Prerequisites

Table of Contents

Some of this content has been taken from Official Quarkus Workshop

cURL

To invoke the REST Web Services described in this workshop, we often use cURL. [1: cURL https://curl.haxx.se] cURL is a command line tool and library to do reliable data transfers with various protocols, including HTTP. It is free, open source (available under the MIT Licence) and has been ported to several operating systems.

Installing cURL

On windows, it's probably best to get it from the source or from chocolatey

If you are on Mac OS X and if you have installed Homebrew, then installing cURL is just a matter of a single command. [2: Homebrew https://brew.sh] Open your terminal and install cURL with the following command:

```
$ brew install curl
```

Checking for cURL Installation

Once installed, check for cURL by running `curl --version` in the terminal. It should display cURL version:

```
$ curl --version
curl 7.54.0 (x86_64-apple-darwin17.0) libcurl/7.54.0 LibreSSL/2.0.20 zlib/1.2.11
nghttp2/1.24.0
Protocols: dict file ftp ftps gopher http https imap imaps ldap ldaps pop3 pop3s rtsp
smb smbs smtp smtps telnet tftp
Features: AsynchDNS IPv6 Largefile GSS-API Kerberos SPNEGO NTLM NTLM_WB SSL libz HTTP2
UnixSockets HTTPS-proxy
```

Some cURL Commands

cURL is a command line utility where you can use several parameters and options to invoke URLs. You invoke `curl` with zero, one or several command-line options to accompany the URL (or set of URLs) you want the transfer to be about. cURL supports over two hundred different options and I would recommend reading the documentation for more help. [3: cURL commands https://ec.haxx.se/cmdline.html] To get some help on the commands and options you can type, use the following command:

```
$ curl --help

Usage: curl [options...] <url>
```

You can also opt to use `curl --manual` which will output the entire man page for cURL plus an appended tutorial for the most common use cases.

Here are some commands that you will be using to invoke the RESTful web service examples in this workshop.

- `curl http://localhost:8083/api/heroes/hello`: HTTP GET on a given URL.
- `curl -X GET http://localhost:8083/api/heroes/hello`: Same effect as the previous command, an HTTP GET on a given URL.
- `curl -v http://localhost:8083/api/heroes/hello`: HTTP GET on a given URL with verbose mode on.
- `curl -H 'Content-Type: application/json' http://localhost:8083/api/heroes/hello`: HTTP GET on a given URL passing the JSON Content Type in the HTTP Header.
- `curl -X DELETE http://localhost:8083/api/heroes/1`: HTTP DELETE on a given URL.

Formatting the cURL JSON Output

Very often when using cURL to invoke a RESTful web service, we get some JSON payload. cURL does not format this JSON, so you will get a flat String such as:

```
$ curl http://localhost:8083/api/heroes
[{"id":"1","name":"Chewbacca","level":"14"},{"id":"2","name":"Wonder
Woman","level":"15"},{"id":"3","name":"Anakin Skywalker","level":"8"}]
```

But what we really want is to format the JSON payload so it is easier to read. For that, there is a neat utility tool called jq that we could use. jq is a tool for processing JSON inputs, applying the given filter to its JSON text inputs and producing the filter's results as JSON on standard output. [4: jq https://stedolan.github.io/jq] You can install it on Mac OSX with a simple `brew install jq`; on Windows you can either get it from the source or chocolatey once again. Once installed, it's just a matter of piping the cURL output to jq like this:

```
$ curl http://localhost:8083/api/heroes | jq
[
  {
    "id": "1",
    "name": "Chewbacca",
    "lastName": "14"
  },
  {
    "id": "2",
    "name": "Wonder Woman",
    "lastName": "15"
  },
  {
    "id": "3",
    "name": "Anakin Skywalker",
    "lastName": "8"
  }
]
```

Installing SDKMAN

Installing SDKMAN! [5: SDKMAN https://sdkman.io] on UNIX-like platforms is as easy as ever. SDKMAN! installs smoothly on Mac OSX, Linux, WLS, Cygwin, Solaris and FreeBSD. We also support Bash and ZSH shells. Simply open a new terminal and enter:

```
$ curl -s "https://get.sdkman.io" | bash
```

Follow the instructions on-screen to complete installation. Next, open a new terminal or enter:

```
$ source "$HOME/.sdkman/bin/sdkman-init.sh"
```

Lastly, run the following code snippet to ensure that installation succeeded:

```
$ sdk version
```

If all went well, the version should be displayed. Something like:

```
  sdkman 5.0.0+51
```

Installing Java

Installing an AdoptOpenJDK.net JDK version 11 :

```
sdk install java 11.0.7.hs-adpt
```

You are prompted :

```
Do you want java 11.0.7.hs-adpt to be set as default? (Y/n): y
```

Type : y

Installing GraalVM

GraalVM is an extension of the *Java Virtual Machine* (JVM) to support more languages and several execution modes. [6: GraalVM https://www.graalvm.org] It supports a large set of languages: Java of course, other JVM-based languages (such as Groovy, Kotlin etc.) but also JavaScript, Ruby, Python, R and C/C++. It includes a new high performance Java compiler, itself called *Graal*, which can be used in a *Just-In-Time* (JIT) configuration on the HotSpot VM, or in an *Ahead-Of-Time* (AOT) configuration on the Substrate VM. [7: SubstrateVM https://github.com/oracle/graal/tree/master/substratevm] One objective of Graal is to improve the performance of Java virtual machine-based languages to match the performance of native languages.

Prerequisites for GraalVM

On Linux, you need GCC, and the glibc and zlib headers. Examples for common distributions:

```
# dnf (rpm-based)
sudo dnf install gcc glibc-devel zlib-devel
# Debian-based distributions:
sudo apt-get install build-essential libz-dev zlib1g-dev
```

On MacOS X, XyCode provides the required dependencies to build native executables:

```
xcode-select --install
```

Installing GraalVM 20 :

```
sdk install java 20.1.0.r11-grl
```

You are prompted with:

```
Do you want java 20.1.0.r11-grl to be set as default? (Y/n): n
```

Install native image in order to allow native compilation on your machine:

```
~/.sdkman/candidates/java/20.1.0.r11-grl/bin/gu install native-image
```

And set the GRAALVM_HOME variable:

```
export GRAALVM_HOME=~/.sdkman/candidates/java/20.1.0.r11-grl/
```

Installing Maven

All the examples of this workshop are built and tested using Maven. [8: Maven https://maven.apache.org] Maven offers a building solution, shared libraries, and a plugin platform for your projects, allowing you to do quality control, documentation, teamwork and so forth. Based on the "convention over configuration" principle, Maven brings a standard project description and a number of conventions such as a standard directory structure. With an extensible architecture based on plugins, Maven can offer many different services.

Installing Maven

The examples of this workshop have been developed with Apache Maven 3.6.x. Once you have installed JDK {jdk-version}, make sure the `JAVA_HOME` environment variable is set. Then, download Maven from http://maven.apache.org/, unzip the file on your hard drive, and add the `apache-maven/bin` directory to your `PATH` variable. More details about the installation process is available on https://maven.apache.org/install.html.

But of course, if you are on Mac OS X and use Homebrew, just install Maven with the following command:

```
$ sdkman install maven 3.6.3
```

You are prompted :

```
Do you want maven 3.6.3 to be set as default? (Y/n): y
```

Type : y

Checking for Maven Installation

Once you've got Maven installed, open a command line and enter `mvn -version` to validate your installation. Maven should print its version and the JDK version it uses (which is handy as you might have different JDK versions installed on the same machine).

```
$ mvn -version
Apache Maven 3.6.2
Maven home: /usr/local/Cellar/maven/3.6.2/libexec
Java version: 1.8.0_201, vendor: Oracle Corporation
OS name: "mac os x", version: "10.14.2", arch: "x86_64", family: "mac"
```

Be aware that Maven needs Internet access so it can download plugins and project dependencies from the Maven Central and/or other remote repositories. [9: Maven Central https://search.maven.org]

Some Maven Commands

Maven is a command line utility where you can use several parameters and options to build, test or package your code. To get some help on the commands you can type, use the following command:

```
$ mvn --help

usage: mvn [options] [<goal(s)>] [<phase(s)>]
```

Here are some commands that you will be using to run the examples in the workshop. Each invoke a different phase of the project life cycle (clean, compile, install etc.) and use the `pom.xml` to download libraries, customise the compilation, or extend some behaviours with plugins:

- `mvn clean`: Deletes all generated files (compiled classes, generated code, artifacts etc.).
- `mvn compile`: Compiles the main Java classes.
- `mvn test-compile`: Compiles the test classes.
- `mvn test`: Compiles the main Java classes as well as the test classes and executes the tests.
- `mvn package`: Compiles, executes the tests and packages the code into an archive.
- `mvn install`: Builds and installs the artifacts in your local repository.
- `mvn clean install`: Cleans and installs (note that you can add several commands separated by a space, like `mvn clean compile test`).

Docker

Docker is a set of platform-as-a-service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels.

Installing Docker

Our infrastructure is going to use Docker to ease the installation of the different technical services (database, monitoring...). So for this, we need to install `docker` and `docker-compose` Installation

instructions are available on the following page:

- Mac OS X - https://docs.docker.com/docker-for-mac/install/ (version 18.03+)
- Windows - https://docs.docker.com/docker-for-windows/install/ (version 18.03+)
- CentOS - https://docs.docker.com/install/linux/docker-ce/centos/
- Debian - https://docs.docker.com/install/linux/docker-ce/debian/
- Fedora - https://docs.docker.com/install/linux/docker-ce/fedora/
- Ubuntu - https://docs.docker.com/install/linux/docker-ce/ubuntu/

On Linux, don't forget the post-execution steps described on https://docs.docker.com/install/linux/linux-postinstall/.

Checking for Docker Installation

Once installed, check that both `docker` and `docker-compose` are available in your `PATH`:

```
$ docker version
Client: Docker Engine - Community
Version:           19.03.2
API version:       1.40
Go version:        go1.12.8
Git commit:        6a30dfc
Built:             Thu Aug 29 05:26:49 2019
OS/Arch:           darwin/amd64
Experimental:      false

Server: Docker Engine - Community
Engine:
Version:          19.03.2
API version:      1.40 (minimum version 1.12)
Go version:       go1.12.8
Git commit:       6a30dfc
Built:            Thu Aug 29 05:32:21 2019
OS/Arch:          linux/amd64
Experimental:     false
containerd:
Version:          v1.2.6
GitCommit:        894b81a4b802e4eb2a91d1ce216b8817763c29fb
runc:
Version:          1.0.0-rc8
GitCommit:        425e105d5a03fabd737a126ad93d62a9eeede87f
docker-init:
Version:          0.18.0
GitCommit:        fec3683

$ docker-compose version
docker-compose version 1.24.1, build 4667896b
docker-py version: 3.7.3
CPython version: 3.6.8
OpenSSL version: OpenSSL 1.1.0j  20 Nov 2018
```

To save some download time, you can also pull in advance the quarkus image (600MB)

```
$ docker pull quay.io/quarkus/ubi-quarkus-native-image:19.3.1-java11
```