**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.



In the deployment of verticles it is important to pay attention to any action or function or task or tasks that might attempt to block the eventLoop or life cycle of the said verticle. In Vertx, we have a threadchecker that checks to be sure that no process or action or function or task is taking more than a predefined internally set thread checker time limit. The default might be found to be 2000 as of the time I am doing all these tests and experiments. Meanwhile, there are certain environments such as embedded devices, where processing power is a bit slower, and it is normal to increase the thread-checker threshold for such cases.

You can do that by setting the the system properties to change the value as seen below :

-Dvertx.options.blockedThreadCheckInterval=5000

Or if you prefer to disable it :

-Dvertx.threadChecks=false

It must be noted that this configuration is global and cannot be fine-tuned on a per-verticle basis.

It is such a great robust way to or practice to use asynchronous method variants that accept a callback to notify of any errors, example the listen method in the creation of an HTTP server using vertx.

Do check from screenshot below :



**Deploying Verticles:**

We can have a verticle successfully deploy another verticle and repeat same for another verticle. Meanwhile, there is not direct parent or child relationship setup for verticle deployment.

One other thing to note about verticles are that Vert.x creates double number of event-loop threads as the number of CPU Cores present, by default. Basically implies that if you have 4cores, then a Vert.x application has 8 event loops. And the assignment of the verticles to event loops is/are done in a rounded-robin fashion or style.

In the deployment of verticles and by extension the number of event loops, can be managed in a way so as to obtain a certain number of event loops that has to be available. Meanwhile it is not possible to allocate a given verticle to a specific event loop manually. In practice, this should never be a problem whatsoever. It must be noted that we can plan the deployment order of verticles.



**Passing Configuration Data**

**Worker Verticles**

**Threads Contextualisation**

**Threads Mix Monitoring and Management**

**Event Bus : A very important part of Vert.x**

There are three major patterns in event bus communications :

1. Point-to-point messaging
2. Request-reply messaging
3. Publish/subscribe messaging

The event bus is not a message broker but rather carries volatile event that are been processed asynchronously by verticles, more specifically the Vert.x event bus is not able to do the following :

1. Support message acknowledgements
2. Support message priorities
3. Support message durability to recover from crashes
4. Provide routing rules
5. Provide transformation rules (schema adaptation, scatter/gather, etc.)
6. Distribute fewer messages to an overloaded consumer.

If the developer or engineer wants a way to handle communications relating to events without ever loosing them then a middleware would be very necessary.

**Point-to-Point messaging**

Messages from producers are shared proportionally in a round-robin fashion amongst consumers.

**Latter Sudden challenges in getting vert.x running smoothly & the way out :**

As I progressed into the deeper parts of vert.x development, deployment and its general usage. I was trying to run some vert.x application using version 4.5.10 and I had this error below :

Nov 06, 2024 11:58:45 AM 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'

I then compared the ‘pom.xml’ (under properties) file from a previous successfully running vert.x application :

<properties>  
 <maven.compiler.source>1.8</maven.compiler.source>  
 <maven.compiler.target>1.8</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <vertx.version>4.0.3</vertx.version>

The new vertx application’s pom.xml file under the properties :

<properties>  
 <maven.compiler.source>1.8</maven.compiler.source>  
 <maven.compiler.target>1.8</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <vertx.version>4.5.10</vertx.version>

The new vertx application’s pom.xml file above was then changed to :

<properties>  
 <maven.compiler.source>1.8</maven.compiler.source>  
 <maven.compiler.target>1.8</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <vertx.version>4.0.3</vertx.version>

The application was then able to run smoothly without any issues whatsoever.

Access the index.html file :

On a macbook M1 using inteliJ and with the “index.html” at the root of the project, the file could not be rendered.

The file was only able to be rendered after placing the file inside the “resource” folder or package.

**Clustering and distributed event bus**

In attempting to run a cluster that would have brought the application development in vertx to a whole new level of deployment and management, I started getting errors to ensure some infinispan.xml is created before the cluster would run.

I tried running from the main : did not work without the infinispan.xml

I tried running using ./gradlew run : did not work without the infinispan.xml

I tried running using mvn compile exec:java : did not work without the infinispan.xml

**Infinispan Replicated Cache**



**Starting Infinispan Server**

Before you start coding fun stuff, you need to start Infinispan Server. For this tutorial, you need a locally running server instance.

You can do one of the following:

* Pull the container image and run with [Docker](https://www.docker.com/) or [Podman](https://podman.io/).
* Download the server distribution and extract it to your filesystem.

Credentials

By default, Infinispan Server requires user authentication. This tutorial uses admin and secret credentials but you can use any username and password.

**Running the Container Image**

The easiest way to run Infinispan Server locally is to pull the container image.

* Podman

podman run --net=host -p 11222:11222 -e USER="admin" -e PASS="secret" quay.io/infinispan/server:latest

* Docker

docker run -it -p 11222:11222 -e USER="admin" -e PASS="password" infinispan/server:latest

**Running the Server Distribution**

Infinispan Server comes as a bare metal distribution that you can run locally.

1. Download the server distribution from [Infinispan Downloads](https://infinispan.org/download/#stable) and extract it.
2. Open a terminal window in the resulting directory. This is $ISPN\_HOME.
3. Add credentials.

$ ./bin/cli.sh user create admin -p secret

Run Infinispan Server.

$ ./bin/server.sh

**Accessing the Infinispan Console**

Open <http://localhost:11222/> in any browser.

You’ll see the **Welcome to Infinispan Server** page.

[](https://github.com/infinispan/infinispan-server-tutorial/blob/main/images/welcomeConsole.png)

To start using the Infinispan Console, do the following:

1. Select **Go to the console**.
2. Enter your credentials (admin/secret).

Successfully have gotten infinispan running :



I then went on to generate this configuration in XML and JSON :

XML :

<?xml version="1.0"?>

<distributed-cache owners="2" mode="ASYNC" statistics="true">

<encoding media-type="application/x-protostream"/>

<locking concurrency-level="32" acquire-timeout="10"/>

<memory max-size="1024MB" when-full="REMOVE"/>

<persistence passivation="false" availability-interval="1000" connection-attempts="10" connection-interval="50">

<file-store>

<data path="data"/>

<index path="index"/>

</file-store>

</persistence>

</distributed-cache>

Improved to :

<distributed-cache *owners*="2" *mode*="ASYNC" *statistics*="true">  
 <encoding *media-type*="application/x-protostream"/>  
 <locking *concurrency-level*="32" *acquire-timeout*="10"/>  
 <memory *max-size*="1024MB" *when-full*="REMOVE"/>  
 <persistence *passivation*="false" *availability-interval*="1000" *connection-attempts*="10" *connection-interval*="50">  
 <file-store>  
 <data *path*="data"/>  
 <index *path*="index"/>  
 </file-store>  
 </persistence>  
 <infinispan>  
 <cache-container>  
 <transport *initial-cluster-size*="4"  
 *initial-cluster-timeout*="30000" />  
 </cache-container>  
 </infinispan>  
</distributed-cache>

Changed to :

<distributed-cache>

<memory storage="HEAP"/>

</distributed-cache>

Changed to :

<distributed-cache>  
 <memory  
 *storage*="HEAP"  
 *max-count*="500"  
 *when-full*="REMOVE"  
 />  
 <infinispan>  
 <cache-container>  
 <transport *initial-cluster-size*="4"  
 *initial-cluster-timeout*="30000" />  
 </cache-container>  
 </infinispan>  
  
</distributed-cache>

JSON :

{

"distributed-cache": {

"owners": "2",

"mode": "ASYNC",

"statistics": true,

"encoding": {

"media-type": "application/x-protostream"

},

"locking": {

"concurrency-level": "32",

"acquire-timeout": "10"

},

"memory": {

"max-size": "1024MB",

"when-full": "REMOVE"

},

"persistence": {

"passivation": false,

"availability-interval": "1000",

"connection-attempts": "10",

"connection-interval": "50",

"file-store": {

"data": {

"path": "data"

},

"index": {

"path": "index"

}

}

}

}

}

I then copied the XML version into the infinispan.xml file.

**Replicated cache:**

<replicated-cache>

<expiration lifespan="5000" max-idle="1000" />

</replicated-cache>

<replicated-cache>  
 <expiration *lifespan*="5000" *max-idle*="1000" />  
 <memory *storage*="OFF\_HEAP" *max-count*="500"/>  
  
 <infinispan>  
 <cache-container>  
 <transport *initial-cluster-size*="4"  
 *initial-cluster-timeout*="30000" />  
 </cache-container>  
 </infinispan>  
  
</replicated-cache>

<?*xml version*="1.0"?>  
<?*xml version*="1.0" *encoding*="UTF-8"?>  
<infinispan *xmlns:xsi*="http://www.w3.org/2001/XMLSchema-instance"  
 *xsi:schemaLocation*="urn:infinispan:config:7.1 http://www.infinispan.org/schemas/infinispan-config-7.1.xsd"  
 *xmlns*="urn:infinispan:config:7.1">  
  
 <cache-container *default-cache*="default">  
 <local-cache *name*="tableCache">  
 <eviction *max-entries*="200" />  
 <expiration *lifespan*="600000" />  
 </local-cache>  
 <transport  
 *initial-cluster-size*="4"  
 *initial-cluster-timeout*="30000" />  
 </cache-container>  
  
</infinispan>

Other configurations :

<distributed-cache>

<persistence>

<table-jdbc-store xmlns="urn:infinispan:config:store:sql:15.0"

dialect="H2"

shared="true"

table-name="books">

<connection-pool connection-url="jdbc:h2:mem:infinispan"

username="sa"

password="changeme"

driver="org.h2.Driver"/>

<write-behind modification-queue-size="2048"

fail-silently="true"/>

</table-jdbc-store>

</persistence>

</distributed-cache>

<distributed-cache>

<persistence>

<connection-pool connection-url="jdbc:h2:mem:infinispan;DB\_CLOSE\_DELAY=-1"

username="sa"

password="changeme"

driver="org.h2.Driver"/>

</persistence>

</distributed-cache>

##### Managed datasource configuration

XML

JSON

YAML

<server xmlns="urn:infinispan:server:15.0">

<data-sources>

<!-- Defines a unique name for the datasource and JNDI name that you

reference in JDBC cache store configuration.

Enables statistics for the datasource, if required. -->

<data-source name="ds"

jndi-name="jdbc/postgres"

statistics="true">

<!-- Specifies the JDBC driver that creates connections. -->

<connection-factory driver="org.postgresql.Driver"

url="jdbc:postgresql://localhost:5432/postgres"

username="postgres"

password="changeme">

<!-- Sets optional JDBC driver-specific connection properties. -->

<connection-property name="name">value</connection-property>

</connection-factory>

<!-- Defines connection pool tuning properties. -->

<connection-pool initial-size="1"

max-size="10"

min-size="3"

background-validation="1000"

idle-removal="1"

blocking-timeout="1000"

leak-detection="10000"/>

</data-source>

</data-sources>

</server>

##### Connection pool tuning properties

You can tune JDBC connection pools for managed datasources in your Infinispan Server configuration.

| **Property** | **Description** |
| --- | --- |
| initial-size | Initial number of connections the pool should hold. |
| max-size | Maximum number of connections in the pool. |
| min-size | Minimum number of connections the pool should hold. |
| blocking-timeout | Maximum time in milliseconds to block while waiting for a connection before throwing an exception. This will never throw an exception if creating a new connection takes an inordinately long period of time. Default is 0 meaning that a call will wait indefinitely. |
| background-validation | Time in milliseconds between background validation runs. A duration of 0 means that this feature is disabled. |
| validate-on-acquisition | Connections idle for longer than this time, specified in milliseconds, are validated before being acquired (foreground validation). A duration of 0 means that this feature is disabled. |
| idle-removal | Time in minutes a connection has to be idle before it can be removed. |
| leak-detection | Time in milliseconds a connection has to be held before a leak warning. |

After entering the command :

$ ./gradlew run

[ for vertx version 4.0.3 ] : The error message was seen below :

ERROR [vert.x-eventloop-thread-0] VertxImpl - Failed to initialize clustered Vert.x

org.infinispan.commons.CacheConfigurationException: ISPN000327: Cannot find a parser for element 'distributed-cache' in namespace ''. Check that your configuration is up-to date for Infinispan '11.0.5.Final' and if you have the proper dependency in the classpath

[ for vertx version 4.4.4 ] : The error message was seen below :

ERROR [vert.x-eventloop-thread-0] VertxImpl - Failed to initialize clustered Vert.x

org.infinispan.commons.CacheConfigurationException: ISPN000343: Must have a transport set in the global configuration in order to define a clustered cache

##### Distributed caches

XML

JSON

YAML

<distributed-cache owners="2"

segments="256"

capacity-factor="1.0"

l1-lifespan="5000"

mode="SYNC"

statistics="true">

<encoding media-type="application/x-protostream"/>

<locking isolation="REPEATABLE\_READ"/>

<transaction mode="FULL\_XA"

locking="OPTIMISTIC"/>

<expiration lifespan="5000"

max-idle="1000" />

<memory max-count="1000000"

when-full="REMOVE"/>

<indexing enabled="true"

storage="local-heap">

<index-reader refresh-interval="1000"/>

<indexed-entities>

<indexed-entity>org.infinispan.Person</indexed-entity>

</indexed-entities>

</indexing>

<partition-handling when-split="ALLOW\_READ\_WRITES"

merge-policy="PREFERRED\_NON\_NULL"/>

<persistence passivation="false">

<!-- Persistent storage configuration. -->

</persistence>

</distributed-cache>

##### Replicated caches

XML

JSON

YAML

<replicated-cache segments="256"

mode="SYNC"

statistics="true">

<encoding media-type="application/x-protostream"/>

<locking isolation="REPEATABLE\_READ"/>

<transaction mode="FULL\_XA"

locking="OPTIMISTIC"/>

<expiration lifespan="5000"

max-idle="1000" />

<memory max-count="1000000"

when-full="REMOVE"/>

<indexing enabled="true"

storage="local-heap">

<index-reader refresh-interval="1000"/>

<indexed-entities>

<indexed-entity>org.infinispan.Person</indexed-entity>

</indexed-entities>

</indexing>

<partition-handling when-split="ALLOW\_READ\_WRITES"

merge-policy="PREFERRED\_NON\_NULL"/>

<persistence passivation="false">

<!-- Persistent storage configuration. -->

</persistence>

</replicated-cache>

##### Multiple caches

XML

JSON

YAML

<infinispan

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="urn:infinispan:config:15.0 https://infinispan.org/schemas/infinispan-config-15.0.xsd

urn:infinispan:server:15.0 https://infinispan.org/schemas/infinispan-server-15.0.xsd"

xmlns="urn:infinispan:config:15.0"

xmlns:server="urn:infinispan:server:15.0">

<cache-container name="default"

statistics="true">

<distributed-cache name="mycacheone"

mode="ASYNC"

statistics="true">

<encoding media-type="application/x-protostream"/>

<expiration lifespan="300000"/>

<memory max-size="400MB"

when-full="REMOVE"/>

</distributed-cache>

<distributed-cache name="mycachetwo"

mode="SYNC"

statistics="true">

<encoding media-type="application/x-protostream"/>

<expiration lifespan="300000"/>

<memory max-size="400MB"

when-full="REMOVE"/>

</distributed-cache>

</cache-container>

</infinispan>

After trying all possible ways of getting the infinispan right, but with no success, I rather resorted to what I do when I stay with a problem for just so so long and there is still no breakthrough : I ask myself : “Is this going to be needed in production ? and is it the best for the current challenge or problem I ultimately need to solve, globally ?

So I went on to do a quick search :

And in my discovery, I came to realise, I needed to leave the problem behind and rather solve a more relevant one : how to deploy an enterprise application or server solution with all the high standards and not just some fancy localhost or localside ephemeral solutions with infinispan.

Now the twist is that :

Both Hazelcast and Infinispan, by default use multicast communications to discover nodes. This is great for local testing and many bare-metal server deployments, but multicast communications are not possible in a Kubernetes

cluster. If you run the containers as is on Kubernetes, the heat sensor services and sensor gateway instances will not be able to communicate over the event bus.

These cluster managers can, of course, be configured to perform service discovery in Kubernetes. We will briefly cover the case of **Hazelcast**, where two discovery modes are possible:

 Hazelcast can connect to the Kubernetes API to listen for and discover pods matching a request, such as a desired label and value.

 Hazelcast can periodically make DNS queries to discover all pods for a given Kubernetes (headless) service.

The DNS approach is more limited.

Instead, let’s use the Kubernetes API and configure Hazelcast to use it. By default, the Hazelcast Vert.x cluster manager reads configuration from a cluster.xml resource. The following listing shows the relevant configuration excerpt of the heat-sensor-service/

src/main/resource/cluster.xml file.

2024/Nov/10th : successful live broadcasting using vertx.

Push and pull/fetch modes.

**Beyond callbacks :**

**Futures & Promises :**

@Override

public void start(Promise<Void> promise) {

vertx.createHttpServer()

.requestHandler(this::handleRequest)

.listen(8080) //returns a future <HttpServer>

.onFailure(promise::fail) //called when the server could not be started

.onSuccess(ok -> {//called on success

System.out.println("http://localhost:8080/");

promise.complete();

});

}

/\*

The listen method(s) or prior httpservers that we saw in earlier examples took a callback, and here it returns a Future<HttpServer>. We then can chain calls to onFailure and onSuccess to define what to do when the server starts, or when an error occurs.

The promise/future interfaces starting from Vert.x 3.8, but the future-based APIs are only available in Vert.x 4.

\*/

**Interoperability with CompletionStage APIs**

CompletionStage<String> cs = promise.future().toCompletionStage();//converts a future to a completionStage

cs

.thenApply(String::toUpperCase) //just like map in Future

.thenApply(str -> "~~~ " + str) //just like map in Future

.whenComplete((str, err) -> {//takes a value or error (ie. str or err)

if (err == null) {

System.out.println(str);

} else {

System.out.println("Oh... " + err.getMessage());

}

});

/\*

we convert the string result to uppercase, prefix it with a string, and eventually call

whenComplete. Note that this is a BiConsumer, and you need to test which of the values or exception parameters is null to know whether the promise completed successfully.

It is also important to note that unless you call an asynchronous CompletionStage

method, the calls are performed on a Vert.x thread.

\*/

**Convert a CompletionStage to a Future**

CompletableFuture<String> cf = CompletableFuture.supplyAsync(() -> {

try {

Thread.sleep(5000);

} catch (InterruptedException e) {

e.printStackTrace();

}

return "5 seconds have elapsed";

});

Future

.fromCompletionStage(cf, vertx.getOrCreateContext())

.onSuccess(System.out::println)

.onFailure(Throwable::printStackTrace);

**Reactive extensions**

Aka : Observable(s) design pattern

Reactive extensions are defined by three things:

 Observing event or data streams (e.g., an incoming HTTP request can be observed)

 Composing operators to transform streams (e.g., merge multiple HTTP request streams as one)

 Subscribing to streams and reacting to events and errors

RxJava v1.x.x does not support back-pressure

RxJava v2.x.x supports back-pressure

**Observable<T>**

* A stream of events of type T. Does not support back-pressure.
* Timer events, observable source where we cannot apply back-pressure like GUI events

**Flowable<T>**

* A stream of events of type T where back-pressure can be applied
* Network data, filesystem inputs

**Single<T>**

* A source that emits exactly one event of type T
* Fetching an entry from a data store by key

**Maybe<T>**

* A source that may emit one event of type T, or none
* Fetching an entry from a data store by key, but the key may not exist

**Completable**

* A source that notifies of some action having completed, but no value is being given
* Deleting files

**Hot Source :**

1. Would still emit event even when there are no subscribers.
2. The subscriber does not get all the events.
3. The subscriber only gets events from the time or period of subscription.
4. Example of hot source(ing) is a periodic timer.

**Cold Source :**

1. Would only emit event if there are subscribers.
2. The subscriber does get all the events.
3. The subscriber only gets events from the time or period of subscription.
4. Example of cold source(ing) is reading file content.

private Single<JsonObject> sendToSnapshot(Single<JsonObject> data) {

return data.flatMap(json -> webClient

.post(4000, "localhost", "")

.expect(ResponsePredicate.SC\_SUCCESS)

.rxSendJsonObject(json)

.flatMap(resp -> Single.just(json))); //and then…

}

**Coroutines**

**Edge Services using coroutines**

**Coroutine trivia :**

“as” is a keyword in Kotlin, so it has to be escaped when used as a method name.

**Choosing asynchronous model appropriate for your project :**

**Some major summary points :**

 Callbacks have expressiveness limitations in relation to composing asynchronous operations, and they can yield that is harder to understand if care is not taken.

 Parallel and sequential asynchronous operations can be composed with other (pure)

asynchronous programming models: futures and promises, reactive extensions,

and coroutines.

 Reactive extensions have an advanced set of composable operators very well suited for event streams.

 Futures and promises are perfect for simple chaining of asynchronous operations.

 Kotlin coroutines affords a language-level support for asynchronous operations.

 There is no universally perfect asynchronous programming model. The choice truly depends on the case of use of such async programming model. The exciting thing about Vert.x is that you can combine these models depending on the challenge or problem domains at hand.

**Beyond the event bus**

**Designing a reactive application**

**Criteria of a reactive application:**

 Back-pressure, as a necessary ingredient in asynchronous stream processing to regulate event throughput.

 Reactive programming as a way to compose asynchronous operations

**One application many services**

**Databases**

PostgresSQL

MongoDB

**Queue Management Services**

**SMTP**

**Event Stats Services**

**Running the application**

**The Web Stack**

1. The construction of an edge service and a public API
2. The Vert.x web client
3. JSON web tokens (JWT) and cross-origin resource sharing (CORS)
4. Serving and integrating a Vue.js reactive application with Vert.x
5. Testing an HTTP API with REST Assured

**Elements from the vertx web stack :**

Advanced routing

Routing with regex

Authentication

HTTP Client

Cookies

Server side session

Server side template rendering

Cross-site request forgery protection

**Modules required :**

Web (router, request handler, HTTP request dispatcher, handler : {

BodyHandler : {

HTTP authentication,

CORS,

CSRF,

Favicon,

HTTP session,

Serving static files,

Virtual Host,

Template rendering

}

})

Client

Auth-JWT

**Routing HTTP Requests**

BodyHandler bodyHandler = BodyHandler.create();// BodyHandler is a predefined handler that extracts HTTP request body payloads.

router.post().handler(bodyHandler);// Here bodyHandler is called for all HTTP POST and PUT requests.

router.put().handler(bodyHandler);

String prefix = "/api/v1";

router.post(prefix + "/register").handler(this::register);

router.post(prefix + "/token").handler(this::token);

// (...) defines jwtHandler, more later

router.get(prefix + "/:username/:year/:month")//We can extract path parameters by prefixing elements with ":".

.handler(jwtHandler) //Handlers can be chained

.handler(this::checkUser)

.handler(this::monthlySteps);

// (...)

**Making HTTP Requests**

**Creating a JWT Handler**

**JWT Handler in a route**

**Checking that a valid JWT token is present**

**Issuing JWT Tokens in vertx**

Working with JWTs

Common JWT libraries make working with JWTs easy. For example, with JJWT1, creating a

new JWT is straightforward, as shown in Example 6-1.

Example 6-1 Creating and validating a signed JWT using JJWT

import java.time.Instant;

import java.time.temporal.ChronoUnit;

import java.util.Date;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jws;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

public class JwtExample {

public String createJwt(String secret) throws Exception {

// create and sign the JWT, including a hint

// for the key used to sign the request (kid)

String newJwt = Jwts.builder()

.setHeaderParam("kid", "meaningfulName")

.setSubject("user-12345")

.setAudience("user")

.setIssuedAt(Date.from(Instant.now()))

.setExpiration(Date.from(Instant.now().plus(15, ChronoUnit.MINUTES)))

.signWith(SignatureAlgorithm.HS512, secret)

.compact();

return newJwt;

}

public void validateJwt(String jwtParameter, String secret) throws Exception {

// Validate the Signed JWT!

// Exceptions thrown if not valid

Jws<Claims> jwt = Jwts.parser()

.setSigningKey(secret)

.parseClaimsJws(jwtParameter);

// Inspect the claims, like make a new JWT

// (need a signing key for this)

Claims jwtClaims = jwt.getBody();

System.out.println(jwtClaims.getAudience());

System.out.println(jwtClaims.getIssuer());

System.out.println(jwtClaims.getSubject());

System.out.println(jwtClaims.getExpiration());

System.out.println(jwtClaims.getIssuedAt());

System.out.println(jwtClaims.getNotBefore());

}

}

**Cross-Origin Resource Sharing ( C.O.R.S )**

In vertx Corsehandler we can specify :

1. Allowed Origin path
2. Allowed HTTP Header(s)
3. Allowed HTTP Method(s)

Example :

Set<String> allowedHeaders = new HashSet<>();

allowedHeaders.add("x-requested-with");

allowedHeaders.add("Access-Control-Allow-Origin");

allowedHeaders.add("origin");

allowedHeaders.add("Content-Type");

allowedHeaders.add("accept");

allowedHeaders.add("Authorization");

Set<HttpMethod> allowedMethods = new HashSet<>();

allowedMethods.add(HttpMethod.GET);

allowedMethods.add(HttpMethod.POST);

allowedMethods.add(HttpMethod.OPTIONS);

allowedMethods.add(HttpMethod.PUT);

router.route().handler(

CorsHandler

.create("\*")//A CORS handler for all routes

.allowedHeaders(allowedHeaders)

.allowedMethods(allowedMethods)

);

**Checking C.O.R.S support**

**A modern web frontend with vertx**

**A canvas of Vue.js components**

**VueJS application structure and build integration**

**VueJS & Vertx**

$ npm -version

$ brew upgrade npm

$ npm -version

$ sudo npm install -g @vue/cli

$ vue -version

$ cd <path-to-folder>

$ mkdir <name-of-project>

$ vue create <name-of-vue-project>

$ npm run serve

//sample output :

**➜ basic-vuejs-project** **git:(master) $** npm run serve

> basic-vuejs-project@0.1.0 serve

> vue-cli-service serve

INFO Starting development server...

DONE Compiled successfully in 2848ms 2024-11-25 8:43:06 AM

App running at:

- Local: http://localhost:**8080**/

- Network: http://192.000.111.222:**8080**/

Note that the development build is not optimized.

To create a production build, run

npm run build.

**VueJS + Vite + TailwindCSS**

**CLIENT SIDE LOGIC | JAVASCRIPT | PAGE(S) FLUIDITY**

Then input this into the terminal :

**npm init vite vue**

<press Enter>

? Select a framework: › - Use arrow-keys. Return to submit.

❯   Vanilla

    Vue

    React

    Preact

    Lit

    Svelte

    Solid

    Qwik

    Others

**Choose : Vue**

? Select a variant: › - Use arrow-keys. Return to submit.

❯   TypeScript

    JavaScript

    Customize with create-vue ↗

    Nuxt ↗

**Choose : Javascript**

Done. Now run:

**cd vue**

**npm install**

**npm run dev**

vue % npm install

vue % npm run dev

> vue@0.0.0 dev

> vite

  VITE v5.2.11  ready in 950 ms

  ➜  Local:   http://localhost:5173/

  ➜  Network: use --host to expose

  ➜  press h + enter to show help

h

  Shortcuts

  press r + enter to restart the server

  press u + enter to show server url

  press o + enter to open in browser

  press c + enter to clear console

  press q + enter to quit

u

  ➜  Local:   http://localhost:5173/

  ➜  Network: use --host to expose

o

vue % **npm install -S vuex@next**

or :

**npm install vue-router vuex axios**

Remember to remove the  ‘setup’ tag from the <script setup></script>

Now we can comment out the <pre>{{user}}</pre> in the html of the App.vue

**STYLING :**

Upnext , let’s install Tailwindcss, by following the instructions at :

[tailwindcss.com/docs/guides/vite](http://tailwindcss.com/docs/guides/vite)

Commands :

**npm install -D tailwindcss postcss autoprefixer**

<press Enter>

**npx tailwindcss init -p**

<press Enter>

**npm install @headlessui/vue @heroicons/vue @tailwindcss/forms -S**

<press Enter>

We can get UI components here :

<https://tailwindui.com/components>

LESSON(S) LEARNT :

1. All the npm commands and the installations relating to the tailwindcss should all be made while at the root folder path of “vue” within the project in the terminal for all the designs and CSSes to work perfectly.

All the tailwindcss things with the form design is perfectly working now as of 1119GMT.

1. In creating “vue-routes” do not put a route there when its components have not been created yet. Stick with what exist. The best practice may be to create the component before you ever call it anywhere-else.
2. The  value for component in the set of unique path(s) should match the name given to the import at the top of the line.

At the point of linking the JS and functions such as dispatch, then and so on with promises, it is therefore necessary to install “ axios “ using npm.

Command :

**$ npm i -S axios**

If we ever need icons we can check out heroicons :

<https://www.heroicons.com>

### process 2 :

**npm install**

**CLIENT SIDE LOGIC | JAVASCRIPT | PAGE(S) FLUIDITY**

Then input this into the terminal :

**npm init vite vue**

<press Enter>

? Select a framework: › - Use arrow-keys. Return to submit.

❯   Vanilla

    Vue

    React

    Preact

    Lit

    Svelte

    Solid

    Qwik

    Others

**Choose : Vue**

? Select a variant: › - Use arrow-keys. Return to submit.

❯   TypeScript

    JavaScript

    Customize with create-vue ↗

    Nuxt ↗

**Choose : Javascript**

Scaffolding project in

airsurvey\_app/vue...

Done. Now run:

***cd vue***

***npm install***

***npm run dev***

vue % **npm install**

The next command was to help get vite + vue setup nicely :

**npm install vue-router vuex axios**

After the command is issued, set up the styling, the root folder still set to “ vue “ :

**STYLING :**

Upnext , let’s install Tailwindcss, by following the instructions at :

[tailwindcss.com/docs/guides/vite](http://tailwindcss.com/docs/guides/vite)

Commands :

**npm install -D tailwindcss postcss autoprefixer**

<press Enter>

**npx tailwindcss init -p**

<press Enter>

**npm install @headlessui/vue @heroicons/vue @tailwindcss/forms -S**

<press Enter>

## Had to paste this code into the “ tailwind.config.js”

module.exports = {

content: [

'./index.html',

'./src/\*\*/\*.{vue,js,ts,jsx,tsx}',

'./pages/\*\*/\*.{html,js}',

'./components/\*\*/\*.{html,js}',

],

theme: {

extend: {},

},

plugins: [

require('@tailwindcss/forms')

],

}

Running the app :

**npm run dev**

**npm run watch**

**npm run test**

>>>> 2024/11/26 :

This time it took me far lesser time to setup a vue.js project from scratch … just a few minutes, then testing to make sure it all works.

**VueJS router configuration**

**Backend integration in VueJS**

**HTML Template components**

<https://tailwindtemplates.io/templates>

<https://github.com/PhatStraw/Free-Tailwind-CSS-Templates-and-Components>

<https://github.com/slim-python/tailwind-css-free-components>

<https://github.com/markmead/hyperui>

<https://tw-elements.com/docs/standard/integrations/vue-integration/>

<https://www.tailwind-kit.com/components#forms>

<https://tailwindflex.com/>

<https://flowbite.com/docs/components/alerts/>

<https://flowbite.com/docs/getting-started/vue/>

<https://tailwindui.com/components/preview>

<https://tailwindui.com/components>

**Javascript code components**

**Using JWT Token with axios**

<https://jwt.io>

**Serving static content with Vertx**

**Writing integration tests**

Some fundamental tests set in the IntegrationTest class :

1 Register some users.

2 Get a JWT token for each user.

3 Fetch a user’s data.

4 Try to fetch the data of another user.

5 Update a user’s data.

6 Check some activity stats for a user.

7 Try to check the activity of another user.

Test dependencies to run the integration tests

Preamble of the integration test class

Preparing a REST assured request specification

Utility hash maps for the integration testing

Test for registering users

Test code for retrieving JWT tokens

Extracting JSON with REST assured

\*\*\* Attempt to run the test(s) : 2024 Nov., 25th

Setting up docker for Mac M1 : {

* Docker Desktop is free for small businesses (fewer than 250 employees AND less than $10 million in annual revenue), personal use, education, and non-commercial open source projects.
* Otherwise, it requires a paid subscription for professional use.
* Paid subscriptions are also required for government entities.
* Docker Pro, Team, and Business subscriptions include commercial use of Docker Desktop.

}

Setting up podman for Mac M1

Setting up Colima for Mac M1

install colima, and the docker CLI. We then launch a VM using special configuration flags to use macOS’s virtualisation layer and the translation of x86/amd64 into Apple Silicon.

Colima is only a wrapper to create Lima VMs.

Lima is the virtual machine that will run with the rosetta enhaced compatibility with x86 / amd64 and provide the docker runtime.

brew install colima # we will create the lima vm with this wrapper  
brew install docker # The CLI only

# Create and Configure a super performant vm

Note: adjust your settings (CPU, Memory and Disk according to your needs and hardware)

colima start \  
--profile default \  
--activate \  
--arch aarch64 \  
--cpu 10 \  
--disk 48 \  
--memory 24 \  
--mount ${HOME}:w \  
--mount-inotify \  
--ssh-agent \  
--vm-type vz \  
--vz-rosetta \  
--verbose

# Key Configuration Settings

* **arch**: AARCH64 specifies that we will run an ARM64 machine and not an x86\_64 machine
* **vm-type**: VZ (to use Apple’s Hypervisor.Framework)
* **vz-rosseta**: Enables Rosetta (needs macOS 13.0 or newer)

Reference on how to use Rosetta with Lima and the compatibility modes:

## [Intel-on-ARM and ARM-on-Intel](https://lima-vm.io/docs/config/multi-arch/?source=post_page-----da5100e2557d--------------------------------" \l "slow-mode" \t "_blank)

### [Lima supports two modes for running Intel-on-ARM and ARM-on-Intel: Lima can run a VM with a foreign architecture, just…](https://lima-vm.io/docs/config/multi-arch/?source=post_page-----da5100e2557d--------------------------------" \l "slow-mode" \t "_blank)

[lima-vm.io](https://lima-vm.io/docs/config/multi-arch/?source=post_page-----da5100e2557d--------------------------------" \l "slow-mode" \t "_blank)

# Configure the Shell to replace Docker

Wait for the machine to come up and then let’s setup the docker environment. (Actually, none of this is strictly necessary. You could use the bundled nerdctl tool.)

But follow along if you want a drop-in Docker replacement:

* Place those in your shell’s profile or in the current session at will.

export COLIMA\_VM="default"  
export COLIMA\_VM\_SOCKET="${HOME}/.colima/${COLIMA\_VM}/docker.sock"  
export DOCKER\_HOST="unix://${COLIMA\_VM\_SOCKET}"

# Bonus: Multiple machines simultaneously

At a certain point you might want to run more experiments or even split work / load / whatever.

For example, one limitation of **using Apple’s Hypervisor is that it is not possible to resize the VM’s Disk after being created**. So instead of destroying the VM and recreating it (losing all the containers), you can sidekick another alongside and split the load on that one.

Podman does not officially support this kind of workloads, though it is still possible to achieve the same.

# To create a secondary machine, just do issue another colima command

colima start \  
--profile secondary \  
--activate \  
--arch aarch64 \  
--cpu 1 \  
--disk 20\  
--memory 8 \  
--mount ${HOME}:w \  
--mount-inotify \  
--ssh-agent \  
--vm-type vz \  
--vz-rosetta \  
--verbose

export COLIMA\_VM="secondary"  
export COLIMA\_VM\_SOCKET="${HOME}/.colima/${COLIMA\_VM}/docker.sock"  
export DOCKER\_HOST="unix://${COLIMA\_VM\_SOCKET}"

# Introduction

This post introduces a streamlined method to set up a Podman machine (QEMU) on **Apple Silicon** **for running amd64 (x86\_64) containers**. **We explore two approaches**: **multi-architecture support** and **fully emulated** x86\_64 machines.

**Note**: I will keep every script here and the explanation in this **public repo** too**:** <https://github.com/guillem-riera/podman-machine-x86_64>

# Approach Overview

1. **Mixed Mode, Multi-Architecture Support**: This method enables support for multiple architectures, including x86\_64, on a standard aarch64 machine. It **maintains high performance for the native ARM images and has a performance impact on amd64 images**. It operates on a base aarch64 machine and compatibility with amd64 images is as good as the package qemu-user-static can provide (I haven’t tested for full compatibility).
2. **Full x86\_64 Emulation**: This offers **maximum compatibility at the cost of slower performance**. It’s a fully emulated x86\_64 machine, which means that the containers are also run in fully x86\_64 mode.

**Recommendation**: Always try the **first approach (mixed mode)** before considering the second.

# Requirements

To get started, ensure you have the following installed:

* Homebrew
* Homebrew bundle
* Podman
* QEMU (automatically included as a dependency of Podman)
* jq

The required packages are listed in the Brewfile. Install them using:

brew bundle install

# Setting Up

# 1. Multi-Arch Support on Current Podman Machine

This setup installs the necessary package **qemu-user-static** on your current machine.

This script facilitates this process:

export PODMAN\_MACHINE\_NAME=${PODMAN\_MACHINE\_NAME:-podman-machine-default}  
  
### Stop all podman machine instances  
ALL\_PODMAN\_MACHINES=$(podman machine list | awk '{ print $1 }' | tr -d '\*' | sed 1d | tr '\n' ' ')  
for PODMAN\_MACHINE in ${ALL\_PODMAN\_MACHINES}; do  
 podman machine stop ${PODMAN\_MACHINE}  
done  
  
### Start the target podman machine  
podman machine start ${PODMAN\_MACHINE\_NAME}  
  
### wait for the podman machine to be running  
PODMAN\_MACHINE\_STATUS=$(podman machine inspect ${PODMAN\_MACHINE\_NAME} | jq -r '.[].State')  
while [[ "${PODMAN\_MACHINE\_STATUS}" != "running" ]]; do  
 echo "[Info] Waiting for podman machine '${PODMAN\_MACHINE\_NAME}' to be running, current status: ${PODMAN\_MACHINE\_STATUS}..."  
 sleep 1  
 PODMAN\_MACHINE\_STATUS=$(podman machine inspect ${PODMAN\_MACHINE\_NAME} | jq -r '.[].State')  
done  
  
### Now that the podman machine is running we can install the package  
podman machine ssh "${PODMAN\_MACHINE\_NAME}" 'sudo rpm-ostree install qemu-user-static'  
  
### Stop the podman machine to apply the changes  
podman machine stop ${PODMAN\_MACHINE\_NAME}  
  
### Start the podman machine again  
podman machine start ${PODMAN\_MACHINE\_NAME}  
  
echo "[Info] Done. You can now run multi-architecture images in ${PODMAN\_MACHINE\_NAME}."

Podman can now run multi-architecture images with performance impacts limited to x86\_64 containers.

# How it works?

This bash script automates the setup of multi-architecture support for an existing Podman machine. Here’s a summary of how it works:

1. **Setup**: It sets the PODMAN\_MACHINE\_NAME variable, defaulting to "podman-machine-default" if not already specified.
2. **Stopping All Podman Machine Instances**: The script lists all existing Podman machines, excluding the header line and any active (marked with an asterisk) machines. It then stops each of these machines to ensure a clean setup environment.
3. **Starting the Target Podman Machine**: It starts the target Podman machine specified in PODMAN\_MACHINE\_NAME.
4. **Waiting for the Machine to Run**: The script continuously checks if the target Podman machine has reached the “running” state. It waits in a loop, checking the machine’s status every second.
5. **Installing the Package**: Once the target machine is running, the script remotely connects to it via SSH and installs the qemu-user-static package using sudo rpm-ostree install. This package is crucial for enabling multi-architecture support.
6. **Restarting the Podman Machine**: After the installation, the script stops the Podman machine to apply the changes and then starts it again

# 2. Full x86\_64 Emulation Setup

**Note**: Follow this step only if the first solution doesn’t meet your needs.

## Creating a new emulated Podman Machine (x86\_64)

The following script creates a podman machine and alters it to make it an x86\_64 machine (using QEMU):

# Setup the podman machine for x86\_64 (QEMU), supports only Apple Silicon (Mx) Macs  
  
# Keep all shell arguments in a variable to pass to the podman machine init command:  
EXTRA\_ARGS=${EXTRA\_ARGS:-$@}  
  
## 1. Download Fedora CoreOS image for x86\_64 (QEMU)  
PODMAN\_X86\_64\_MACHINE\_NAME=${PODMAN\_X86\_64\_MACHINE\_NAME:-x86\_64}  
PODMAN\_X86\_64\_MACHINE\_NAME\_EXISTS=$(podman machine list | grep ${PODMAN\_X86\_64\_MACHINE\_NAME} | wc -l | tr -d '[:space:]')  
PODMAN\_QEMU\_IMAGE="fedora-coreos-39.20231101.3.0-qemu.x86\_64.qcow2.xz"  
DOWNLOAD\_DIR=${DOWNLOAD\_DIR:-.}  
  
if [ ${PODMAN\_X86\_64\_MACHINE\_NAME\_EXISTS} -lt 1 ]; then  
 curl -C- -O "https://builds.coreos.fedoraproject.org/prod/streams/stable/builds/39.20231101.3.0/x86\_64/${PODMAN\_QEMU\_IMAGE}"  
 podman machine init --image-path ${DOWNLOAD\_DIR}/${PODMAN\_QEMU\_IMAGE} ${PODMAN\_X86\_64\_MACHINE\_NAME} ${EXTRA\_ARGS}  
else  
 echo "[Info] Machine ${PODMAN\_X86\_64\_MACHINE\_NAME} already exists. If you want to recreate it, run 'podman machine rm ${PODMAN\_X86\_64\_MACHINE\_NAME}'"  
fi  
  
## 2. Change machine settings  
  
### Get the machine config file name  
machineConfigFile="$(podman machine inspect ${PODMAN\_X86\_64\_MACHINE\_NAME} | jq -r '.[].ConfigPath.Path')"  
  
### Change the QEMU binary to x86\_64  
sed -i '' 's/qemu-system-aarch64/qemu-system-x86\_64/g' ${machineConfigFile}  
### Change the firmware to x86\_64  
sed -i '' 's/edk2-aarch64-code/edk2-x86\_64-code/g' ${machineConfigFile}  
### Delete the additional UEFI firmware file (ovmf) and the preceding '-drive' option. The '-drive' option is in a line above the line containing the path to 'x86\_64\_ovmf\_vars.fd'. Both lines must be deleted, but other -drive options must be kept.  
#### using sed to match 2 lines: '-drive' followed by 'x86\_64\_ovmf\_vars.fd'  
sed -i '' '/-drive/{N;/x86\_64\_ovmf\_vars.fd/d;}' ${machineConfigFile}  
### Delete the HVF (Hypervisor Framework) acceleration, which is only available for macOS. This are also 2 lines: '-accel' followed by 'hvf'  
sed -i '' '/-accel/{N;/hvf/d;}' ${machineConfigFile}  
### Delete the TCG acceleration, which seems to work only for Alpha and ARM architectures. This are also 2 lines: '-accel' followed by 'tcg'  
sed -i '' '/-accel/{N;/tcg/d;}' ${machineConfigFile}  
### Change the machine type to q35  
sed -i '' 's/virt,highmem=on/q35/g' ${machineConfigFile}  
### Change the cpu type from 'host' to 'qemu64'  
sed -i '' 's/host/qemu64/g' ${machineConfigFile}

# How it works?

This script is designed to set up a Podman machine specifically for x86\_64 architecture on Apple Silicon (Mx) Macs by modifying the QEMU template that podman generates when it creates a new machine.

Here’s a summary of its functionality and workflow:

**Shell Arguments**: The script stores any arguments passed to it in the EXTRA\_ARGS variable, which will later be used in the podman machine init command.

**Downloading** Fedora CoreOS Image for x86\_64 (QEMU):

* It sets a default name for the Podman x86\_64 machine (PODMAN\_X86\_64\_MACHINE\_NAME) and checks if a machine with this name already exists.
* If the machine does not exist, the script downloads the specified Fedora CoreOS image for x86\_64 using curl.
* After downloading, it initializes a new Podman machine with this image and any extra arguments provided.

**Changing Machine Settings**:

* The script retrieves the configuration file path of the newly created Podman machine.
* Several modifications are made to the machine’s configuration file to adapt it for x86\_64 emulation:
* **Changing QEMU Binary**: Updates the QEMU binary from qemu-system-aarch64 to qemu-system-x86\_64.
* **Changing Firmware**: Adjusts the firmware from edk2-aarch64-code to edk2-x86\_64-code.
* **Removing UEFI Firmware File**: Deletes lines related to the UEFI firmware file (x86\_64\_ovmf\_vars.fd) and its preceding '-drive' option.
* **Removing HVF Acceleration**: Eliminates the Hypervisor Framework (HVF) acceleration settings, as they are only available for macOS.
* **Removing TCG Acceleration**: Removes TCG acceleration settings, which are typically for Alpha and ARM architectures.
* **Changing Machine Type**: Updates the machine type from virt,highmem=on to q35.
* **Changing CPU Type**: Changes the CPU type from host to qemu64.

## Conclusion

The podman offers a convenient way to run x86\_64 containers on Apple Silicon, but you have to do extra steps to enable that.

Whether you require high performance or maximum compatibility, these methods provide a flexible solution to meet your containerization needs.

This is possible because QEMU, the underlaying virtualisation and emulation tool is really awesome!.

# Alternatives

If you are looking for a very high performance and fully open source alternative to Docker Desktop that supports x86\_64 / amd64 architecture with Rosetta, check my newer post on colima:

<https://github.com/guillem-riera/podman-machine-x86_64>

**Messaging and event streaming with Vertx**

Messaging with AMQP

Event streaming with Apache Kafka

Examples of messaging queue brokers :

* 1. AMQP : Advanced Messaging Queuing Protocol
  2. STOMP : Simple Text Oriented Messaging Protocol
  3. RabbitMQ Client
  4. MQTT : Messaging Queuing Telemetric Transport



Sending emails

Integration testing with messaging and event streaming

middleware

Event-Bus TCP Bridge

**Event driven services beyond HTTP with vertx**

**Ingestion from AMQP**

**AMQP client configuration :**

private AmqpClientOptions amqpConfig() {

return new AmqpClientOptions()

.setHost("localhost")//Credentials are the default ones from the docker image

.setPort(5672) //Credentials are the default ones from the docker image

.setUsername("artemis")//Credentials are the default ones from the docker image

.setPassword("simetraehcapa");//Credentials are the default ones from the docker image

}

// (...)

AmqpClientOptions amqpOptions = amqpConfig();

AmqpReceiverOptions receiverOptions = new AmqpReceiverOptions()

.setAutoAcknowledgement(false) //We would manually acknowledge incoming messages

.setDurable(true); //We want durable messages

**AMQP event-processing pipeline :**

AmqpClient.create(vertx, amqpOptions) //Create an AMQP client

.rxConnect()

.flatMap(conn -> conn.rxCreateReceiver("step-events", receiverOptions)) //Create a message receiver from the ‘step-events’ destination

.flatMapPublisher(AmqpReceiver::toFlowable) //Create a flowable AMQP messages

.doOnError(this::logAmqpError) //Error logging

.retryWhen(this::retryLater) //Retry logic

.subscribe(this::handleAmqpMessage); //Subscription that dispatches incoming messages

**Logging AMQP errors**

private void logAmqpError(Throwable err) {

logger.error("Woops AMQP", err);

}

**Recovering from errors with a delayed AMQP resubscription**

private Flowable<Throwable> retryLater(Flowable<Throwable> errs) {

return errs.delay(10, TimeUnit.SECONDS, RxHelper.scheduler(vertx));// It is important to use the scheduler parameter to process events on a Vert.x event loop

}

**Translating AMQP messages to Kafka records**

Handling AMQP messages

private void handleAmqpMessage(AmqpMessage message) {

if (!"application/json".equals(message.contentType()) || invalidIngestedJson(message.bodyAsJsonObject())) {

logger.error("Invalid AMQP message (discarded): {}", message.bodyAsBinary());

message.accepted();

return;

}

JsonObject payload = message.bodyAsJsonObject();

KafkaProducerRecord<String, JsonObject> record = makeKafkaRecord(payload);

updateProducer.rxSend(record).subscribe(

ok -> message.accepted(),

err -> {

logger.error("AMQP ingestion failed", err);

message.rejected();

});

}

Checking for valid JSON data

Preparing a kafka record

Configuring a Kafka Producer

HTTP server for ingestion

Ingesting updates from HTTP

Sending emails using MailHog

Listening for specific update events

Kafka RxJava Pipeline for receiving and processing specific updates

Predicate for certain specific events within the program

Creating an SMTP client

Mail client configuration

Implementation of the sendmail method

Request to retrieve the email address

Preparing an email message / template

Integration Tests

Ingestion testing

Ingest test preparation

AMQP Ingestion test preamble

AMQP Ingestion test: checking for Kafka record

Ingesting a bad JSON document

Sending of email testing

Testing for email NOT sent based on a Boolean check

**PERSISTENT STATE MANAGEMENT WITH DATABASES**

**Data/Object Mapping in Java: Advantages and Disadvantages**

Data/object mapping, often referred to as Object-Relational Mapping (ORM), is a technique that allows developers to interact with databases using object-oriented programming languages like Java. This approach bridges the gap between the relational database schema and the object-oriented model.

**Advantages**

1. **Improved Productivity:**
   * **Reduced boilerplate code:** ORM frameworks automate the repetitive tasks of SQL queries, result set parsing, and object instantiation.
   * **Simplified database interactions:** Developers can interact with the database using object-oriented concepts, making code more readable and maintainable.
2. **Enhanced Data Integrity:**
   * **Data consistency:** ORM frameworks can enforce data integrity constraints, such as unique keys and foreign key relationships, at the object level.
   * **Reduced data inconsistencies:** By abstracting database operations, ORM helps prevent errors that could lead to inconsistent data.
3. **Improved Code Reusability:**
   * **Reusable data access components:** ORM frameworks often provide a layer of abstraction that can be reused across different parts of an application.
   * **Simplified data access logic:** By encapsulating database interactions, ORM promotes code reusability and modularity.
4. **Facilitated Testing:**
   * **Mocking and stubbing:** ORM frameworks can be easily mocked or stubbed for unit testing, making it easier to isolate and test different parts of the application.
   * **Simplified test setup:** By abstracting database interactions, ORM reduces the complexity of setting up test environments.

**Disadvantages**

1. **Increased Complexity:**
   * **Steep learning curve:** ORM frameworks can have a significant learning curve, especially for developers new to the concept.
   * **Configuration overhead:** Configuring ORM frameworks can be complex and time-consuming, especially for large and complex applications.
2. **Performance Overhead:**
   * **Performance impact:** ORM frameworks can introduce performance overhead, particularly for complex queries and large datasets.
   * **Inefficient query generation:** Some ORM frameworks may generate inefficient SQL queries, leading to performance degradation.
3. **Vendor Lock-in:**
   * **Dependency on specific ORM framework:** Once an application is heavily reliant on an ORM framework, it can be difficult to switch to another framework or database system.
   * **Tight coupling:** ORM frameworks can tightly couple the application to a specific database technology, limiting flexibility.
4. **Limited Control Over Database Interactions:**
   * **Reduced flexibility:** ORM frameworks can limit the flexibility to write custom SQL queries or optimize database performance.
   * **Dependency on framework features:** Developers may need to rely on the specific features and limitations of the ORM framework.

**Conclusion**

While data/object mapping offers significant advantages in terms of productivity, data integrity, and code reusability, it's essential to weigh the potential disadvantages, such as increased complexity, performance overhead, and vendor lock-in. The decision to use an ORM framework should be based on the specific needs of the application and the development team's expertise.

<https://github.com/search?q=totp+generator+language%3AJava&type=repositories&l=Java>

**TOTP generation explained**

Time-based verification codes, often referred to as Time-based One-Time Passwords (TOTP), are generated using a cryptographic algorithm that relies on a shared secret key and the current time. Here's a breakdown of how it works:

**1. Shared Secret Key:**

* A unique, randomly generated secret key is created and shared between the user and the authentication server. This key is typically stored securely in the user's authentication app or device.

**2. Time-Based Counter:**

* The current Unix time (seconds since January 1, 1970) is retrieved.
* This time is divided by a predefined time interval (usually 30 or 60 seconds) to obtain a time-based counter value.

**3. HMAC-Based One-Time Password (HOTP) Algorithm:**

* The HOTP algorithm is used to generate the verification code. It takes the following inputs:
  + Shared secret key
  + Time-based counter value

* The algorithm applies a cryptographic hash function (such as HMAC-SHA1 or HMAC-SHA256) to these inputs to produce a hash value.
* The hash value is truncated to a specific number of digits (usually 6 or 8) to form the verification code.

**4. Verification Code Generation:**

* The generated verification code is displayed to the user on their authentication app or device.
* The user enters this code on the authentication website or app to verify their identity.

**5. Server-Side Verification:**

* The server performs the same calculations using the shared secret key and the current time to generate its own verification code.
* The server compares the user-provided code with the generated code.
* If the codes match within a certain time window (to account for clock skew), the authentication is successful.

**Key Points:**

* **Time-Based Validity:** The verification code is valid for a limited time (e.g., 30 or 60 seconds). After this time, a new code is generated.
* **Security:** The security of TOTP relies on the strength of the shared secret key and the cryptographic hash function used.
* **Flexibility:** TOTP can be implemented in various ways, using different time intervals and code lengths.
* **Wide Adoption:** TOTP is widely used in two-factor authentication (2FA) systems due to its simplicity and security.

By following these steps, TOTP provides a secure and convenient way to verify user identity without relying on traditional passwords.

**1. Install the JavaScript Obfuscator Module**

To proceed with the obfuscation of any JS code (for the browser, node.js, etc) with Node.js, you will need to rely on the JavaScript obfuscator module. JavaScript obfuscator is a powerful free obfuscator for JavaScript and Node.js with a wide number of features that provide protection for your source code. This module:

* has no limits or restrictions.
* runs on your local machine - does not send data to a server.
* compatible with es2015, es2016 and partially es2017.

You can install this module using the following command:

npm install javascript-obfuscator

After the installation you will be able to require the module in your scripts using require("javascript-obfuscator"). The obfuscator is free and open source (BSD-2-Clause licensed) and is written in TypeScript, you can see [an online implementation of the module here](https://javascriptobfuscator.herokuapp.com/). For more information about this library, please [visit the official repository at Github here](https://github.com/javascript-obfuscator/javascript-obfuscator).

## 2. Using the Obfuscator

The logic to obfuscate some code with the module is really simple. You create an instance of the Module, from the instance you can use the obfuscate method that expects as first argument the code that you want to obfuscate. This method returns the obfuscated synchronously. Through a series of transformations, such as variable / function / arguments renaming, strings removal, and others, your source code is transformed into something unreadable, while working exactly as before:

// Require the JavaScript obfuscator

var JavaScriptObfuscator = require('javascript-obfuscator');

// Obfuscate the code providen as first argument

var obfuscationResult = JavaScriptObfuscator.obfuscate(`

(function(){

var variable1 = '5' - 3;

var variable2 = '5' + 3;

var variable3 = '5' + - '2';

var variable4 = ['10','10','10','10','10'].map(parseInt);

var variable5 = 'foo ' + 1 + 1;

console.log(variable1);

console.log(variable2);

console.log(variable3);

console.log(variable4);

console.log(variable5);

})();

`);

// Display obfuscated result

console.log(obfuscationResult.getObfuscatedCode());

### Obfuscator Options

The obfuscator can be customized if you provide the configuration object as second argument in the obfuscate method. The following snippet shows all the available properties on the module:

JavaScriptObfuscator.obfuscate(YourCode, {

compact: true,

controlFlowFlattening: false,

controlFlowFlatteningThreshold: 0.75,

deadCodeInjection: false,

deadCodeInjectionThreshold: 0.4,

debugProtection: false,

debugProtectionInterval: false,

disableConsoleOutput: false,

domainLock: [],

log: false,

mangle: false,

renameGlobals: false,

reservedNames: [],

rotateStringArray: true,

seed: 0,

selfDefending: false,

sourceMap: false,

sourceMapBaseUrl: '',

sourceMapFileName: '',

sourceMapMode: 'separate',

stringArray: true,

stringArrayEncoding: false,

stringArrayThreshold: 0.75,

target: 'browser',

unicodeEscapeSequence: false

});

It's worth to read the documentation of the library as in the future new options may appear. The official repository offers already made presets to provide a feeling of "low", "medium" or "high" obfuscation with a special combination of options. Note that the better the obfuscation is, the slower the processing step is:

### A. Low Obfuscation

{

compact: true,

controlFlowFlattening: false,

deadCodeInjection: false,

debugProtection: false,

debugProtectionInterval: false,

disableConsoleOutput: true,

log: false,

mangle: true,

renameGlobals: false,

rotateStringArray: true,

selfDefending: true,

stringArray: true,

stringArrayEncoding: false,

stringArrayThreshold: 0.75,

unicodeEscapeSequence: false

}

### B. Medium Obfuscation

{

compact: true,

controlFlowFlattening: true,

controlFlowFlatteningThreshold: 0.75,

deadCodeInjection: true,

deadCodeInjectionThreshold: 0.4,

debugProtection: false,

debugProtectionInterval: false,

disableConsoleOutput: true,

log: false,

mangle: false,

renameGlobals: false,

rotateStringArray: true,

selfDefending: true,

stringArray: true,

stringArrayEncoding: 'base64',

stringArrayThreshold: 0.75,

unicodeEscapeSequence: false

}

### C. High Obfuscation

{

compact: true,

controlFlowFlattening: true,

controlFlowFlatteningThreshold: 1,

deadCodeInjection: true,

deadCodeInjectionThreshold: 1,

debugProtection: true,

debugProtectionInterval: true,

disableConsoleOutput: true,

log: false,

mangle: false,

renameGlobals: false,

rotateStringArray: true,

selfDefending: true,

stringArray: true,

stringArrayEncoding: 'rc4',

stringArrayThreshold: 1,

unicodeEscapeSequence: false

}

## Example

In the following example we're going to read the content of a JS file and we'll write a new one with the obfuscated version of the code:

// Require Filesystem module

var fs = require("fs");

// Require the Obfuscator Module

var JavaScriptObfuscator = require('javascript-obfuscator');

// Read the file of your original JavaScript Code as text

fs.readFile('./your-original-code.js', "UTF-8", function(err, data) {

if (err) {

throw err;

}

// Obfuscate content of the JS file

var obfuscationResult = JavaScriptObfuscator.obfuscate(data);

// Write the obfuscated code into a new file

fs.writeFile('./your-code-obfuscated.js', obfuscationResult.getObfuscatedCode() , function(err) {

if(err) {

return console.log(err);

}

console.log("The file was saved!");

});

});

Remember that, while it's almost impossible to recover the exact original source code from the obfuscated version, someone with the time, knowledge and patience can reverse-engineer it. The example doesn't use a special obfuscation but the default, so if you want a custom obfuscation, provide a configuration object.

**Critical installations :**

After carefully going through the Terms & Conditions of Docker, I have no desire to use it now nor anytime into the future. It is a very closed system and there are some conditions in the terms such as owner on usage data, certain biased declaration of the who and what a business is, who and what a government institution is, etc.

So I would rather take my chances on how to setup the following very necessary setup directly and configuring them to the level I want without the need for any docker application.

If now or in the very near future, I find a way to use Podman, that is something I could try. Meanwhile, this is also NOT highly desired. The goal is basically to get to the bare bones of the setup for each of the following :

PostgreDB

MongoDB

Kafka

Artemis AMQP

Consul

Infinispan

**Setting up PostgresDB**

**Setting up MongoDB**

**Setting up Kafka**

**Setting up Artemis AMQP**



I am assuming that you have already setup homebrew on your MacOS Apple Silicon system. If not refer to [this](https://stackoverflow.com/questions/66666134/how-to-install-homebrew-on-m1-mac) stackoverflow link to set it up or refer the official documentation.

The first steps are pretty straightforward:

$ brew install java  
$ brew install kafka

Once these commands run successfully your kafka package is installed. Now to run kafka open two seperate terminals to run Zookeeper and Kafka services seperately as:

**Start Zookeeper:**

$ **zookeeper-server-start /opt/homebrew/etc/kafka/zookeeper.properties**

**Start Kafka:**

$ **kafka-server-start /opt/homebrew/etc/kafka/server.properties**

After the services are successfully running, the next step is to create a kafka topic. *A topic is a category or feed name to which records are stored and published.* Before running any producer or consumer API it is mandatory to create a topic. Here is an example to create a sample topic named foobar :

$ **kafka-topics --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic foobar**

**NOTE:** Notice here we aren’t using --zookeeper flag as in standard documentations because Kafka versions for new MacOS do not support/ require that flag anymore.

Once the topic is created we can test out Kafka producer and consumer APIs:

* Open two terminals.
* In the first one initialize a producer console for topic thefoobar . And try to send some test messages as follows:

$ **kafka-console-producer --broker-list localhost:9092 --topic foobar  
>** foo  
**>** bar

* In the second one initalize a consumer console for the topic foobar . This will listen to the bootstrap server at port 9092 at topic foobar:

$ **kafka-console-consumer --bootstrap-server localhost:9092 --topic foobar --from-beginning**foo  
bar

If you can see the outputs kafka is set up and running neatly on your Apple Silicon system.

Cheers !

Terminal output upon running : $ brew install kafka

brew install kafka

==> **Auto-updating Homebrew...**

Adjust how often this is run with HOMEBREW\_AUTO\_UPDATE\_SECS or disable with

HOMEBREW\_NO\_AUTO\_UPDATE. Hide these hints with HOMEBREW\_NO\_ENV\_HINTS (see `man brew`).

==> **Downloading https://ghcr.io/v2/homebrew/portable-ruby/portable-ruby/blobs/sha256:303bed4c7fc431a685db3c3c151d873740114adbdccd23762ea2d1e39ea78f47**

########################################################################################################### 100.0%

==> **Pouring portable-ruby-3.3.6.arm64\_big\_sur.bottle.tar.gz**

==> **Auto-updated Homebrew!**

Updated 3 taps (homebrew/services, homebrew/core and homebrew/cask).

==> **New Formulae**

aliae dum kraftkit pie topiary

cargo-run-bin funzzy lla rshijack

directx-headers harlequin nrm scooter

==> **New Casks**

beaver-notes font-sketchybar-app-font tabtab zen-browser@twilight

font-cica-without-emoji ishare website-audit

You have **56** outdated formulae installed.

Warning: You are using macOS 12.

We (and Apple) do not provide support for this old version.

It is expected behaviour that some formulae will fail to build in this old version.

It is expected behaviour that Homebrew will be buggy and slow.

Do not create any issues about this on Homebrew's GitHub repositories.

Do not create any issues even if you think this message is unrelated.

Any opened issues will be immediately closed without response.

Do not ask for help from Homebrew or its maintainers on social media.

You may ask for help in Homebrew's discussions but are unlikely to receive a response.

Try to figure out the problem yourself and submit a fix as a pull request.

We will review it but may or may not accept it.

==> **Fetching dependencies for kafka: python-packaging, ca-certificates, openssl@3, sqlite, xz, pkgconf, python@3.13, libunistring, bison, meson, python-setuptools, glib, pixman, icu4c@76, gobject-introspection, harfbuzz, openjdk, automake, cppunit, libtool, maven and zookeeper**

==> **Fetching python-packaging**

==> **Downloading https://ghcr.io/v2/homebrew/core/python-packaging/manifests/24.2**

########################################################################################################### 100.0%

==> **Downloading https://ghcr.io/v2/homebrew/core/python-packaging/blobs/sha256:81d0db4704a8a4d53322164f860947baa0b**

########################################################################################################### 100.0%

==> **Fetching ca-certificates**

==> **Downloading https://ghcr.io/v2/homebrew/core/ca-certificates/manifests/2024-11-26**

########################################################################################################### 100.0%

==> **Downloading https://ghcr.io/v2/homebrew/core/ca-certificates/blobs/sha256:7a3b5f75ca44d330e0f37432af09f58e37bf**

########################################################################################################### 100.0%

==> **Fetching openssl@3**

==> **Downloading https://raw.githubusercontent.com/Homebrew/homebrew-core/a441d45b6da9b1ca05566933c51bf52a99b532a1/**

########################################################################################################### 100.0%

==> **Downloading https://github.com/openssl/openssl/releases/download/openssl-3.4.0/openssl-3.4.0.tar.gz**

==> **Downloading from https://objects.githubusercontent.com/github-production-release-asset-2e65be/7634677/ff52f315**

########################################################################################################### 100.0%

==> **Fetching sqlite**

==> **Downloading https://raw.githubusercontent.com/Homebrew/homebrew-core/a441d45b6da9b1ca05566933c51bf52a99b532a1/**

########################################################################################################### 100.0%

==> **Downloading https://www.sqlite.org/2024/sqlite-autoconf-3470100.tar.gz**

########################################################################################################### 100.0%

==> **Fetching xz**

==> **Downloading https://raw.githubusercontent.com/Homebrew/homebrew-core/a441d45b6da9b1ca05566933c51bf52a99b532a1/**

########################################################################################################### 100.0%

==> **Downloading https://github.com/tukaani-project/xz/releases/download/v5.6.3/xz-5.6.3.tar.gz**

==> **Downloading from https://objects.githubusercontent.com/github-production-release-asset-2e65be/553665726/054f4c**

########################################################################################################### 100.0%

==> **Fetching pkgconf**

==> **Downloading https://raw.githubusercontent.com/Homebrew/homebrew-core/a441d45b6da9b1ca05566933c51bf52a99b532a1/**

########################################################################################################### 100.0%

==> **Downloading https://distfiles.ariadne.space/pkgconf/pkgconf-2.3.0.tar.xz**

########################################################################################################### 100.0%

==> **Fetching python@3.13**

==> **Downloading https://raw.githubusercontent.com/Homebrew/homebrew-core/a441d45b6da9b1ca05566933c51bf52a99b532a1/**

########################################################################################################### 100.0%

==> **Downloading https://raw.githubusercontent.com/Homebrew/formula-patches/8b5bcbb262d1ea4e572bba55043bf7d2341a682**

########################################################################################################### 100.0%

==> **Downloading https://files.pythonhosted.org/packages/c4/e6/c1ac50fe3eebb38a155155711e6e864e254ce4b6e17fe2429b4c**

########################################################################################################### 100.0%

==> **Downloading https://files.pythonhosted.org/packages/4d/87/fb90046e096a03aeab235e139436b3fe804cdd447ed2093b0d70**

########################################################################################################### 100.0%

==> **Downloading https://files.pythonhosted.org/packages/27/b8/f21073fde99492b33ca357876430822e4800cdf522011f180413**

########################################################################################################### 100.0%

==> **Downloading https://files.pythonhosted.org/packages/b7/a0/95e9e962c5fd9da11c1e28aa4c0d8210ab277b1ada951d2aee33**

########################################################################################################### 100.0%

==> **Downloading https://www.python.org/ftp/python/3.13.0/Python-3.13.0.tgz**

########################################################################################################### 100.0%

==> **Fetching libunistring**

==> **Downloading https://raw.githubusercontent.com/Homebrew/homebrew-core/a441d45b6da9b1ca05566933c51bf52a99b532a1/**

########################################################################################################### 100.0%

==> **Downloading https://ftp.gnu.org/gnu/libunistring/libunistring-1.3.tar.gz**

########################################################################################################### 100.0%

==> **Fetching bison**

==> **Downloading https://ghcr.io/v2/homebrew/core/bison/manifests/3.8.2**

########################################################################################################### 100.0%

==> **Downloading https://ghcr.io/v2/homebrew/core/bison/blobs/sha256:78ce4e93936c37005e944b21e4b4d305725bc66f6c675a**

########################################################################################################### 100.0%

==> **Fetching meson**

==> **Downloading https://ghcr.io/v2/homebrew/core/meson/manifests/1.6.0**

########################################################################################################### 100.0%

==> **Downloading https://ghcr.io/v2/homebrew/core/meson/blobs/sha256:77eb91483991e1b615c3fe1dcea843e43706ac3eb011b0**

########################################################################################################### 100.0%

==> **Fetching python-setuptools**

==> **Downloading https://ghcr.io/v2/homebrew/core/python-setuptools/manifests/75.6.0**

########################################################################################################### 100.0%

==> **Downloading https://ghcr.io/v2/homebrew/core/python-setuptools/blobs/sha256:0c0edf5d7de8a5255a90ac7daa168db31d**

########################################################################################################### 100.0%

==> **Fetching glib**

==> **Downloading https://raw.githubusercontent.com/Homebrew/homebrew-core/a441d45b6da9b1ca05566933c51bf52a99b532a1/**

########################################################################################################### 100.0%

==> **Downloading https://raw.githubusercontent.com/Homebrew/formula-patches/43467fd8dfc0e8954892ecc08fab131242dca02**

########################################################################################################### 100.0%

==> **Downloading https://download.gnome.org/sources/gobject-introspection/1.82/gobject-introspection-1.82.0.tar.xz**

########################################################################################################### 100.0%

==> **Downloading https://download.gnome.org/sources/glib/2.82/glib-2.82.2.tar.xz**

########################################################################################################### 100.0%

==> **Fetching pixman**

==> **Downloading https://raw.githubusercontent.com/Homebrew/homebrew-core/a441d45b6da9b1ca05566933c51bf52a99b532a1/**

########################################################################################################### 100.0%

==> **Downloading https://cairographics.org/releases/pixman-0.44.0.tar.gz**

########################################################################################################### 100.0%

==> **Fetching icu4c@76**

==> **Downloading https://raw.githubusercontent.com/Homebrew/homebrew-core/a441d45b6da9b1ca05566933c51bf52a99b532a1/**

########################################################################################################### 100.0%

==> **Downloading https://github.com/unicode-org/icu/releases/download/release-76-1/icu4c-76\_1-src.tgz**

==> **Downloading from https://objects.githubusercontent.com/github-production-release-asset-2e65be/49244766/b85410b**

########################################################################################################### 100.0%

==> **Fetching gobject-introspection**

==> **Downloading https://raw.githubusercontent.com/Homebrew/homebrew-core/a441d45b6da9b1ca05566933c51bf52a99b532a1/**

########################################################################################################### 100.0%

==> **Downloading https://gitlab.gnome.org/tschoonj/gobject-introspection/-/commit/a7be304478b25271166cd92d110f251a8**

-#O#- # #

==> **Downloading https://files.pythonhosted.org/packages/67/03/fb5ba97ff65ce64f6d35b582aacffc26b693a98053fa831ab43a**

########################################################################################################### 100.0%

==> **Downloading https://files.pythonhosted.org/packages/54/28/3af612670f82f4c056911fbbbb42760255801b3068c48de792d3**

########################################################################################################### 100.0%

==> **Downloading https://files.pythonhosted.org/packages/87/5b/aae44c6655f3801e81aa3eef09dbbf012431987ba564d7231722**

########################################################################################################### 100.0%

==> **Downloading https://files.pythonhosted.org/packages/3e/2c/f0a538a2f91ce633a78daaeb34cbfb93a54bd2132a6de1f6cec0**

########################################################################################################### 100.0%

==> **Downloading https://download.gnome.org/sources/gobject-introspection/1.82/gobject-introspection-1.82.0.tar.xz**

Already downloaded: /Users/userXXZ/Library/Caches/Homebrew/downloads/bec68772f24244dc47ca35bbb5a4e323bc9996c70727b85986d2826626fa4e30--gobject-introspection-1.82.0.tar.xz

==> **Fetching harfbuzz**

==> **Downloading https://raw.githubusercontent.com/Homebrew/homebrew-core/a441d45b6da9b1ca05566933c51bf52a99b532a1/**

########################################################################################################### 100.0%

==> **Downloading https://github.com/harfbuzz/harfbuzz/raw/fc0daafab0336b847ac14682e581a8838f36a0bf/test/shaping/fon**

==> **Downloading from https://raw.githubusercontent.com/harfbuzz/harfbuzz/fc0daafab0336b847ac14682e581a8838f36a0bf/**

########################################################################################################### 100.0%

==> **Downloading https://github.com/harfbuzz/harfbuzz/archive/refs/tags/10.1.0.tar.gz**

==> **Downloading from https://codeload.github.com/harfbuzz/harfbuzz/tar.gz/refs/tags/10.1.0**

########################################################################################################### 100.0%

==> **Fetching openjdk**

==> **Downloading https://raw.githubusercontent.com/Homebrew/homebrew-core/a441d45b6da9b1ca05566933c51bf52a99b532a1/**

########################################################################################################### 100.0%

==> **Downloading https://github.com/openjdk/jdk/commit/ba5a4670b8ad86fefb41a939752754bf36aac9dc.patch?full\_index=1**

########################################################################################################### 100.0%

==> **Downloading https://download.java.net/java/GA/jdk22.0.2/c9ecb94cd31b495da20a27d4581645e8/9/GPL/openjdk-22.0.2\_**

########################################################################################################### 100.0%

==> **Downloading https://github.com/openjdk/jdk23u/archive/refs/tags/jdk-23.0.1-ga.tar.gz**

==> **Downloading from https://codeload.github.com/openjdk/jdk23u/tar.gz/refs/tags/jdk-23.0.1-ga**

-=O=- # # # #

==> **Fetching automake**

==> **Downloading https://ghcr.io/v2/homebrew/core/automake/manifests/1.17**

########################################################################################################### 100.0%

==> **Downloading https://ghcr.io/v2/homebrew/core/automake/blobs/sha256:aaf3cb57d50c48af4886c0cd24340aa6ca5628feac4**

########################################################################################################### 100.0%

==> **Fetching cppunit**

==> **Downloading https://ghcr.io/v2/homebrew/core/cppunit/manifests/1.15.1**

########################################################################################################### 100.0%

==> **Downloading https://ghcr.io/v2/homebrew/core/cppunit/blobs/sha256:7acd81de05bc607287b7133050c269e2ea86ac4e1141**

########################################################################################################### 100.0%

==> **Fetching libtool**

==> **Downloading https://raw.githubusercontent.com/Homebrew/homebrew-core/a441d45b6da9b1ca05566933c51bf52a99b532a1/**

########################################################################################################### 100.0%

==> **Downloading https://ftp.gnu.org/gnu/libtool/libtool-2.5.4.tar.xz**

########################################################################################################### 100.0%

==> **Fetching maven**

==> **Downloading https://ghcr.io/v2/homebrew/core/maven/manifests/3.9.9**

########################################################################################################### 100.0%

==> **Downloading https://ghcr.io/v2/homebrew/core/maven/blobs/sha256:106bdaaec0342b1656442dd5d1521b3edf69df22576726**

########################################################################################################### 100.0%

==> **Fetching zookeeper**

==> **Downloading https://raw.githubusercontent.com/Homebrew/homebrew-core/a441d45b6da9b1ca05566933c51bf52a99b532a1/**

########################################################################################################### 100.0%

==> **Downloading https://raw.githubusercontent.com/apache/zookeeper/release-3.9.3/conf/logback.xml**

########################################################################################################### 100.0%

==> **Downloading https://www.apache.org/dyn/closer.lua?path=zookeeper/zookeeper-3.9.3/apache-zookeeper-3.9.3.tar.gz**

==> **Downloading from https://dlcdn.apache.org/zookeeper/zookeeper-3.9.3/apache-zookeeper-3.9.3.tar.gz**

########################################################################################################### 100.0%

==> **Fetching kafka**

==> **Downloading https://raw.githubusercontent.com/Homebrew/homebrew-core/a441d45b6da9b1ca05566933c51bf52a99b532a1/**

########################################################################################################### 100.0%

==> **Downloading https://www.apache.org/dyn/closer.lua?path=kafka/3.9.0/kafka\_2.13-3.9.0.tgz**

==> **Downloading from https://dlcdn.apache.org/kafka/3.9.0/kafka\_2.13-3.9.0.tgz**

########################################################################################################### 100.0%

Warning: Your Xcode (13.1) is outdated.

Please update to Xcode 14.2 (or delete it).

Xcode can be updated from the App Store.

Warning: A newer Command Line Tools release is available.

Update them from Software Update in System Preferences.

If that doesn't show you any updates, run:

sudo rm -rf /Library/Developer/CommandLineTools

sudo xcode-select --install

Alternatively, manually download them from:

https://developer.apple.com/download/all/.

You should download the Command Line Tools for Xcode 14.2.

==> **Installing dependencies for kafka: python-packaging, ca-certificates, openssl@3, sqlite, xz, pkgconf, python@3.13, libunistring, bison, meson, python-setuptools, glib, pixman, icu4c@76, gobject-introspection, harfbuzz, openjdk, automake, cppunit, libtool, maven and zookeeper**

==> **Installing kafka dependency: python-packaging**

==> **Downloading https://ghcr.io/v2/homebrew/core/python-packaging/manifests/24.2**

Already downloaded: /Users/userXXZ/Library/Caches/Homebrew/downloads/a654ed84d67e434e714ac548959721dfcfc2297e1cb87be9237baee7632a2b76--python-packaging-24.2.bottle\_manifest.json

==> **Pouring python-packaging--24.2.all.bottle.tar.gz**

🍺 /opt/homebrew/Cellar/python-packaging/24.2: 56 files, 491.3KB

==> **Installing kafka dependency: ca-certificates**

==> **Downloading https://ghcr.io/v2/homebrew/core/ca-certificates/manifests/2024-11-26**

Already downloaded: /Users/userXXZ/Library/Caches/Homebrew/downloads/e16b55434e8bc1472ffb41e1a71c1b853417578c631fa1b69e8730f8cafca76c--ca-certificates-2024-11-26.bottle\_manifest.json

==> **Pouring ca-certificates--2024-11-26.all.bottle.tar.gz**

==> **Downloading https://formulae.brew.sh/api/formula.jws.json**

########################################################################################################### 100.0%

==> **Regenerating CA certificate bundle from keychain, this may take a while...**

🍺 /opt/homebrew/Cellar/ca-certificates/2024-11-26: 4 files, 239.4KB

==> **Installing kafka dependency: openssl@3**

Warning: Your Xcode (13.1) is outdated.

Please update to Xcode 14.2 (or delete it).

Xcode can be updated from the App Store.

Warning: A newer Command Line Tools release is available.

Update them from Software Update in System Preferences.

If that doesn't show you any updates, run:

sudo rm -rf /Library/Developer/CommandLineTools

sudo xcode-select --install

Alternatively, manually download them from:

https://developer.apple.com/download/all/.

You should download the Command Line Tools for Xcode 14.2.

==> **perl ./Configure --prefix=/opt/homebrew/Cellar/openssl@3/3.4.0 --openssldir=/opt/homebrew/etc/openssl@3 --libd**

==> **make**

==> **make install MANDIR=/opt/homebrew/Cellar/openssl@3/3.4.0/share/man MANSUFFIX=ssl**

==> **make HARNESS\_JOBS=8 test TESTS=-test\_afalg**

🍺 /opt/homebrew/Cellar/openssl@3/3.4.0: 7,227 files, 33.3MB, built in 4 minutes 27 seconds

==> **Installing kafka dependency: sqlite**

Warning: Your Xcode (13.1) is outdated.

Please update to Xcode 14.2 (or delete it).

Xcode can be updated from the App Store.

Warning: A newer Command Line Tools release is available.

Update them from Software Update in System Preferences.

If that doesn't show you any updates, run:

sudo rm -rf /Library/Developer/CommandLineTools

sudo xcode-select --install

Alternatively, manually download them from:

https://developer.apple.com/download/all/.

You should download the Command Line Tools for Xcode 14.2.

==> **./configure --enable-dynamic-extensions --enable-readline --disable-editline --enable-session**

**Setting up Kafdrop :**

<https://citizix.com/how-to-install-and-set-up-kafdrop-kafka-web-ui/>   
  
<https://github.com/obsidiandynamics/kafdrop/releases>

**Kafka Cheetsheet**

# Kafka Cheat Sheet

This is a list of commonly used CLI examples, when you work with **Kafka**, **Kafka Connect** and **Schema Registry**. Feel free to use it as well as post extensions to it.

All commands should be executed from **Apache Kafka** or **Confluent Platform** home directory. It is also assumed, that Zookeeper, Brokers, Connect Workers and Schema Registry operate on standard ports. Adjust when necessary.

## Table of contents

* [Kafka Cluster Management](https://github.com/whatsupbros/kafka-cheat-sheet#Kafka-Cluster-Management)
* [Kafka Topics](https://github.com/whatsupbros/kafka-cheat-sheet#Kafka-Topics)
* [Kafka Consumers](https://github.com/whatsupbros/kafka-cheat-sheet#Kafka-Consumers)
* [Kafka Producers](https://github.com/whatsupbros/kafka-cheat-sheet#Kafka-Producers)
* [kafkacat](https://github.com/whatsupbros/kafka-cheat-sheet#kafkacat)
* [Kafka Connect](https://github.com/whatsupbros/kafka-cheat-sheet#Kafka-Connect)
* [Schema Registry](https://github.com/whatsupbros/kafka-cheat-sheet#Schema-Registry)

## Kafka Cluster Management

See also:

* ZooKeeper configuration options: <https://docs.confluent.io/platform/current/zookeeper/deployment.html#configuration-options>
* Brokers configuration options: <https://docs.confluent.io/platform/current/installation/configuration/broker-configs.html>
* REST Proxy v3 for cluster management: <https://docs.confluent.io/platform/current/kafka-rest/api.html#crest-api-v3>

### Environment setup

export KAFKA\_HEAP\_OPTS="-Xmx2G -Xms128M"

export CONFLUENT\_HOME="/path/to/confluent-X.X.X"

export PATH="${CONFLUENT\_HOME}/bin:$PATH"

export LOG\_DIR=/tmp

### Starting Kafka Cluster

Start Zookeeper:

./bin/zookeeper-server-start ./path/to/zookeeper.properties > ./output/zookeeper.out &

Start Kafka Broker:

./bin/kafka-server-start ./path/to/broker0.properties > ./output/broker0.out &

Start Kafka Connect Worker (distributed mode):

./bin/connect-distributed ./path/to/worker0.properties > ./output/worker0.out &

Start Schema Registry:

./bin/schema-registry-start ./path/to/schema-registry.properties > ./output/schema-registry.out &

Start kSQL Server:

./bin/ksql-server-start ./path/to/ksql-server.properties > ./output/ksql-server.out &

Note: Standard output of Kafka Cluster processes will be saved in the specified files in this case. It also can be configured in log4j.properties which logs should be written and where by each process.

### Stopping Kafka Cluster

Note: This command searches for PIDs of all Kafka processes, and stops them gracefully.

#!/bin/bash

jps -m | grep 'QuorumPeerMain\|Kafka\|ConnectDistributed\|SchemaRegistryMain\|KsqlServerMain' | awk '{print $1}' | xargs kill

## Kafka Topics

See also:

* <https://docs.confluent.io/platform/current/installation/configuration/topic-configs.html>
* REST Proxy v3 for topics management: <https://docs.confluent.io/platform/current/kafka-rest/api.html#create-a-topic>

### List topics

./bin/kafka-topics --zookeeper localhost:2181 --list

..or..

./bin/kafka-topics --bootstrap-server localhost:9092 --list

### Describe a topic

./bin/kafka-topics --zookeeper localhost:2181 --describe --topic my-topic

..or..

./bin/kafka-topics --bootstrap-server localhost:9092 --describe --topic my-topic

### Create a topic

Note: To read more about topic configuration options, refer to [official docs](https://docs.confluent.io/platform/current/installation/configuration/topic-configs.html)

./bin/kafka-topics --create --bootstrap-server localhost:9092 --topic my-topic --replication-factor 3 --partitions 3 --config cleanup.policy=compact

### Alter topic config

./bin/kafka-configs --bootstrap-server localhost:9092 --topic my-topic --alter --add-config 'cleanup.policy=compact,retention.ms=86400000,segment.bytes=1073741824'

..or..

# deprecated

./bin/kafka-topics --zookeeper localhost:2181 --topic my-topic --alter --config cleanup.policy=compact --config retention.ms=86400000 --config segment.bytes=1073741824

Note: Usage of kafka-topics command to alter topics configuration is deprecated, usage of kafka-configs command is recommended.

### Delete topic config (reset to default)

./bin/kafka-configs --bootstrap-server localhost:9092 --topic my-topic --alter --delete-config 'cleanup.policy,retention.ms'

### Purge a topic

./bin/kafka-configs --bootstrap-server localhost:9092 --topic my-topic --alter --add-config 'cleanup.policy=delete,retention.ms=100'

...wait a minute...

./bin/kafka-configs --bootstrap-server localhost:9092 --topic my-topic --alter --delete-config 'cleanup.policy,retention.ms'

### Delete a topic

./bin/kafka-topics --bootstrap-server localhost:9092 --delete --topic my-topic

## Kafka Consumers

See also:

* <https://docs.confluent.io/platform/current/tutorials/examples/clients/docs/kafka-commands.html#basic-producer-and-consumer>
* <https://docs.confluent.io/platform/current/installation/configuration/consumer-configs.html>

### Simple consumer console

Plain text:

./bin/kafka-console-consumer --bootstrap-server localhost:9092 --topic my-topic --from-beginning

Avro:

./bin/kafka-avro-console-consumer --bootstrap-server localhost:9092 --topic my-topic --property schema.registry.url=http://localhost:8081 --from-beginning

### Print key together with value

./bin/kafka-avro-console-consumer --bootstrap-server localhost:9092 \

--topic my-topic \

--property schema.registry.url=http://localhost:8081 \

--property print.key=true \

--property print.value=true \

--property key.separator=":" \

--from-beginning

### Use different deserializers for key and value

./bin/kafka-avro-console-consumer --bootstrap-server localhost:9092 \

--topic my-topic \

--property schema.registry.url=http://localhost:8081 \

--property print.key=true \

--property print.value=true \

--property key.separator=":" \

--key-deserializer "org.apache.kafka.common.serialization.StringDeserializer" \

--value-deserializer "io.confluent.kafka.serializers.KafkaAvroDeserializer" \

--from-beginning

### Use basic auth for Schema Registry

export API\_KEY="USERNAME"

export API\_SECRET="PASSWORD"

./bin/kafka-avro-console-consumer --bootstrap-server localhost:9092 \

--topic my-topic \

--property schema.registry.url=http://localhost:8081 \

--property schema.registry.basic.auth.user.info="$API\_KEY:$API\_SECRET" \

--property basic.auth.credentials.source=USER\_INFO \

--property print.key=true \

--property print.value=true \

--property key.separator=":" \

--key-deserializer "org.apache.kafka.common.serialization.StringDeserializer" \

--value-deserializer "io.confluent.kafka.serializers.KafkaAvroDeserializer" \

--from-beginning

### Use SASL SSL security for Kafka Broker and Schema Registry connection

export API\_KEY="USERNAME"

export API\_SECRET="PASSWORD"

export KAFKA\_TRUSTSTORE\_LOCATION="/path/to/truststore.jks"

export KAFKA\_TRUSTSTORE\_PASSPHRASE="<TRUSTSTORE\_PASSPHRASE>"

export KAFKA\_KEYSTORE\_LOCATION=/path/to/keystore.jks

export KAFKA\_KEYSTORE\_PASSPHRASE="<KEYSTORE\_PASSPHRASE>"

export KAFKA\_KEY\_LOCATION="/path/to/key.pem"

export KAFKA\_KEY\_PASSPHRASE="<KEY\_PASSPHRASE>"

export SCHEMA\_REGISTRY\_OPTS="-Djavax.net.ssl.keyStore=$KAFKA\_KEYSTORE\_LOCATION -Djavax.net.ssl.trustStore=$KAFKA\_TRUSTSTORE\_LOCATION -Djavax.net.ssl.keyStorePassword=$KAFKA\_KEYSTORE\_PASSPHRASE -Djavax.net.ssl.trustStorePassword=$KAFKA\_TRUSTSTORE\_PASSPHRASE"

./bin/kafka-avro-console-consumer --bootstrap-server localhost:9092 \

--topic my-topic \

--property print.key=true \

--property print.value=true \

--property key.separator=":" \

--key-deserializer "org.apache.kafka.common.serialization.StringDeserializer" \

--value-deserializer "io.confluent.kafka.serializers.KafkaAvroDeserializer" \

--property schema.registry.url=http://localhost:8081 \

--property schema.registry.basic.auth.user.info="$API\_KEY:$API\_SECRET" \

--property basic.auth.credentials.source=USER\_INFO \

--consumer-property security.protocol=SASL\_SSL \

--consumer-property sasl.mechanism=PLAIN \

--consumer-property sasl.jaas.config="org.apache.kafka.common.security.plain.PlainLoginModule required username=\"$API\_KEY\" password=\"$API\_SECRET\";" \

--consumer-property ssl.truststore.location=$KAFKA\_TRUSTSTORE\_LOCATION \

--consumer-property ssl.truststore.password=$KAFKA\_TRUSTSTORE\_PASSPHRASE \

--consumer-property ssl.keystore.location=$KAFKA\_KEYSTORE\_LOCATION \

--consumer-property ssl.keystore.password=$KAFKA\_KEYSTORE\_PASSPHRASE \

--consumer-property ssl.key.password=$KAFKA\_KEY\_PASSPHRASE \

--consumer-property ssl.truststore.type=JKS \

--consumer-property ssl.keystore.type=JKS \

--from-beginning

### Save consumed Kafka messages to a file

./bin/kafka-avro-console-consumer --bootstrap-server localhost:9092 \

--topic my-topic \

--property print.key=true \

--property print.value=true \

--property key.separator=":" \

--property schema.registry.url=http://localhost:8081 \

--from-beginning

--timeout-ms 25000 \

> ./my-topic-data.json

Note: After consuming all topic messages, the console waits for 25 seconds timeout without new messages, and then exits

## Kafka Producers

See also:

* <https://docs.confluent.io/platform/current/tutorials/examples/clients/docs/kafka-commands.html#basic-producer-and-consumer>
* <https://docs.confluent.io/platform/current/installation/configuration/producer-configs.html>

### Simple producer console

Plain text:

./bin/kafka-console-producer --bootstrap-server localhost:9092 --topic my-topic

Avro:

./bin/kafka-avro-console-producer --bootstrap-server localhost:9092 --topic my-topic --property schema.registry.url=http://localhost:8081 --property value.schema='<value-schema-as-json>'

### Parse key together with value

Note: Topic value schema is read from the file /path/to/my-topic-value-schema.json, where schema must be formatted as normal minified JSON (no extra spaces and other whitespace characters).

./bin/kafka-avro-console-producer --bootstrap-server localhost:9092 \

--topic my-topic \

--property schema.registry.url=http://localhost:8081 \

--property value.schema.file=/path/to/my-topic-value-schema.json \

--property parse.key=true \

--property key.separator=":"

### Use different serializer for key

./bin/kafka-avro-console-producer --bootstrap-server localhost:9092 \

--topic my-topic \

--property schema.registry.url=http://localhost:8081 \

--property value.schema.file=/path/to/my-topic-value-schema.json \

--property parse.key=true \

--property key.separator=":" \

--property key.serializer="org.apache.kafka.common.serialization.StringSerializer"

### Read Kafka messages to produce from a file

./bin/kafka-avro-console-producer --bootstrap-server localhost:9092 \

--topic my-topic \

--property schema.registry.url=http://localhost:8081 \

--property value.schema.file=/path/to/my-topic-value-schema.json \

--property parse.key=true \

--property key.separator=":" \

--property key.serializer="org.apache.kafka.common.serialization.StringSerializer" \

< ./my-topic-data.json

## [kafkacat](https://github.com/edenhill/kafkacat)

Note: kafkacat is a powerful tool to work with Kafka Cluster written in C. It can operate in both consumer and producer mode, it is fast, it can read topic headers, but it currently does not support all Kafka features (i.e. there is no producer mode for Avro serialized topics).

### kafkacat for Windows

To build kafkacat for Windows from sources, you will need to build it from win32 directory and have these pre-requisites:

1. NuGet package manager (<https://www.nuget.org/downloads>)
2. MS Visual C++ Build Tools 14 for Visual Studio 2015 (<https://visualstudio.microsoft.com/ru/vs/older-downloads/>) - this is not Visual Studio IDE itself, but a subset of stuff to build existing projects. Note this, that exactly this version is currently required to build the project, this can change in future.
3. Add MSBuild location to PATH environment variable (usually it is C:\Program Files (x86)\MSBuild\14.0\Bin) Change version manually to the current one in win32/win32\_config.h (1.6.0 at the moment of writing this)

Now build the project using the official instructions:

cd win32

nuget restore

msbuild # or, with full path: C:\Program Files (x86)\MSBuild\14.0\Bin\msbuild.exe

This should do the trick, you should have the binary then in %kafkacat\_source\_base\_dir%\win32\x64\Debug:

C:\kafkacat\kafkacat-1.6.0\win32\x64\Debug>kafkacat.exe -V

kafkacat - Apache Kafka producer and consumer tool

https://github.com/edenhill/kafkacat

Copyright (c) 2014-2019, Magnus Edenhill

Version 1.6.0 (Transactions, librdkafka 1.5.0 builtin.features=gzip,snappy,ssl,sasl,regex,lz4,sasl\_gssapi,sasl\_plain,sasl\_scram,plugins,zstd,sasl\_oauthbearer)

Note: Currently, the Windows version of kafkacat does not support JSON, Avro and Schema Registry

### kafkacat for Linux

#### Option 1. Install from repository

Ubuntu/Debian example:

sudo apt install kafkacat

Note: kafkacat, installable from Ubuntu (1.5.0-1.1) or Debian (1.6.0-1) repositories does not support working with Avro and Schema Registry, despite the fact that this functionality was added in version 1.5.0.

#### Option 2. Build from sources

To be able to use kafkacat with Avro and Schema registry, download its sources from GitHub and build it yourself from sources with support of libavro-c and libserdes (as it is mentioned here: <https://github.com/edenhill/kafkacat#requirements>)

On Ubuntu 20.04, first, you will need these additional package to build the app:

sudo apt install pkg-config build-essential cmake libtool libssl-dev zlib1g-dev libzstd-dev libsasl2-dev libjansson-dev libcurl4-openssl-dev

Then, you can use bootstrap script to build kafkacat with all dependencies:

./bootstrap.sh

Now you are ready to use it:

$ ./kafkacat -V

kafkacat - Apache Kafka producer and consumer tool

https://github.com/edenhill/kafkacat

Copyright (c) 2014-2019, Magnus Edenhill

Version 1.6.0 (JSON, Avro, Transactions, librdkafka 1.5.0 builtin.features=gzip,snappy,ssl,sasl,regex,lz4,sasl\_gssapi,sasl\_plain,sasl\_scram,plugins,zstd,sasl\_oauthbearer)

### kafkacat examples

See kafkacat usage examples here: <https://github.com/edenhill/kafkacat#examples>

## Kafka Connect

Note: For Kafka Connect and Schema Registry you will need curl and jq utilities to make requests to their APIs.

Note: Alternatively, you can use [this postman workspace](https://www.postman.com/whatsupbros/workspace/03c475b6-174a-40f4-8111-f09d13082d23) to work with Kafka Connect and Schema Registry REST APIs

See also:

* Workers configuration options: <https://docs.confluent.io/platform/current/connect/references/allconfigs.html>
* Connectors configuration options: <https://docs.confluent.io/platform/current/installation/configuration/connect/index.html>
* Kafka Connect REST API Reference: <https://docs.confluent.io/platform/current/connect/references/restapi.html>

### List installed Kafka Connect plugins

curl -Ss -X GET http://localhost:8083/connector-plugins | jq

### List the connectors

curl -Ss -X GET http://localhost:8083/connectors | jq

### Deploy a connector

curl -Ss -X POST -H "Content-Type: application/json" --data @/path/to/my-topic-connector-config.json http://localhost:8083/connectors | jq

### Get connector overview (configuration and tasks overview)

curl -Ss -X GET http://localhost:8083/connectors/<connector-name> | jq

### Get connector config

curl -Ss -X GET http://localhost:8083/connectors/<connector-name>/config | jq

### Get connector status

curl -Ss -X GET http://localhost:8083/connectors/<connector-name>/status | jq

### Restart connector

curl -Ss -X POST http://localhost:8083/connectors/<connector-name>/restart | jq

### Get connector tasks

curl -Ss -X GET http://localhost:8083/connectors/<connector-name>/tasks | jq

### Get connector task status

curl -Ss -X GET http://localhost:8083/connectors/<connector-name>/tasks/0/status | jq

### Restart connector task

curl -Ss -X POST http://localhost:8083/connectors/<connector-name>/tasks/0/restart | jq

### Remove connector

curl -Ss -X DELETE http://localhost:8083/connectors/<connector-name> | jq

### Get current logging levels

curl -Ss http://localhost:8083/admin/loggers | jq

### Set logging level for a particular logger

curl -Ss -X PUT -H "Content-Type:application/json" http://localhost:8083/admin/loggers/<logger-name> -d '{"level": "DEBUG"}' | jq

**Examples:**

# sets debug log level for JDBC connector (source and sink)

curl -Ss -X PUT -H "Content-Type:application/json" http://localhost:8083/admin/loggers/io.confluent.connect.jdbc -d '{"level": "DEBUG"}' | jq

# sets debug log level for Kafka Connect Worker Sink tasks

curl -Ss -X PUT -H "Content-Type:application/json" http://localhost:8083/admin/loggers/org.apache.kafka.connect.runtime.WorkerSinkTask -d '{"level": "DEBUG"}' | jq

# sets debug log level for Kafka Connect in general

curl -Ss -X PUT -H "Content-Type:application/json" http://localhost:8083/admin/loggers/org.apache.kafka.connect -d '{"level": "DEBUG"}' | jq

## Schema Registry

See also:

* Schema Registry REST API Reference: <https://docs.confluent.io/platform/current/schema-registry/develop/api.html>
* REST API Usage Examples: <https://docs.confluent.io/platform/current/schema-registry/develop/using.html#common-sr-api-usage-examples>

### Retrieve currently registered schemas (subjects)

curl -Ss -X GET http://localhost:8081/subjects | jq

### Retrieve schema versions of a subject

curl -Ss -X GET http://localhost:8081/subjects/my-topic-value/versions | jq

### Retrieve schema of a subject of a particular version

With metadata (schema ID, version, schema value as escaped JSON):

curl -Ss -X GET http://localhost:8081/subjects/my-topic-value/versions/1 | jq

Without metadata (schema value as normal JSON):

curl -Ss -X GET http://localhost:8081/subjects/my-topic-value/versions/1/schema | jq

### Retrieve last schema of a subject

With metadata (schema ID, version, schema value as escaped JSON):

curl -Ss -X GET http://localhost:8081/subjects/my-topic-value/versions/latest | jq

Without metadata (schema value as normal JSON):

curl -Ss -X GET http://localhost:8081/subjects/my-topic-value/versions/latest/schema | jq

### Retrieve last schema of a subject, use certificate for the connection to Schema Registry

export API\_KEY="USERNAME"

export API\_SECRET="PASSWORD"

export KEY\_LOCATION="/path/to/key.pem"

export KEY\_PASSPHRASE="<KEY\_PASSPHRASE>"

curl -Ss --cert $KEY\_LOCATION:$KEY\_PASSPHRASE \

-X GET http://$API\_KEY:$API\_SECRET@localhost:8081/subjects/my-topic-value/versions/latest/schema

### Retrieve last schema of a subject and save it into file

Note: Schema saved using this command, is suitable for value.schema.file property of kafka-avro-console-producer utility

curl -Ss -X GET http://localhost:8081/subjects/my-topic-value/versions/latest/schema > ./my-topic-value-schema.json

### Deploy schema for a subject

Note: Schema is read from my-topic-value-schema.json file from the current directory. Mind the fact, that schema value format must be formatted as mimified escaped JSON, i.e.: {"schema":"{\"type\":\"string\"}"}.

curl -Ss -X POST -H "Content-Type: application/json" --data @/path/to/my-topic-value-schema.json http://localhost:8081/subjects/my-topic-value/versions | jq

## About

Kafka, Kafka Connect and Schema Registry commands cheat sheet

## Key Kafka commands

The Kafka CLI is a powerful tool. However, the user experience can be challenging if you don’t already know the exact command needed for your task. The table below shows commonly used CLI commands to interact with Kafka.

|  |  |
| --- | --- |
| Create topics | bin/kafka-topics.sh --bootstrap-server <URL> --create --replication-factor 3 --partitions 4 --topic topic-name |
| List all topics | bin/kafka-topics.sh --bootstrap-server <URL> --list |
| Add a topic partition | bin/kafka-topics.sh --bootstrap-server <URL> --alter --topic <topic-name> --partitions 16 |
| Run a producer | bin/kafka-console-producer.sh --topic <topic-name> --broker-list <URL> |
| Run a consumer | bin/kafka-console-consumer.sh --bootstrap-server <URL> --topic <topic-name> |
| Consume messages from the beginning | bin/kafka-console-consumer.sh --topic <topic-name> --bootstrap-server <URL> --group <group-name>--from-beginning |
| Get information on a specific consumer group | bin/kafka-consumer-groups.sh --bootstrap-server <URL> --describe --group <group-name> |
| Delete a consumer group | /bin/kafka-consumer-groups.sh --bootstrap-server <URL> --delete --group my-group --group <group-name> |
| Reset the offset of a topic | bin/kafka-consumer-groups.sh --bootstrap-server <URL> --reset-offsets --group <group-name> --topic <topic-name> --to-earliest |

## [How to install and set up Kafdrop – Kafka Web UI](https://citizix.com/how-to-install-and-set-up-kafdrop-kafka-web-ui/)

Apr 16, 2022

8 minute read

Kafdrop is a web UI for viewing Kafka topics and browsing consumer groups. The tool displays information such as brokers, topics, partitions, consumers, and lets you view messages.

Apache Kafka is an open-source platform. Kafka was originally developed by Linkedin and was later incubated as the Apache Project. It can process over 1 million messages per second.

Kafka is an amazing platform for processing a huge number of messages very quickly. However, Kafka has one disadvantage that it does not come with an inbuilt User Interface where the users can see the information related to Kafka.

Kafdrop helps us in solving this problem. It gives us a simple, lightweight, and easy-to-use User Interface where one can not only see the required information but can also create and delete Kafka topics.

#### Features

* **View Kafka brokers** - topic and partition assignments, and controller status
* **View topics** - partition count, replication status, and custom configuration
* **Browse messages** - JSON, plain text, Avro and Protobuf encoding
* **View consumer groups** - per-partition parked offsets, combined and per-partition lag
* **Create new topics**
* **View ACLs**
* **Support for Azure Event Hubs**

#### Requirements

* Java 11 or newer
* Kafka (version 0.11.0 or newer) or Azure Event Hubs

Optional, additional integration:

* Schema Registry

Kafdrop can be installed by executing a JAR file or via docker or on Kubernetes. In this guide we will set up Kafdrop JAR directly.

Also checkout:

* [How to install Apache Kafka on Rocky Linux or AlmaLinux8](https://citizix.com/how-to-install-apache-kafka-on-rocky-linux-or-almalinux8/)

## Installing Java

Kafdrop needs Java to run, hence first we need to install that on our local environment and it must be equal or greater than Java 11. Well, we don’t need to add any third repository because the package to get JAVA is already there on the system base repo.

Let us install latest Java on RHEL 8 based distributions with the following command. For other Linux Distributions please consult your package manager manual.

|  |  |
| --- | --- |
| 1 | sudo dnf install java-11-openjdk |

Type y and press enter when prompted to accept the installation.

## Get the latest Kafdrop

Kafdrop is available as a jar file. Get the latest release from github release page [here](https://github.com/obsidiandynamics/kafdrop/releases).

We are going to download and install kafka in the /opt/kafka directory. First become root and set up the required directory structure

|  |  |
| --- | --- |
| 1  2 | sudo mkdir/opt/kafdrop  cd /opt/kafdrop/ |

Next, download the latest kadrop and rename the file. In this guide we are downloading version 3.30.0.

|  |  |
| --- | --- |
| 1  2  3 | curl -LO https://github.com/obsidiandynamics/kafdrop/releases/download/3.30.0/kafdrop-3.30.0.jar  mv kafdrop-3.30.0.jar kafdrop.jar |

## Running the Jar

Once the jar is downloaded, we can run it with the java -jar command. This is the command:

|  |  |
| --- | --- |
| 1  2  3 | java --add-opens=java.base/sun.nio.ch=ALL-UNNAMED \  -jar /opt/kafdrop/kafdrop.jar \  --kafka.brokerConnect=localhost:9092 |

If unspecified, kafka.brokerConnect defaults to localhost:9092.

**Note:** As of Kafdrop 3.10.0, a ZooKeeper connection is no longer required. All necessary cluster information is retrieved via the Kafka admin API.

Once it starts, Open a browser and navigate to http://server\_ip:9000/. The port can be overridden by adding the following config:

|  |  |
| --- | --- |
| 1 | --server.port=<port> --management.server.port=<port> |

Optionally, configure a schema registry connection with:

|  |  |
| --- | --- |
| 1 | --schemaregistry.connect=http://localhost:8081 |

and if you also require basic auth for your schema registry connection you should add:

|  |  |
| --- | --- |
| 1 | --schemaregistry.auth=username:password |

Finally, a default message format (e.g. to deserialize Avro messages) can optionally be configured as follows:

|  |  |
| --- | --- |
| 1 | --message.format=AVRO |

Valid format values are DEFAULT, AVRO, PROTOBUF. This can also be configured at the topic level via dropdown when viewing messages.

## Create a Systemd unit for Kafdrop service

When running Kafdrop Service in a production server we have to run it in the background. Hence, create systemd units for both the scripts.

Create a kafdrop systemd service file

|  |  |
| --- | --- |
| 1 | sudo vim /etc/systemd/system/kafdrop.service |

Add this content to the file

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | [Unit]  Description=Kafdrop server  Documentation=https://github.com/obsidiandynamics/kafdrop  Requires=network.target remote-fs.target  After=network.target remote-fs.target  [Service]  Type=simple  ExecStart=/bin/java --add-opens=java.base/sun.nio.ch=ALL-UNNAMED \  -jar /opt/kafdrop/kafdrop.jar \  --kafka.brokerConnect=localhost:9092  Restart=on-abnormal  [Install]  WantedBy=multi-user.target |

Finally, save and close the file. To ensure that the service is recognized, reload systemd units:

|  |  |
| --- | --- |
| 1 | sudo systemctl daemon-reload |

## Start and enable Kafdrop systemd service

Now, let’s start and enable the service to make sure they will also get active even after the system reboot.

Start kafdrop

|  |  |
| --- | --- |
| 1 | sudo systemctl start kafdrop |

Confirm the services status to ensure that they are both running as expected:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | $ sudo systemctl status kafdrop  kafdrop.service - Kafdrop server  Loaded: loaded (/etc/systemd/system/kafdrop.service; disabled; vendor preset: disabled)  Active: active (running) since Sat 2022-04-16 10:49:28 UTC; 21s ago  Docs: https://github.com/obsidiandynamics/kafdrop  Main PID: 78642 (java)  Tasks: 22 (limit: 23167)  Memory: 334.7M  CGroup: /system.slice/kafdrop.service  &#x2514;&#x2500;78642 /bin/java --add-opens=java.base/sun.nio.ch=ALL-UNNAMED -jar /opt/kafdrop/kafdrop.jar --kafka.brokerConnect=localhost:9092  Apr 16 10:49:35 rockysrv.citizix.com java[78642]: 2022-04-16 10:49:35.046 INFO 78642 [ main] k.c.KafkaConfiguration : Checking keystore file kafka.keystore.jks  Apr 16 10:49:35 rockysrv.citizix.com java[78642]: 2022-04-16 10:49:35.046 INFO 78642 [ main] k.c.KafkaConfiguration : Checking properties file kafka.properties  Apr 16 10:49:35 rockysrv.citizix.com java[78642]: 2022-04-16 10:49:35.128 INFO 78642 [ main] k.s.BuildInfo : Kafdrop version: 3.30.0, build time: 2022-04->  Apr 16 10:49:36 rockysrv.citizix.com java[78642]: 2022-04-16 10:49:36.039 INFO 78642 [ main] o.s.b.a.e.w.EndpointLinksResolver : Exposing 13 endpoint(s) beneath base path '/a>  Apr 16 10:49:37 rockysrv.citizix.com java[78642]: 2022-04-16 10:49:36.996 INFO 78642 [ main] i.u.Undertow : starting server: Undertow - 2.2.16.Final  Apr 16 10:49:37 rockysrv.citizix.com java[78642]: 2022-04-16 10:49:37.011 INFO 78642 [ main] o.x.Xnio : XNIO version 3.8.6.Final  Apr 16 10:49:37 rockysrv.citizix.com java[78642]: 2022-04-16 10:49:37.029 INFO 78642 [ main] o.x.n.NioXnio : XNIO NIO Implementation Version 3.8.6.Final  Apr 16 10:49:37 rockysrv.citizix.com java[78642]: 2022-04-16 10:49:37.075 INFO 78642 [ main] o.j.t.Version : JBoss Threads version 3.1.0.Final  Apr 16 10:49:37 rockysrv.citizix.com java[78642]: 2022-04-16 10:49:37.139 INFO 78642 [ main] o.s.b.w.e.u.UndertowWebServer : Undertow started on port(s) 9000 (http)  Apr 16 10:49:37 rockysrv.citizix.com java[78642]: 2022-04-16 10:49:37.727 INFO 78642 [ main] o.s.b.StartupInfoLogger : Started Kafdrop in 7.411 seconds (JVM runnin |

Finally enable the service on boot:

|  |  |
| --- | --- |
| 1 | sudo systemctl enable kafdrop |

Once the service is successfully started, you can access the UI.

## Navigating the UI

Open a browser and navigate to http://server\_ip:9000/.

The **Cluster Overview** screen is the landing page of the web UI.

You get to see the overall layout of the cluster - the individual brokers that make it up, their addresses and some key broker stats - whether they are a controller and the number of partitions each broker owns. The latter is quite important - as your cluster size and the number of topics (and therefore partitions) grows, you generally want to see an approximately level distribution of partitions across the cluster.

Next is the **Topics List**, which in most cases is what you’re really here for. Any reasonably-sized microservices-based ecosystem might have hundreds, if not thousands of topics. As you’d expect, the list is searchable. The stats displayed alongside each topic are fairly ho-hum. The one worth noting is the under-replicated column. Essentially, it’s telling us the number of partition replicas that have fallen behind the primary. Zero is a good figure. Anything else is indicative of either a broker or a network issue that requires immediate attention.

Click on a topic in the list to get to the **Topic Overview** screen.

The screen is subdivided into four sections.

On the top-left, there is a summary of the topic stats - a handy view, not dissimilar to what you would have seen in the cluster overview.

On the top-right, you can view the custom configuration. In the example above, the topic runs a stock-standard config, so there’s nothing to see. Had the configuration been overridden, you’d see a set of custom values like in the example below.

The bottom-left section enumerates over the partitions. The partition indexes are links - clicking through will reveal the first 100 messages in the topic.

The **consumers** section on the bottom-right lists the consumer group names as well as their aggregate lag (the sum of all individual partition lags).

Clicking on the consumer group on the Topic Overview gets you into the **Consumer View**. This screen provides a comprehensive breakdown of a single consumer group.

The view is sectioned by topic. For each topic, a separate table lists the underlying partitions. Against each partition, we see the committed offset, which we can compare against the first and last offsets to see how our consumer is tracking. Conveniently, Kafdrop displays the computed lag for each partition, which is aggregated at the footer of each topic table.

The **Message View** screen is the coveted topic viewer that has in all likelihood brought you here. You can get to the message view in one of two ways:

1. Click the **View Messages** button in the Topic Overview screen.
2. Click the individual partition link in the Topic Overview.

It’s exactly what you’d expect - a chronologically-ordered list of messages (or records, in Kafka parlance) for a chosen partition.

Each entry conveniently displays the offset, the record key (if one is set), the timestamp of publication, and any headers that may have been appended by the producer.

There’s another little trick up Kafdrop’s sleeve. If the message happens to be a valid JSON document, the topic viewer can nicely format it. Click on the green arrow on the left of the message to expand it.

## Conclusion

In this guide we learnt how to install and use the Kafdrop Kafka UI.



·

Sep 29, 2019

Do you make changes to your Kafka cluster using the CLI?  
Do you always have to look up the CLI commands and options?  
My Apache Kafka, CLI cheat sheet might be helpful for you!

In this short blog post, you find my Kafka CLI cheatsheet (I’ll add more commands to the cheat sheet regularly).

# Overview

[**Docker**](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#471a)

* [Kafka & Zookeeper Docker Compose file](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#e260)
* [Start Kafka & Zookeeper using Docker Compose](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#e260)
* [Attach to the Kafka Broker running in Docker](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#5745)
* [Unset the JMX port in the Kafka Docker container](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#5745)
* [Stop Kafka & Zookeeper using Docker Compose](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#84e3)

[**Cluster and Broker(s)**](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#a689)

* [Show the version of the Kafka broker](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#8198)
* [Connect to Zookeeper](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#8198)
* [Show the Kafka cluster id](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#6260)
* [List brokers in the Kafka cluster](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#497b)
* [Show details of a Kafka broker](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#81cb)
* [Show all the topics that exist in the cluster](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#93c2)
* [Show details of a specific topic](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#cd67)
* [Exit the zookeeper shell](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#c481)

[**Topics**](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#6fd2)

* [Create a Kafka topic](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#8c2f)
* [Create a Kafka topic in case the topic doesn’t exist](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#09e8)
* [Create a Kafka topic with a short retention period](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#47c2)
* [List the Kafka topics](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#a1a2)
* [List Kafka topics and exclude internal topics](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#c9bd)
* [Show the Kafka topic details](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#477b)
* [Increase the number of partitions of a Kafka topic](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#1bbb)
* [Change the retention time of a Kafka topic](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#2020)
* [Purge a Kafka topic](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#b94e)
* [List Kafka topics (with configuration values) that have specific configuration overrides](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#38bc)
* [Show specific Kafka topic configuration overrides](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#f61a)
* [Show partition offsets of a Kafka topic](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#0bf4)
* [Show offset for specific partition(s) of a Kafka topic](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#f2c5)
* [Delete a Kafka topic](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#0d0c)

[**Producers**](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#0bd9)

* [Produce a message on a Kafka topic](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#f574)
* [Produce messages on a Kafka topic from a file](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#abe2)
* [Produce messages to Kafka with both key and value](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#8719)

[**Consumers**](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#5c14)

* [Consume from a Kafka topic](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#fe4f)
* [Consume a Kafka topic and show both key, value and timestamp](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#8719)
* [Consume a topic from a Kafka topic from the beginning](https://medium.com/@TimvanBaarsen/apache-kafka-cli-commands-cheat-sheet-a6f06eac01b#995a)

## Kafka CLI

All the commands used in this blogpost are included in the Apache Kafka distribution. The file extension of the scripts in the Apache Kafka distribution is .sh

Be aware that the Confluent Kafka distribution dropped the .sh file extension!

# Docker

For every command, I also provide a practical working example you can execute in a running Kafka Docker container. I’m using the Confluent Kafka distribution in this cheat sheet.

## Kafka & Zookeeper Docker Compose file

You start a Kafka broker (including Zookeeper) using this docker-compose example file:

Copy the content in a file with the name: docker-compose.yml

docker-compose.yml file to run Kafka and Zookeeper. <https://gist.github.com/j-tim/cb630c3646b5987e45e2b8cf847f4149>

## Start Kafka & Zookeeper using Docker Compose

Start the official [Confluent Kafka](https://hub.docker.com/r/confluentinc/cp-kafka/tags) and [Zookeeper Docker](https://hub.docker.com/r/confluentinc/cp-zookeeper/tags) containers using [Docker Compose](https://docs.docker.com/compose/):

docker-compose up -d

In case you don’t have the docker images on your system, docker will pull the images from Dockerhub.

## Attach to the Kafka Broker running in Docker

Attach to the Kafka Docker container to execute operations on your Apache Kafka cluster.

docker exec -it kafka bash

## Unset the JMX port in the Kafka Docker container

In case you configured a JMX\_PORT in your docker-compose file for the Kafka Docker container you have to unset the JMX\_PORT environment variable

unset JXM\_PORT

otherwise, you will run into issues when executing the CLI like:

JMX port already in use

## Stop Kafka & Zookeeper using Docker Compose

docker-compose down -v

# Cluster and broker(s)

## Show the version of the Kafka broker

$KAFKA\_HOME/bin/kafka-broker-api-versions.sh --bootstrap-server localhost:9092 --version

🐳 Example to execute in Docker:

./usr/bin/kafka-broker-api-versions --bootstrap-server localhost:9092 --version

## Connect to Zookeeper

Apache Kafka is depending\* on Apache Zookeeper for:

* cluster membership
* controller election
* topic configuration
* access control lists
* quotas.

The meta-data for those operations are stored outside of Kafka in a separate Zookeeper cluster.

\* from version 2.8 onwards Apache Kafka is not depending on Zookeeper anymore. For more details see: [Apache Kafka Needs No Keeper: Removing the Apache ZooKeeper Dependency](https://www.confluent.io/blog/removing-zookeeper-dependency-in-kafka/) on the Confluent blog.

To be able to see metadata of the Kafka cluster from Zookeeper first connect to Zookeeper using the zookeeper-shell command that ships with the Kafka distribution.

$KAFKA\_HOME/bin/zookeeper-shell.sh $ZK\_HOSTS:2181

🐳 Example to execute in Docker:

./usr/bin/zookeeper-shell zookeeper:2181

Output:

Connecting to zookeeper:2181  
Welcome to ZooKeeper!  
JLine support is disabledWATCHER::WatchedEvent state:SyncConnected type:None path:null

You are now connected to the ZooKeeper cluster.

## Show the Kafka cluster id

$KAFKA\_HOME/bin/zookeeper-shell $ZK\_HOSTS:2181 get /cluster/id

🐳 Example to execute in Docker:

./usr/bin/zookeeper-shell zookeeper:2181 get /cluster/id

Output:

Connecting to zookeeper:2181WATCHER::WatchedEvent state:SyncConnected type:None path:null  
{"version":"1","id":"cDpWBoJpQraTnNqSwB\_4Tg"}

In this example, the id of the Kafka cluster is: cpWBoJpQraTnNqSwB\_4Tg

## List brokers in the Kafka cluster

Let’s list the broker in the Kafka cluster

ls /brokers/ids

Output:

[1]

In my docker setup, there is only one Kafka broker in the cluster.  
The id of the Kafka broker is: 1

## Show details of a Kafka broker

get /brokers/ids/1

## Show all the topics that exist in the cluster

ls /brokers/topics

Output:

[my-first-topic]

## Show details of a specific topic

get /brokers/topics/my-first-topic

## **Exit the zookeeper shell**

exit

From Kafka version 3.0.0. onwards ‘exit’ doesn’t work ‘quit’ should be used instead.

# Topics

In this chapter, you find the CLI command and options that are related to Kafka topics.

## Create a Kafka topic

$KAFKA\_HOME/bin/kafka-topics.sh --zookeeper $ZK\_HOSTS **--create** --topic $TOPIC\_NAME --partitions 3 --replication-factor 1

🐳 Example to execute in Docker:

./usr/bin/kafka-topics --zookeeper zookeeper:2181 **--create** --topic my-first-topic --partitions 3 --replication-factor 1

## Create a Kafka topic in case the topic doesn’t exist

$KAFKA\_HOME/bin/kafka-topics.sh --zookeeper $ZK\_HOSTS --create --topic $TOPIC\_NAME --partitions 3 --replication-factor 1 **--if-not-exists**

🐳 Example to execute in Docker:

./usr/bin/kafka-topics --zookeeper zookeeper:2181 --create --topic my-first-topic --partitions 3 --replication-factor 1 **--if-not-exists**

Note the CLI will not give an error in case the topic already exists.

## Create a Kafka topic with a short retention

By default, a Kafka topic has a retention period of [7 days](https://kafka.apache.org/documentation/#topicconfigs_retention.ms).  
This example shows the command to create a topic with a retention period of 10 seconds.

./usr/bin/kafka-topics --bootstrap-server localhost:9092 --create --topic my-topic-with-short-retention-period --partitions 3 --replication-factor 1 --config retention.ms=10000 --config segment.ms=10000

## List Kafka topics

$KAFKA\_HOME/bin/kafka-topics.sh --zookeeper $ZK\_HOSTS **--list**

🐳 Example to execute in Docker:

./usr/bin/kafka-topics --zookeeper zookeeper:2181 **--list**

## List Kafka topics and exclude internal topics

Exclude listing of external topics like “\_\_consumer\_offsets”

$KAFKA\_HOME/bin/kafka-topics.sh --zookeeper $ZK\_HOSTS **--list** --exclude-internal

🐳 Example to execute in Docker:

./usr/bin/kafka-topics --zookeeper zookeeper:2181 **--list** --exclude-internal

## Show the Kafka topic details

$KAFKA\_HOME/bin/kafka-topics.sh --zookeeper $ZK\_HOSTS --topic $TOPIC\_NAME **--describe**

🐳 Example to execute in Docker:

./usr/bin/kafka-topics --zookeeper zookeeper:2181 --topic my-first-topic **--describe**

## Increase the number of partitions of a Kafka topic

$KAFKA\_HOME/bin/kafka-topics.sh --zookeeper $ZK\_HOSTS   
**--alter** --topic my-first-topic **--partitions** 5

🐳 Example to execute in Docker:

./usr/bin/kafka-topics --zookeeper zookeeper:2181 **--alter** --topic my-first-topic **--partitions** 5

Note:

* The number of partitions for a topic can only be increased.
* If partitions are increased for a topic that has a key, the partition logic or ordering of the messages will be affected.

## Change the retention time of a Kafka topic

259200000 ms = 3 days

$KAFKA\_HOME/bin/kafka-configs.sh --zookeeper $ZK\_HOSTS **--alter** --entity-type topics --entity-name my-first-topic **--add-config retention.ms=259200000**

🐳 Example to execute in Docker:

./usr/bin/kafka-configs --zookeeper zookeeper:2181 **--alter** --entity-type topics --entity-name my-first-topic **--add-config retention.ms=259200000**

## Purge a Kafka topic

At the time of writing, there is no single command to purge a topic.  
As a workaround, you can purge a topic by changing the retention time of a topic to one minute.

$KAFKA\_HOME/bin/kafka-configs.sh --zookeeper $ZK\_HOSTS **--alter** --entity-type topics --entity-name my-first-topic **--add-config retention.ms=1000**

Wait one minute. Kafka will now remove records older than one minute from the topic. Now delete the retention.ms configuration from the topic so it will default back to the retention configuration of the Kafka cluster. The default retention period of a Kafka topic is seven days (unless you configured it differently).

$KAFKA\_HOME/bin/kafka-configs.sh --zookeeper $ZK\_HOSTS **--alter** --entity-type topics --entity-name my-first-topic **--delete-config retention.ms**

In case you had a specific retention period specified on the topic. You need to apply that retention period again (in this example 3 days retention period)

$KAFKA\_HOME/bin/kafka-configs.sh --zookeeper $ZK\_HOSTS --alter --entity-type topics --entity-name my-first-topic **--add-config retention.ms=259200000**

🐳 Example to execute in Docker:

./usr/bin/kafka-configs --zookeeper zookeeper:2181 **--alter** --entity-type topics --entity-name my-first-topic **--add-config retention.ms=1000**

Wait one minute:

./usr/bin/kafka-configs --zookeeper zookeeper:2181 **--alter** --entity-type topics --entity-name my-first-topic **--delete-config retention.ms**

## List Kafka topics (with configuration values) that have specific configuration overrides

$KAFKA\_HOME/bin/kafka-topics.sh --zookeeper $ZK\_HOSTS --describe --topics-with-overrides

🐳 Example to execute in Docker:

./usr/bin/kafka-topics --zookeeper zookeeper:2181 --describe --topics-with-overrides

## Show specific Kafka topic configuration overrides

$KAFKA\_HOME/bin/kafka-topics.sh --zookeeper $ZK\_HOSTS --describe --entity-type topics --entity-name my-first-topic

🐳 Example to execute in Docker:

./usr/bin/kafka-configs --zookeeper zookeeper:2181 --describe --entity-type topics --entity-name my-first-topic

Note in case there a no particular property overrides for the topic this command will not show default cluster properties applied to the topic.

## Show partition offsets of a Kafka topic

$KAFKA\_HOME/bin/kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list localhost:9092 --topic $TOPIC\_NAME

🐳 Example to execute in Docker:

./usr/bin/kafka-run-class kafka.tools.GetOffsetShell --broker-list localhost:9092 --topic my-first-topic

Example output:

my-first-topic:0:3  
my-first-topic:1:3  
my-first-topic:2:3

Format is: topicname:partition-id:offset  
In this example, the output means: on all 3 partitions of topic with the name my-first-topicthere are 3 records.

By default, the latest offset for all partitions is shown.   
Configuration options:

* latest offset: --time -1
* earliest offset: --time -2

to show the earliest offset:

./usr/bin/kafka-run-class kafka.tools.GetOffsetShell --broker-list localhost:9092 --topic my-first-topic --time -2

Example output:

my-first-topic:0:0  
my-first-topic:1:0  
my-first-topic:2:0

## Show offset for specific partition(s) of a Kafka topic

$KAFKA\_HOME/bin/kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list localhost:9092 --topic $TOPIC\_NAME --partitions partition-id, another-partition-id

🐳 Example to execute in Docker:

./usr/bin/kafka-run-class kafka.tools.GetOffsetShell --broker-list localhost:9092 --topic my-first-topic --partitions 0,2

Example output:

my-first-topic:0:3  
my-first-topic:2:3

## Delete a Kafka topic

$KAFKA\_HOME/bin/kafka-topics.sh — zookeeper $ZK\_HOSTS **--delete**   
--topic $TOPIC\_NAME

🐳 Example to execute in Docker:

./usr/bin/kafka-topics --zookeeper zookeeper:2181 **--delete** --topic my-first-topic

# Producers

## Produce a message to a Kafka topic

$KAFKA\_HOME/usr/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic $TOPIC\_NAME

A prompt will open:

>

Type in a message and press enter to publish it to the topic:

>Hello World

Note:

* Repeat the same step to publish more messages
* The message published to Kafka has a null key
* Press Ctrl+C to exit

🐳 Example to execute in Docker:

./usr/bin/kafka-console-producer --broker-list localhost:9092 --topic my-first-topic

## Produce messages to a Kafka topic from a file

Example file topic-input.txt (make sure each message is on a new line):

Hello World  
Kafka Rocks!  
Happy Streaming

Produce messages to the topic from the file:

$KAFKA\_HOME/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic my-first-topic < topic-input.txt

🐳 Example to execute in Docker:

./usr/bin/kafka-console-producer --broker-list localhost:9092 --topic my-first-topic < topic-input.txt

## Produce messages to Kafka with both key and value

By default messages sent to a Kafka topic will result in messages with nullkeys. In this example, the separator between the key and the value is: :

$KAFKA\_HOME/usr/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic $TOPIC\_NAME **--property parse.key=true --property key.separator=:**

🐳 Example to execute in Docker:

./usr/bin/kafka-console-producer --broker-list localhost:9092 --topic some-topic **--property parse.key=true** **--property key.separator=\***

Example input:

>key:value>foo:bar>anotherKey:another value

# Consumers

## Consume a Kafka topic

$KAFKA\_HOME/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic $TOPIC\_NAME

🐳 Example to execute in Docker:

./usr/bin/kafka-console-consumer --bootstrap-server localhost:9092 --topic my-first-topic

If a consumer group id is not specified, the kafka-console-consumer generates a random consumer group.

## Consume a Kafka topic and show both key, value and timestamp

By default, the console consumer will only the value of the Kafka record.  
Using this command you can show both the key and value.

$KAFKA\_HOME/bin/kafka-console-consumer --bootstrap-server localhost:9092 --topic some-topic **--formatter kafka.tools.DefaultMessageFormatter --property print.timestamp=true --property print.key=true --property print.value=true**

🐳 Example to execute in Docker:

./usr/bin/kafka-console-consumer --bootstrap-server localhost:9092 \ --topic some-topic \  
**--formatter kafka.tools.DefaultMessageFormatter \  
--property print.timestamp=true** **\  
--property print.key=true \   
--property print.value=true**

## Consume a Kafka topic from the beginning

$KAFKA\_HOME/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic my-first-topic **--from-beginning**

🐳 Example to execute in Docker:

./usr/bin/kafka-console-consumer --bootstrap-server localhost:9092 --topic my-first-topic **--from-beginning**

# Miscellaneous

Find all the partitions where one or more of the replicas for the partition are not in-sync with the leader.

$KAFKA\_HOME/bin/kafka-topics.sh --zookeeper localhost:2181 --describe --under-replicated-partitions

🐳 Example to execute in Docker:

./usr/bin/kafka-topics --zookeeper zookeeper:2181 --describe --under-replicated-partitions

**Setting up Artemis AMQP**

# High availability and failover configurations are set up on two VMs of ActiveMQ Artemis.

Aug 17, 2023

--

High availability and Failover is when the system can keep working even if one or more servers stop working.

Failover is a part of high availability and it means that if one server (live server) fails, the client connections can move to another server (backup server) so that the client applications can keep working without any interruption.

Apache ActiveMQ Artemis supports two different strategies for backing up a server shared store and replication. Which is configured via the ha-policy configuration element.

**Replication**

Replicated clusters in ActiveMQ Artemis are setups where the data is synchronized between the master and slave brokers. When messages are sent to the master broker, the data is replicated to all the configured slave brokers. This ensures that even if the master broker goes down, the data is available on the slave brokers, allowing them to take over the role of the master and continue serving clients.

**Shared-store**

When using a shared store, both live and backup servers share the same entire data directory using a shared file system. When failover occurs and a backup server takes over, it will load the persistent storage from the shared file system and clients can connect to it.

Both servers we use replicated clustered configurations.

# Pre-Installations

Before install Artemis make sure you have Java 11 or later version installed on your machine. If not you can install JDK from the following link. <https://www.oracle.com/java/technologies/downloads/>

In this project, I used two different operating systems. It is not a necessity for them to be different, but I personally use MacOS and set up a Linux virtual machine through Parallels. For this reason, I will provide installation details for both operating systems. If you also want to install Parallels for Mac (M1), you can do so through this website: <https://www.parallels.com/eu/>

# Install and Run ActiveMQ on MacOS (M1)

• **Download Apache ActiveMQ Artemis**: Visit the official Apache ActiveMQ Artemis website (<https://activemq.apache.org/components/artemis/download/>) and download the latest version of the software.

• **Extract the Archive**: Once the download is complete, extract the downloaded archive to a location of your choice on your Mac.

* Set Environment Variables: Open the Terminal and set the ARTEMIS\_HOME environment variable to point to the extracted folder’s location. For example:

$ export ARTEMIS\_HOME=/path/to/extracted/folder

* **Create a Broker Instance**: Run the following command to create a new broker instance:

$ cd $ARTEMIS\_HOME/bin  
$ ./artemis create <instance-name>

**Error! Filename not specified.**

Creating a broker instance

* **Start the Server:** After creating the broker instance, navigate to its bin directory and run the following command to start broker:

$ ./artemis run

* Now you can open the login page <http://localhost:8161/console> in your browser.

**Error! Filename not specified.**

Login page

# Install and Run ActiveMQ on Ubuntu 22.04

* We should not run Artemis under the root user for security reasons. So, we need to create a group artemis and add a user artemis and we are going to install Artemis under /opt/artemis directory:

$ sudo groupadd artemis  
$ sudo useradd -s /bin/false -g artemis -d /opt/artemis artemis

* Now we can download Artemis version 2.29.0 on the /opt directory

$ cd /opt  
$ sudo wget https://archive.apache.org/dist/activemq/activemq-artemis/2.29.0/apache-artemis-2.29.0-bin.tar.gz

Version of slave device should be same version as master. My master’s version is 2.29.0.

* Ater finish the download extract the tar package and rename the extracted directory to **artemis**

$ sudo tar -xvzf apache-artemis-2.8.1-bin.tar.gz  
$ sudo mv apache-artemis-2.8.1 artemis

* We need to change the permission and the ownership of the Artemis Home directory. We will also give executed permission to opt/artemis/bin/ directory.

$ sudo chown -R artemis: artemis  
$ sudo chmod o+x /opt/artemis/bin/

* And finally we can create an instance like we do in the Mac. I will create into /var/lib directory.

$ cd /var/lib  
$ sudo /opt/artemis/bin/artemis create test-broker

**Error! Filename not specified.**

Create a broker instance

* We can run using the below command

$ sudo /var/lib/test-broker/bin/artemis run

* Once the broker instance starts we can access the admin console at <http://localhost:8161/console> and the login page will appear.

**Error! Filename not specified.**

Login page

# Create Replicated Brokers

After setting up the installations and test brokers for the two different operating systems, we can now proceed to configure the replica cluster configurations. To make servers replicated we can create as replicated clustered. On master device we need to create master instance like the below command:

$ ./artemis create –-replicated –-clustered master

For slave broker we need to run slave command and create the backup instance on Ubuntu.

$ ./artemis create –-replicated –-clustered slave

After we run these commands both of the systems ask you for additional inputs:

* **— user :** it is the default username to login Artemis.
* **— password :** it is the login password.
* **— host :** your device IP (master’s or slave’s).
* **— cluster-user :** a username for cluster connection.
* **— cluster-password :** password for cluster connection.
* **— allow-anonymous | — require-login** **:** allow anonymous access or not.

**Error! Filename not specified.**

Replicated broker master example

However, some manual changes may still be required to set up the replication. We need to edit broker.xml files to each instances as I will explained below.

# Cluster Configurations Of Master And Slave Brokers

Artemis provides cluster configuration to create a group of brokers that work together. This cluster setup allows multiple brokers to share the workload and handle failover seamlessly. In case one broker fails, another can take over to ensure uninterrupted message processing.

**Both brokers include these three main components:**

## <broadcast-groups>

* Helps brokers discover each other even if they are on different network segments

<broadcast-groups>  
 <broadcast-group name="gss-broadcast">  
 <group-address>${udp-address:231.7.7.7}</group-address>  
 <group-port>9876</group-port>  
 <broadcast-period>100</broadcast-period>  
 <connector-ref>artemis</connector-ref>  
 </broadcast-group>  
 </broadcast-groups>

## <discovery-groups>

* Allows brokers to learn about each other dynamically and join the cluster.

<discovery-groups>  
 <discovery-group name="gss-discovery">   
 <group-address>${udp-address:231.7.7.7}</group-address>  
 <group-port>9876</group-port>  
 <refresh-timeout>10000</refresh-timeout>  
 </discovery-group>  
 </discovery-groups>

## <cluster-connections>

* Enables brokers to share information, such as message state and data replication, to achieve high availability and failover. Slave cluster can discover if a live server is already running, see [check-for-live-server](https://activemq.apache.org/components/artemis/documentation/1.1.0/ha.html).
* Master broker: include [<use-duplicate-detection>](https://activemq.apache.org/components/artemis/documentation/1.1.0/ha.html) for filtering out duplicate messages without you having to code your own fiddly duplicate detection logic at the application level.

<cluster-connections>  
 <cluster-connection name="gss-cluster">  
 <address>jms</address>  
 <connector-ref>artemis</connector-ref>  
 <retry-interval>500</retry-interval>  
 <use-duplicate-detection>true</use-duplicate-detection>  
 <message-load-balancing>ON\_DEMAND</message-load-balancing>  
 <max-hops>1</max-hops>  
 <static-connectors>  
 <connector-ref>artemis</connector-ref>  
 </static-connectors>  
 </cluster-connection>  
 </cluster-connections>

* Slave broker : same configuration but without duplicate detection.
* To connect clusters we need to add user and password information

<cluster-user>admin</cluster-user>  
 <cluster-password>admin</cluster-password>

We should make the master and slave configurations to allows to be linked together as live — backup groups.

# Configuration Of Master Broker

Master cluster is the live server until it is down… After we done with cluster and groups we need to add HA Policies.

This is the HA policies of master:

<ha-policy>  
 <replication>  
 <master>  
 <check-for-live-server>true</check-for-live-server>  
 </master>  
 </replication>  
 </ha-policy>

**<check-for-live-server> :** This option is only necessary for performing ‘fail-back’ on replicating servers. If set, backup servers will only pair with live servers with matching group-name.

## **NOTE:**

We need to start with <connectors> and <acceptors>. For master cluster we need to change the connection and also acceptors IP’s with master device IP.

<connectors>  
 <connector name="artemis">tcp://Master\_device\_IP:61616</connector>   
</connectors>

<acceptor name= "artemis">tcp://Master\_device\_IP:61616?tcpSendBufferSize= 1048576;tcpReceiveBufferSize=1048576;amqpMinLargeMessageSize=102400;protocols=CORE,AMQP,STOMP,HORNETQ,MQTT,OPENWIRE;useEpoll=true;amqpCredits=1000;amqpLrnalManagementObjects=false</acceptor>

# Configuration Of Slave Broker

Slave broker is the backup server that when live server is down the backup server will replace and become the current live server. Here are some HA policies for the slave broker that we need to make:

* **< allow-failback> :** Whether a server will automatically stop when a another places a request to take over its place. The use case is when the backup has failed over

<ha-policy>  
 <replication>  
 <slave>  
 <allow-failback>true</allow-failback>  
 </slave>  
 </replication>  
 </ha-policy>

* **<connectors> :** These connectors allow clients to connect to both the master and slave brokers in our messaging system.

<connectors>  
 <connector name="master-broker">tcp://Master\_device\_IP:61616</connector>  
 <connector name="slave-broker">tcp://Slave\_device\_IP:61616</connector>  
 </connectors>

* **<acceptors>** : Allows TCP settings and protocol options for Artemis messaging

<acceptor name= "artemis">tcp://Slave\_device\_IP:61616?tcpSendBufferSize= 1048576;tcpReceiveBufferSize=1048576;amqpMinLargeMessageSize=102400;protocols=CORE,AMQP,STOMP,HORNETQ,MQTT,OPENWIRE;useEpoll=true;amqpCredits=1000;amqpLrnalManagementObjects=false</acceptor>

# Run Brokers

After we done with configurations we can test by running each broker.

* First run the master broker on Mac and open the login page <http://localhost:8161/console> from your browser. Enter the user and password as you configured.
* Now we can run the slave broker on Ubuntu and open the login page.
* We can see from Cluster Info section that backup server is connected to the master server.
* If we kill the master the slave automatically will be the current live server (but like copy of master).

## **Master broker before kill**

It shows is live and backup true.

* Lives : 1
* Backups : 1
* HA Policy : Replicated
* replicating : true

**Error! Filename not specified.**

Master broker

## **Slave broker after kill the Master broker**

The master broker is no longer live, and the slave broker has taken its place. All addresses and clients that were previously connected are now linked to the slave broker (the new live broker).

* Lives : 1
* Backups : 0
* HA Policy : Replica

**Error! Filename not specified.**

Slave broker

**In conclusion, it’s not relevant where your master and slave brokers are located; these configurations are equally effective for both scenarios.**

# My own ActiveMQ Artemis Cheat Sheet

[ActiveMQ Artemis](https://activemq.apache.org/components/artemis/) is a high-performance messaging system for highly scalable microservices or asynchronous messaging between different systems.

This cheat sheet includes the most common commands to install, deploy, administrate or operate a messaging system based in ActiveMQ Artemis.

**NOTE**: ActiveMQ Artemis is the upstream project of [Red Hat AMQ 7 Broker](https://access.redhat.com/products/red-hat-amq#broker-gs), so these commands are also valid for this product.

## Cluster Topologies

ActiveMQ Artemis allows defining different of topologies to build simple or complex messaging solutions. The most standard topologies to cover High Availability, scalable and failover scenarios are:

* Symmetric cluster with Replicated journal
* Symmetric cluster with Shared journal

For other cluster topologies, review [Cluster Topologies](https://activemq.apache.org/components/artemis/documentation/latest/clusters.html#cluster-topologies).

### Symmetric Cluster with Replicated Journal

Topology based in:

* Set of live brokers with network journal replication (Symmetric Cluster and High Availability)
* Set of backup brokers for the live brokers (Failover)

This diagram shows us this topology:

### Symmetric Cluster with Shared Journal

Topology based in:

* Set of live brokers with a shared journal (Symmetric Cluster and High Availability)
* Set of backup brokers for the live brokers (Failover)

This diagram shows us this topology:

## Deploying Cluster Topologies

### Deploying a Symmetric Cluster with Replicated Journal

This command will create a live broker instance in a replicated journal topology:

$ARTEMIS\_HOME/bin/artemis create /opt/brokers/live-replicated-broker-XX \

--http-host $HOSTNAME \

--host $HOSTNAME \

--aio \

--clustered \

--cluster-user $ARTEMIS\_CLUSTER\_USER \

--cluster-password $ARTEMIS\_CLUSTER\_PASSWORD \

--name live-replicated-broker-XX \

--max-hops 1 \

--user $ARTEMIS\_ADMIN\_USER \

--password $ARTEMIS\_ADMIN\_PASSWORD \

--require-login \

--port-offset 0 \

--no-autocreate \

--replicated \

--failover-on-shutdown

This other command will create a backup broker instance for a replicated journal topology:

$ARTEMIS\_HOME/bin/artemis create /opt/brokers/backup-replicated-broker-XX \

--http-host $HOSTNAME \

--host $HOSTNAME \

--aio \

--clustered \

--cluster-user $ARTEMIS\_CLUSTER\_USER \

--cluster-password $ARTEMIS\_CLUSTER\_PASSWORD \

--name live-replicated-broker-XX \

--max-hops 1 \

--user $ARTEMIS\_ADMIN\_USER \

--password $ARTEMIS\_ADMIN\_PASSWORD \

--require-login \

--port-offset 0 \

--no-autocreate \

--replicated \

--failover-on-shutdown \

--slave

By default, ActiveMQ Artemis will use a broadcast network to discover the instances of the topology. The values are identified at runtime of the command and store them in the configuration files.

In case you need to identify the different instances of the topology, the staticCluster property is needed. This property identifies the static list of other brokers of the topology.

--staticCluster $ARTEMIS\_STATIC\_CLUSTER\_LIST

Samples:

* **ARTEMIS\_STATIC\_CLUSTER\_LIST**: tcp://live-broker-01:61616,...,tcp://live-broker-XX:61616

### Deploying a Symmetric Cluster with Shared Journal

This command will create a live broker of a live/backup pair using a shared journal:

$ARTEMIS\_HOME/bin/artemis create /opt/brokers/live-broker \

--http-host $HOSTNAME \

--host $HOSTNAME \

--aio \

--clustered \

--cluster-user $ARTEMIS\_CLUSTER\_USER \

--cluster-password $ARTEMIS\_CLUSTER\_PASSWORD \

--name live-broker \

--max-hops 1 \

--user $ARTEMIS\_ADMIN\_USER \

--password $ARTEMIS\_ADMIN\_PASSWORD \

--require-login \

--port-offset 0 \

--no-autocreate \

--data $ARTEMIS\_SHARED\_STORAGE\_PATH \

--shared-store \

--failover-on-shutdown

For the backup broker instance:

$ARTEMIS\_HOME/bin/artemis create /opt/brokers/backup-broker \

--http-host $HOSTNAME \

--host $HOSTNAME \

--aio \

--clustered \

--cluster-user $ARTEMIS\_CLUSTER\_USER \

--cluster-password $ARTEMIS\_CLUSTER\_PASSWORD \

--name backup-broker \

--max-hops 1 \

--user $ARTEMIS\_ADMIN\_USER \

--password $ARTEMIS\_ADMIN\_PASSWORD \

--require-login \

--port-offset 100 \

--no-autocreate \

--data $ARTEMIS\_SHARED\_STORAGE\_PATH \

--shared-store \

--slave \

--failover-on-shutdown \

Here, the key point is that $ARTEMIS\_SHARED\_STORAGE\_PATH mount a shared storage between both brokers. It is very common to use a NFSv4 mounted folder.

More details in [Persistence](https://activemq.apache.org/components/artemis/documentation/latest/persistence.html).

## HA and Failover life cycle

ActiveMQ Artemis includes a failover feature based in the combination of Live and Backup brokers. This feature allows defining an automatic client failover connection.

A client can receive information about all live and backup brokers, so that in the event of a connection failure, it can reconnect to the backup broker. The backup broker then automatically re-creates any sessions and consumers that existed on each connection before failover. This feature saves you from having to hand-code manual reconnection logic in your applications.

When a session is re-created on the backup, it does not have any knowledge of messages already sent or acknowledged. Any in-flight sends or acknowledgements at the time of failover might also be lost. However, even without 100% transparent failover, it is simple to guarantee once and only once delivery, even in the case of failure, by using a combination of duplicate detection and retrying of transactions.

Client connection strings identify a connection string using the Core protocol with the right values to enable the high availability and failover features from the client side. For example:

* Single broker: tcp://HOSTNAME1:61616?ha=true&reconnectAttempts=10
* Failover connection: (tcp://HOSTNAME1:616161,tcp://HOSTNAME2:61616)?ha=true&reconnectAttempts=10

Every time a live broker shutdown, the backup instance will be promoted as lived instance to continue the service to producers and consumers, and also to distribute the messages to other members of the topology. The backup instance could delegate again the role to the live instance as soon is is ready. The following definition in the backup instance automate that life cycle:

<ha-policy>

<shared-store>

<slave>

<allow-failback>true</allow-failback>

</slave>

</shared-store>

</ha-policy>

More details in [High Availability and Failover](https://activemq.apache.org/components/artemis/documentation/latest/ha.html).

## Message Redistribution

To enable [messaging redistribution](https://activemq.apache.org/components/artemis/documentation/latest/clusters.html#message-redistribution) between brokers in a cluster, the **redistribution-delay** property must be enabled to zero in the <address-setting> in broker.xml file:

<!--default for catch all-->

<address-setting match="#">

<redistribution-delay>0</redistribution-delay>

</address-setting>

## Protocols

ActiveMQ Artemis has a pluggable protocol architecture, so that it can easily enable one or more protocols for a network connection. The broker supports the following protocols:

* Core
* AMQP
* MQTT
* OpenWire
* STOMP
* HornetQ

Each protocol could be defined as secure channel (SSL). These channels could be used to define one-way or two-ways to validate the communications from the clients.

More details in [Protocols and Interoperability](https://activemq.apache.org/components/artemis/documentation/latest/protocols-interoperability.html).

## Starting a broker

Start a broker instance is very simple:

$ARTEMIS\_BROKER\_INSTANCE\_HOME/bin/artemis run

Or you can run the broker in the background using:

$ARTEMIS\_BROKER\_INSTANCE\_HOME/bin/artemis-service start

## Installing as OS service (systemd)

Create a file called artemis.service file in the /etc/systemd/system folder. This file will have the following content:

[Unit]

Description = ActiveMQ Artemis - Broker

After = syslog.target network.target

[Service]

ExecStart = /opt/brokers/live/bin/artemis run

ExecStop = /opt/brokers/live/bin/artemis stop

User = amq-broker

Group = amq-broker

SuccessExitStatus = 0 143

RestartSec = 60

Restart = on-failure

LimitNOFILE = 102642

[Install]

WantedBy = multi-user.target

To Enable the OS services:

systemctl enable artemis

To start, stop, check the status or restart the OS service:

systemctl (start|stop|status|restart) artemis

## Modularizing Broker Configuration

ActiveMQ Artemis supports XML inclusions so the configuration can be broken out into separate files. It is an interesting feature to prevent errors and to ease populating common configuration between clustered brokers.

By default, the etc/broker.xml file declares XInclude XML namespace.

<configuration xmlns="urn:activemq"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:xi="http://www.w3.org/2001/XInclude"

...

For example, we could move the addresses tag to an external directory in each host. The live and backup brokers of the same host will share an addresses.xml configuration file.

❯ cat /opt/brokers/config/addresses.xml

<addresses xmlns="urn:activemq:core">

<address name="DLQ">

<anycast>

<queue name="DLQ" />

</anycast>

</address>

<address name="ExpiryQueue">

<anycast>

<queue name="ExpiryQueue" />

</anycast>

</address>

<!-- Queues and Topics -->

<address name="SampleQueue">

<anycast>

<queue name="SampleQueue" />

</anycast>

</address>

<address name="SampleTopic">

<multicast/>

</address>

</addresses>

This file must be copied in any ActiveMQ Artemis cluster hosts. Finally each broker.xml file will include it using the xi:include tag:

<xi:include href="/opt/brokers/config/addresses.xml"/>

**WARNING**: External files are not monitored by the broker so it is needed to refresh or restart it to take the latest status. touch command could be useful to modify the timestamp of the broker.xml configuration file:

touch /opt/brokers/live/etc/broker.xml

## Automatic Configuration

ActiveMQ Artemis is defined to automatically create an address/queue when a new sender/receiver is connected. It is a great feature because it allows us to avoid having to manage the address in the broker.xml file. Also ActiveMQ Artemis also deletes an address/queue when there is not a sender/receiver connected and there are no messages persisted. This feature is also great however it includes some extra staff to manage this process.

The following properties in the address-setting section of broker.xml file manages the automatic configuration:

<address-setting match="#">

<auto-create-addresses>false</auto-create-addresses>

<auto-create-queues>false</auto-create-queues>

<auto-create-jms-queues>false</auto-create-jms-queues>

<auto-create-jms-topics>false</auto-create-jms-topics>

<auto-delete-addresses>false</auto-delete-addresses>

<auto-delete-queues>false</auto-delete-queues>

</address-setting>

More details in [Automatic Configuration](https://activemq.apache.org/components/artemis/documentation/latest/address-model.html#automatic-configuration)

## Producer Command

To send messages to a resource (and easy way to test the broker):

./bin/artemis producer --url '$ARTEMIS\_CORE\_PROTOCOL\_URL' \

--destination $ARTEMIS\_RESOURCE \

--user $ARTEMIS\_ADMIN\_USER --password $ARTEMIS\_ADMIN\_PASSWORD \

--message-size 1024 --message-count 10 --verbose

Where:

* **ARTEMIS\_CORE\_PROTOCOL\_URL**: Identifies a connection string using the Core protocol. Valid values:
  + Single broker: tcp://HOSTNAME1:61616?ha=true&reconnectAttempts=10
  + Failover connection: (tcp://HOSTNAME1:616161,tcp://HOSTNAME2:61616)?ha=true&reconnectAttempts=10
* **ARTEMIS\_RESOURCE**: Identifies the resource. Sample values for:
  + A Queue: queue://SampleQueue
  + A Topic: topic://SampleTopic

## Consumer Command

To consume messages from a resource (and easy way to test the broker):

./bin/artemis consumer --url '$ARTEMIS\_CORE\_PROTOCOL\_URL' \

--destination $ARTEMIS\_RESOURCE \

--user $ARTEMIS\_ADMIN\_USER --password $ARTEMIS\_ADMIN\_PASSWORD \

--message-count 10 --verbose

Where:

* **ARTEMIS\_CORE\_PROTOCOL\_URL**: Identifies a connection string using the Core protocol. Valid values:
  + Single broker: tcp://HOSTNAME1:61616?ha=true&reconnectAttempts=10
  + Failover connection: (tcp://HOSTNAME1:616161,tcp://HOSTNAME2:61616)?ha=true&reconnectAttempts=10
* **ARTEMIS\_RESOURCE**: Identifies the resource. Valid values:
  + Queue: queue://SampleQueue
  + Topic: topic://SampleTopic

## Durable Subscription Queue

The broker saves messages for any inactive subscribers when a queue is configured as a Durable Subscription. The broker delivers them to the subscribers when they reconnect. Clients are therefore guaranteed to receive each message delivered to the queue after subscribing to it.

A sample definition will be similar to:

<address name="topic">

<multicast>

<queue name="subscription1">

<durable>true</durable>

</queue>

<queue name="subscription2">

<durable>true</durable>

</queue>

</multicast>

</address>

This is a sample definition of a Durable Subscription Queue to be used with the Artemis producer and consumer commands:

<address name="topic.events">

<multicast>

<queue name="c1.Consumer ActiveMQTopic[topic.foo], thread=0">

<durable>true</durable>

</queue>

<queue name="c2.Consumer ActiveMQTopic[topic.foo], thread=0">

<durable>true</durable>

</queue>

</multicast>

</address>

This command produces a number of messages to the topic:

./bin/artemis producer --url tcp://$HOSTNAME:5672 \

--user admin --password admin \

--protocol amqp \

--destination topic://topic.events \

--threads 1 \

--message-count 100 \

--text-size 1024

This command consumes the messages from the c1 subscription:

./bin/artemis consumer --url tcp://$HOSTNAME:5672 \

--user admin --password admin \

--protocol amqp \

--destination topic://topic.events \

--threads 1 \

--message-count 100 \

--durable \

--clientID c1

This other command consumes the messages from the c2 subscription:

./bin/artemis consumer --url tcp://$HOSTNAME:5672 \

--user admin --password admin \

--protocol amqp \

--destination topic://topic.events \

--threads 1 \

--message-count 100 \

--durable \

--clientID c2

**NOTE**: These queues could be consumed using JMS Message Consumers using [Fully Qualified Queue Names (FQQN)](https://activemq.apache.org/components/artemis/documentation/latest/address-model.html) as:

String FQQN = "topic::subscription1";

Queue queueDestination session.createQueue(FQQN);

MessageConsumer consumer = session.createConsumer(queueDestination);

More details in [Addressing Model](https://activemq.apache.org/components/artemis/documentation/latest/address-model.html).

## Clustered Message Grouping

This feature allows to process messages with a particular group ID in the same order by the consumers. Each clustered broker therefore uses a grouping handler to manage the complexity of routing of grouped messages. Each clustered broker should choose should choose a grouping handler type: Local or Remote.

Local grouping handler broker (only one per cluster topology):

<grouping-handler name="my-grouping-handler">

<type>LOCAL</type>

<address>SampleQueue</address>

<timeout>5000</timeout>

</grouping-handler>

Remote grouping handler broker (the rest of instances of the cluster topology):

<grouping-handler name="my-grouping-handler">

<type>REMOTE</type>

<address>SampleQueue</address>

<timeout>5000</timeout>

</grouping-handler>

To produce grouped messages:

./bin/artemis producer --url $ARTEMIS\_CORE\_PROTOCOL\_URL \

--destination $ARTEMIS\_RESOURCE \

--user $ARTEMIS\_ADMIN\_USER --password $ARTEMIS\_ADMIN\_PASSWORD \

--message "Sample Grouped Message" \

--message-count 10 \

--group mygroup

To consume grouped messages:

./bin/artemis consumer --url $ARTEMIS\_CORE\_PROTOCOL\_URL \

--destination $ARTEMIS\_RESOURCE \

--user $ARTEMIS\_ADMIN\_USER --password $ARTEMIS\_ADMIN\_PASSWORD \

--message-count 10 \

--verbose

More details in [Clustered Message Grouping](https://activemq.apache.org/components/artemis/documentation/latest/message-grouping.html)

## AMQP Secured Connection

Client connection string for AMQP secure protocol:

amqps://HOSTNAME:5671?sslEnabled=true&transport.trustAll=true&transport.verifyHost=false

More details in [Configuring Transports](https://activemq.apache.org/components/artemis/documentation/latest/configuring-transports.html)

## Monitoring

ActiveMQ Artemis includes [Jolokia](https://jolokia.org/) endpoints to execute administrative tasks or query administrative information.

Samples:

* Query the up time of the broker:

❯ curl -u admin:admin 'http://localhost:8161/console/jolokia/read/org.apache.activemq.artemis:broker=!%22live!%22/Uptime' | jq

{

"request": {

"mbean": "org.apache.activemq.artemis:broker=\"live\"",

"attribute": "Uptime",

"type": "read"

},

"value": "1 minute",

"timestamp": 1653917729,

"status": 200

}

* Query to get the total number of messages added:

❯ curl -u admin:admin 'http://localhost:8161/console/jolokia/read/org.apache.activemq.artemis:broker=!%22live!%22/TotalMessagesAdded' | jq

{

"request": {

"mbean": "org.apache.activemq.artemis:broker=\"live\"",

"attribute": "TotalMessagesAdded",

"type": "read"

},

"value": 700,

"timestamp": 1653917823,

"status": 200

}

## Performance and Limits

ActiveMQ Artemis manages several resources (descriptors, connections, …) and it is needed to define the OS limits to allow it for the user that runs it.

Review the /etc/security/limits.conf file to add the following definition for this user:

amq-broker soft nofile 65001

amq-broker hard nofile 65001

**WARNING**: In RHEL 8 this step is no longer needed since nofile defaults have been increased to 1048576 max open files.

ActiveMQ Artemis includes a general server thread pool used for most asynchronous actions on the server side. This pool is defined by default to use only 30 threads and it is very useful to improve the performance.

Definition at broker.xml file:

<thread-pool-max-size>120</thread-pool-max-size>

There are a few things that can go wrong in a production environment (bugs, IO errors, memory issues, …), so ActiveMQ Artemis includes a protection to shut itself down when bad things happen (as a safeguard). This method includes different policies:

* **LOG** (default): Log messages into artemis.log to inform that something is wrong.
* **HALT** (default at broker creation): Stop the messaging process but not the VM.
* **SHUTDOWN**: Shutdown the VM process.

To check easily if a broker suffered an issue the best practice is to use SHUTDOWN policy. It is very easy to check if the broker is running or not checking the service or the java process in OS.

Definition at broker.xml file:

<critical-analyzer-policy>SHUTDOWN</critical-analyzer-policy>

More details about performance, tunning and extra features, please review the following references:

* [Performance Tuning](https://activemq.apache.org/components/artemis/documentation/latest/perf-tuning.html)
* [Thread management](https://activemq.apache.org/components/artemis/documentation/latest/thread-pooling.html)
* [Critical Analysis of the broker](https://activemq.apache.org/components/artemis/documentation/latest/critical-analysis.html)

## Managing users

Add a user:

./bin/artemis user add --url $ARTEMIS\_CORE\_PROTOCOL\_URL \

--user admin --password admin \

--user-command-user user1 \

--user-command-password user1-password1 \

--role role1

Reset a user (change user password and/or role/s):

./bin/artemis user reset --url $ARTEMIS\_CORE\_PROTOCOL\_URL \

--user admin --password admin \

--user-command-user user1 \

--user-command-password user1-password2 \

--role role2,role3

Remove a user:

./bin/artemis user rm --url $ARTEMIS\_CORE\_PROTOCOL\_URL \

--user admin --password admin \

--user-command-user user1

## Mask passwords

To mask password to be added in broker.xml file:

$ARTEMIS\_HOME/bin/artemis mask $PASSWORD

More details in [Generating a masked password](https://activemq.apache.org/components/artemis/documentation/latest/masking-passwords.html#generating-a-masked-password).

## Getting more Help

ActiveMQ Artemis CLI tool includes an option to get in line help and documentation for each command implemented.

To get detailed information of any kind of command:

./bin/artemis help COMMAND

## ActiveMQ Artemis version

This cheat-sheet was tested and verified with the ActiveMQ Artemis 2.18 version.

**Writing test cases for DB : { mongoDB & PostgresDB }**

**End to End realtime reactive event processing**

1.Combining RxJava operators and Vert.x clients to support advanced processing

2.Using RxJava operators to perform content enrichment and aggregate data processing on top of event streams

3.Extending the Vert.x event bus to web applications to unify backend and frontend communication models

4.Managing state in a stream-processing setting

**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 identifying each of the elements as expected. the root, just like magic, the IDE suddenly seems to have recovered and then started

