**VERT.X PROJECTS**

After a very thorough research into what sort of tools or toolkit or platform for some API I have been planning for some time, vertx came up as the final and possible immediate solution or option for the level of performance I was aiming for.

In my research I used the following websites for my Benchmarking checks and confirmations :

|  |
| --- |
| **Other Benchmarking websites :** |
| https://dev.to/tuananhpham/popular-backend-frameworks-performance-benchmark-1bkh |
| <https://www.tiobe.com/tiobe-index/> |
| https://pypl.github.io/PYPL.html#google\_vignette |
| https://trends.builtwith.com/framework |
| https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html |
| https://programming-language-benchmarks.vercel.app/ |
| https://programming-language-benchmarks.vercel.app/java |
| https://programming-language-benchmarks.vercel.app/java-vs-go |
| <https://www.quora.com/What-are-the-best-alternatives-to-Java-for-high-performance-backend-development> |
| https://news.ycombinator.com/item?id=17254152 |
| https://github.com/rwf2/Rocket/issues/710 |
| https://just.billywhizz.io/blog/on-javascript-performance-01/ |
| [https://www.techempower.com](https://www.techempower.com/) |

Although, one could argue that the way benchmarking is done or executed, it might not be the same as what you might experience in a production environment. And for that I say : If the framework or platform or tool or toolkit or language of choice performed at this level for a very simple hello world app or sample project or in applying a. very simple hello world json structure, then , that is just it. All the other technologies are been benchmarked with same. You better believe it is just what it is. If you did pull the sample technologies and tested them your results would not be too far away nor apart from what has already been done.  
  
Below is a composite framework scoring ( source : techempower.com )

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **­­­** | | | |  | **Source** : *techempower.com* | | | | |  |  |  |  |
| *Each framework's peak performance in each test type (shown in the colored columns below) is multiplied by the weights shown above. The results are then summed to yield a weighted score. Only frameworks that implement all test types are included. 159 total frameworks ranked, 10 visible, 149 hidden by filters. See filter panel above.* | | | | | | | | | |  | **Hardware :** Citrine Dell R440 Xeon Gold + 10 GbE | | |
| Rnk | Framework | JSON | 1-query | 20-query | Fortunes | Updates | Plaintext | Weighted score | | Min | Max | Average |  |
| 5 | redkale | 1,210,086 | 457,935 | 32,272 | 413,537 | 22,900 | 6,981,831 | **7,096** | 87.80% | 22,900 | 6,981,831 | 1,519,760 |  |
| 7 | vert.x | 1,194,427 | 581,542 | 32,505 | 453,406 | 17,122 | 5,624,745 | **6,847** | 84.70% | 17,122 | 5,624,745 | 1,317,291 |  |
| 11 | jooby | 1,162,248 | 515,795 | 32,196 | 427,682 | 15,889 | 4,801,653 | **6,370** | 78.80% | 15,889 | 4,801,653 | 1,159,244 |  |
| 16 | vertx-web | 1,121,889 | 566,804 | 30,669 | 388,512 | 16,598 | 3,774,206 | **6,107** | 75.50% | 16,598 | 3,774,206 | 983,113 |  |
| 19 | inverno | 1,042,404 | 467,759 | 32,441 | 309,684 | 15,864 | 4,043,496 | **5,566** | 68.80% | 15,864 | 4,043,496 | 985,275 |  |
| 38 | quarkus | 903,185 | 318,897 | 17,610 | 214,275 | 6,697 | 2,861,479 | **3,637** | 45.00% | 6,697 | 2,861,479 | 720,357 |  |
| 50 | actframework | 964,004 | 231,641 | 16,942 | 124,422 | 1,985 | 3,273,101 | **2,911** | 36.00% | 1,985 | 3,273,101 | 768,683 |  |
| 55 | javalin | 512,495 | 211,243 | 16,582 | 161,275 | 10,405 | 897,788 | **2,755** | 34.10% | 10,405 | 897,788 | 301,631 |  |
| 57 | revenj.jvm | 527,667 | 290,147 | 14,330 | 177,853 | 5,206 | 730,158 | **2,543** | 31.40% | 5,206 | 730,158 | 290,894 |  |
| 125 | wicket | 378,043 | 24,624 | 1,426 | 25,882 | 551 | 516,369 | **679** | 8.40% | 551 | 516,369 | 157,816 |  |

I had been pretty much used to developing some very sophisticated API solutions using micronaut for some fintech backend that had processed in excess of $40million by the time it was been considered for an acquisition deal/discussion. And at the point, the final part of the negotiation(s) was left to my C.E.O to take over.

So in my quest to decide on what tools, or tool-kit or framework and platform to use for my next big fintech backend vertx had become the obvious. Meanwhile, I took some time off to consider the differences and general comparison between micronaut and vertx.

**Micronaut Framework vs Vert.x: What are the differences?**

**Introduction**

Micronaut Framework and Vert.x are both popular frameworks used for developing microservices and reactive applications. While they share some similarities, there are key differences that set them apart from each other.

1. **Execution Model**: *Micronaut* adopts a more traditional Java execution model, utilizing compile-time annotation processing to minimize reflection and enhance performance. On the other hand, *Vert.x* is event-driven and non-blocking, making it well-suited for highly concurrent applications.
2. **Language Support**: *Micronaut* primarily focuses on providing support for the Java programming language, although it also offers limited compatibility with Kotlin and Groovy. Alternatively, *Vert.x* is polyglot, meaning it supports multiple languages, including Java, Kotlin, JavaScript, Groovy, Ruby, Python, and more.
3. **Dependency Injection**: *Micronaut* has a built-in dependency injection framework that leverages compile-time DI, leading to faster startup times and decreased memory consumption. Conversely, *Vert.x* does not have its own DI framework and instead encourages the use of external libraries such as Dagger or Spring for dependency injection.
4. **Web Support**: *Micronaut* provides comprehensive support for building RESTful APIs and web applications out-of-the-box, including features like server-side templating and built-in support for HTTP clients. *Vert.x*, on the other hand, offers more low-level control over the web stack and allows for the creation of various types of applications, including not only traditional web apps but also real-time websockets and event-driven web systems.
5. **Concurrency Model**: *Micronaut* leverages thread pools and CompletableFuture for performing tasks concurrently and handling asynchronous operations. *Vert.x*, being an event-driven framework, uses an event loop model and employs a single-threaded model, wherein a single event loop can handle multiple requests concurrently by utilizing non-blocking I/O operations.
6. **Development Approach**: *Micronaut* utilizes a compile-time approach, where it analyzes your project's classpath during the build phase to generate factory classes and metadata. This, in turn, reduces the amount of reflection required at runtime. *Vert.x*, on the other hand, takes a more runtime approach and relies more heavily on dynamic features and runtime reflection.

In Summary, Micronaut Framework focuses on optimizing performance, enables compile-time DI, and provides efficient Java support, while Vert.x emphasizes its polyglot nature, event-driven architecture, and flexible web support.

*[ source : stackshare.io ]*

So basically after trying out a number of the technologies listed thereof, I settled on vert.x.



Starting a vertx project is as simple as getting your preferred IDE eg. IntelliJ Community Package version or Eclipse or Visual Studio Code and creating a maven or gradle project and ensuring a dependency to and for io.vertx is set in place.  
  
Or you can start from the vertx starter link to generate a downloadable package or project to start from after successfully importing into the/an IDE as a project.

You can use the Vert.x starter web application at https://start.vertx.io and generate a project skeleton to download



After successfully importing the project,

You can then run the following command :

$ mvn clean install

You can expect some <Build Successful> kind of message at the end to be sure that everything worked out perfectly, as seen below :

[INFO] Copying com.aireceive.firstvertxapp:starter:pom:1.0.0-SNAPSHOT to project local repository

[INFO] Copying com.aireceive.firstvertxapp:starter:jar:1.0.0-SNAPSHOT to project local repository

[INFO] Copying com.aireceive.firstvertxapp:starter:pom:consumer:1.0.0-SNAPSHOT to project local repository

[INFO] --------------------------------------------------------------------------------------------------------------------------

[INFO] **BUILD SUCCESS**

[INFO] --------------------------------------------------------------------------------------------------------------------------

[INFO] Total time: 3.020 s

[INFO] Finished at: 2024-11-04T14:38:33Z

[INFO] --------------------------------------------------------------------------------------------------------------------------

To install the application and the respective dependencies.

**Caution :**

I tried the following commands and it did not work out as expected :

$ vertx run com.arieceive.firstvertxapp.starter.MainMerticle

And the application did not run but rather I had the following error :

Unrecognized VM option 'UseBiasedLocking'

Error: Could not create the Java Virtual Machine.

Error: A fatal exception has occurred. Program will exit.

I also tried the same command and later this error :

Can not find io.netty.resolver.dns.macos.MacOSDnsServerAddressStreamProvider in the classpath, fallback to system defaults. This may result in incorrect DNS resolutions on MacOS. Check whether you have a dependency on 'io.netty:netty-resolver-dns-native-macos'

Failed in deploying verticle

java.lang.ClassNotFoundException: com.arieceive.firstvertxapp.starter.MainMerticle

at java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.java:581)

at java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.java:178)

at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:527)

at io.vertx.core.impl.JavaVerticleFactory.createVerticle(JavaVerticleFactory.java:41)

at io.vertx.core.impl.VerticleManager.doDeployVerticle(VerticleManager.java:217)

at io.vertx.core.impl.VerticleManager.doDeployVerticle(VerticleManager.java:193)

at io.vertx.core.impl.VerticleManager.doDeployVerticle(VerticleManager.java:180)

at io.vertx.core.impl.VerticleManager.deployVerticle(VerticleManager.java:156)

at io.vertx.core.impl.VertxImpl.deployVerticle(VertxImpl.java:794)

at io.vertx.core.impl.VertxImpl.deployVerticle(VertxImpl.java:800)

at io.vertx.core.impl.launcher.commands.VertxIsolatedDeployer.deploy(VertxIsolatedDeployer.java:42)

at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)

at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

at java.base/java.lang.reflect.Method.invoke(Method.java:566)

at io.vertx.core.impl.launcher.commands.ClasspathHandler.deploy(ClasspathHandler.java:169)

at io.vertx.core.impl.launcher.commands.RunCommand.deploy(RunCommand.java:398)

at io.vertx.core.impl.launcher.commands.RunCommand.run(RunCommand.java:260)

at io.vertx.core.impl.launcher.VertxCommandLauncher.execute(VertxCommandLauncher.java:248)

at io.vertx.core.impl.launcher.VertxCommandLauncher.dispatch(VertxCommandLauncher.java:383)

at io.vertx.core.impl.launcher.VertxCommandLauncher.dispatch(VertxCommandLauncher.java:346)

at io.vertx.core.Launcher.main(Launcher.java:45)

So the next thing was to try running the generated directly.

I tried : ( from the root folder of the project )

$ java -jar target/starter-1.0.0-SNAPSHOT.jar

And I got this feedback :

no main manifest attribute, in target/starter-1.0.0-SNAPSHOT.jar

What worked out find in getting application running :

But at last, after trying this :

$ java -jar target/starter-1.0.0-SNAPSHOT-fat.jar

I was then able to get the application running with a successful test using a web browser.  
  
Nov 04, 2024 2:34:20 PM io.netty.resolver.dns.DnsServerAddressStreamProviders <clinit>

WARNING: Can not find io.netty.resolver.dns.macos.MacOSDnsServerAddressStreamProvider in the classpath, fallback to system defaults. This may result in incorrect DNS resolutions on MacOS. Check whether you have a dependency on 'io.netty:netty-resolver-dns-native-macos'

HTTP server started on port 8888

Nov 04, 2024 2:34:20 PM io.vertx.core.impl.launcher.commands.VertxIsolatedDeployer

INFO: Succeeded in deploying verticle

My first vertx web app :

[INFO] Copying com.aireceive.firstvertxwebapp:vertx-web-starter:pom:1.0.0-SNAPSHOT to project local repository

[INFO] Copying com.aireceive.firstvertxwebapp:vertx-web-starter:jar:1.0.0-SNAPSHOT to project local repository

[INFO] Copying com.aireceive.firstvertxwebapp:vertx-web-starter:pom:consumer:1.0.0-SNAPSHOT to project local repository

[INFO] --------------------------------------------------------------------------------------------------------------------------

[INFO] **BUILD SUCCESS**

[INFO] --------------------------------------------------------------------------------------------------------------------------

[INFO] Total time: 3.472 s

[INFO] Finished at: 2024-11-04T16:07:02Z

[INFO] --------------------------------------------------------------------------------------------------------------------------

➜ vertx-web-starter git:(main) ✗ java -jar target/vertx-web-starter-1.0.0-SNAPSHOT-fat.jar

Nov 04, 2024 4:07:18 PM io.netty.resolver.dns.DnsServerAddressStreamProviders <clinit>

WARNING: Cannot find io.netty.resolver.dns.macos.MacOSDnsServerAddressStreamProvider in the classpath, fallback to system defaults. This may result in incorrect DNS resolutions on MacOS. Check whether you have a dependency on 'io.netty:netty-resolver-dns-native-macos'

HTTP server started on port 8889

Nov 04, 2024 4:07:18 PM io.vertx.core.impl.launcher.commands.VertxIsolatedDeployer

INFO: Succeeded in deploying verticle



Alternative to Vert.x for asynchronous and reactive programming :

1. NodeJS
2. Akka
3. Spring Framework
4. Quarkus
5. Netty
6. Scripting languages
7. Native languages

Vert.x is the best compared to all the above in so many ways, thanks to the JVM to its advantage. Some of the other options above might have their own benefits in various ways meanwhile for a high performant production system, Vertx would always win in over 82% of the key metrics been checked against.

Summary to this point :

1. Asynchronous programming allows you to handle multiple multiplex networked connections on a single thread.
2. Handling or managing non-blocking I/O is more complex than the equivalent imperative code base on blocking I/O, even for simple protocols.
3. Asynchronous event processing is simplified by the event loop and the reactor pattern.
4. Despite the demanding workloads and failures, a reactive system is both scalable and resilient, producing responses with consistent latencies.
5. Vert.x is an efficient and approachable toolkit for writing asynchronous and reactive applications on the JVM.

**Verticles**

A verticle is the fundamental processing unit in Vert.x, with a life cycle. The role of a verticle is fundamentally to envelope a single technical functional unit for processing events, such as ;

1. Exposing an HTTP API
2. Responding to requests
3. Providing a repository interface on top of a database
4. Issuing requests to a third-party system

Characteristics of Verticles :

1. Verticles exclusively can communicate with other entities by sending and responding to messages.
2. Verticles have private state that may be updated when receiving events, they can deploy other verticles.
3. Verticles can communicate via message-passing
4. Verticles do not necessarily follow the orthodox definition of actors

Now, let’s build a cute little verticle that processes just two types of events for now :

1. Periodic timers
2. Handle HTTP requests

The life cycle of a verticle is basically :

1. Start , and
2. Stop

The start method is typically used in setting things up and fundamentally initializing handlers.

The stop method is used in doing housekeeping tasks, such as closing an opened database connection(s).

To run a verticle, it can basically be run from the mian of the java class that it is in.

Or you can also run it on the command line using Gradle such as :

**$ ./gradlew run -PmainClass=com.example.hello.HelloVerticle**

**Some few experiments :**

So I had to run a few experiments. I decided to check if I could run a single verticle and rather apply two port numbers to see how the behaviour of a dual starting of the same verticle would be and to my surprise, it is only the first verticle start via either the terminal or by directly running it from the “main” method that responds to all the requests to the running verticle.



**May be just IDE trivial :**

So in the way I like to use my IDE (eg. InteliJ) is fundamentally that I have taken time to define different types of elements in the said programming language by certain specific colours. Now with all of the above I shared so far on my progress with Vert.x, the colours were not coming up . . . everything was just plain white.



In order to get the colours to show, after trying to set the project properties, or try to set the compiler levels and or types, none of those helped until , I added a pom to the root of my project and defined the respective modules by name. After adding the pom file to the root, just like magic, the IDE suddenly seems to have recovered and then started identifying each of the elements as expected.

