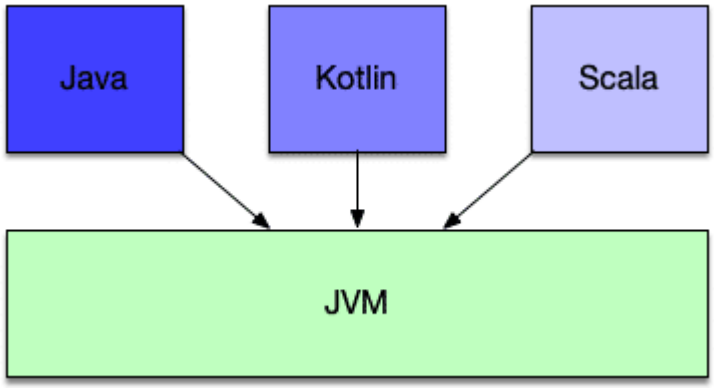
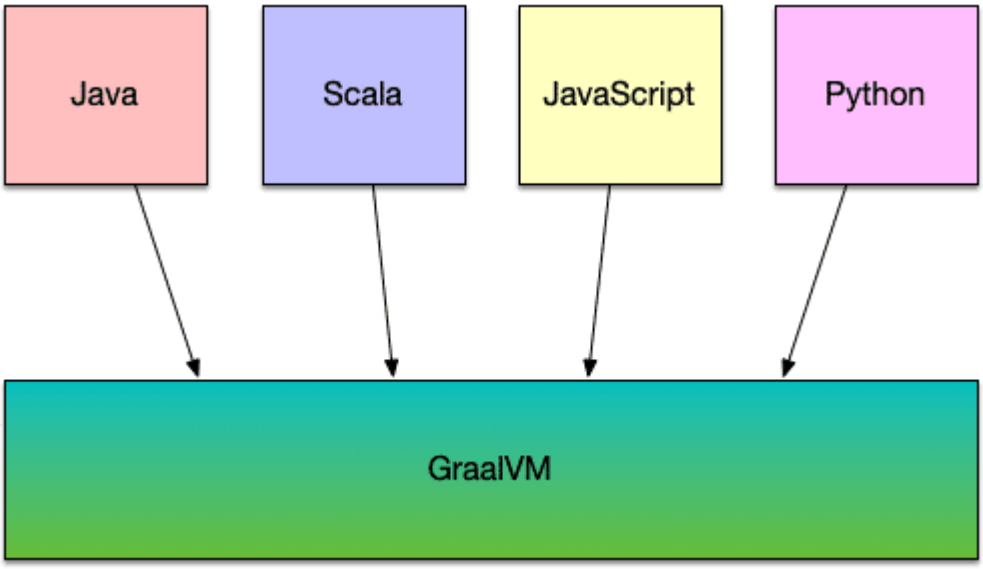
GraalVM es una Maquina Virtual que es capaz de ejecutar código de diversos lenguajes de programación. Esto en principio nos puede sonar un poco raro porque ya existen maquinas virtuales que ejecutan código . Sin ir más lejos la propia maquina virtual de Java JVM es capaz de ejecutar código Java y otros lenguajes basados en su propia plataforma.



Sin embargo cada vez aparecen más opciones y una de las que hoy en día empieza a destacar es Graal.

GraalVM y sus ventajas

La ventaja de GraalVM como maquina virtual es que puede ejecutar código de diferentes lenguajes de programación , no solo de Java sino que puede ejecutar C++ , Python o JavaScript. Algo que abre las puertas a muchas cosas.



Es cierto que los desarrolladores Java estamos acostumbrados a usar la maquina virtual para ejecutar código en otros lenguajes cómo puede ser Kotlin o Scala. . Ahora bien siempre estamos ubicados en nuestro propio entorno es decir no podemos ejecutar código de Python o de JavaScript de una forma tan directa.

Graal es políglota

Con GraalVM las cosas cambian ya que esta máquina virtual es capaz de ejecutar código de muchos lenguajes de programación concretamente destacan aquellos bajo el paraguas de Node.js o de OpenJDK.

Esto la hace especialmente interesante ya que hoy por hoy muchos de los desarrolladores Java trabajamos como lenguaje complementario con JavaScript y plataformas como Node y viceversa muchos desarrolladores de Node trabajan con Java. El hecho de añadir a esta lista de lenguajes la capacidad de ejecutar código de python o código de C++ la hacen todavía más interesante ya que por ejemplo el mundo de BigData esta plagado de gente que programa con Python.

Un par de Ejemplos

Vamos a construir un par de ejemplos con GraalVM y ejecutarlos con un contenedor de Docker. Para que veamos la versatilidad que tiene el uso de esta Maquina Virtual. Para ello partiremos de un proyecto de servicio REST con Spring Boot . Veamos el contenido del pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.2.2.RELEASE</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.arquitecturajava</groupId>
	<artifactId>rest</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>rest</name>
	<description>Demo project for Spring Boot</description>

	<properties>
		<java.version>1.8</java.version>
	</properties>

	<dependencies>
		<dependency>
```

```
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-
web</artifactId>
                </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-
test</artifactId>
                        <scope>test</scope>
                        <exclusions>
                                <exclusion>
<groupId>org.junit.vintage</groupId>
                                        <artifactId>junit-vintage-
engine</artifactId>
                                </exclusion>
                        </exclusions>
                </dependency>
        </dependencies>

        <build>
                <plugins>
                        <plugin>
<groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-
plugin</artifactId>
                        </plugin>
                </plugins>
        </build>

</project>
```

Una vez tenemos construido el pom.xml es momento de definir un servicio REST de holamundo.

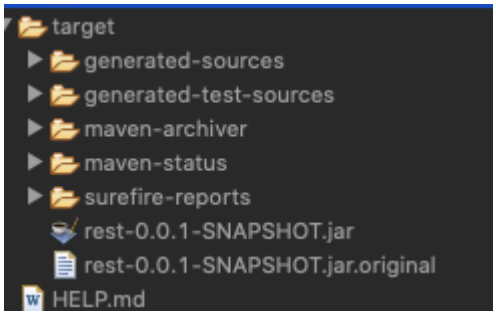
```
package com.arquitecturajava.rest;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HolaService {

	@RequestMapping
	public String hola() {
		return "hola";
	}
}
```

Hechas estas dos cosas el siguiente paso es ejecutar mvn package y empaquetar la aplicación con Spring Boot como FatJAR.



Usando Docker

En este caso se trata del fichero rest-0.0.1.jar lo podemos copiar en cualquier carpeta. Lo que nos queda es tener instalado docker y construir un DockerFile. Recordemos que un DockerFile lo que hace es prepara los comandos elementales para poder lanzar un

contenedor docker con la aplicación que nosotros necesitemos copiando los ficheros y ejecutando las ordenes que hagan falta ,veamos su contenido.

```
FROM oracle/graalvm-ce
COPY miproyecto /home
CMD java -jar /home/rest-0.0.1-SNAPSHOT.jar RestApplication
EXPOSE 8080
```

En este caso vamos a ver paso a paso que contiene el fichero de docker.

1. En primer lugar esta el comando FROM que lo que hace es se descarga una imagen de la maquina virtual de GraalVM convertida ya en contenedor
2. El segundo paso que realizo es usar el comando COPY para copiar mi FatJAR en la carpeta /home del contenedor que tiene una estructura Linux
3. El tercer paso es abrir un terminal con el comando CMD y ejecutar el jar con:
   java -jar /home/miproyecto/rest-0.0.1-SNAPSHOT.jar RestApplication

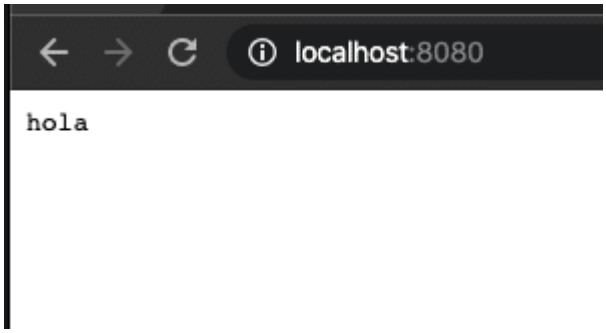
4. Finalmente usamos el comando EXPOSE para exponer el puerto 8080 del contenedor en el sistema operativo principal.
   Una vez disponemos del fichero docker file es suficiente con hacer

   ```
   docker build . -t springboot
   ```

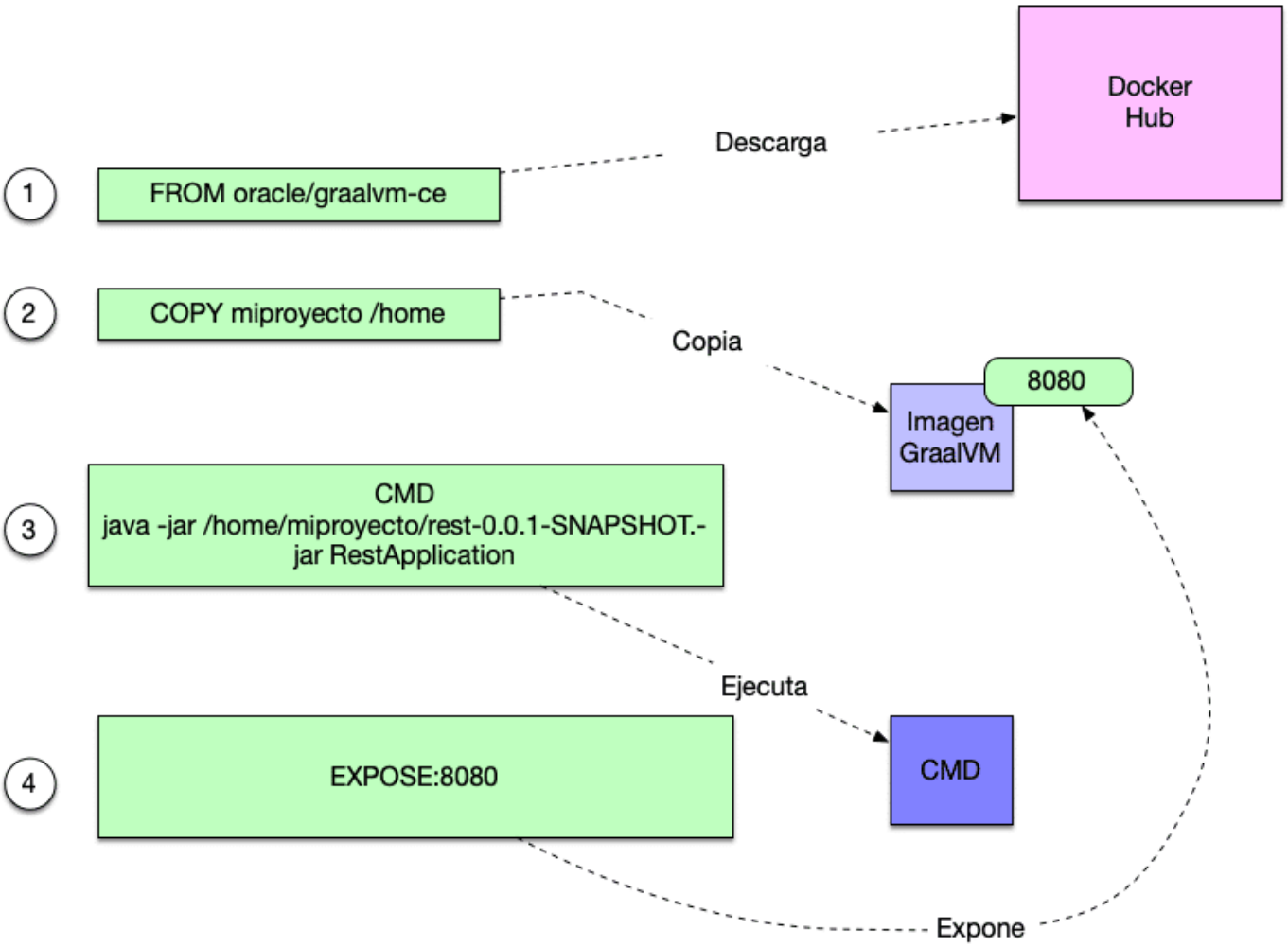
   Construyendo el contenedor que despliega la máquina con nuestra aplicación dentro de ella. Por último ejecutaremos

   ```
   docker run -p:8080:8080 springboot
   ```

   El cual lanza el contenedor y publica la información en el puerto 8080 mapeandolo. Si cargamos la página de localhost en nuestro equipo podremos ver la aplicación de SpringBoot en funcionamiento.



Acabamos de desplegar nuestra primera aplicación de Spring Boot sobre GraalVM usando docker. El diagrama clarifica un poco la estructura del docker file.



Es momento de hacer algo similar con Node.js y ver el resultado:

GraalVM y Node.js

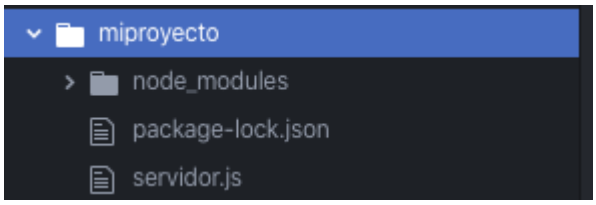
Vamos a construir el mismo ejemplo con Node.js .Al tratarse de JavaScript tendremos que construir una aplicación con el framework Express.js. En este caso lo que vamos a hacer es usar el ejemplo de Hello world.

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('hola express desde graal');
});

app.listen(8081, function () {
  console.log('Example app listening on port 8081!');
});
```

Acabamos de construir nuestra aplicación de node en el puerto 8081 y disponemos de esta estructura de carpetas en la cual hemos ejecutado npm install express para que nos instale express.js como framework.



Es momento de disponer de un dockerfile que nos despliegue nuestra aplicación como contenedor. En este caso lo curioso es que aunque estemos trabajando con otro lenguaje el contenedor docker que usamos es el mismo.

FROM oracle/graalvm-ce
COPY miproyecto /home

```
CMD node /home/miproyecto/servidor.js
EXPOSE 8081
```

En este caso copiamos el contenido que tenemos en la carpeta de miproyecto a la carpeta /home del contenedor. Recordemos que el contenido son ficheros de Javascript. Ejecutamos y exponemos la aplicación en el puerto correspondiente. Para ello necesitaremos el siguiente dockerfile

```
FROM oracle/graalvm-ce
COPY miproyecto/* /home/miproyecto/
WORKDIR /home/miproyecto
COPY /miproyecto/package*.json ./
RUN npm install
CMD [ “node”, “servidor.js” ]
EXPOSE 8081
```

En este caso estamos realizando una operación similar a la anterior pero con unos comandos mas concretos para node y express.js que es el framework MVC de Node. Ejecutamos

```
docker build -t minode
```

Acto seguido

```
docker run -p:8081:8081 minode
```

Es momento de acceder a la aplicación en nuestro equipo a través del puerto 8081.



Acabamos de publicar dos aplicaciones en tecnologías diferentes utilizando el mismo sistema de maquina virtual o interprete a través de GraalVM.

Otros artículos relacionados

- ¿Que es Spring Boot?
- Spring Boot JPA
- Spring Boot Thymeleaf
- https://hub.docker.com/r/oracle/graalvm-ce/