


Vert.x





For vertx demo

- Generate the project at <https://start.vertx.io/> by giving the specifications

Create a new Vert.x application

Version	<input type="button" value="4.2.6"/> <input type="button" value="4.3.0-SNAPSHOT"/>
Language	<input type="button" value="Java"/> <input type="button" value="Kotlin"/>
Build	<input type="button" value="Maven"/> <input type="button" value="Gradle"/>
Group Id	<input type="text" value="com.example.firstVertx"/>
Artifact Id	<input type="text" value="vertex-starter"/>
Dependencies (0/78)	<input type="text" value="Web, MQTT, etc."/>
+ Show dependencies panel	
Advanced options 	

Package	<input type="text" value="Your project package name"/>
JDK Version	<input type="button" value="JDK 1.8"/> <input type="button" value="JDK 11"/> <input type="button" value="JDK 17"/>

- Build that in an IDE (Eclipse or IntelliJ) and Run as JUnit
- Output : HTTP server started on port 8888
- No errors

Features of Vertex

The whole Vert.x toolkit is built around the vertx object, which is the control centre of Vert.x. With this Vertx object we can do a lot of things like :

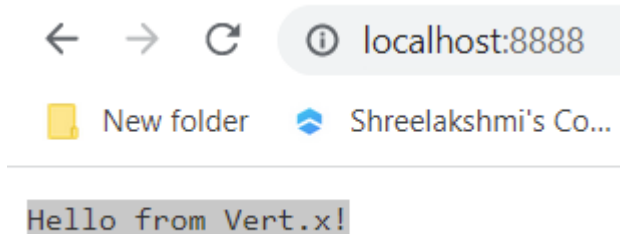
- Creating servers
- Creating clients
- Interact with event bus
- Setting timers ?!

To start our application we need a vert.x object. We can achieve this by creating 1 vert.x object in the Main.java class as the Main.java class acts as an entry point for any Java Application because it has public static void main(String[] args) in it.

Steps

- Create main method in MainVerticle class

- Assign vertx instance to a new variable
`var vertx = Vertx.vertx();`
`vertx.deployVerticle(new MainVerticle());`
- Vertx is deploying new verticle and the `start()` method below is called.
- Run main method
- Output : HTTP server started on port 8888
- Make a REST API call from the browser by typing
`localhost:8888`



>> I added emoji and the text and made the text compatible with emoji. Now, even after I removed the emoji, I'm not able to build and run the file. So, I deleted the project !
 And I think it was erased from the disk and I unzipped the folder (project from vertx.io) and gave it another name and opened it in Eclipse. The build gradle failed !

```

public class MainVerticle extends AbstractVerticle {
    private static final Logger LOG = LoggerFactory.getLogger(MainVerticle.class);

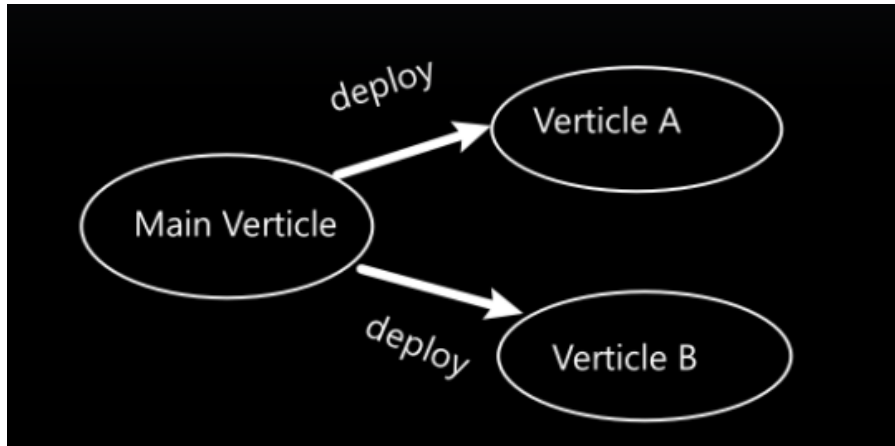
    public static void main(String[] args) {
        var vertx : Vertx = Vertx.vertx();
        vertx.deployVerticle(new MainVerticle());
    }

    @Override
    public void start(Promise<Void> startPromise) throws Exception {
        vertx.createHttpServer().requestHandler(req -> {
            req.response()
                .putHeader( name: "content-type", value: "text/plain")
                .end( chunk: "Hello World!");
        }).listen( port: 8888, http -> {
            if (http.succeeded()) {
                startPromise.complete();
                LOG.info("HTTP server started on port 8888");
            } else {
                startPromise.fail(http.cause());
            }
        });
    }
}

```

We have learned how to create Vertex object.

- Normally only 1 vertex instance is needed for each application. So make sure not to create multiple ones and (always rely on instance references from the different verticals ?)
- Vert.x comes with scalable actor like deployment and concurrency model vertical objects
- Vertices are running on the **event loop** threads which are used for non-blocking operations and a vertical can be deployed multiple times for scalability.
- Example : Main verticle deploys Verticle A and Verticle B and so on...



- Each child vertical can be deployed multiple times.
- Each vertical has its own thread
- So, multiple threads can be used without concurrent access issues.
- Verticle code is executed on non-blocking event loop
- Vert.x uses multiple **event loops** for an application (called **Multi - Reactor pattern**)
- An event loop must not execute blocking code to guarantee a fast processing of events.
- So, inside a vertical thread safety is guaranteed.
- Vertices typically **communicate** over the Vert.x through **eventbus**.
- The whole vertical model is optional (the whole project setup). But it's important to use it to avoid concurrency issues in an easy way and to make it easier to scale your application.
- Typically all Verticles use the same Vert.x instance. So don't create your own Vert.x instance, because Vert.x is handling this for you.

>> I restarted and redownloaded the whole project from the <https://start.vertx.io/> 2 times, but even after importing the project and building the gradle, it says the project has some errors and it can't be run both in IntelliJ and Eclipse.

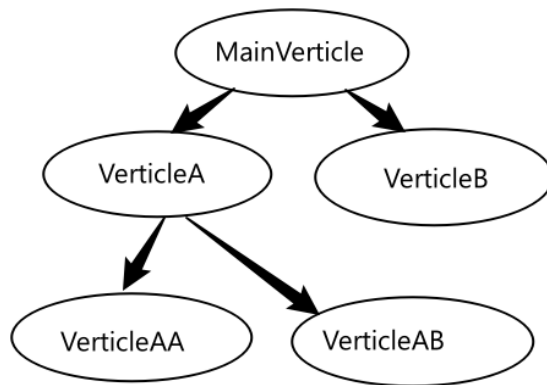
Now I'm watching this video :

https://www.youtube.com/watch?v=sAVJDyQd4j4&ab_channel=TechPrimers

Still showed the errors which I couldn't resolve, that's why I restarted the whole project once again from the beginning.

Now, the build is successful and it does not show any errors !

- After deploying this



- After running the main method -> Build gradle daemon -> compile Java class
-> it goes down the tree and the parents are always deployed first. (Like preorder in tree)

Undeployment of Verticles

- Stop method helps in cleaning up resource properly when the application shuts down. Just like cleaning up the resources like database connections after disconnecting with the server
- whenDeployed is an asynchronous call back mechanism ??
- Output :

```

> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes

> Task :MainVerticle.main()
Start com.danielprinz.udemy.vertex_starter.vertices.MainVerticle
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleA
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleB
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleAA
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleAB
Deployed com.danielprinz.udemy.vertex_starter.vertices.VerticleAA
Stop com.danielprinz.udemy.vertex_starter.vertices.VerticleAA
Deployed com.danielprinz.udemy.vertex_starter.vertices.VerticleAB
|

```

Scaling Verticles

- Deploying a vertical multiple times.
- Creating a new VerticleN (same as VerticleB)
- Deploy it in the MainVerticle

- When deploying multiple instances, only pass the name as first parameter (Don't add new). 2nd parameter can be deployment options to set the number of instances.
- Don't run all the vertices multiple times! As they would compete for resources.
- Output :

```
> Task :MainVerticle.main()
Start com.danielprinz.udemy.vertex_starter.vertices.MainVerticle
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleA
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleB
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleAA
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleAB
Deployed com.danielprinz.udemy.vertex_starter.vertices.VerticleAA
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleN ←
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleN ←
Stop com.danielprinz.udemy.vertex_starter.vertices.VerticleAA
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleN ↵
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleN ↵
Deployed com.danielprinz.udemy.vertex_starter.vertices.VerticleAB
```

1:32:14 PM: Executing ':MainVerticle.main()'...

```
> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes

> Task :MainVerticle.main()
Start com.danielprinz.udemy.vertex_starter.vertices.MainVerticle
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleA
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleB
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleAA
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleAB
Deployed com.danielprinz.udemy.vertex_starter.vertices.VerticleAA
Stop com.danielprinz.udemy.vertex_starter.vertices.VerticleAA
Deployed com.danielprinz.udemy.vertex_starter.vertices.VerticleAB
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleN on thread vert.x-eventloop-thread-6
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleN on thread vert.x-eventloop-thread-5
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleN on thread vert.x-eventloop-thread-7
Start com.danielprinz.udemy.vertex_starter.vertices.VerticleN on thread vert.x-eventloop-thread-8
```

- Adding Config options by referring to the video 10. Vertical Vert.x config
- If the verticle is deployed multiple times, each verticle gets the same configuration.

- Output :

```
10:19:16 AM: Executing ':MainVerticle.main()'...

Starting Gradle Daemon...
Gradle Daemon started in 2 s 671 ms
> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes

> Task :MainVerticle.main()
Start com.danielprinz.udemy.vertex_starter.verticles.MainVerticle
Start com.danielprinz.udemy.vertex_starter.verticles.VerticleA
Start com.danielprinz.udemy.vertex_starter.verticles.VerticleB
Start com.danielprinz.udemy.vertex_starter.verticles.VerticleAA
Deployed com.danielprinz.udemy.vertex_starter.verticles.VerticleAA
Start com.danielprinz.udemy.vertex_starter.verticles.VerticleAB
Stop com.danielprinz.udemy.vertex_starter.verticles.VerticleAA
Deployed com.danielprinz.udemy.vertex_starter.verticles.VerticleAB
Start com.danielprinz.udemy.vertex_starter.verticles.VerticleN with config {"id":"9e580a97-6501-4c39-9670-ee230c0807eb","name":"VerticleN"}
Start com.danielprinz.udemy.vertex_starter.verticles.VerticleN with config {"id":"9e580a97-6501-4c39-9670-ee230c0807eb","name":"VerticleN"}
Start com.danielprinz.udemy.vertex_starter.verticles.VerticleN with config {"id":"9e580a97-6501-4c39-9670-ee230c0807eb","name":"VerticleN"}
Start com.danielprinz.udemy.vertex_starter.verticles.VerticleN with config {"id":"9e580a97-6501-4c39-9670-ee230c0807eb","name":"VerticleN"}
```

Logging

- Apache Log4j Version 2 is used for logging
- Vertx uses internal API to support other logging packets as well
- Cool benefit of Apache4j 2 is supporting asynchronous call logging which can get 18 times higher throughput and low latency than Version 1 especially in the multi - threaded environment Log4j 2 is better than Version 1.
- I copy pasted the below code exactly as the video 11. Logging told me but I still got Unresolved reference: dependencyManagement

Code : from <https://logging.apache.org/log4j/2.x/maven-artifacts.html>

```
plugins {
    id 'io.spring.dependency-management' version '1.0.1.RELEASE'
}

dependencyManagement {
    imports {
        mavenBom 'org.apache.logging.log4j:log4j-bom:2.17.2'
    }
}

dependencies {
    implementation 'org.apache.logging.log4j:log4j-api'
    implementation 'org.apache.logging.log4j:log4j-core'
    // etc.
}
```

- I commented it and after building the gradle again I uncommented and it worked fine.
- The debug LOGs are not printed while debugging.
- Udemy Video : 11

The next feature of Vert.x is Event Loops

Event Loops

- If we run some code, the code is executed on a thread on the CPU.
- If the application would have only 1 thread nothing could happen in parallel and other requests would have to wait the whole time until the thread is free.
- To avoid this issue, a software programmer normally runs multiple threads and they can execute tasks in parallel. This concept is called **Multi threading** .
- It is ok to run more threads as a software programmer but when more concurrent requests are happening then a lot of threads have to be created and depending on the requests, they would need to wait until the response returns.
- This causes a lot of wasted CPU time as threads are fully not utilised and at some point the CPU will not accept more threads. That means there is limited power and limited resources.
- The solution to waiting time is called : Non - Blocking I/O Event Loop
- The threads which are executing non-blocking I/O are called Event Loops
- Vert.x is utilising the concept of Event Loops heavily
- In the picture, all the events are lined up in a queue and the event loop thread is processing them as fast as possible without any waiting time.
- RULE NO. 1 : Never block any event loop threads to ensure all the events can be processed as fast as possible.
- Each vert.x Verticle is scheduled on an Event Loop thread. With that inside a verticle thread safety is guaranteed and concurrency can happen without concurrent access issues.
- Sometimes there are operations which are blocking by nature, like traditional relational database access, file access, waiting for network responses.
- These blocking operations cannot be avoided. It is possible to execute these kind of events on so-called - **Worker Threads**.
- Additionally Vertx has a thread blocked checker that will warn when the event loop was blocked. If you see these warnings make sure to fix it, otherwise your application might not behave as expected.
- There are 2 ways to do blocking operations,
 1. Vertx.executeBlocking
 2. Deploying worker verticle
- If you encounter a lot of blocked thread checker warnings in your application, remember to schedule your code on a blocking thread to avoid blocking the event loop.

The EventBus

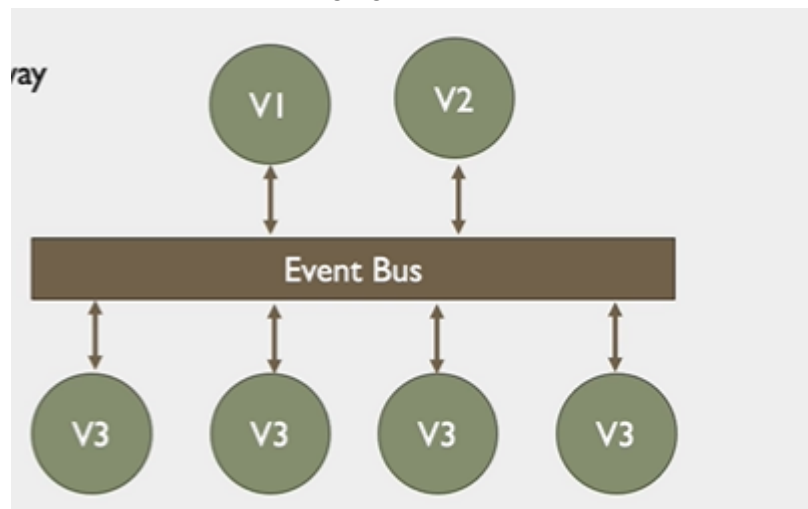
Look at this question at the end : >>>>>>>>>>>>>>>>>>>

<https://www.udemy.com/course/reactive-web-applications-with-vertex-and-vuejs/learn/lecture/22953006#questions/15026774>

Every verticle runs on its own thread and sometimes have to communicate with each other.

For this purpose, we can use the vertx Event Bus.

- Event Bus can be seen as the nervous system of vertx
- It allows event driven communication in a non-blocking thread-safe way.
- There is only 1 event bus instance available per vertx instance available.
- It is possible to hook up the event bus to clients and to distribute messages to multiple server nodes.
- The event bus supports 3 ways of messaging
 1. Publish/ subscribe messaging
 2. Point to point messaging
 3. request-response messaging



JSON Object

- When working with web applications, JSON is the de facto standard.
- A majority of the REST API use it, it is human readable and very flexible.
- Java does not have a JSON Object class out of the box, but Vertx has it.
- Vertx has a built in JSON Object in JSON array implementation, it is possible to convert the Java objects to the Vertx JSON objects or send them over an event bus.
- To see the functionality of JSON Object in the Vertx we are going to create a standard JUnit Test with the name JSONObjectExample
- JUnit 5 is configured already in the Vertx starter website so we don't have to add a dependency.
- Code :


```

package com.danielprinz.udemy.vertex_starter;

import io.vertx.core.json.JsonObject;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class JsonObjectExample {

    @Test
    void jsonObjectCanBeMapped()
    {
        final JsonObject myJsonObject = new JsonObject();
        myJsonObject.put("id", 1);
        //Key value pairs in .put method just like HashMaps
        myJsonObject.put("loves_vertx", true);

        assertEquals( expected: "", myJsonObject.encode());
    }
}

```

- Output : The test fails because we have set the assertEquals to an empty

```

expected: <> but was: <{"id":1,"loves_vertx":true}>
Expected :
Actual    :{"id":1,"loves_vertx":true}
<Click to see difference>

```

variable.

Vertx JSON Array Object

- Array is the same type of values stored sequentially.

Map Java Objects[Person class object] to JSON Objects

- Using JSON Objects is fine, but while developing we would want our own objects.
- With Vertx version 4, one **dependency has to be added**. Then we can convert object from and to JSON Objects.
- So the `canMapJavaObjects()` method is used.
- In the background, Vertx uses jackson as JSON processor for the *conversion*.
<https://github.com/FasterXML/jackson>
 - Jackson is a high performance, JSON Processor that is very common across multiple frameworks.
 - Every object needs to have a **constructor**, **getter** and **setter**, otherwise the conversion will fail.
 - `mapFrom` method is used.

Converting Person class object to JSON Object

- `mapTo` method is used
- Deserializing an object??? ->>>>>>>>> How does it actually happen in the code
<https://www.geeksforgeeks.org/serialization-in-java/>
- Error :
Cannot construct instance of `com.danielprinz.udemy.vertex_starter.Person` (no Creators, like default constructor, exist): cannot deserialize from Object value (no delegate- or property-based Creator)
at [Source: UNKNOWN; byte offset: #UNKNOWN]
- Solution : Adding the fault constructor to the Person object

Event Bus - JSON

- We have seen how to use Vertx EventBus to communicate in a thread safe way.
- Now, we will see how to send Vertx JSON Object over the Event Bus
- How to send custom Java Objects over the event bus ?
- By default Java does not allow custom objects to be sent on an event Bus.
- We require a custom codec for it.

Custom message Codec

- A generic version of an event bus message codec done by Udemy instructor.
- Every vertx message codec must implement interface `MessageCodec<T, T>`
- Provide the type of the message being sent and the message being received.
- `LocalMessageCodec` provide way to send one type and to receive the same type
- Make sure to always send **Immutable objects** when using this codec otherwise concurrency issues may occur

- Immutable object : is an object that cannot be changed after it was created. Immutability is normally achieved by having a constructor to initialise it but not have setters for the properties. So the properties cannot be changed afterwards.
- Output :

```
> Task :PingPongExample.main()
Sending Ping{message='Hello...', enabled=true}
Received message : {}Ping{message='Hello...', enabled=true}
Response: {}customcodec.Pong@73a304eb
```

Vertx Future and Promise

- We know that Vertx uses an asynchronous programming model to achieve scalability and resource efficiency.
- We learn about *event loops* where *only asynchronous tasks* should be executed.
- In an asynchronous environment it is possible to use callbacks to manage tasks and react on different results
- Call back hell : Chaining call-backs together can get very messy. There is a better way.
- In Vertx we call it Future and Promise.
- They allow a simple coordination of asynchronous tasks
- They are pretty common when using functional programming approach.
- Promise was introduced in Version 3.8 and in the previous versions of Vertx only Future was available.
- Promise : Used to write an eventual value and can be completed or it can be marked as field.
- Future : used to read the value from the promise when it is available.
- The promise should live in a small context. For example, inside a function.
- A future on the other hand can be returned and chained together to act on the result of promise.

Promise:

```
Final Promise<String> promise = Promise.promise();
vertx .setTimer(500, id-> promise.complete("Success!") );
```

- Here the type of promise is String, any other type can work as well
- To simulate asynchronous code, a vertx timer is used, and the promise is completed after 500 milliseconds delay with the message "Success!"
- A promise can be completed in multiple ways and can contain different value.

```
final Promise<Void> returnVoid = Promise.promise();
returnVoid.complete();
```

- Promise<Void> does not contain any value.
- Promise<String> can also be used (Previously shown code)
- Promise<JsonObject> can also be used

```
Final Promise<String> returnString = Promise.promise();
returnString.complete("Hello")
```

```
Final Promise<JsonObject> returnJsonObject = Promise.promise();
return JsonObject.complete(new JsonObject().put("message", "Hello!"));
```

- A future is the review of the promise and can react on the result.
- For the given example code, from the promise a future can be created with promise.future()
- The future has the same return type as the promise.

```
Final Promise<String> promise = Promise.promise();
vertx .setTimer(500, id-> promise.complete("Success!") );
```

```
Final Future<String> future = promise.future();
future
    .onSuccess(result ->
        context.completeNow()
    )
    .onFailure(context::failNow);
```

- A future can react on the outcome of promise.
- See composite future.
- Promise = write view
- Future = read view

Vertx Launcher

- Shadow plugin is included in build.gradle file, this means when we build our application, a fat jar is created
- *Fat jar* : single jar file which contains all the compiled java classes of a project and also all the compiled java classes from all dependencies.
- This means no additional dependencies have to be referenced.
- Fat jars are very handy for packaging an application inside a java container or for execution
- We will now bundle a fat jar by using gradle
- Clean and assemble the project using the command : **./gradlew clean assemble**

- For maven project replace gradle by maven
- And then execute this command in the terminal of the given the project
- Instead of using the main method in the main verticle , we can start the project by doing the following (after the gradle wrapper)

```

BUILD SUCCESSFUL in 7s
10 actionable tasks: 10 executed
PS C:\Users\shree\OneDrive\Documents\Project_Workspace\IUDX\practice\without errors I gue
ss\vertex-starter> java -jar build/libs/vertex-starter-1.0.0-SNAPSHOT-fat.jar
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impact perform
ance.
HTTP server started on port 8888

```

The screenshot shows the IntelliJ IDEA interface with a terminal window open. The terminal output indicates a successful build and the execution of a Java command to start a Vert.x application. The status bar at the bottom shows 'Tests passed: 1 (21 minutes ago)' and the current time is 22:29.

Live redeploy

- Vertx launcher has another main benefit which is live redeploy functionality
- IntelliJ - create a Run configuration (Application), set the Main class to `io.vertx.core.Launcher`. In the Program arguments write: `run your-verticle-fully-qualified-name --redeploy=**/*.class --launcher-class=io.vertx.core.Launcher`. To trigger the redeployment, you need to make the project or the module explicitly (Build menu → Make project).

Vert.x Docker - FatJar

- These days most of the applications are running on the cloud
- The de facto standard is to bundle a docker container and deploy it
- Here we will see how to bundle a Vertx fat jar file into a docker container
- In the next video, we will see how to use the google chip plugin to bundle at Docker container, during the gradle build
- To learn about Docker, I watched these videos :

https://www.youtube.com/watch?v=FzwlS2jMESM&t=0s&ab_channel=JetBrainsTV

To Know about Docker features available in IntelliJ watch this

https://www.youtube.com/watch?v=ck6xQqSOlpw&ab_channel=IntelliJIDEAbyJetBrains

- Dockerfile will include the instructions for *how to build the image* for the given hello World Java Application.
- The order of commands in a Dockerfile is very important as each instruction is executed one by one.
- An image is made out of layers

```

=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/adoptopenjdk:11-jre-hotspot 0.0s
=> [internal] load build context 0.1s
=> => transferring context: 9.34MB 0.1s
=> CACHED [1/3] FROM docker.io/library/adoptopenjdk:11-jre-hotspot 0.0s
=> [2/3] COPY build/libs/vertex-starter-1.0.0-SNAPSHOT-fat.jar /usr/app/ 0.1s
=> exporting to image 0.1s
=> => exporting layers 0.1s
=> => writing image sha256:56d348350dafa6a9401fe7f1e301d87cdc9d3ff78aa69b2199f96a 0.0s
=> => naming to docker.io/example/vertx-starter 0.0s

```

Id of the container (SHA for container) is successfully built after running the build command example/vertx-starter is the name of the image

```

=> => naming to docker.io/example/vertx-starter 0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how
to fix them
PS C:\Users\shree\OneDrive\Documents\Project_Workspace\IUDX\practice\without errors I gue
ss\vertex-starter> docker run -t -i -p 8888:8888 example/vertx-starter
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impact perform
ance.
HTTP server started on port 8888

```

- After making some changes in the Dockerfile or any other java class, run this in the terminal
- `./gradlew clean assemble`
- Then build it
- Run it

Vert.x docker Jib

- Now we will learn how to use jib to bundle our Java application into a docker container
- Then suddenly, lib folder with the fat jar file vanished and showed me errors while building the Dockerfile
- So, I ran the `./gradlew clean assemble` again
- Got the compiled Java files in the fat jar file present in the libs folder
- Then built it using the command :
- `docker build -t example/vertx-starter .`
- Run using : in the terminal
- `docker run -t -i -p 8888:8888 example/vertx-starter`
- Output : HTTP server started on port 8888
- In the web browser :

```

< > ↻ ⓘ localhost:8888

```

Hello Shreeeee!

Vert.x Web

- In this section, we will learn how to use vertx to create REST API
- In Vert.x server and client is split up into different packages

Server

- On the server side : Vertx is built on top of *Netty (A very popular reactive HTTP server)*
- It also supports : request routing
- Non - blocking and asynchronous
- The server supports Reactive HTTP Request Processing
- Also supports, Json Binding with Jackson
- Server also supports : Web sockets
- And supports HTTP version 2

Client

- When using Vertx web client, we can use functionality such as asynchronous HTTP and HTTP version 2
- Client supports JSON body encoding and decoding out of the box
- The client can be used for request and response streaming
- Supports RxJava 2 API

Building a rest api Application : **Stock Broker Rest API**

1. Public Endpoints
 - GET
 - /assets
 - /quotes/{asset}
2. Private Endpoints
 - GET
 - /account/watchlist + /account/watchlist-reactive
 - PUT
 - /account/watchlist + /account/watchlist-reactive
 - DELETE
 - /account/watchlist + /account/watchlist-reactive
3. I'm going to send the repository to GitHub
4. Go to the link to generate the project : <https://start.vertx.io/>

Create a new Vert.x application

Version	<div>4.2.6 4.3.0-SNAPSHOT</div>
Language	<div>Java Kotlin</div>
Build	<div>Maven Gradle</div>
Group Id	<div>com.shree.udemy</div>
Artifact Id	<div>vertx-stock-broker</div>
Dependencies (1/78)	<div>Web, MQTT, etc.</div>
+ Show dependencies panel	
Advanced options	

Package	<div>Your project package name</div>
JDK Version	<div>JDK 1.8 JDK 11 JDK 17</div>

Selected dependencies (1)

Vert.x Web ✕

Generate Project alt + ⌘ >_

Generate with Power Shell

Invoke-WebRequest -Uri https://starl

[→ Find a Vert.x how-to](#)
[🐛 Report an issue](#)

Create a folder (named same as Artifact Id) (vertx-stock-broker)

Cd to the folder and open terminal in intellij

Then paste the command which is given in the “Generate with Powershell” box

It will download a zip folder

Does not work {

Unzip the folder using the command

unzip name-of-the-zip-file.zip

unzip vertx-stock-broker.zip

}

Unzip it manually

- For logging in the project , go to pom.xml and write
- After the dependency add a new dependency

```
<dependency>
<groupId>ch.qos.logback</groupId>
<artifactId>logback-classic</artifactId>
<version>1.2.7</version>
</dependency>
```

- Build the maven by clicking on the icon and the dependency will be downloaded

```
private static final Logger LOG = LoggerFactory.getLogger(MainVerticle.class);
```

In the MainVerticle.java class and use the org.slf4j for importing the Logger and LoggerFactory packages

- Creating the LOG configuration in the resources folder
- Copy pasting the log configurations in the logback.xml file in the resources folder (create resources folder under main folder)
- Change the logger name in logback.xml
- Output : With Colour log output !

```
"C:\Program Files\Java\jdk-13.0.1\bin\java.exe" ...
HTTP server started on port 8888
13:41:22.746 INFO [vert.x-eventloop-thread-1] com.shree.udemy.vertx_stock_broker.MainVerticle - Deployed com.shree.udemy.vertx_stock_broker.MainVerticle !
```

HTTP Routing - GET Request

```
public void start(Promise<Void> startPromise) throws Exception {
```

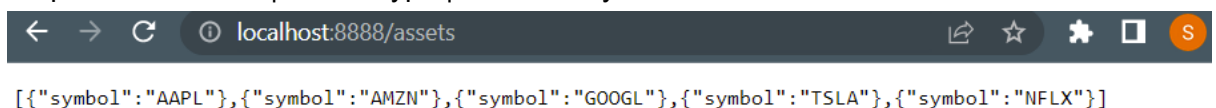
```
    final Router restApi = Router.router(vertx);
```

- Now use the the newly created router and attached it to the HTTP server

Using the below code to do so in the (start method)

```
        vertx.createHttpServer()
            .requestHandler(restApi)
            .exceptionHandler(error -> {
                LOG.error("HTTP Server error: ", error);
            })
            .listen(8888, http -> {
                if (http.succeeded()) {
                    startPromise.complete();
                    LOG.info("HTTP server started on port 8888");
                } else {
                    startPromise.fail(http.cause());
                }
            });
```

- Output : of Rest API | Return type | JSON Array



```
localhost:8888/assets
[{"symbol": "AAPL"}, {"symbol": "AMZN"}, {"symbol": "GOOGL"}, {"symbol": "TSLA"}, {"symbol": "NFLX"}]
```

- Output in IntelliJ

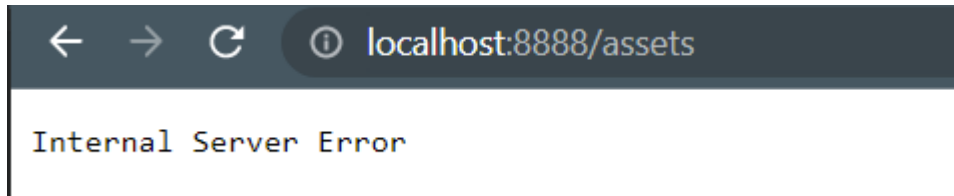
```
"C:\Program Files\Java\jdk-13.0.1\bin\java.exe" ...
14:18:58.703 INFO [vert.x-eventloop-thread-1] com.shree.udemy.vertx_stock_broker.MainVerticle - Deployed com.shree.udemy.vertx_stock_broker.MainVerticle !
14:18:58.703 INFO [vert.x-eventloop-thread-0] com.shree.udemy.vertx_stock_broker.MainVerticle - HTTP server started on port 8888
14:19:26.097 INFO [vert.x-eventloop-thread-0] com.shree.udemy.vertx_stock_broker.MainVerticle - Path /assets responds with [{"symbol": "AAPL"}, {"symbol": "AMZN"}, {"symbol": "GOOGL"}]
```

This makes use of HTTP get using Vert.x JsonObject

Now, we will look into how we can use our Own Java Objects

Custom Objects and Error handling

- Previously we created our First Rest API Endpoint and returned a vertex JSON Array
- Here, we will return a custom Java Object and add some proper error handling to see when something goes wrong in our assets



- That's why always write down failure handler in the start method of the main verticle
- After error handling



```
    {"message ":"Something went wrong :-| "}

// Failure Handler
restApi.route().failureHandler(errorContext ->{
    if(errorContext.response().ended())
    {
//        if the client stops the requests
        return;
    }
    LOG.error("Route Error : ", errorContext.failure());
//    response to the client
//    .end( ) is defining a body
    errorContext.response()
        .setStatusCode(500)
//        Setting the status code as 500 bcs we don't know
//        what happened
        .end(new JsonObject().put("message ", "Something went wrong :-| ").toBuffer());
});
```

- Add this in the pom.xml to solve the problem

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.13.0</version>
</dependency>
```

- Re - load the maven changes and run the main method in the MainVerticle
- Correct Output !!



```
15:12:21.216 INFO [vert.x-eventloop-thread-1] com.shree.udemy.broker.MainVerticle - Deployed com.shree.udemy.broker.MainVerticle !
15:12:21.216 INFO [vert.x-eventloop-thread-0] com.shree.udemy.broker.MainVerticle - HTTP server started on port 8888
15:12:24.132 INFO [vert.x-eventloop-thread-0] assets.AssetsRestApi - Path /assets responds with [{"name":"AAPL"}, {"name":"AMZN"}, {"name":"GOOGL"}, {"name":"TSLA"}, {"name":"NFLX"}]
```

Unit Tests using WebClient

- To see how the Vert.x webclient works and how we can use Unit Tests in our REST API, we will create a JUnit 5 test which will access the AssetsRestApi class and verify that everything works as expected.
- For this, we are copying TestsMainVerticle and renaming it to Test Assets Rest Api (without the spaces)
- Creating Vertx Web client in the TestAssetsRestApi
- Vertx web client is an asynchronous HTTP client which is quite powerful
- Vertx Web can be used for testing purpose and also in the application itself
- For example, it can be used as client for other REST APIs or as proxy
- At first, we are going to use it for testing by writing WebClient.create
- So, while adding the <dependency> in pom.xml file, it was showing me errors
- I clicked on the build maven icon in spite of the errors. Andddd it didn't show any errors!
- And WebClient and its methods are available now !
- The output is tested in the Unit test

Project Lombok

- Project Lombok is a very useful Java Library to avoid a lot of boilerplate code
- It can generate getters, setters, full featured builders and more
- In this video, we will configure lombok for our project
- As we have Maven project, we will need to add dependency in pom.xml from the website projectlombok.org/setup/maven
- Code :

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.22</version>
  <scope>provided</scope>
</dependency>
```

- No errors are found

HTTP routing Path Variables

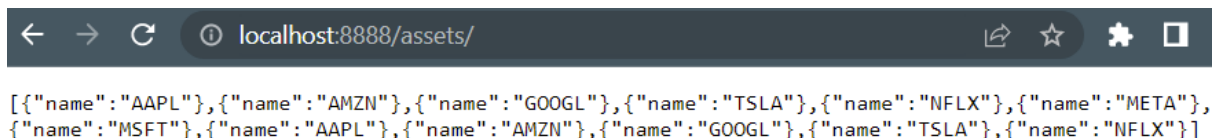
- Right now, stock broker application has 1 rest api endpoint
- In this video we will add another REST API endpoint
- Also, see how to use path parameter and create a unit test to verify the correct behaviour
- We will start by defining a new class called QuotesRestApi
- And calling it in the start method of the main verticle

- To make it a bit more realistic, we will add an in-memory store in the next video

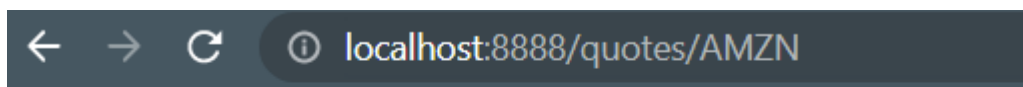
In Memory Store

- To limit the response to a valid asset we will create an in memory store
- :: symbol is used for mapping
- Hashmap is used here for caching in memory store

Custom HTTP Response

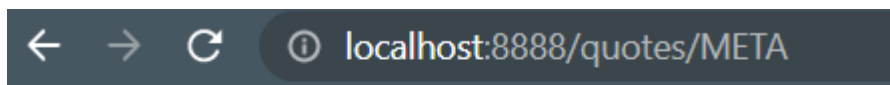


```
[{"name": "AAPL"}, {"name": "AMZN"}, {"name": "GOOGL"}, {"name": "TSLA"}, {"name": "NFLX"}, {"name": "META"}, {"name": "MSFT"}, {"name": "AAPL"}, {"name": "AMZN"}, {"name": "GOOGL"}, {"name": "TSLA"}, {"name": "NFLX"}]
```



```
{}
```

```
{}
```



```
{}
```

```
{}
```

- Output not as expected !

HTTP routing - PUT

- Our application has the REST API for Assets and Quotes
- All the endpoints have been HTTP GET Endpoints
- It is also possible to use HTTP POST, PUT and DELETE
- POST and PUT are very similar
- PUT and DELETE Endpoints are now introduced by creating a new REST API
- We are going to create a new watchlist REST API by defining one in the main verticle
- Below the other REST APIs we are adding watchlist REST API
- WatchlistRestApi.attach(restApi)
- We are passing router with the name restApi

Body Handler

- The body handler to pass HTTP Request bodies is not enabled
- Therefore our put endpoint does not work
- In this video we will see how to register our body handler correctly
- A bodyhandler can be registered via route. For example it could be registered only for the watchlist route or we can add it to the parent router
- A handler can be called before another handler is called by using BodyHandler
- This means when a HTTP request reaches web server a body handler is called
- If the request body is available, the body will be added to the request context and after that the handler will forward it to the next handler
- If there is no next handler, the request will END
- If there is one, it will be forwarded to the next handler.

HTTP Headers - Content Type

- When creating REST APIs it is best practise to set the HTTP content Type header to the value Application/Json
- In this lecture we will test the header and add a test assertion for it

HTTP - Custom Headers

Refactor HTTP Handlers

Vert.x Web server

- Vert.x is a very resource efficient and highly scalable toolkit using a non-blocking HTTP server to handle a lot of HTTP requests.
- Under the hood vertx uses netty (a very popular HTTP server)
- Until now, we deployed our REST API on 1 event loop.
- If your server has more than 1 CPU processor this approach would not utilise the resources fully.
- In this video, we will see how to utilise our resources efficiently
- HTTP server runs on one eventloop thread and all requests are handled on the same thread.
- Multiple HTTP servers can be started on multiple threads with Vertx
- Load will be distributed
- Example : using 1 HTTP server er CPU core

- Depending on what the application is doing, it makes sense to add more or less threads
- When the server is responding to multiple requests, their requests are distributed over multiple threads

```

MainVerticle x TestAssetsRestApi.returns_all_assets x
16:26:45.103 INFO [vert.x-eventloop-thread-7] com.shree.udemy.broker.MainVerticle - HTTP server started on port 8888
16:26:45.103 INFO [vert.x-eventloop-thread-2] com.shree.udemy.broker.MainVerticle - HTTP server started on port 8888
16:26:45.103 INFO [vert.x-eventloop-thread-12] com.shree.udemy.broker.MainVerticle - HTTP server started on port 8888
16:26:45.103 INFO [vert.x-eventloop-thread-11] com.shree.udemy.broker.MainVerticle - HTTP server started on port 8888
16:26:45.103 INFO [vert.x-eventloop-thread-1] com.shree.udemy.broker.MainVerticle - HTTP server started on port 8888
16:26:45.103 INFO [vert.x-eventloop-thread-6] com.shree.udemy.broker.MainVerticle - HTTP server started on port 8888
16:26:45.103 INFO [vert.x-eventloop-thread-10] com.shree.udemy.broker.MainVerticle - HTTP server started on port 8888
16:26:45.103 INFO [vert.x-eventloop-thread-0] com.shree.udemy.broker.MainVerticle - Deployed RESTAPIVerticle with id 84
16:26:45.103 INFO [vert.x-eventloop-thread-9] com.shree.udemy.broker.MainVerticle - HTTP server started on port 8888
16:26:45.108 INFO [vert.x-eventloop-thread-13] com.shree.udemy.broker.MainVerticle - Deployed com.shree.udemy.broker.M

```

• Multiple threads running on the same server

★ How many cores does my laptop have ????

- In this video we have scaled the web server
- In the next video we will see how the scaled web server reacts under load

How the scaled Web server reacts under load- LOAD TEST

VERT.X CONFIGURATIONS

- In this section we will learn how to configure a vertx application properly
- Vert.x config has a lot of functionality
- Supports a lot of different formats like : JSON, YAML (extension), Hocon (extension) etc
- Configuration properties can be stored in multiple stores. Vertx supports :
 - System properties
 - Environment properties
 - Files
 - Event Bus
 - Kubernetes Config Map (extension) etc...
- An environment store can be configured as follows:

-
- ❖ I uploaded the project - stock broker vertx application to GitHub from my Git Bash !
 - ❖ By following the given commands + USED THE PERSONAL TOKEN ! (Because the access was denied while pushing it to the master branch)

- ❖ This helped me a lot -> from <https://stackoverflow.com/questions/4181861/message-src-refspec-master-does-not-match-any-when-pushing-commits-in-git>

Try git show-ref to see what refs you have. Is there a refs/heads/master?

Due to the recent "Replacing master with main in GitHub" action, you may notice that there is a refs/heads/main. As a result, the following command may change from git push origin HEAD:master to git push origin HEAD:main

2. You can try git push origin HEAD:master as a more local-reference-independent solution. This explicitly states that you want to push the local ref HEAD to the remote ref master (see the [git-push refsPEC](#) documentation).

- ❖ git init
- ❖ git add .
- ❖ Git commit -m "Initial Commit"
- ❖ git branch -M main
- ❖ git remote add origin <https://github.com/shreelakshmi-datakaveri/udemy-vertx-stock-broker.git>
- ❖ git push -u origin main -> Here I encountered with the problem!

```
shree@DESKTOP-83AE3P9 MINGW64 ~/OneDrive/Documents/Project_Workspace/IUDX/practice/vertx-stock-broker (main)
$ git show-ref
d7fd7a5fa52042201cb164f5c3dcafd63368c2a refs/heads/main

shree@DESKTOP-83AE3P9 MINGW64 ~/OneDrive/Documents/Project_Workspace/IUDX/practice/vertx-stock-broker (main)
$ git push origin HEAD:master
error: unable to read askpass response from 'C:/Program Files/Git/mingw64/libexec/git-core/git-gui--askpass'
Username for 'https://github.com': shreelakshmi-datakaveri
remote: Support for password authentication was removed on August 13, 2021. Please use a personal access token instead.
remote: Please see https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/ for more information.
fatal: Authentication failed for 'https://github.com/shreelakshmi-datakaveri/udemy-vertx-stock-broker.git/'

shree@DESKTOP-83AE3P9 MINGW64 ~/OneDrive/Documents/Project_Workspace/IUDX/practice/vertx-stock-broker (main)
$ git push origin HEAD:master
Enumerating objects: 55, done.
Counting objects: 100% (55/55), done.
Delta compression using up to 12 threads
Compressing objects: 100% (44/44), done.
Writing objects: 100% (55/55), 136.99 KiB | 6.85 MiB/s, done.
Total 55 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), done.
To https://github.com/shreelakshmi-datakaveri/udemy-vertx-stock-broker.git
 * [new branch]      HEAD -> master

shree@DESKTOP-83AE3P9 MINGW64 ~/OneDrive/Documents/Project_Workspace/IUDX/practice/vertx-stock-broker (main)
$ git status
On branch main
nothing to commit, working tree clean
```

- ❖ End result :

