MapReduce job mit Java in IntelliJ IDE

1. Einleitung

IntelliJ community Edition (lizenzfrei) bzw. für Studenten auch die Ultimate Edition (Registrierung mit gültiger FH-Email-Adresse) von https://www.jetbrains.com/idea/download/?section=windows herunterladen. Für diese Übung wäre Community Edition ausreichend, für das Studium generell würde ich die Ultimate Edition empfehlen.

2. Generell

Folgende 2 Varianten bieten sich an, das Projekt zu bauen:

1. Mit Hilfe eines externen build systems (vorbereitetes build.gradle File)
2. Mit Hilfe von IntelliJ selbst als Build-System (erfordert etwas mehr Vorbereitung, ist aber vielleicht besser zu verstehen)

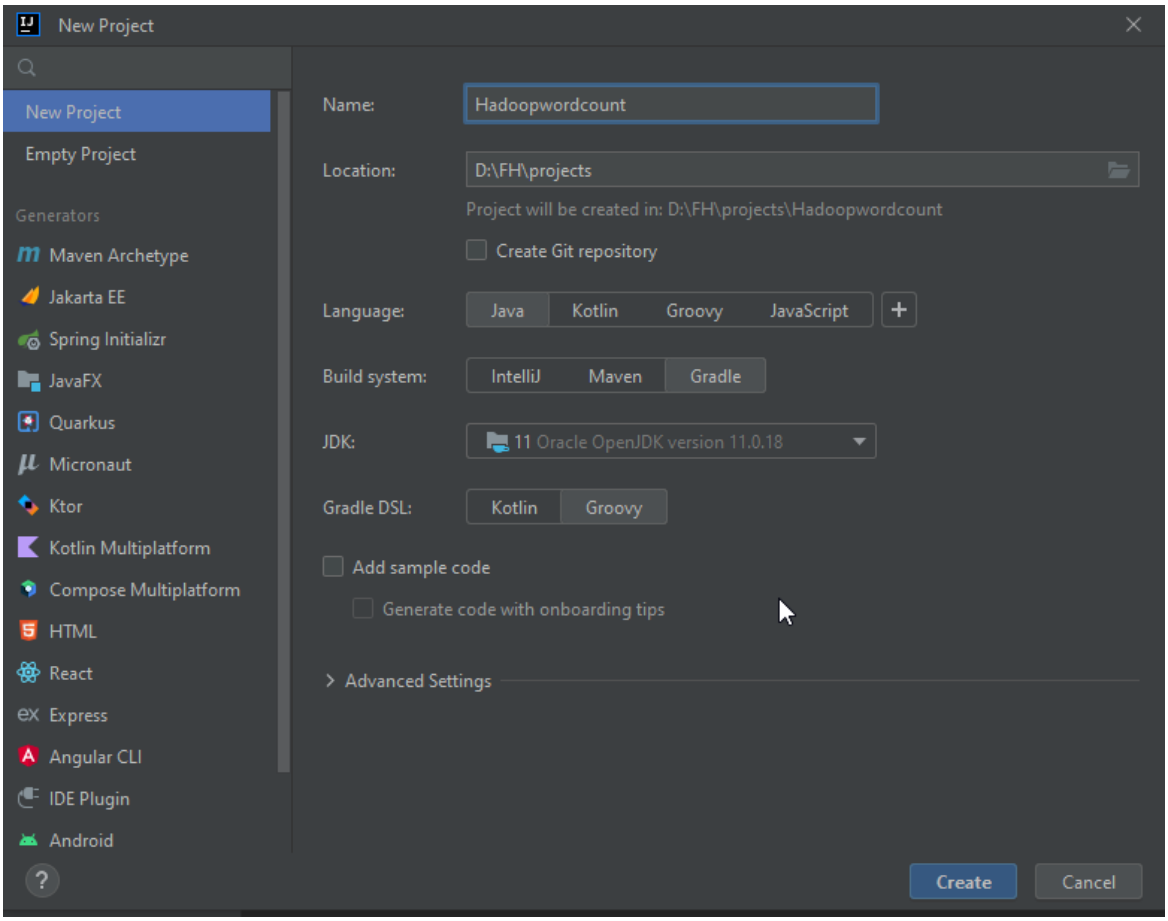
Im Folgenden wird bei Abweichungen zwischen den Varianten stets die Notation @gradle bzw. @intellij verwendet.

3. Projekt anlegen

Nach Installation ein neues Java-Projekt anlegen, passende JDK-Version auswählen.

@gradle: „Gradle“ als Buildsystem auswählen

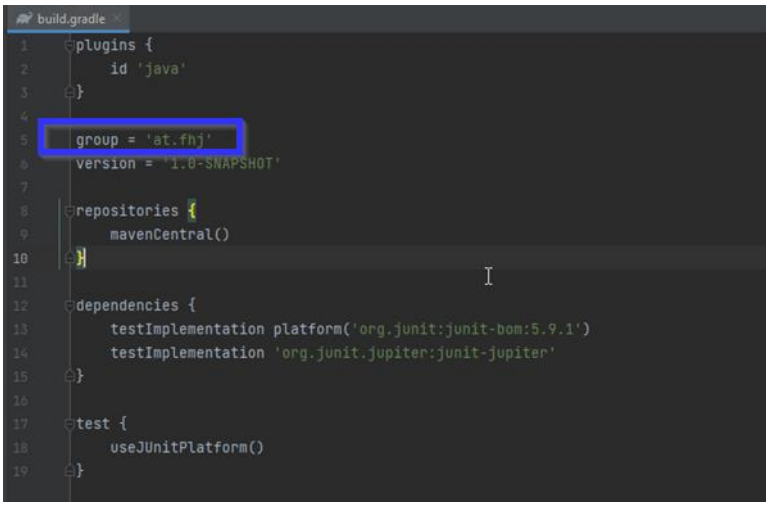
@intellij: „IntelliJ“ als Buildsystem auswählen



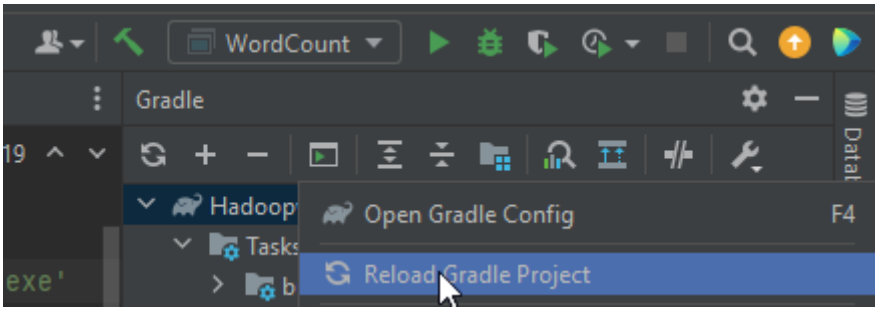
4. Buildsystem Vorbereitungen

@gradle:

das automatisch generierte build.gradle sollte wie folgt aussehen, in einem ersten Schritt ist nur die "group" ist zu ändern.



Besser jedoch das build.gradle aus dem github-Repo (https://github.com/Woberegger/BigData/tree/main/src/wordcount) abholen, dort sind auch die dependencies schon eingetragen – danach folgendes ausführen, damit das build-File neu geladen wird – es sollten darauf hin etliche Jar-Files aus dem zentralen Repo im Internet runtergeladen werden:

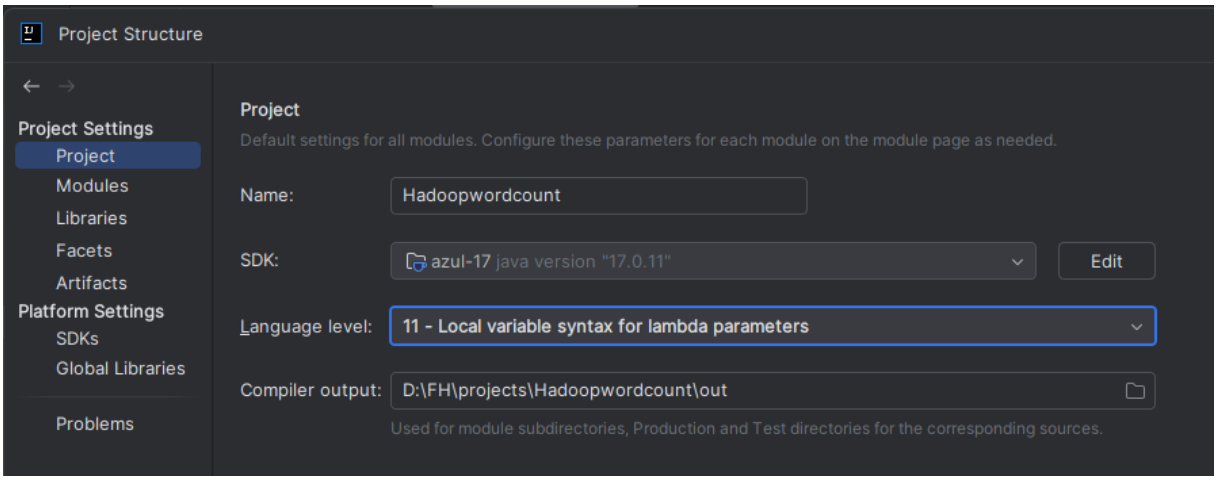


Im build.gradle ist wichtig, dass die Kompatibilität zu älterer Java-Version gesetzt ist:

```
java {
   sourceCompatibility = JavaVersion.VERSION_11
   targetCompatibility = JavaVersion.VERSION_11
}
```

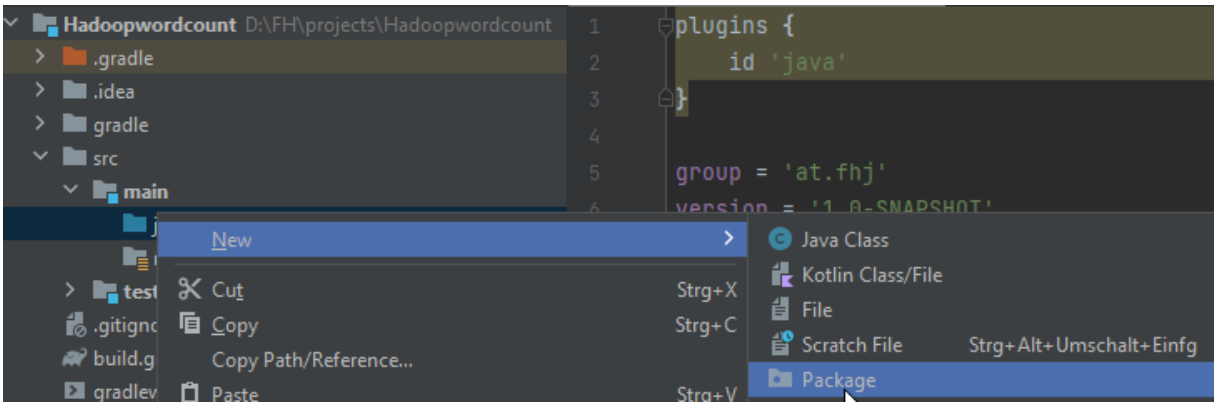
@intellij

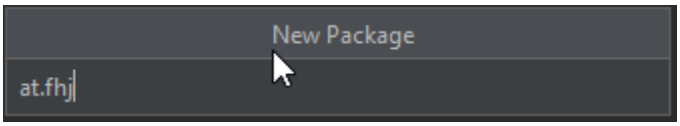
Wenn eine neuere Java-Version als 11 verwendet wird, hier die Kompatibilität zu Version 11 eintragen (bei Verwendung von gradle wird dies als „compatibility“ im build.gradle eingetragen):



5. Existierende Sourcefile vorbereiten

ein neues Package at.fhj anlegen, damit die Struktur der Defaultstruktur enspricht.





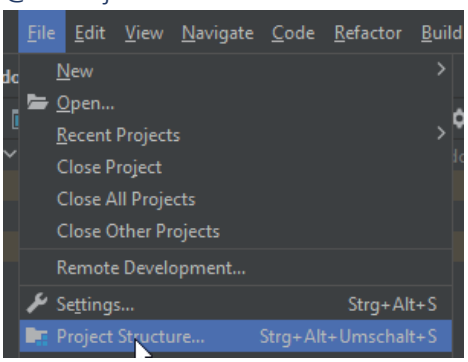
Dort die 3 Sourcefiles ablegen (am einfachsten durch Kopieren von https://github.com/Woberegger/BigData/tree/main/src/wordcount in das zuvor angelegte Package-Directory)

6. Projektstruktur anpassen

@gradle

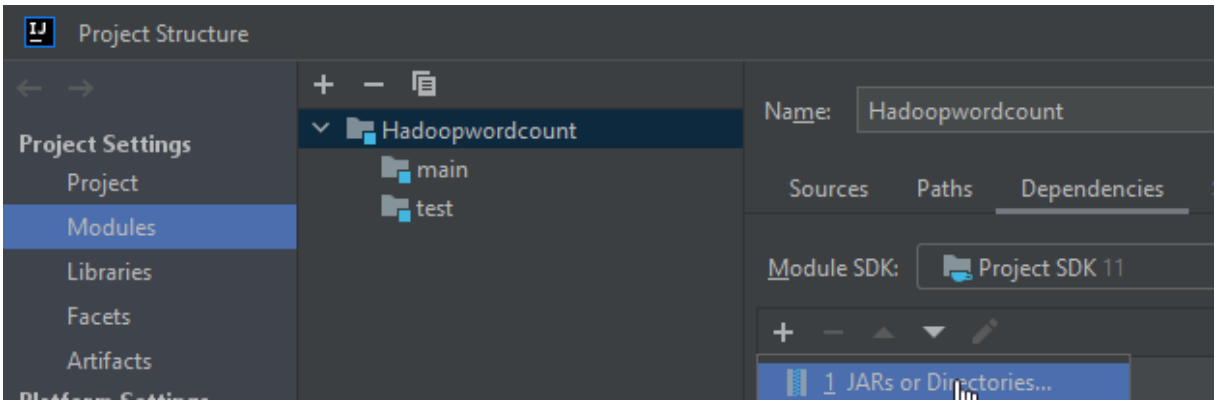
Nichts zu tun

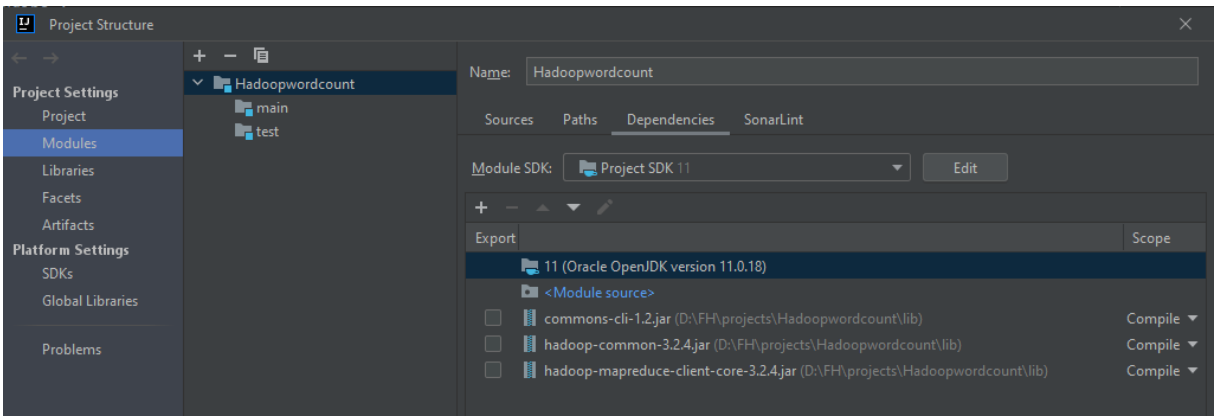
@intellij



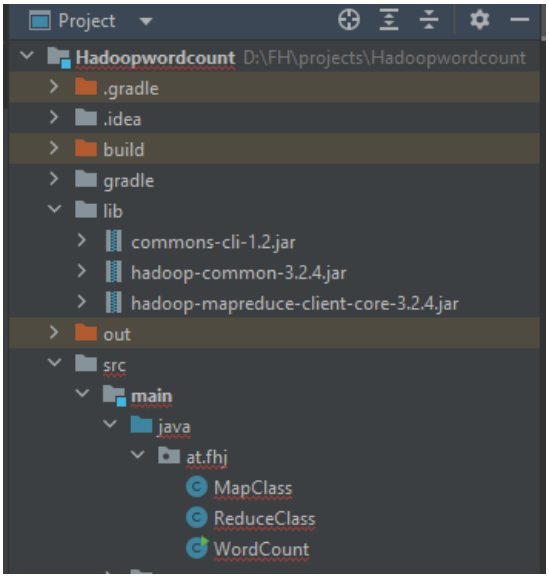
Damit die Dependencies aufgelöst werden, gibt es mehrere Möglichkeiten:

a) MavenCentral repo verwenden (siehe build.gradle)
b) Jar-Dateien zum Projekt hinzufügen (die Dateien entweder aus der E-Learning-Plattform oder noch besser vom installierten hadoop in der virtuellen Maschine – ACHTUNG: Versionen müssen zusammenpassen, es ist aber möglich, mit niedrigerer Version zu bauen, als die Version in der virtuellen Maschine):





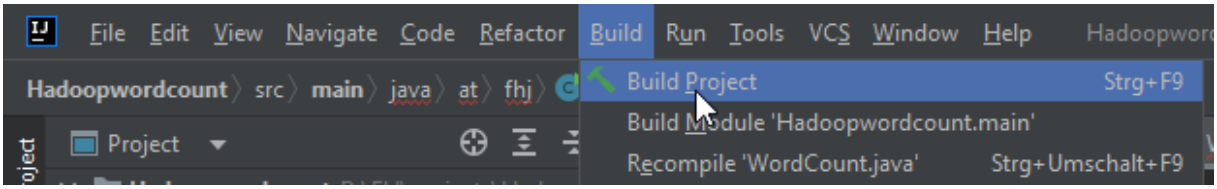
Projektstruktur, nachdem die Sourcedateien und Libraries dort abgelegt wurden



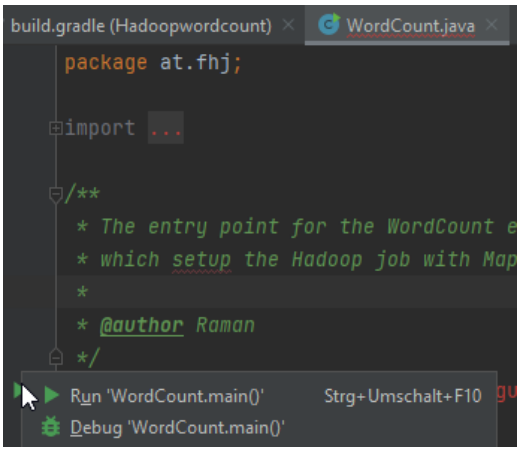
7. Project bauen

8. @intellij

Projekt bauen, d.h. kompilieren

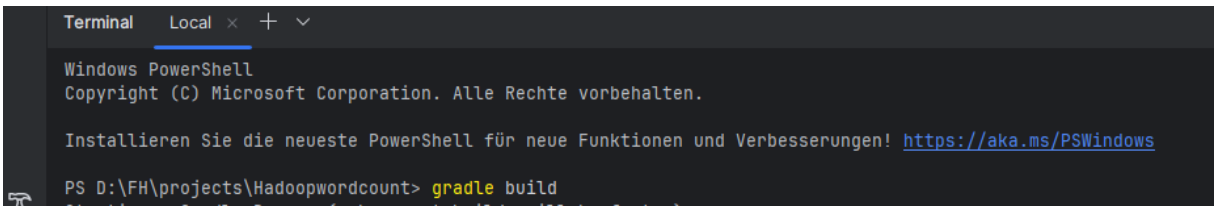


Ausführen oder Debuggen direkt bei den jeweiligen Methoden (müsste man remote debuggen auf der Zielplattform verwenden, wo DFS installiert ist), d.h. dieser Schritt ist im Normalfall zu überspringen.



@gradle

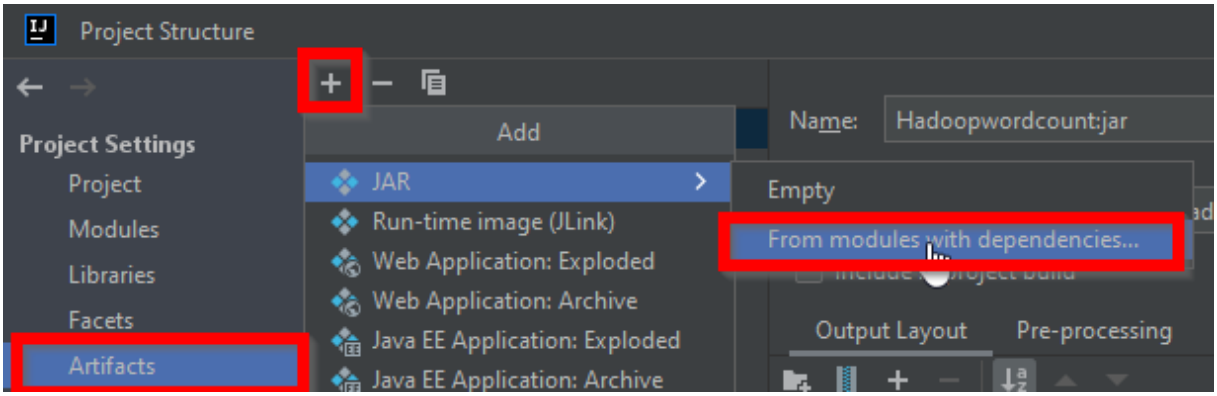
Wie bei IntelliJ bzw. optional über Kommandozeile

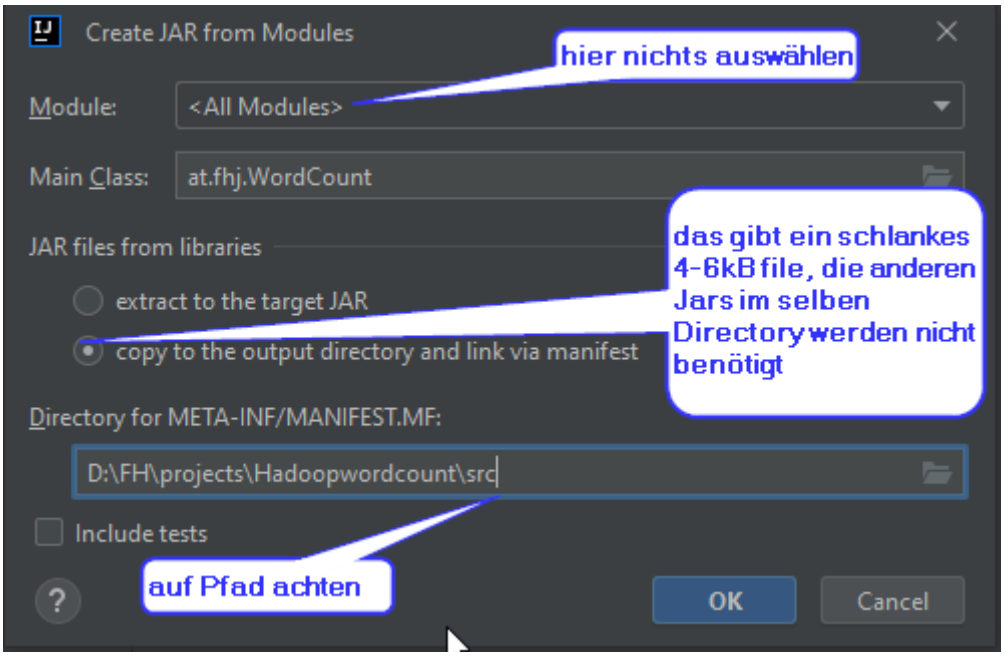


9. Jar-Datei erstellen

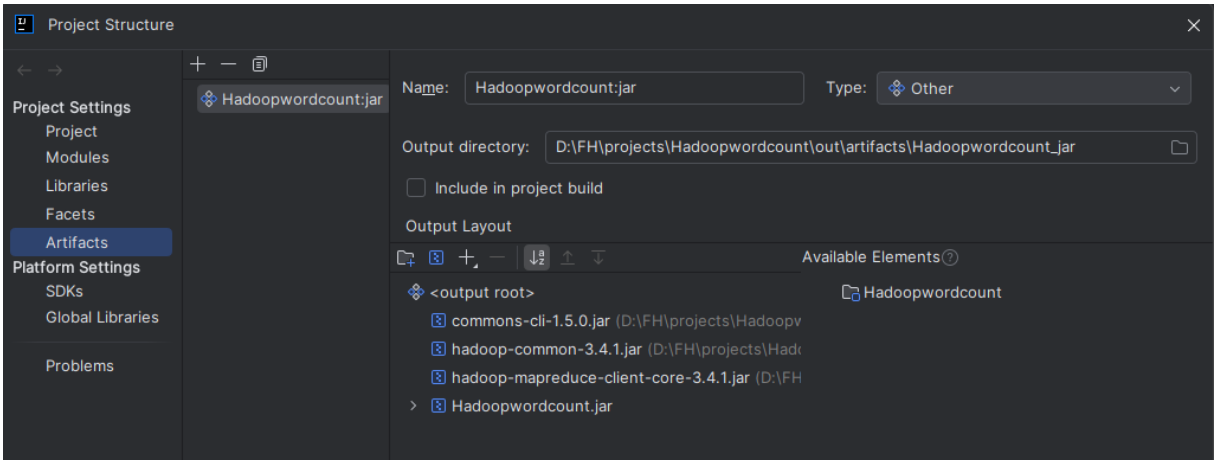
@intellij

Artifakt anlegen, d.h. über "Project structure" das Erstellen des jar-File konfigurieren und dem Jar-File mitteilen, wo der Einsprungspunkt, d.h. die Main-Klasse, zu finden ist.

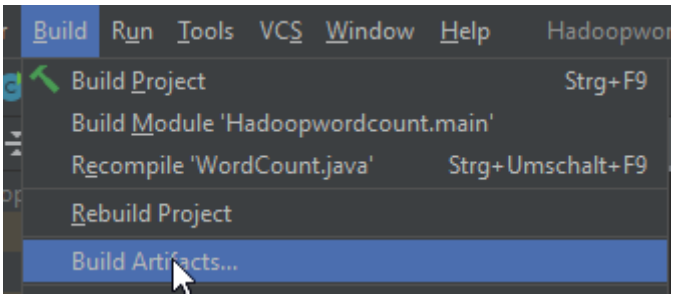




erwarteter Output danach ungefähr wie folgt, diesen mit "Apply" bestätigen (wenn mit gradle davor schon ein build angestoßen wurde, kann sein, dass hier die Liste auch bereits länger ist):



und dann kann man das Artifakt bauen:



@gradle

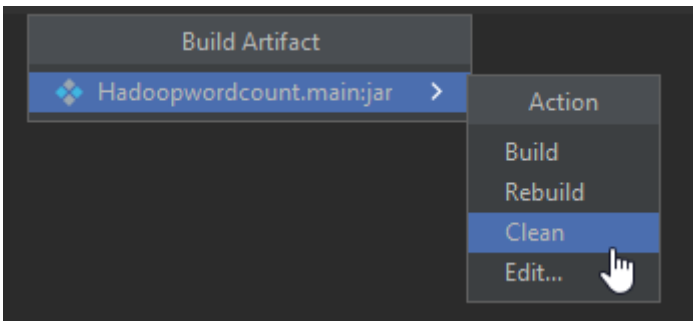
Entweder ebenfalls über die GUI oder über Kommandozeile:

gradle task Jar

10. Prüfung des erstellten Artifacts und Kopieren auf Zielplattform

Je nach verwendetem Buildsystem und deren Settings findet man das generierte Jar-File unter z.B. out/artifacts oder build/libs und beinhaltet der Dateiname eine Versionsnr. oder nicht.

Falls zuvor schon eine Version dort gebildet wurde, die nicht funktioniert (weil z.B. fehlende Dependencies), diese besser über "Clean" zuerst löschen oder noch besser gesamtes "out" Verzeichnis löschen.



Das build.gradle beinhaltet einen „deploy“ Task, dieser funktioniert jedoch nur bei Passwort-based login bzw. müsste man die Credentials und Hostnamen ändern.

Daher am besten händisch die Datei manuell kopieren als user „student“ und dann die Datei dem hduser „schenken“:

```
scp Hadoopwordcount-1.0 student@<hostname>:/tmp/Hadoopwordcount-1.0.jar
ssh student@<hostname>
sudo chown hduser:hadoop /tmp/Hadoopwordcount-1.0.jar
su - hduser
mv /tmp/Hadoopwordcount-1.0.jar ~/Hadoopwordcount.jar
```

11. Test des generierten jar-Files

Hadoop starten und Test aufrufen mit dem zuvor generierten und auf die Zielplattform kopierten Jar-File.

Job ausführen

1.) Input Ordner und Beispiel Textdatei im hdfs anlegen
2.) Job ausführen (Vorsicht Pfade sind ggf. anzupassen!)

```
su - hduser
start-dfs.sh
start-yarn.sh
hdfs dfs -mkdir /input
hdfs dfs -mkdir /output
hdfs dfs -put ~/BigData/data/Bibel.txt /input/
OutputDir=/output/Bibel
# folgendes für jeden Versuch neu ausführen
hdfs dfs -rm -R $OutputDir
hadoop jar Hadoopwordcount.jar /input/Bibel.txt $OutputDir
```

Bei Problemen die Jar-Datei überprüfen ("jar -tvf" funktioniert meines Wissens nur unter Linux), sie muss zumindest folgende Dateien beinhalten:

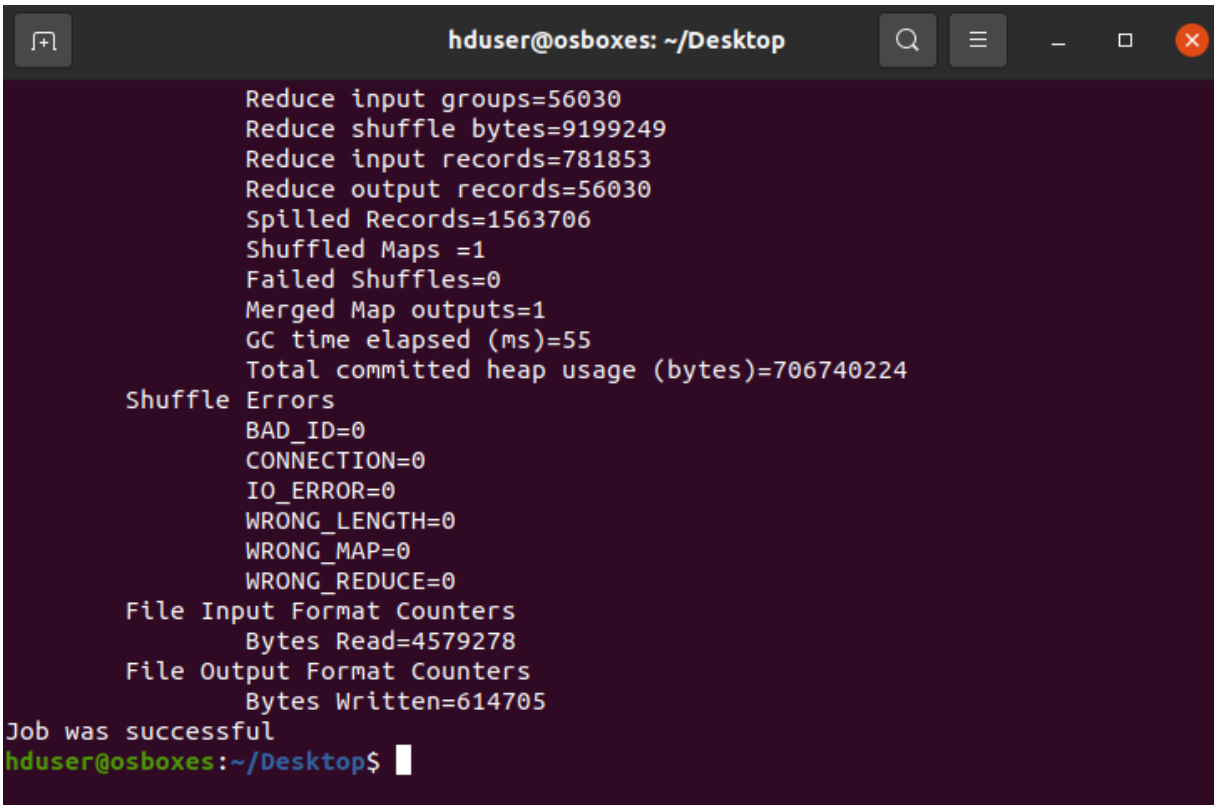
```
jar -tvf Hadoopwordcount.jar | awk '{ print $8 }'
META-INF/MANIFEST.MF
```

```
at/fhj/MapClass.class
at/fhj/ReduceClass.class
at/fhj/WordCount.class
```

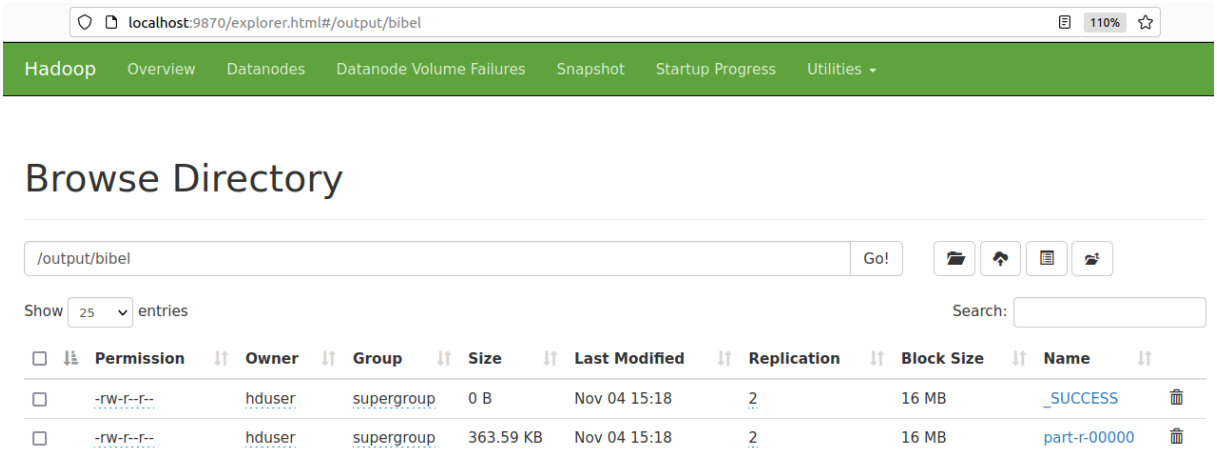
Der Inhalt der Datei MANIFEST.MF sollte wie folgt sein (optional stehen dahinter noch ClassPaths):

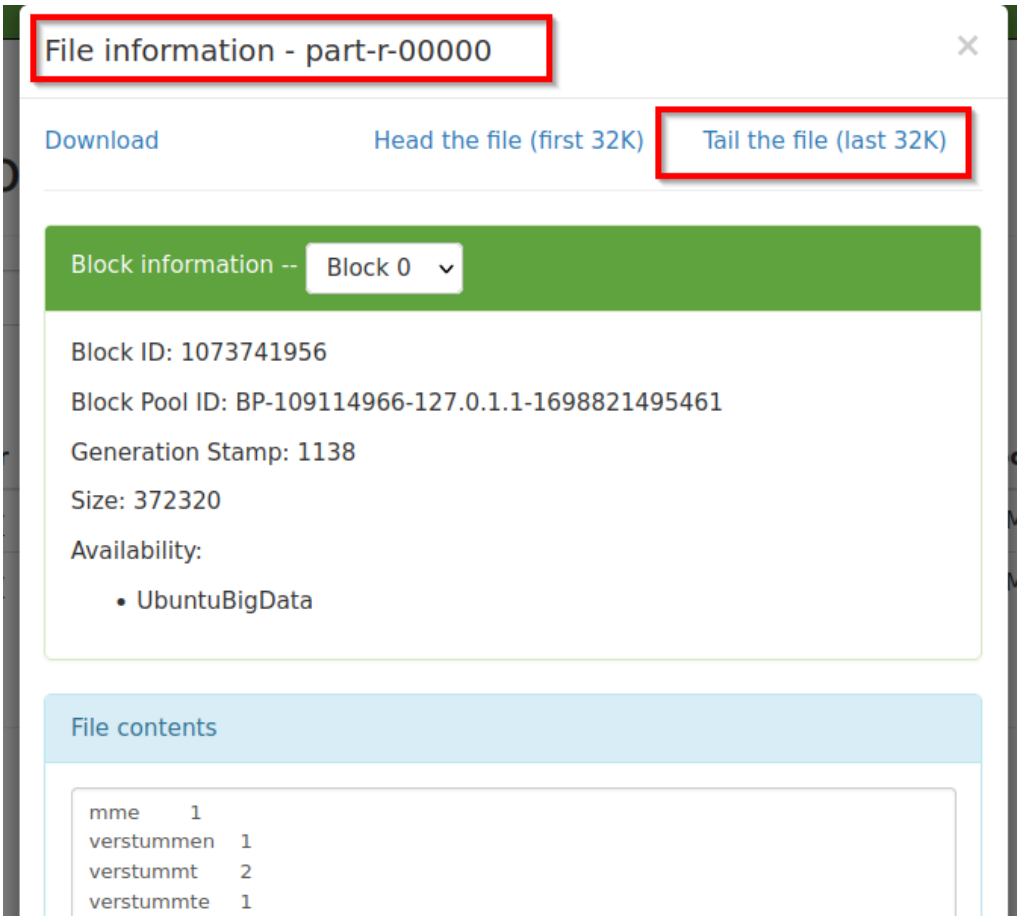
```
jar -xf Hadoopwordcount.jar META-INF/MANIFEST.MF
cat META-INF/MANIFEST.MF
Manifest-Version: 1.0
Main-Class: at.fhj.WordCount
```

Erwarteter Output nach erfolgreichem MapReduce Job wie folgt:



Im angegebenen Output-Verzeichnis befinden sich dann 2 Dateien, eine leere Datei "_SUCCESS" und eine Datei mit dem Ergebnis des Jobs.





Das “Tail the last file” funktioniert üblicherweise nicht über die GUI, daher besser über Kommandozeile:

```
hdfs dfs -tail /output/Bibel/part-r-00000
```

Wichtig: wenn der Job erneut gestartet werden soll, muss zuvor das Output-Verzeichnis gelöscht werden (sowohl wenn man Output ins lokale Dateisystem als auch wenn man dies in hdfs schreibt)!