` in order to authenticate. This file particularly contains the name of the service account plus a private key for authentication. We can create this file like so:

```
gcloud services enable compute.googleapis.com
export PROJECT_NUMBER=$(gcloud projects list --filter=$DEVSHELL_PROJECT_ID --format='value(PROJECT_NUMBER)')
export SERVICE_ACCOUNT=$PROJECT_NUMBER-compute@developer.gserviceaccount.com
gcloud iam service-accounts keys create ~/credentials.json --iam-account=$SERVICE_ACCOUNT
```

Now upload this `credentials.json` to our bucket, too, in order to make it accessible to our virtual machines later on which run parcer as a SQL client.

```
gsutil cp ~/credentials.json gs://${DEVSHELL_PROJECT_ID}
```

WARNING As the name suggests, `credentials.json` contains secrets like the private key that are necessary to authenticate with the `sqlinstance-01`. Therefore, accessbility of our bucket should be as restricted as possible. In particular, it should not be made accessible via internet, which is quite easy to achieve for buckets.

6.5. Virtual Machines

In this exercise you will run and test parcer in a single virtual machine named `vminstance-01`.

6.5.1. Create Virtual Machine Instance

For a test, we create a virtual machine `vminstance-01` via the Cloud Shell

```
gcloud compute instances create vminstance-01 \
--zone=europe-west3-c \
--image-project ubuntu-os-cloud \
--image-family ubuntu-2004-lts \
--metadata=startup-script='#! /bin/bash
sudo apt update -y
sudo apt install openjdk-11-jdk-headless -y
curl -sSO https://dl.google.com/cloudagents/add-google-cloud-ops-agent-repo.sh
sudo bash add-google-cloud-ops-agent-repo.sh --also-install
export PROJECT_ID=$(curl "http://metadata.google.internal/computeMetadata/v1/project/project-id" -H "Metadata-Flavor:
Google")
export GOOGLE_APPLICATION_CREDENTIALS=$HOME/credentials.json
export CLOUD_SQL_INSTANCE_CONNECTION_NAME=$PROJECT_ID:europe-west3:sqlinstance-01
gsutil cp gs://$PROJECT_ID/credentials.json $GOOGLE_APPLICATION_CREDENTIALS
gsutil cp gs://$PROJECT_ID/parcer-with-postgres.jar $HOME/parcer-with-postgres.jar
java -Dspring.cloud.gcp.sql.instance-connection-name=$CLOUD_SQL_INSTANCE_CONNECTION_NAME -
Dspring.datasource.username=postgres -Dspring.datasource.password=x79bilvzU77 -Dspring.cloud.gcp.sql.database-
name=postgres -Dspring.datasource.continue-on-error=true -Dspring.datasource.initialization-mode=always -
Dspring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQL92Dialect -Dspring.jpa.hibernate.ddl-
auto=update -Dspring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true -jar $HOME/parcer-with-postgres.jar'
```

This lengthy command line demands for some explanations:

festlegen der Zones, falls man vergessen hat am Anfang einzustellen

- `zone=europe-west3-c` means that `vminstance-01` is created in zone `europe-west3-c`

die Virtuelle Maschine soll Ubuntu sein

- `--image-project ubuntu-os-cloud --image-family ubuntu-2004-lts` points to an image that contains a bare Ubuntu 20.04 LTS operating system.

soll beim Hochfahren eine paare Sache installieren, indem man die metadata in (key values) eingibt. Der Key lautet: (startup-script='#! /bin/bash) und der value ist alles was von Unter steht.

- `--metadata=startup-script` command line option has a whole inline Bash script as its value. Please note, that line breaks do **not** have to be escaped via \ in this script. The startup script does the following:

sucht nach Update

Installation der openjdk11

  - `sudo apt update -y && sudo apt install openjdk-11-jdk-headless -y` installs an OpenJDK 11 (i.e. Java)
  - `curl -sSO https://dl.google.com/cloudagents/add-google-cloud-ops-agent-repo.sh && sudo bash add-google-`

`cloud-ops-agent-repo.sh --also-install` installs a OPS Agent on the virtual machine that particularly forwards log files produced by $parcer to a central storage such that it can be inspected in the Cloud Console (cf. Observability)

- `export PROJECT_ID=$(curl "http://metadata.google.internal/computeMetadata/v1/project/project-id" -H "Metadata-Flavor: Google")` determines the project ID within the `vminstance-01` Elaubt des Zugriffs der Metadata mit dem IP
- `export GOOGLE_APPLICATION_CREDENTIALS=$HOME/credentials.json` stores the location of the `credentials.json` file
  setzen wir die Connection mit der Daten Bank
- `export CLOUD_SQL_INSTANCE_CONNECTION_NAME=$PROJECT_ID:europe-west3:sqlinstance-01` stores the connection string to the `sqlinstance-01` database. The Spring Boot Java program parcer will later look for this environment variable `GOOGLE_APPLICATION_CREDENTIALS`. Please reassure yourself that the other two literal segments really apply to your setting, i.e., that you have a `sqlinstance-01` which runs in zone `europe-west3`. Also please verify that the password
  setzen der Username der Daten Bank mit dem Passwort
  set in `spring.datasource.password` is actually the password that you have chosen in Set Password for Postgres User
- The `gsutil` command download the `credentials.json` and the `parcer-with-postgres.jar` to the `vminstance-01`. Please note that the `startup-script` is executed in the `root` context, i.e. the files can be found at `vminstance-01:/root`
- The `java` command finally starts parcer.

TIP

Thanks to the startup script, there is nothing left to configure or install in the `vminstance-01`. However, it is sometimes handy to login into virtual machines via SSH. You can do this from the Cloud Shell via `gcloud compute ssh vminstance-01`. This reuses the Cloud Shell terminal such that you have to keep an eye on the command prompt which tells you wether you are currently logged in into a virtual machine or the Cloud Console. Another option is to open a dedicated (browser based) window using the respective `SSH` button in the list of virtual machines in the web UI of the Cloud Console.

6.5.2. Store `vminstance-01` External IP

The exteral IP of the `vminstance-01` in your project is dynamically assigned by Google Cloud. In the following, it is convenient to have the `EXTERNAL_IP` of `vminstance-01` in an environment variable within the Cloud Shell. In the Cloud Shell, you can extract this value into the environment variable like this

```
export EXTERNAL_IP=$(gcloud compute instances describe vminstance-01 --
format='get(networkInterfaces[0].accessConfigs[0].natIP)')
```

6.5.3. Open Firewall

Man soll die Firewall mit dem port 8080ö ffnen. Denn man kann nicht vom Draußen unsere virtuelle Maschine zugreifen

Although there is an external IP for your `vminstance-01`, TCP connections are blocked by default. Therefore, add a firewall rule named `parcerhttp` so that TCP connections to resources in your project's network are accepted:

```
gcloud compute firewall-rules create parcerhttp --allow tcp:8080
```

You can list firewall rules that are in place with

```
gcloud compute firewall-rules list
```

in the Cloud Shell or alternatively in the web UI of the Cloud Console. (Only in the Cloud Shell you see the column `Filters` which tell you about allowed source IP ranges.)

Make sure that your `vminstance-01` is reachable from your local computer. First, print out the `EXTERNAL_IP` in the Cloud Shell

```
echo $EXTERNAL_IP
```

Then open a **local** (Git) Bash shell and execute

```
export EXTERNAL_IP=insert printed IP from above here
ping ${EXTERNAL_IP}
```

6.5.4. Test Application

parcer is a web application that offers just a RESTful web API. You can use the Linux command line tool `curl` to test the application, i.e., create test requests like in the following snippets. The following commands can be executed in either the Cloud Shell or your local (Git) bash, provided that `EXTERNAL_IP` is set properly.

Create a customer

```
curl -sSL -D - -H "Content-Type: application/json" -X POST -d '{"name":"TH Koeln"}' http://${EXTERNAL_IP}:8080/customers
```

Validate a valid postal address

```
curl -sSL -D - -H "Content-Type: application/json" -X POST -d '{"line1": "Rene Woerzberger", "line2": "Claudiusstrasse 16", "line3": "51503 Roesrath"}' http://${EXTERNAL_IP}:8080/addressValidator
```

Validate an invalid postal address

```
curl -sSL -D - -H "Content-Type: application/json" -X POST -d '{"line1": "Rene Woerzberger", "line2": "Claudiusstrasse 16", "line3": "5150 Roesrath"}' http://${EXTERNAL_IP}:8080/addressValidator
```

Create a label

```
curl -sSL -D - -H "Content-Type: application/json" -X POST -d '{"shipmentNumber": "0815", "recipientAddress": {"line1": "Rene Woerzberger", "line2": "Claudiusstrasse 16", "line3": "51503 Roesrath"}}' http://${EXTERNAL_IP}:8080/labels
```

Create a shipment for customer 1

```
curl -sSL -D - -H "Content-Type: application/json" -X POST -d '{"recipientAddress": {"line1": "Rene Woerzberger", "line2": "Claudiusstrasse 16", "line3": "51503 Roesrath"}}' http://${EXTERNAL_IP}:8080/customers/1/shipments
```

You can list all customers by aiming your browser at

```
http://${EXTERNAL_IP}:8080/customers
```

and the shipments of customer 1 by

```
http://${EXTERNAL_IP}:8080/customers/1/shipments
```

Make a local Bash request

```
curl -sSL -D - -H "Content-Type: application/json" -X POST -d '{"name":"TH Koeln"}' http://${EXTERNAL_IP}:8080/customers
```

Open http://${EXTERNAL_IP}:8080/customers in the browser and check whether a customer has been created.

6.6. Scaling

After completing the previous exercise, your project consists of

- a VM instance `vminstance-01`, which acts as an application server
- a Cloud SQL instance `sqlinstance-01`, which serves as database server

In this exercise, `vminstance-01` will be scaled horizontally by using multiple instances. In detail,

- we create an *instance template*, which serves as a base for

- Es wäre mühsam, die Schritte unter Virtuelle Maschineninstanz erstellen mehrmals zu wiederholen, nur um einen Cluster gleicher virtueller Maschineninstanzen zu erhalten. Stattdessen erstellen wir eine Instanzvorlage instancetemplate-01, die angibt, wie die Instanzen unserer virtuellen Maschine aussehen sollen
- Wir nennen unsere Instanzvorlage „Instacegroup-01“, die auf dem gleichen Betriebssytem basiert.
- In einem „Instancegroupe-01“ kann mehere Instance (zum Beispiel: instancegroupe-01-aaaa) versammelt, die aus der „Instancegroupe-01“ abgeleitet wird.

- an *instance group* wherein the instances based on the images are added.

6.6.1. Instance Templates

It would be tedios to repeat the steps in Create Virtual Machine Instance several times just to get a cluster of equal virtual machine instances. Instead, we create an instance template `instancetemplate-01` that specifies how the our virtual machine instances should look like.

- Die „Instancegroupe-01“ wird von draußen durch der Port 8080 exportiert
- Die instance groupe, damit sie weißt, wie die Instanzen unserer virtuellen Maschine aussehen sollen, erstellen wir eine Instanzvorlage „instancetemplate-01“ wo die Start-Up-Script-Datei liegt.
- Diese „instancetemplate-01“ referenziert einen virtual Maschine Image „Ubuntu-virtuel-Image“. Dann weißt die virtual Maschine Image“, wie er eine neue „Ubuntu-virtuel-Image“ erzeugen sollte

```
gcloud compute instance-templates create instancetemplate-01 \
--image-project ubuntu-os-cloud \
--image-family ubuntu-2004-lts \
--tags=http-server,https-server \
--metadata=startup-script='#! /bin/bash
sudo apt update -y
sudo apt install openjdk-11-jdk-headless -y
curl -sSO https://dl.google.com/cloudagents/add-google-cloud-ops-agent-repo.sh
sudo bash add-google-cloud-ops-agent-repo.sh --also-install
export PROJECT_ID=$(curl "http://metadata.google.internal/computeMetadata/v1/project/project-id" -H "Metadata-Flavor:
Google")
export GOOGLE_APPLICATION_CREDENTIALS=$HOME/credentials.json
export CLOUD_SQL_INSTANCE_CONNECTION_NAME=$PROJECT_ID:europe-west3:sqlinstance-01
gsutil cp gs://$PROJECT_ID/credentials.json $GOOGLE_APPLICATION_CREDENTIALS
gsutil cp gs://$PROJECT_ID/parcer-with-postgres.jar $HOME/parcer-with-postgres.jar
java -Dspring.cloud.gcp.sql.instance-connection-name=$CLOUD_SQL_INSTANCE_CONNECTION_NAME -
Dspring.datasource.username=postgres -Dspring.datasource.password=x79bilvzU77 -Dspring.cloud.gcp.sql.database-
name=postgres -Dspring.datasource.continue-on-error=true -Dspring.datasource.initialization-mode=always -
Dspring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQL92Dialect -Dspring.jpa.hibernate.ddl-
auto=update -Dspring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true -jar $HOME/parcer-with-postgres.jar'
```

The additional command line parameter `--tags` opens ports `80`, `8080` (http) and `443` (https) for the virtual machine instances so you don’t have to open them manually in the firewall.

NOTE

Please note that we are basing our template on the existing Ubuntu 20.04 LTS image. We also could have made an image `vmimage-01` by means of `gcloud compute image create vmimage-01` out of the `vminstance-01` with the already backed-in `startup-script` and used that image here instead of the Ubuntu image.

6.6.2. Instance Groups

Virtual machine images, which are based on the same instance template, are grouped into managed instance groups. All instances contained within such groups are based on the same template.

```
gcloud compute instance-groups managed create instancegroup-01 --template instancetemplate-01 --size=2
```

Verify that the VM instances are running with

```
gcloud compute instance-groups managed list-instances instancegroup-01
```

Unfortunately, the output leaves out `EXTERNAL_IP`. Identify the `EXTERNAL_IP` using the Cloud Console. Then open one of the two `EXTERNAL_IP`s in the browser

```
http://${EXTERNAL_IP}:8080/customers
```

where you have to replace `${EXTERNAL_IP}` with the actual IP on your own.

6.7. SQL Failover Standby Instances and Read Replica

In this exercise, you will create a failover standby instance and a read replica `sqlinstance-02` for the running `sqlinstance-01`

- Es wird sqlinstance-01 als a failover stand by instance and a read replica sqlinstance-02 skaliert.

6.7.1. Setup Backup

In preperation for the database replica, you have to make a database backup

```
gcloud sql instances patch sqlinstance-01 --backup-start-time=01:00
```

NOTE

The `--backup-start-time=01:00` set the start time of the backup process between 1 and 7 a.m. GMT. Since backups decrease the throughput of a database, you should set the backup start time to times of low application usage.

6.7.2. Setup Failover Replica

Additionally, a failover SQL instance is created that will step in if the main SQL instance stops working.

- Es wird hier auf der selben zone und Region außerhalb von sqlinstance-01 laufen

```
gcloud sql instances patch sqlinstance-01 --availability-type REGIONAL
```

This process takes some minutes. In the mean time, observe the progress in the web UI of the Cloud Console.

6.7.3. Configure Read Replica

To improve the reading performance of your database, create a read replica.

- Es wird hier auf der gleiche zone und Region von sqlinstance-01 laufen

```
gcloud sql instances create sqlinstance-03 --master-instance-name=sqlinstance-01 --region=europe-west3
```

INFO: Please note that you have to delete this read replica if you want to restore a backup (and recreate it afterwards).

6.7.4. Test Failover

Provoke a failover of the main SQL instance

```
gcloud sql instances failover sqlinstance-01
```

During the failover, aim your browser to http://${EXTERNAL_IP}:8080/customers again and reload the page frequently. For some seconds, the application should respond with http 500 error pages before it responds with http 200 and the result page again.

6.8. Load-Balancing

- wir müssen erst ein backendservice-01 anlegen. Er wird konfiguriert und wird damit verweist an welchen instancegroupe, er weiterleiten muss.

parcer should of course be available from a single external IP. Moreover, the request workload should be distributed between all VM instances. To achieve this, you will create a load balancer in the following steps.

6.8.1. Named Ports

- Es besitzt einen healthckeck-01. Er prüft, welche Instance gesund ist und an welchen Instance Request tatsäzlich abbekommen kann.

You could run multiple processes in a VM instance of an instance group, which use different ports. Named Ports are a means to give a certain process and protocol a name and abstract from the somewhat arbitrarly chose port number.

```
gcloud compute instance-groups managed set-named-ports instancegroup-01 --named-ports=parcer-http:8080
```

6.8.2. Health Checks

Backend services check by means of health checks which VM instances in an instance group are considered healthy, i.e., are able to respond to requests.

```
gcloud compute health-checks create http healthcheck-01 --port-name=parcer-http --request-path=/customers
```

TIP

It is important to choose a request path * that is existent (mapped in implementation) * that responds a `200` (OK) http status code if and only if the VM instance is healthy * that ideally does not produce a great load on the system (which /customers will, if there are many of them in the database) * but still is a good indicator if

the program that runs on the VM instance is fully functional.

6.8.3. Backend Service

A backend service handles the real load balacing. A backend service distributes requests on healthy VM instances with free capacity which should be geographically as close as possible to the client.

Create a backend service:

```
gcloud compute backend-services create backendservice-01 --health-checks=healthcheck-01 --port-name=parcer-http --global
```

Link the backend service to the instance group

```
gcloud compute backend-services add-backend backendservice-01 --instance-group=instancegroup-01 --instance-group-zone=europe-west3-c --global
```

6.8.4. URL Map

A URL map allows for routing requests to different backend services which originally have been routed to the same IP. The routing is based on path segments in the request URL.

NOTE In this way, you can put different microservices under a single external IP and domain name an make them look like a single, integrated system.

Since there is only one backend service at the moment, you can create a trivial URL map with `--default-service`.

```
gcloud compute url-maps create urlmap-01 --default-service=backendservice-01
```

6.8.5. Target HTTP Proxy

Target HTTP proxies direct HTTP(S) requests to backends based on their respective URL map

Note: There are other types of target proxies, e.g., target TCP proxies, which work without a URL map and direct requests directly towards specific backend services.

Create a target HTTP proxy:

```
gcloud compute target-http-proxies create targethttpproxy-01 --url-map=urlmap-01
```

6.8.6. IP Address

Static, non-ephemeral IP addresses are valuable thus billed resources in Google Cloud. Create a IPV4 address:

```
gcloud compute addresses create address-01 --ip-version=IPV4 --global
```

6.8.7. Forwarding Rule

- Damit der targethttpproxy von Außer aufgeruft wird, braucht man Forwarding Rule.

The new IP address will now be linked to a forwarding rule:

```
gcloud compute forwarding-rules create forwardingrule-01 \
--address=address-01 --target-http-proxy=targethttpproxy-01 \
--ports=80 --global
```

`--ports=80` causes the forwarding rule to accept connection on the standard http port 80, instead of the alternative HTTP port 8080. Conversely, the `backendservice-01` is directing request to port `8080` of the VM instances. Therefore, parcer has not to be started with root privileges in the VM instances. This is necessary only for privileged ports between `0` and `1024`.

WARNING Although the Cloud Shell reports successful execution within seconds, the propagation of forwarding rule (changes) takes some minutes to become effective in the Google Cloud global network. So take a break of, let's say, five minutes before you proceed with Load-Balanced parcer Testing.

6.8.8. Load-Balanced parcer Testing

Determine the reserved IP address created before

```
gcloud compute addresses describe address-01 --global
```

Note down this external IP of the forwarding rule below `ADDRESS/RANGE`.

After that, open aim the browser at

```
http://FW_EXTERNAL_IP:80/customers
```

TIP The standard-HTTP-Port `:80` can be of course omitted.

6.9. Domain Names

Up till now, we have used the external IP of your virtual machines or our load balancer to access parcer from clients in the internet. Of course, this is not best practise for real applications. Instead, parcer should be accessible using a (mnemonic) domain name. Moreover, domain names are a neccessity if we want to secure traffic between external clients and parcer via TLS (SSL).

6.9.1. Create Managed Zone

A DNS zone in the Google-Cloud (Cloud DNS) is nothing else than another resource that can be created. In the following, we create a zone for a second level domain name, which we will register in Register Domain Name. This domain name is equal to your Project ID with a top level domain `.com`, e.g. `parcer-rwoerzbe-01`. Although, this domain name is not very mnemonic it should be at least quite unlikely to be registered already.

```
gcloud services enable domains.googleapis.com
gcloud services enable dns.googleapis.com
gcloud dns managed-zones create managedzone-01 --dns-name=${DEVSHELL_PROJECT_ID}.com --description=
```

TIP

- If you create a new or another zone with the same `--dns-name`, you have to prove that you are the owner. There are many ways to accomplish all of which required the participation of an authorized user.
- The `--description=` has to be included even if it is left empty (at least in `gcloud`-Version 228. This seems to be a bug.)

6.9.2. Register Domain Name

We can do the actual registration with

```
gcloud domains registrations register ${DEVSHELL_PROJECT_ID}.com --contact-data-from-file=contacts.yaml
```

However, it is quite tedious to craft the `contacts.yaml` with your personal data by hand. This time, we use the domain registration in the Cloud Console.

Reassure yourself, that you have selected the right project in the Cloud Console, e.g. `parcer-rwoerzbe-01`. Then:

1. In **Search Domain** search for a second level domain name that is your Project ID, e.g. `parcer-rwoerzbe-01`. Select the one with the top level domain `.com` from the suggestions, e.g. `parcer-rwoerzbe-01.com`. This should be the first entry in the suggestions list. Be aware of that the other suggestions are sublte variants of that name. Click `Continue`.
2. In **DNS configuration** leave `Use Cloud DNS (Recommended)` selected. `Cloud DNS zone` should be `managedzone-01`. Click `Continue`.
3. In **Privacy protection** leave `Privacy protection on` selected and click `Continue`.
4. In **Contact details** fill in your contact data (at least the mandatory fields). Please use your real smail email address.

Wait for the registration to be completed. After some minutes you should receive a `Get started with Google Domains` email and `Action required: Please verify your email address for…` email. In the latter one you should follow the link in that email in order to verify your email address.

6.9.3. Create Resource Records

Now create an `A resource record`, which lets the registered domain point towards the `FW_EXTERNAL_IP`. Execute the following in the Cloud Shell:

```
export FW_EXTERNAL_IP=$(gcloud compute addresses describe address-01 --global --format='value(address)')
gcloud dns record-sets transaction start --zone=managedzone-01
gcloud dns record-sets transaction add $FW_EXTERNAL_IP --name=${DEVSHELL_PROJECT_ID}.com. --ttl=7200 --type=A --
zone=managedzone-01
gcloud dns record-sets transaction execute --zone=managedzone-01
```

Wait for a minute.

WARNING | The additional dot at the end of `--name=${DEVSHELL_PROJECT_ID}.com.` is **not** a typo!

6.9.4. Domain Name Testing

Open the following URL in your browser http://${PROJECT_ID}.com/customers , where you have to replace `${PROJECT_ID}.com` by your domain name (e.g. `http://parcer-rwoerzbe-01.com/customers`)

6.10. SSL Certificate

Now clients (browsers) are able to connect over HTTP with the load balancer. It is state of the art that at least connections over the internet are secured via "Transport Layer Security (TLS)" better know as "Secure Socket Layer (SSL)". HTTP connections over SSL are also called HTTPS connections.

6.10.1. Create SSL Certificate

HTTPS connections require SSL certificates, which are installed on the server. The creation of a SSL certificate that is signed from a certificate authority is a lengthy process which is greatly simplified by Google Cloud.

```
gcloud compute ssl-certificates create sslcertificate-01 \
--domains=${DEVSHELL_PROJECT_ID}.com
```

6.10.2. Target HTTPS Proxy

Create a target HTTPS proxy, which uses the just-created SSL certificate.

```
gcloud compute target-https-proxies create targethttpsproxy-01 \
--url-map=urlmap-01 \
--ssl-certificates=sslcertificate-01
```

6.10.3. Forwarding Rule

A new global forwarding rule `forwardingrule-02` redirects traffic directed to port `443` to the new target HTTPS proxy.

```
gcloud compute forwarding-rules create forwardingrule-02 \
--address=address-01 \
--target-https-proxy=targethttpsproxy-01 \
--ports=443 \
--global
```

6.10.4. Show SSL Certificate status

Closely watch the status of the certificate creation process:

```
watch -n 30 gcloud compute ssl-certificates list
```

It will stay in status `PROVISIONING` for around 30 to 60 minutes and should then switch to `ACTIVE`. Wait till this has happened. Sometimes the state switches to `FAILED_NOT_VISIBLE`. Then you have wait a little longer until it finally arrives in `ACTIVE`.

WARNING Please note, that the Cloud Shell disconnects after some time of inactivity. Then you have just to `Reconnect` and repeat the `watch` command.

6.10.5. HTTPS Connection Testing

Open the following URL in your browser https://${PROJECT_ID}.com/customers , where you have to replace `${PROJECT_ID}.com` by your domain name (e.g. `https://parcer-rwoerzbe-01.com/customers`)

WARNING You have to start the URL with `https`. The common redirect from `http` to `https` has not been configured.

6.11. Observability

After the successful go live of our application parcer we have to observe it in case of reported anomalies. This includes the convenient access to a central Logging of our application and the Monitoring of the respective infrastructure.

6.11.1. Logging

Log entries are produced by userland applications like parcer. They are rather specific and reveal chronologies of events and internal state during runtime. They must be maintained by a developer within the respective source code.

In the Cloud Console, enable the logging API

```
gcloud services enable logging.googleapis.com
```

On each virtual machine, you have to install an agent that collects log data. Please note that this agent has already been installed by means of the `startup-script.sh` in the Instance Templates.

```
curl -sSO https://dl.google.com/cloudagents/add-logging-agent-repo.sh
sudo bash add-logging-agent-repo.sh --also-install
```

Now, browse to Logging > Logs Explorer and verify if you can see log entries that have been produced by parcer.

6.11.2. Monitoring

Monitoring is rather application agnostic and targets the infrastructure itself together with rather generic metrics. Each type of resource has numerous types of metrics, which can be arranged in a dashboard in the Google Cloud.

For that, navigate to Metrics > Dashboard. Add some graphs that show you the current CPU utilization of your virtual machine instances as well as that of your Cloud SQL instances.

---

7. Kubernetes

In this exercise, we will touch the most common parts of Kubernetes. It is partially derived from the tutorial at https://cloud.google.com/kubernetes-engine/docs/tutorials/guestbook

7.1. GCP Project

Once more, create a new GCP project. Thus, open a Cloud Shell, google_cloud_en.html and

```
and…
```

```
gcloud projects create k8s-${CAMPUS_ID}-01
gcloud config set project k8s-${CAMPUS_ID}-01
gcloud config set compute/region us-central1
gcloud config set compute/zone us-central1-b
gcloud beta billing accounts list
gcloud beta billing projects link ${DEVSHELL_PROJECT_ID} --billing-account=<AccountID>
```

7.2. Install kubectl

Install `kubectl` in your Cloud Shell

```
sudo apt-get install kubectl
```

7.3. Kubernetes Cluster

Enable the Kubernetes API for your project

```
gcloud services enable container.googleapis.com
```

Create a Kubernetes cluster with two worker nodes

```
gcloud container clusters create guestbook --num-nodes=2
```

Check if everything worked as expected

```
gcloud container clusters describe guestbook
```

7.4. Redis Leader Deployment

Redis is an in-memory database with a key-value data model that supports replication with a single Leader. Run

```
kubectl create -f https://raw.githubusercontent.com/GoogleCloudPlatform/kubernetes-engine-samples/main/guestbook/redis-leader-deployment.yaml
```

You should also open the URL in the browser in parallel to see what is actually applied. Execute

```
kubectl get pods -o=wide
```

to get details about running pods and

```
kubectl logs -f PODNAME
```

to peek into the logs.

7.5. Redis Leader Service

Though the Redis Leader deployment just contains a single pod, we need to create a service for the Redis Leader in order to make it discoverable

```
kubectl create -f https://raw.githubusercontent.com/GoogleCloudPlatform/kubernetes-engine-samples/main/guestbook/redis-leader-service.yaml
```

7.6. Redis Follower Deployment

Also create Redis Follower (a.k.a. workers)

```
kubectl create -f https://raw.githubusercontent.com/GoogleCloudPlatform/kubernetes-engine-samples/main/guestbook/redis-follower-deployment.yaml
```

7.7. Redis Follower Service

And a service for the slave pods

```
kubectl create -f https://raw.githubusercontent.com/GoogleCloudPlatform/kubernetes-engine-samples/main/guestbook/redis-follower-service.yaml
```

7.8. Frontend Deployment

```
kubectl create -f https://raw.githubusercontent.com/GoogleCloudPlatform/kubernetes-engine-samples/main/guestbook/frontend-deployment.yaml
```

7.9. Frontend Service

```
kubectl create -f https://raw.githubusercontent.com/GoogleCloudPlatform/kubernetes-engine-samples/main/guestbook/frontend-service.yaml
```

7.10. Inspect Pods and Services

By

```
kubectl get pods -o=wide
```

you can list all running pods and by

```
kubectl get services -o=wide
```

the created services. Using labels, you can filter the latter ones

```
kubectl get service -o=wide -l=tier=frontend
```

7.11. Frontend Service with Load Balancer

By default, services just have an internal cluster IP (class A, i.e. starting with 10.) .

Download the frontend service YAML

```
curl -sL -o frontend-service.yaml https://raw.githubusercontent.com/GoogleCloudPlatform/kubernetes-engine-samples/main/guestbook/frontend-service.yaml
```

comment in the line with LoadBalancer

```
nano frontend-service.yaml
```

and re-apply it

```
kubectl apply -f frontend-service.yaml
```

If you call

```
kubectl get service -o=wide -l=tier=frontend
```

it should show an `EXTERNAL-IP` (after a while in <pending>)

7.12. Test in Web Browser

Copy the `EXTERNAL-IP` in your web browser. You are now supposed to see the guestbook.

7.13. Login into a Container

In order to log in into a running container, you first have to figure out the containerID and before that, the podID

```
kubectl get pods -o=wide
```

Get some details of a pod

```
kubectl get pod frontend-<some_generated_suffix> -o=yaml
```

As you can derive from the output, there is just one container in the pod. With

```
kubectl exec -it frontend-<some_generated_suffix> bash
```

you can login into that first and only container just like with `docker exec`.

7.14. DNS

Being logged in the container, you can demonstrate via

```
ping redis-slave
```

that the `redis-slave` service can be resolved via DNS.

7.15. Manual Scaling

Download the `frontend-deployment.yaml`

```
curl -sL -o frontend-deployment.yaml https://raw.githubusercontent.com/GoogleCloudPlatform/kubernetes-engine-samples/main/guestbook/frontend-deployment.yaml
```

Decrease the number of `replicas` in `frontend-deployment.yaml` to `2`

```
nano frontend-deployment.yaml
```

and re-apply the YAML

```
kubectl apply -f frontend-deployment.yaml
```

Via

```
kubectl get pods -o=wide
```

you can see that an additional pod has been deleted.

Reset the `replicas` to `3` and once more re-apply the YAML.

7.16. Rolling Updates

Checkout the guestbook sources

```
git clone https://github.com/kubernetes/examples.git
```

Edit `index.html`

```
nano ~/examples/guestbook/php-redis/index.html
```

and replace class `btn-primary` with `btn-success` in order to make the success button green instead of blue.

Rebuild the `gb-frontend` Docker image

```
docker image build --tag gb-frontend:v5 ~/examples/guestbook/php-redis/
```

and recheck if the image can be found locally

```
docker image ls
```

Every GCP project creates a docker image namespace with the project ID in the `gcr.io` registry

Tag the image accordingly

```
docker image tag gb-frontend:v5 gcr.io/${DEVSHELL_PROJECT_ID}/gb-frontend:v5
docker image push gcr.io/${DEVSHELL_PROJECT_ID}/gb-frontend:v5
```

Before doing the rollout, check to rollout status for the deployment called `frontend`

```
kubectl rollout status deployment frontend
```

Now do the rolling update

```
kubectl set image deployment frontend php-redis=gcr.io/${DEVSHELL_PROJECT_ID}/gb-frontend:v5 --record=true
```

This takes some seconds. You can track the status with

```
kubectl rollout status deployment frontend --watch
```

Meanwhile, repeatedly reload (F5) the guestbook in your browser. If you are lucky, the button flips between blue and green and stays green after a while.

You can list the history of rollouts for the deployment "frontend" with

```
kubectl rollout history deployment frontend
```

7.17. Rollback

If something goes wrong or you are unhappy with the latest revision, you can rollback to the penultimate revision again:

```
kubectl rollout undo deployment frontend
```

7.18. Clean up

Either disconnect the project from billing or do the following:

```
kubectl delete service frontend
```

Check if the load balancer has been deleted by

```
gcloud compute forwarding-rules list
```

If so, delete the whole Kubernetes cluster

```
gcloud container clusters delete guestbook
```

Appendix A: Appendix

A.1. Additional Exercise Resources

This is a curated list for web resources that provides additional exercises. Please understand that since these resources are out of my control, I (René Wörzberger) cannot support you if problems arise.

Table 1. Additional Exercise Resources

Resource	Description	Docker	Kubernetes	Google Cloud	Messaging and Streaming	Web Technology
Google Codelabs	free labs	-	X (search for "Kubernetes")	X	X	X
QWIKLABS	**non-free** labs in real Google and Amazon Cloud projects			X		
Katacoda	Some free labs. Since acquisition by O'Reilly, no more free labs are being added anymore.	X	X			

Labs

1. Praktikumsaufabe zu Git und Maven

Ziel dieser Aufgabe ist, anhand eines kleinen Java-Projekts den Umgang mit Git, GitLab und Maven zu üben.

IMPORTANT Nach Lösung aller Aufgaben muss das in dieser Praktikumsaufgabe muss in Ihrem Git-Remote-Repository `toolbox` das Tag `abgabe` vorhanden und geeignet gesetzt worden sein.

1.1. Lokales Repository erzeugen

Erzeugen Sie ein initial leeres Working-Directory `toolbox` mit darin enthaltenem Repository (.git).

IMPORTANT Wie machen Sie das?

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=init_workdir]

Sie werden im Folgenden ein kleines Java-Projekt erstellen und diese in Aufgabe Mavenizing in ein Maven-Projekt überführen. Beim Ausführen des Java-Compilers `javac` und später beim Ausführen von Maven entstehen generierte Dateien, beispielsweise **`.class`- und** `.jar`-Dateien.

IMPORTANT Wieso ist es nicht empfehlenswert, solche Dateien unter Versionskontrolle zu stellen? Was passiert konkret, wenn zwei Branches gemergt werden (beim `git merge` oder `git pull`), in denen eine Klasse `X.java` jeweils geändert wurde und die kompilierte `X.class`-Datei ebenfalls jeweils comitted wurde.

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=merge_class]

IMPORTANT Wie kann man ein bestimmtes Git-Repository so konfigurieren, dass bestimmte Dateien nicht unter Versionskontrolle gestellt werden können?

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=exclude_files]

1.2. Working-Tree

In dieser Aufgabe werden im Working-Tree Änderungen vorgenommen und diese in das lokale Repository committed.

Erstellen Sie im Working-Tree `toolbox` eine Datei- und Verzeichnisstruktur wie folgt:

```
toolbox
|
+-src
| |
| +-Toolbox.java
|
+-README.md
```

1.3. Initiale Implementierung von Toolbox.java

Erstellen Sie in `Toolbox.java` eine Klasse `Toolbox`, die eine `main`-Methode implementiert. Implementieren Sie die Klasse so, dass sie beim Aufruf wie

```
java Toolbox length zähleZeichenDiesesWorts
```

Folgendes ausgibt:

```
23
```

Bei einem fehlerhaften oder unvollständigen Aufruf wie

```
java Toolbox
```

soll

```
Usage: java Toolbox <command> <command-specific arguments>
```

ausgegeben werden.

1.4. Exkurs: Markdown

Machen Sie sich mit der Auszeichnungssprache Markdown. Einen Markdown-Editor mit Preview-Funktion finden Sie unter https://dillinger.io/. Aktuelle IDEs wie IntelliJ und Visual Studio Code bieten ebenfalls (ggf. über Extensions) Unterstützung für die Editierung von Markdown-Dateien.

Erzeugen Sie in `README.md` folgenden Text

Toolbox

Toolbox is a set of commandline tools.

Prerequisites

You need working JDK installation.

Installing

Compile `src/Toolbox.java`

```
javac Toolbox.java
```

Usage

The basic usage pattern is

```
java Toolbox <command> <command-specific arguments>
```

where `<command>` is one of the following

- **length**: Prints out the length of the string passed in the `<command-specific argument>`
- (more to follow)

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=create_readme_md]

1.5. Staging

Vergewissern Sie sich, dass die zwei Dateien `Toolbox.java` und `README.md` aktuell nicht unter Versionskontrolle stehen.

IMPORTANT | Wie machen Sie das mit Git?

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=git_status1]

Wichtig ist hier, dass die beiden Dateien unter `Untracked files` stehen.

Fügen Sie beide Dateien dem Index hinzu.

IMPORTANT | Wie machen Sie das mit Git?

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=add_index]

Vergewissern Sie sich, dass die beiden Dateien nun im Index sind und somit bei nächsten Commit berücksichtigt werden.

IMPORTANT | Mit welchem `git`-Kommando können Sie das erkennen?

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=git_status2]

1.6. Commit

Erzeugen Sie eine neue Version in der Commit-History mit der Commit-Message `"Initiale Version. Command 'length' wird unterstützt."`.

IMPORTANT | Wie machen Sie das mit Git?

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=git_commit]

Vergewissern Sie sich wiederum, dass es zwischen Working-Tree, Index und dem gerade erzeugte HEAD-Commit keine Abweichungen gibt.

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=git_status3]

```
git status
```

```
On branch master
nothing to commit, working tree clean
```

1.7. Changelog in README.md

Fügen Sie einen weiteren Abschnitt `Changelog` am Ende von `README.md` ein, der kurz die Änderung aus dem ersten Commit beschreibt. Beim Format für das Changelog orientieren Sie sich bitte an http://keepachangelog.com/en/1.0.0/

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=changelog]

1.8. Javadoc-Kommentar für main-Methode

Fügen sie in `src/Toolbox.java` einen Javadoc-Kommentar für die `main`-Methode ein (direkt über der Methode).

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=javadoc]

1.9. Änderungen committen

Ihr Working-Directory enthält nun zwei Änderungen, die nichts miteinander zu tun haben. Ein Commit sollte jedoch immer eine in sich kohärente Änderung beinhalten. Committen Sie die daher Änderungen im Working-Directory in **zwei** Commits mit den Commit-Messages

```
"Changelog in README.md eingefügt"
"Javadoc für main in src/Toolbox.java eingefügt."
```

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=git_commit2]

1.10. Tagging

Versehen Sie Ihren letzten Commit mit einem Tag namens `v1.0.0`.

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=git_tag]

Der Tag-Name besteht aus den drei Segmenten `MAJOR.MINOR.PATCH` und orientiert sich damit an der Konvention des Semantic Versioning.

IMPORTANT Nennen Sie jeweils eine hypothetische Änderung in `Toolbox.java` für, die eine Version `v2.0.0` bzw. `v1.1.0` bzw. `v1.0.1` zur Folge hätte.

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=semver]

1.11. Remote-Repository auf GitLab anlegen (Teammitglied A)

Toolbox ist nun so weit, im Team weiterentwickelt zu werden. Im Folgenden werden die Teammitglieder als A und B bezeichnet, wobei davon ausgegangen wird, dass die vorherigen Aufgaben auf dem Rechner von A gelöst wurden und dort das lokale Repository `toolbox/.git` zu finden ist.

Sofern noch nicht geschehen, sollten sich beide Teammitglieder bei `https://gitlab.nt.fh-koeln.de` registrieren.

Teammitglied A soll nun ein neues Gitlab-Projekt `toolbox` anlegen mit

- einer Projekt-URL https://gitlab.nt.fh-koeln.de/<username>/toolbox
- Visibility Level = Private.
- Verzichten Sie darauf, dass eine `README.md` automatisch generiert wird.

Laden Sie Teammitglied B und User `rwoerzbe` in der Rolle Maintainer in das Projekt `toolbox` ein.

Fügen Sie das Remote-Repository als Remote-Referenz mit dem Namen `origin` dem lokalen Repository hinzu und pushen Sie sämtliche Branches und Tags. Tipp: Unter `Project / Details` im Projekt `toolbox` steht die Lösung bereits.

IMPORANT: Wie machen Sie das mit Git?

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=remote_repo]

1.12. Remote-Repository clonen (Teammitglied B)

Das Teammitglied B soll nun das Remote-Repository auf seinem Rechner clonen.

IMPORANT: Wie machen Sie das mit Git?

TIP Auch hier steht unter `Project / Details` im Wesentlichen schon, was zu tun ist.

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=clone_toolbox]

1.13. Feature-Branch command-reverse (Teammitglied B)

Teammitglied B führt in toolbox Folgendes aus

```
git checkout -b command-reverse master
```

NOTE Was bewirkt dieser Aufruf? Tipp: Mit `git <command> --help` hier also `git checkout --help` kann man sich über Kommandozeilenparameter einzelner git-Kommandos informieren.

Implementieren Sie als Teammitglied B in `Toolbox.java` ein `<command> reverse`, das ein `<command-specific argument>`

einfach in umgekehrter Reihenfolge ausgibt.

Beispiel:

```
java Toolbox reverse HalloWelt
tleWollaH
```

Testen Sie Ihre Implementierung und erweitern Sie den Abschnitt `Change Log` in `README.md` folgendermaßen

```
Change Log
----------
## [Unreleased]
### Added
- Implemented command 'reverse'.*

## [1.0.0] - 2018-10-16
### Added
- Initial version. Supports command 'length'.
```

NOTE | Weshalb ist es günstig, den Eintrag unterhalb von `[Unreleased]` zu platzieren?

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=unreleased]

Committen Sie die Änderungen in den aktuellen Branch `command-reverse` mit der Commit-Message `"Implemented command 'reverse'"`.

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=commit_reversed]

Pushen Sie als Teammitglied B den Branch command-reverse in das Remote-Repository.

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=push_reversed]

Erzeugen Sie in Gitlab im Projekt `toolbox` einen Merge-Request mit

- Source branch: `command-reverse`
- Target branch: `master`
- Assignee: <Teammitglied A>

1.14. Feature-Branch "markdown-remark" (Teammitglied A)

Als Teamitglied A erzeugen Sie einen Branch `markdown-remark` und wechselt auf diesen Branch.

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=branch_markdown]

Fügen Sie dann in der ersten Zeile von `README.md` folgenden Text ein

```
This document is written in Markdown.
```

und im Abschnitt `Change Log` folgende Änderung

```
Change Log
----------
## [Unreleased]
### Added
- remark about Markdown in README.md.

## [1.0.0] - 2017-09-21
### Added
- Initial version. Supports command 'length'.
```

Committen Sie die Änderungen mit der Commit-Message `"remark about Markdown"`.

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=commit_markdown]

Pushen Sie den Branch `markdown-remark`.

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=push_markdown]

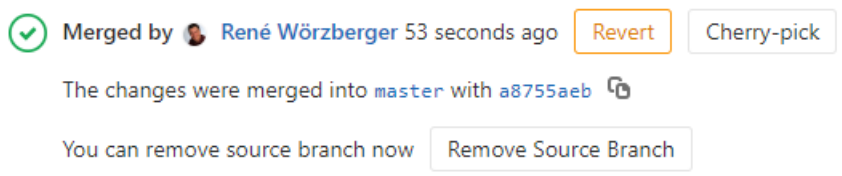
Erzeugen Sie in Gitlab im Projekt `toolbox` einen Merge-Request mit

- Source branch: `markdown-remark`
- Target branch: `master`
- Assignee: <Teammitglied B>

1.15. unproblematischer Merge-Request (Teammitglied A)

Führen Sie in GitLab den ersten Merge-Request (von Teammitglied B) aus.

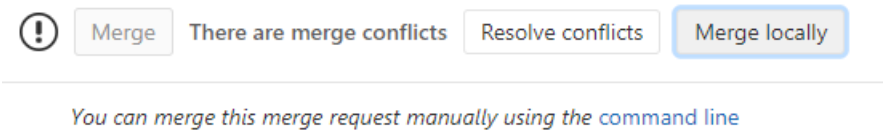
Vergewissern Sie sich dabei, dass er erfolgreich ausgeführt wurde:



1.16. problematischer Merge-Request (Teammitglied B)

Führen Sie in GitLab den zweiten Merge-Request (als Teammitglied B) aus.

Wie Sie sehen, erkennt GitLab hier Konflikte.



Entscheiden Sie sich für die für eine der beiden Optionen (Resolve conflicts ermöglicht die Konflikte über eine WebUI zu lösen, Merge locally weist Sie an, wie man die Konflikte in Ihrem lokalen Repository lösen kann.)

1.17. Mavenizing

(Es ist Ihnen überlassen, welches Teammitglied diese Aufgabe durchführt)

"Mavenizen" Sie das Projekt toolbox auf einem Branch namens `mavenize`, d.h. indem Sie (im Eltern-Verzeichnis von `toolbox`) mittels

```
mvn archetype:generate \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-quickstart \
-DarchetypeVersion=1.4
```

zunächst einen Rahmen generieren, wobei Sie

```
'groupId': de.thk.sm
'artifactId': toolbox
'version': 1.0-SNAPSHOT
'package': de.thk.sm
```

wählen und `Toolbox.java` geeignet verschieben und anpassen.

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=git_mv]

Führen Sie `mvn` geeignet aus, sodass sich die Toolbox wie folgt aufrufen lässt:

```
java -cp target/toolbox-1.0-SNAPSHOT.jar de.thk.sm.Toolbox reverse Hallo
```

Unresolved directive in labs/lab_git_maven_de.asciidoc - include::lab_git_maven_solutions_de.asciidoc[tag=mvn_package]

Integrieren Sie die Änderung im Remote-Repository wiederum mittels eines Merge-Requests.

CAUTION | @SM-Teilnehmer: Taggen Sie die Abgabeversion mit dem Tag `abgabe`.

---

2. Praktikumsaufgabe zur Qualitätssicherung

In diesem Praktikum übernehmen Sie die Entwicklung eines bestehenden Systems namens "Parcel Services (kurz: Parcer)". In der Vorlesung wurden die architektonischen und entwicklungsprozessualen Unzulänglichkeiten von Parcer bereits beschrieben (s. Folie "Fallstudie"). Im Folgenden werden Sie Schritte zur Verbesserung unternehmen. Im Einzelnen werden Sie

- den Build um weitere Qualitätssicherungs-Schritte erweitern.
- (optional) die monolithische Struktur von Parcer ein Stückweit aufbrechen, indem Sie das Parcer-Maven-Modul in mehrere Maven-Module zerlegen,

In manchen Aufgaben werden Verständnisfragen gestellt. Bitte machen Sie sich bei der Bearbeitung der Aufgaben entsprechende Notizen.

IMPORTANT | Als Abgabe zählt bei diesem Praktikumstermin der mit `abgabe` getaggte Commit in Ihrem `parcer`-Remote-Repository.

2.1. Parcer im Überblick

2.1.1. Anwendungsfälle

Parcer ist ein Versandlogistik-System, das Versandkunden eines Logistikdienstleisters nutzen, um Pakete für den Versand vorzubereiten. Es unterstützt die Anwendungsfälle wie in [Use-Cases von Parcer] dargestellt.

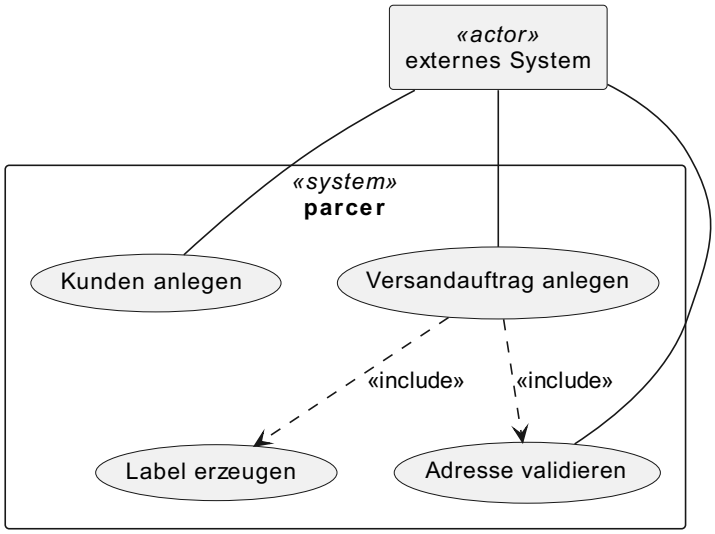


Figure 2. Use-Cases von Parcer

Parcer exponiert WebAPIs für externe Systeme, die beispielsweise Warenwirtschaftssysteme oder WebShop-Systeme von Versandkunden aufrufen können.

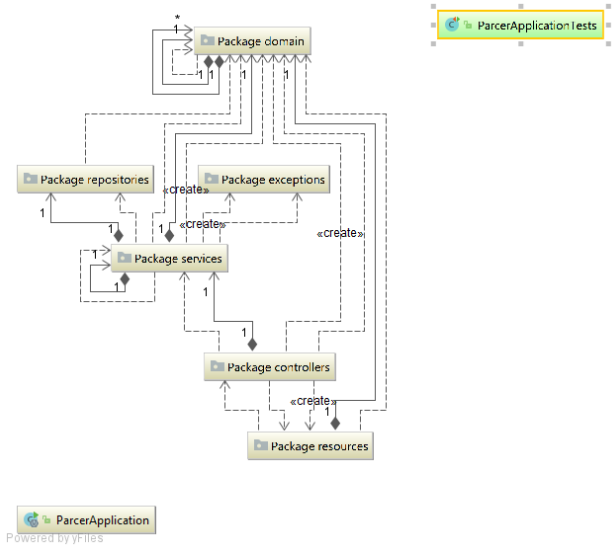
NOTE	Parcer ist real existierenden Systemen entlehnt. Der Code- und Funktionsumfang von Parcer ist aus didaktischen Gründen und aus Aufwandsgründen natürlich um mehrere Größenordnungen kleiner als bei den real existierenden Systemen.

2.1.2. Monolithische Ausgangs-Architektur

Java-Package-Struktur

Die Softwarearchitektur besteht aus mehreren Komponenten, die in Java als Java-Packages realisiert sind. Diese Packages und ihre Abhängigkeiten sind in [Package und Dependencies] dargestellt.

Packages und Dependencies



Die Verantwortlichkeiten der Komponenten sind dabei wie folgt:

- Package `controllers`:
  - Klassen in diesem Paket besitzen Methoden, die bei Requests an die WebAPIs aufgerufen werden und denen eventuelle Daten im Request übergeben werden. Diese Methoden delegieren die Verarbeitung an Klassen im Package services.
  - Beispiel: Der Methode `CustomerController.createCustomer(Customer)` wird ein `Customer`-Objekt übergeben. Dieses

reicht die Methode weiter an `CustomerService.createCustomer(Customer)`.

- Package `domain`
  - Fachliche Daten werden in Objekten von Klassen gehalten, die in Package `domain` definiert sind. Die Erzeugung dieser Objekte geschieht mitunter automatisch, beispielsweise als Formalparameter (Eingabeparameter) beim Aufruf von Methoden im Package `controllers`.
  - Beispiel: Die Klasse `Customer` repräsentiert einen Kunden, der eine eindeutige `id`, einen `name` und Menge von zugeordneten `Shipment`s hat.
- Package `resources`:
  - WebAPI-Responses werden aus Objekten serialisiert, deren Klassen im Package `resources` zu finden sind. Diese Klassen kapseln die aus `domain`, erben jedoch von einer Spring-Boot-Framework-Klasse `ResourceSupport`, wodurch die WebAPI-Responses gemäß der JSON-HAL-Spezifikation zusätzliche Daten beinhalten können.
  - Beispiel: Ein `CustomerResource`-Objekt kapselt ein `Customer`-Objekt. Im Constructor von `CustomerResource` wird zusätzlich dafür gesorgt, dass sich in WebAPI-Responses nach Aufruf von `CustomerController.getCustomer(id)` zusätzlich ein Link (URL) zum Abruf der zugehörigen Shipments befindet.
- Package `services`:
  - Klassen in diesem Paket verarbeiten Aufrufe fachlich, wobei Sie andere Services in Package `services` aufrufen oder Anfragen an die Datenbank erzeugen durch Verwendung von Klassen aus Package `repositories`. Jeder Service besteht dabei aus einem Java-Interface und einer implementierenden Klasse.
  - Beispiel: `ShipmentServiceImpl.createShipment(id)`
    - delegiert die Adressprüfung an `AddressValidationService.getValidationErrors(Address)`
    - delegiert die Label-Erzeugung an `LabelService.createLabel(Address, id)`
    - und speichert bei Erfolg die Sendung mittels `ShipmentRepository.save(Shipment)`
- Package `exceptions`:
  - Bei bestimmten Fehlersitutationen werden in Methoden von Klassen innerhalb von Package `services` Exceptions erzeugt, die im Package `exceptions` implementiert sind.
  - Beispiel: `ShipmentServiceImpl.createShipment(id)` erzeugt eine `InvalidAddressException`, wenn ein Shipment (Versandauftrag) auf Basis einer ungültigen Empfängeradresse erzeugt werden soll.
- Package `repositories`:
  - Das Package `repositories` enthält Interfaces, die von der Spring-Boot-Framework-Klasse `CrudRepository` erben. Das Spring-Boot-Framework erzeugt hieraus automatisch zur Laufzeit implementierende Klassen und Objekte, über die einfache und übliche Datenbankzugriffe realisiert werden.
  - Beispiel: `ShipmentServiceImpl.createShipment(id)` ruft `ShipmentRepository.save(Shipment)` auf. Die Implementierung geschieht durch Spring-Boot automatisch, d.h. es gibt keine `ShipmentRepositoryImpl.save(Shipment)`.

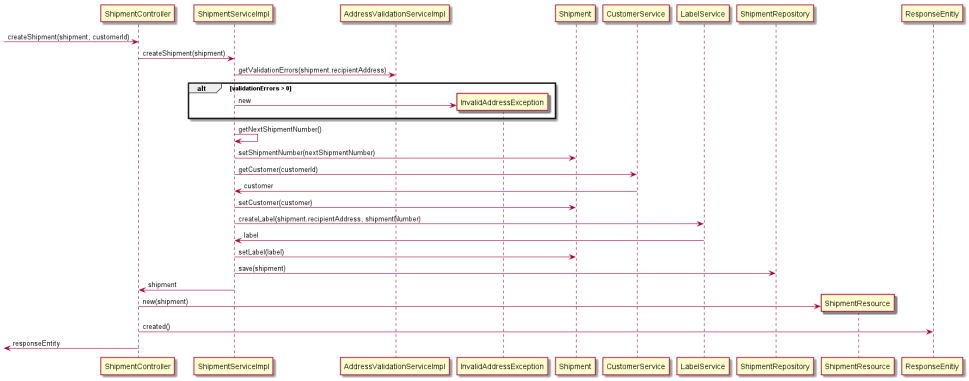
Die Klassen, die die vier Anwendungsfälle implementieren, sind gemäß ihrer technischen Rolle auf die oben dargestellten Packages verteilt. Die Package-Struktur unterteilt die Klassen also nicht bezüglich ihrer fachlichen Zugehörigkeit zu Anwendungsfällen. Beispielsweise befinden sich in Package `services` die Klassen `ShipmentServiceImpl` und `CustomerServiceImpl`, die zur Implementierung verschiedener Anwendungsfälle dienen.

Verhalten

In [Verarbeitung eines createShipment-WebAPI-Requests] ist die Verarbeitung eines WebAPI-Requests als UML-

Sequenzdiagramm dargestellt am Beispiel eines createShipment-Requests. Aus jedem Package ist hierbei mindestens eines Klasse beteiligt.

Verarbeitung eines createShipment-WebAPI-Requests



Maven-Modul-Struktur

Parcer besteht aus einem einzigen Maven-Modul, d.h. der Build wird auch nur durch eine ein `pom.xml` beschrieben und erzeugt genau ein Artefakt `parcer.jar`.

Die `pom.xml` deklariert mehrere Abhängigkeiten zu 3rd-Party-Libraries, insbesondere:

- Spring-Boot: Parcer nutzt das im Java-Umfeld beliebte Spring-Boot-Framework. Spring-Boot befreit Entwickler von bestimmten Implementierungsaufgaben, wie die (De-)Serialisierung von WebAPI-Request/Responses oder die Implementierung einfacher Datenbankzugriffe.
- Thymeleaf: Der Erzeugung eines Labels als PDF geschieht unter Zuhilfenahme der Bibliothek Thymeleaf.

Git-Repository-Struktur

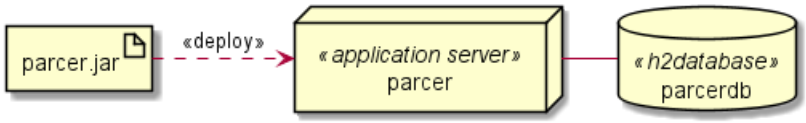
Für die Entwicklung von Parcer gibt es ein führendes Git-Repository. Dort ist sämtlicher Quellcode von Parcer zu finden.

Datenbank

Parcer verwendet eine H2-Datenbank. Annotationen wie `@Entity` der Java-Persistence-API in den Klassen des `domain`-Packages sorgen dafür, dass das Datenbank-Schema automatisch aufgebaut wird. Separate DDL-SQL-Dateien sind hier nicht notwendig. (Aus Gründen der Einfachheit, wird hier die Datenbank nur "in memory" gehalten.)

Abbildung 4 fasst die monolithische Struktur aus Deployment-Sicht zusammen.

Monolithische Struktur als UML-Deployment-Diagramm



2.2. Parcer herunterladen

Führen Sie in einer Bash folgenden Befehl aus, der Parcer in der Ausgangsstellung herunterlädt und im aktuellen Verzeichnis ~/sm/parcer entpackt. (Das Verzeichnis können Sie natürlich selbst wählen.)

```
mkdir -p ~/sm/parcer
curl -skL https://bit.ly/sm1819-parcer | tar xvfz - -C ~/sm/parcer
```

Wenn Sie ein JDK 11 oder höher einsetzen (mittels java -version kontrollierbar), fügen Sie bitte in ~/sm/parcer/pom.xml noch folgende <dependency> ein

```
<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.0</version>
</dependency>
```

2.3. Parcer unter Versionskontrolle stellen

Sorgen Sie dafür, dass das `parcer`-Verzeichnis unter (zunächst lokale) Git-Versionskontrolle gestellt wird.

IMPORTANT | Wie bewerkstelligen Sie das?

Unresolved directive in labs/lab_quality_assurance_de.asciidoc - include::lab_quality_assurance_solutions_de.asciidoc[tag=under_git]

Erzeugen Sie anschließend für Ihr Team ein eigenes Remote-Repository, in dem Sie die Weiterentwicklungen von Parcer versionieren. Setzen Sie die Remote-Ref origin auf Ihr Remote-Repository.

IMPORTANT | Wie bewerkstelligen Sie das?

Unresolved directive in labs/lab_quality_assurance_de.asciidoc - include::lab_quality_assurance_solutions_de.asciidoc[tag=add_origin]

Rufen Sie Maven derart auf, dass `parcer-0.0.1-SNAPSHOT.jar` erzeugt und im lokalen Repository installiert wird

IMPORTANT | Wie bewerkstelligen Sie das?

Unresolved directive in labs/lab_quality_assurance_de.asciidoc - include::lab_quality_assurance_solutions_de.asciidoc[tag=maven_install]

2.4. Parcer starten

Starten Sie Parcer lokal mittels

```
mvn spring-boot:run
```

CAUTION | Dabei wird vorausgesetzt, dass Port `8080` noch durch keinen Prozess belegt ist.

2.5. Requests absetzen

Parcer stellt keine WebGUI bereit, sondern nur eine http-basierte, "RESTful" API, die JSON+HAL-Repräsentationen in Requests und Responses erwartet bzw. produziert.

Folgender Request erzeugt einen neuen Kunden in der Parcer-Datenbank

```
POST /customers HTTP/1.1
Host: localhost:8080

{"name":"TH Koeln"}
```

Effektiv wird hier nur ein Key-Value-Paar übergeben.

Erzeugen Sie einen solchen Request, beispielsweise unter Zuhilfenahme des Kommandozeilen-Tools `curl`, das auch unter Windows im Zuge der Installation von Git in der GitBash verfügbar sein sollte.

```
curl -sSL -D - -H "Content-Type: application/json" -X POST \
-d '{"name":"TH Koeln"}' http://localhost:8080/customers
```

Als Response wird retourniert:

```
{
    "customer": {
        "id": 1,
        "name": "TH Koeln"
    },
    "_links": {
        "shipments": {
            "href": "http://localhost:8080/customers/1/shipments"
        },
        "self": {
            "href": "http://localhost:8080/customers/1"
        }
    }
}
```

Die übergebenen Daten werden angereichert über eine von Parcer vergebene id für den Kunden. Der Response beinhaltet konform zur JSON+HAL-Spezifikation darüber hinaus Links zu assoziierten Daten, nämlich einen Link zu dem Kunden zugeordnete Shipments (die im Moment noch nicht vorhanden sind) und einem `self`-Link, über den die erzeugte Customer-Ressource später erneut abgerufen werden kann.

In den JavaDocs der Methoden der Klassen innerhalb des Packages `controllers` finden Sie weitere `curl`-basierte Beispielaufrufe.

Nehmen Sie diese als Beispiele, um folgende Testfälle manuell durchzuführen:

1. Anlegen eines Kunden (ist oben bereits geschehen)
2. Validierung einer postalischen Adresse (beispielsweise Ihrer Privat-Adresse)
3. Validierung einer postalischen Adresse mit einer fehlenden Ziffer in der Postleitzahl
4. Erzeugung eines PDF-Labels mit beliebiger, valider recipientAddress und beliebiger shipmentNumber
5. Erzeugung einer Sendung mit beliebiger, valider recipientAddress
6. Abruf des in Schritt 5 erzeugten Label
7. Abruf aller Sendungen des in Schritt 1 erzeugten Kunden.

IMPORTANT | Notieren Sie sich die Aufrufe, die Sie getätigt haben.

Unresolved directive in labs/lab_quality_assurance_de.asciidoc - include::lab_quality_assurance_solutions_de.asciidoc[tag=sol-parcertestcurls]

2.6. H2-Console verwenden

Öffnen Sie im Browser http://localhost:8080/h2-console und loggen Sie sich mit

- der JDBC URL `jdbc:h2:mem:testdb`
- User `sa` und
- leerem Passwort ein.

IMPORTANT | Wie können Sie den Inhalt der Tabelle `SHIPMENT` ausgeben?

Unresolved directive in labs/lab_quality_assurance_de.asciidoc - include::lab_quality_assurance_solutions_de.asciidoc[tag=sol-selectshipment]

2.7. Messung der Code-Abdeckung

Die Automatisierung der Qualitätssicherung ist in Parcer nur spärlich ausgebaut. In `parcer\src\test\java\de\thk\parcer\parcer\services\ShipmentServiceImplTest.java` sind lediglich zwei JUnit-Testfälle implementiert.

Messen Sie im Maven-Build die durch die vorhandenen Testfälle erreichte Code-Abdeckung Verwenden Sie hierfür das Maven-JaCoCo-Plugin, das im Productmodel-Beispiel in der Vorlesung/Übung bereits eingesetzt wurde.

2.8. HTML-Report

Stellen Sie sicher, dass während des Maven-Builds in `target/site/jacoco` ein HTML-Report zur erreichten Code-Abdeckung generiert wird.

2.9. JaCoCo-Dokumentation

Informieren Sie sich mittels

```
mvn help:describe -Dplugin=org.jacoco:jacoco-maven-plugin -Ddetail
```

über weitere Goals von JaCoCo.

2.10. Mindestabdeckung

IMPORTANT	Stellen Sie durch Einbindung des passenden JaCoCo-Goals sicher, dass der Maven-Build abbricht, sobald die Anweisungsüberdeckung im Package `de.thk.parcer.parcer.services` weniger als `85%` beträgt. Verwenden Sie das Goal `check`, das in der Phase `prepare-package` ausgeführt werden soll.

Unresolved directive in labs/lab_quality_assurance_de.asciidoc - include::lab_quality_assurance_solutions_de.asciidoc[tag=sol-mindestabdeckungmaven]

2.11. Erweiterung der Code-Abdeckung

Die vorhandenen JUnit-Testfälle führen zu einer Anweisungsüberdeckung von `76%` im Package `de.thk.parcer.parcer.services`. Der Maven-Build bricht daher ab, sofern die Aufgabe Mindestabdeckung korrekt gelöst wurde.

IMPORTANT	Analysieren Sie mittels des generierten Reports in `target/site/jacoco`, welche Anweisungen in Package `de.thk.parcer.parcer.services` nicht überdeckt sind. Erzeugen Sie eine Klasse `parcer\src\test\java\de\thk\parcer\parcer\services\LabelServiceImplTest.java`, mit der Sie die Abdeckung in `de.thk.parcer.parcer.services.LabelService` erhöhen, so dass Sie in dem Package eine Anweisungsabdeckung von mindestens `85%` erreichen. Orientieren Sie sich dabei an `parcer\src\test\java\de\thk\parcer\parcer\services\ShipmentServiceImplTest.java`.

Unresolved directive in labs/lab_quality_assurance_de.asciidoc - include::lab_quality_assurance_solutions_de.asciidoc[tag=sol-erweiterungabdeckung]

2.12. JMeter

Starten Sie JMeter. Sie können die JMeter-GUI ohne Installation als Maven-Plugin beispielsweise direkt von der

Kommandozeile starten, ohne ein <plugin> in der pom.xml konfigurieren zu müssen:

```
mvn com.lazerycode.jmeter:jmeter-maven-plugin:2.8.0:gui
```

Erstellen Sie einen Testplan, den Sie unter src/test/resources/parcer.jmx speichern.

Der Testplan soll einen Customer anlegen und in 100 nebenläufigen Threads jeweils 10 (also insgesamt 1000) Shipments.

Hinweise:

- Maven wird in einer Version 3.5.0 oder höher benötigt.
- Sie müssen bei allen http-Requests den http-Header `Content-Type: application/json` setzen.
- Über die `Config Element`s `HTTP Header Manager` und `HTTP Requests Defaults` können Sie Gemeinsamkeiten der http-Requests an einer Stelle konfigurieren.
- Sie können bei der Anlage der Shipments die ID des angelegten Customers hartkodieren. Sie sollte bei Anlage nur eines Customers `1` sein.
- Vergessen Sie bitte nicht, die `parcer.jmx` unter Versionskontrolle zu stellen.

2.13. Sonarqube (optionale Aufgabe)

Wenn Sie Docker lokal installiert haben, starten Sie einen Sonarqube-Container mittels

```
docker container run -d --name sonarqube -p 9000:9000 -p 9092:9092 sonarqube
```

Überprüfen Sie, ob die Instanz wirklich läuft, indem Sie http://localhost:9000 im Browser öffnen.

Analysieren Sie das Projekt mittels

```
mvn org.sonarsource.scanner.maven:sonar-maven-plugin:3.5.0.1254:sonar
```

und refreshen Sie die Seite im Browser.

Welche Mängel wurden gefunden?

Stoppen Sie anschließend den Sonarqube-Docker-Container mittels

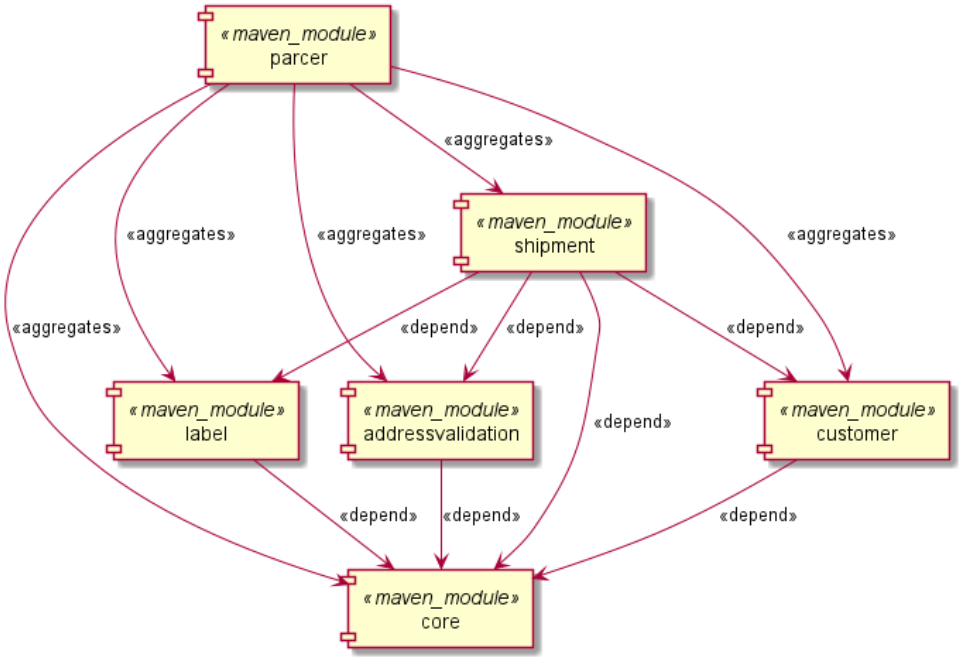
```
docker container stop sonarqube
```

2.14. Zerlegung in Maven-Module (optionale Aufgabe)

Zerlegen Sie das Maven-Modul `parcer`. Das Modul `parcer` soll nur noch als Aggregator-Modul verbleiben und die Module `addressvalidation`, `label`, `shipment`, `customer` und `core` aggregieren. Der unverändert bleibende Quellcode (`*.java`) soll wie folgt aufgeteilt werden:

- Modul `core` erhält die Packages `domain` und `exceptions` und sämtliche bislang darin enthaltende Klassen.
- Module `label`, `addressvalidation`, `customer` und `shipments` haben die Packages `controllers`, `services`, `resources` und `repositories`. Darin enthalten sind die jeweils jedoch nur die für den Anwendungsfall relevanten Klassen. Beispielsweise ist im Modul `customer` im Package `controllers` nur die Klasse `CustomerController`.
- Die Dependencies zwischen den Modulen soll in der jeweiligen `pom.xml` so konfiguriert werden, wie in Abbildung 6 dargestellt.

Maven-Modulstruktur mit Abhängigkeiten



Stellen Sie sicher, dass nach der Zerlegung Parcer nach wie vor mit

```
mvn install
mvn spring-boot:run
```

gebaut und gestartet werden kann.

Hinweise:

- Die Aufgabe besteht im Wesentlich im Anlegen von Verzeichnissen, Anlegen oder Änderung von `pom.xml`s und im Verschieben von `*.java`-Dateien, die nur angepasst werden sollen, wenn Abhängigkeiten unbedingt aufgelöst werden müssen.
- Sie können die aggregierende `parcer/pom.xml` auch als Parent-POM verwenden. Überlegen Sie sich gut, welche `<dependencies>` in welchem Modul überhaupt gebraucht werden. (Wenn eine fehlt, merken Sie das spätestens beim `mvn install` oder ggf. früher in der Entwicklungsumgebung: IntelliJ IDEA markiert fehlende Importe (nach Änderungen in pom.xmls + Klick auf Import Changes) als Fehler). Nutzen Sie auch das `<dependencyManagement>` in der Parent-POM.
- Leider ist das Zusammenspiel von Spring-Boot, Unit-Tests und Maven-Submodulen nicht ganz einfach. Auch wenn es architektonisch unsauber ist, verschieben Sie sämtliche Testfälle (unterhalb von `src/test`) in `shipment/src/test` und die `ParcerApplication.java` auch an die entsprechende Stelle unterhalb von `shipement/src`.

---

3. Praktikumsaufgabe zur Google Cloud und Docker-Containern

Ziel dieses Praktikums ist, die Container-Virtualisierung mittels Docker im Zusammenspiel mit der Google Cloud zu üben.

Da Sie bei der Bearbeitung der Aufgaben zwischen verschiedenen Shells wechseln (Cloud-Shell und SSH-Shell auf `vminstance-01`) achten Sie bitte darauf, dass Sie jeweils in der richtigen arbeiten.

3.1. Google Cloud Projekt aufsetzen

Bereiten Sie ein GCP-Projekt mit Project-ID `parcerdocker-CampusID-01` vor (s. [GCP-Projekt anlegen] und bitte hier **nicht** den `--machine-type=f1-micro` verwenden wie in [Virtuelle Maschine erzeugen], da dieser für die Aufgabe zu schlecht ausgestattet ist). Sie sollten also anschließend eine `vminstance-01` mit installiertem Java und `sqlinstance-01` haben.

Installieren Sie in der `vminstance-01` auch Docker und Docker Compose.

3.2. Firewall öffnen

Öffnen Sie vier TCP-Ports in der Firewall gemäß folgender Vorgaben:

TCP-Port	Name
8080	shipmenthttp
8081	customerhttp
8083	addressvalidationhttp
8084	labelhttp

3.3. Parcer.Customer direkt ausführen

Installieren Sie ein OpenJDK 11 LTS.

Erzeugen Sie in der `vminstance-01` ein Verzeichnis `~/sm/parcer` und entpacken Sie dabei eine Revision von Parcer mittels

```
curl -sL http://bit.ly/smparcermultimodule | tar xvfz - -C ~/sm/parcer
```

Die Revision ist eine Lösung für die Aufgabe aus Zerlegung in Maven-Module (optionale Aufgabe). Die einzelnen Module `shipment`, `customer`, `addressvalidation` und `label` können hier separat gestartet werden und hören auf den freigeschalteten Ports.

Sämtliche Module hängen von einem Modul `core` ab. Installieren Sie dies zunächst im lokalen Maven-Repository

```
cd ~/sm/parcer/core
../mvnw install
```

Starten Sie beispielsweise `customer`

```
cd ~/sm/parcer/customer
../mvnw spring-boot:run
```

und erzeugen einen Customer mittels `curl` von Ihrem lokalen Rechner aus. Notieren Sie sich wieder das absetzte `curl`-Kommando.

Unresolved directive in labs/lab_google_cloud_docker_de.asciidoc - include::lab_google_cloud_docker_solutions_de.asciidoc[tag=curl_customer]

3.4. Docker installieren

Wie Sie sehen, ist Docker in der Cloud-Shell vorinstalliert. Zeigen Sie mittels des hello-world-Images, dass die Installation funktioniert.

Installieren Sie Docker auch in der `vminstance-01`. Zeigen Sie mittels des hello-world-Images auch hier, dass die Installation funktioniert.

3.5. Dockerfiles

> Im Folgenden wird davon ausgegangen, dass Sie mit Docker in der Cloud-Shell arbeiten. Sie können sich

CAUTION jedoch auch ein Docker auf Ihrem lokalen Rechner installieren, was aber aufgrund der Vielfalt Ihrer Betriebssysteme und Versionen in dieser Lehrveranstaltung nicht supported wird.

Laden Sie den Quellcode ebenfalls in die Cloud-Shell herunter mittels

```
curl -sL http://bit.ly/smparcermultimodule | tar xvfz - -C ~/sm/parcer
```

Sie können jetzt die heruntergeladenen Dateien direkt in der Cloud-Shell mittels des (recht einfach gehaltenen) Editors über den Button editieren (was in der Cloud-Shell i.d.R. besser klappt als über herkömmliche Editoren wie vi, emacs oder nano).

Erstellen Sie für jeden der vier Parcer-Module ein Dockerfile für ein Image mit den folgenden Eigenschaften:

- Es soll auf dem Image `openjdk:11` basieren.
- Es soll das jeweilige Build-Artefakt-Jar als `/app.jar` (im Root-Verzeichnis) beinhalten
- Beim Starten des Containers soll `java -Djava.security.egd=file:/dev/./urandom -jar /app.jar` ausgeführt werden.

CAUTION @SM-Teilnehmer: Im Praktikumstermin müssen Sie eins Ihrer vier (gleichartigen) Dockerfiles zeilenweise erläutern können.

Unresolved directive in labs/lab_google_cloud_docker_de.asciidoc - include::lab_google_cloud_docker_solutions_de.asciidoc[tag=dockerfile]

3.6. Addressvalidation-Container starten

Übertragen Sie via `gcr.io`-Registry die vier Images auf die `vminstance-01` und starten Sie dort nur einen Addressvalidation-Container. Verwenden Sie Semantic-Versioning beim Tagging der Images.

Unresolved directive in labs/lab_google_cloud_docker_de.asciidoc - include::lab_google_cloud_docker_solutions_de.asciidoc[tag=start_addressvalidation_container]

CAUTION @SM-Teilnehmer: Im Praktikumstermin müssen Sie zeigen, dass Sie erfolgreich (mittels `curl`) einen Request an den auf `vminstance-01` laufenden Addressvalidation-Container absetzen können.

Unresolved directive in labs/lab_google_cloud_docker_de.asciidoc - include::lab_google_cloud_docker_solutions_de.asciidoc[tag=curl_addressvalidation_container]

3.7. Docker-Compose

Erstellen Sie ein `docker-compose.yml`, mit dem alle Images in der Cloud-Shell mittels

```
docker-compose build
```

gebaut. Mit demselben `docker-compose.yml` sollen auch in `vminstance-01` mittels eines einzelnen Aufrufs

```
docker-compose up
```

je Image ein Container gestartet werden können.

Bedenken Sie hierbei, dass `shipment` von `addressvalidation` und `label` abhängt.

CAUTION @SM-Teilnehmer: Im Praktikumstermin müssen Sie Ihr `docker-compose.yml` zeilenweise erläutern können.

Unresolved directive in labs/lab_google_cloud_docker_de.asciidoc - include::lab_google_cloud_docker_solutions_de.asciidoc[tag=docker_compose]

3.8. Weiterentwicklung

Entwickeln Sie Parcer weiter. Beispielsweise können sie in `addressvalidation` die Addressvalidierung erweitern oder in label das Label-Template anpassen. (Letztlich ist die Fachlichkeit der Änderung nicht wichtig. Wichtig ist nur, dass testbar etwas geändert wurde.)

Ändern Sie das `docker-compose.yml` nun so, dass die von der Änderung betroffenen Images mit einer gemäß Semantic-Versioning erhöhten Versionsnummer getaggt werden.

Übertragen Sie die geänderten Images wieder mittels `gcr.io`-Registry auf die `vminstance-01` und starten Sie Parcer wieder mittels `docker-compose`.

IMPORTANT

@SM-Teilnehmer: Im Praktikumstermin müssen Sie

- die betreffenden Zeilen in `docker-compose.yml` erläutern können
- zeigen, dass die betreffenden Images in der `gcr.io`-Registry sind
- dass das Tag sinnvoll gewählt wurde (Stichwort: Semantic Versioning)
- und dass die Änderung im laufenden Parcer wirksam und sichtbar sind.

Last updated 2022-12-19 13:08:50 +0100