Installation

Damit wir mit GraalVM starten zu können, müssen wir es zunächst installieren. Dafür besuchen wir die Website https://www.graalvm.org/downloads/. Hier sehen wir schon die beiden unterschiedlichen Versionen. Für die Übung reicht die Community Edition. Diese ist kostenlos.

Für Unix-Systeme kann direkt der folgende Befehl verwendet werden.

bash <(curl -sL https://get.graalvm.org/jdk)

Dieser installiert alles direkt und man folgt den Anweisungen aus dem Skript.

Für Windows-Nutzer gibt es keinen Installer, deswegen machen wir das einfach schnell manuell. Installationsanleitungen für die einzelnen Systeme sind in der offiziellen Dokumentation zu finden:
https://www.graalvm.org/latest/docs/getting-started/windows/
https://www.graalvm.org/latest/docs/getting-started/linux/
https://www.graalvm.org/latest/docs/getting-started/macos/

In den Übungen machen wir das auf einer Ubuntu Desktop 22.04 Installation bzw. in einem GitHub Codespace. Andere Systeme sind aber analog und unterscheiden sich nur wenig.

Wir laden uns das entsprechende Release von GitHub https://github.com/graalvm/graalvm-ce-builds/releases/tag/vm-22.3.0 herunter:

wget https://github.com/graalvm/graalvm-ce-builds/releases/download/vm-22.3.0/graalvm-ce-java17-linux-amd64-22.3.0.tar.gz

Anschließend entpacken wir das Verzeichnis an einem Ort, an dem es ausgeführt werden kann.

tar -xvzf graal.tar.gz

Im Zweifel noch mal die Rechte setzen. Hier hab ich es in /opt/graal abgelegt.

chmod +x -R /opt/graal

Anschließend müssen wir die Umgebungsvariablen setzen. Auf Linux z. B. über die folgenden Befehle:

export JAVA_HOME="/opt/graal"
export GRAALVM_HOME="/opt/graal"
export PATH="/opt/graal:$PATH"

Diese beiden Home-Variablen werden verwendet, damit Tools, die Java bzw. Graal benötigen, diese finden. Mit der PATH-Variable machen wir anschließend die Kommandotools im Terminal verfügbar.

Entsprechend können die Variablen auch in /etc/environment bzw. in die .bashrc gesetzt werden, damit diese auch nach einem Neustart noch gültig sind.

In Codespaces muss man Variablen etwas anders konfigurieren, weil es eine Remote-Verbindung ist. Hier nehmen wir die folgende Konfiguration, die das für uns macht. Diese liegt in „.devcontainer/devcontainer.json“. Nach dem Ändern, einmal die Umgebung neu bauen lassen (F1 > „Codespace: Full Rebuild“.

```
{
    "remoteEnv": {
        "PATH": "/opt/graal/bin:${localEnv:PATH}",
        "GRAALVM_HOME": "/opt/graal",
        "JAVA_HOME": "/opt/graal"
    },
    "image": "ubuntu"
}
```

Und fertig aufgesetzt ist alles. Probieren wir das mal aus und schauen uns die installierte Version an:

java -version

Hier sollte etwas mit „OpenJDK“ und „GraalVM“ zu sehen sein. Mit „which java” sehen wir auch, dass der Pfad auf unsere Installation zeigt.

Lift-and-Shift Ansatz

Probieren wir Graal direkt mal aus und wir schreiben ein kleines Programm. Wenn wir Java ohne GraalVM verwenden würden, dann sähe ein einfaches HelloWorld so aus:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello Java!");
    }
}
```

Dieses würden wir dann im einfachsten Fall wie folgt compilieren und ausführen:

javac HelloWorld.java
java HelloWorld

Wenn wir das in Graal ausführen, sehen wir, dass es genauso auch hier geht. Das Schöne an Graal ist, dass es kompatibel zu einer Standard-Java-VM ist, wir aber trotzdem die Vorteile von Graal bekommen. Das macht es super einfach einen Lift-and-Shift Ansatz zu fahren. Das heißt, wir können einfach die Runtime auf Graal umstellen, ohne dass wir Änderungen am Quellcode durchführen müssen.