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 direct. Skripte aus dem Internet einfach auszuführen, sollten wir aber vermeiden. 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"

Diese beiden werden verwendet, damit Tools, die auf Java basieren, dieses findet und auch Tools, die speziell auf Graal aufsetzen (später sehen wir noch das Native Image Building) auch dieses findet. Anschließend müssen wir noch die Kommandotools verfügbar machen:

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

Entsprechend kann das auch in /etc/environment bzw. in die .profile.rc, dann ist das auch dauerhaft gesetzt. Ich pack es in die /etc/environment. Eventuell muss aber die Umgebung neu geladen werden.

In Codespaces ist es etwas anders, 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. (TODO: Hier sind die Pfade noch falsch)

```
{
    "remoteEnv": {
        "PATH": "/workspaces/codespaces-blank/graalvm-ce-java17-
22.3.0/bin:${localEnv:PATH}",
        "GRAALVM_HOME": "/workspaces/codespaces-blank/graalvm-ce-java17-
22.3.0",
        "JAVA_HOME": "/workspaces/codespaces-blank/graalvm-ce-java17-22.3.0"
    },
    "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.

Lift-and-Shift Ansatz

Probieren wir Graal direkt mal aus und probieren uns an einem kleinen Programm. Wenn wir Java 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 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 beliebige Änderungen am Quellcode durchführen müssen.