**Chapter 2**

**Micronaut**

Sanjay Patnia\*

a UCSC Extension, Santa Clara, California, USA

**Abstract**

Micronaut is a JVM based modern full-stack Microservice framework. This new framework has been developed by Grails team with an intention to solve problems which have been identified over the years while building the real world Microservices applications.

One of the most exciting features of Micronaut is its compile time dependency injection mechanism. Most frameworks use reflection and proxies to perform dependency injection at runtime. Micronaut, however, builds its dependency injection data at compile time. The result is faster application startup and smaller memory footprints.

I think it is not an exaggeration if I say we are living in the age of Microservices. Microservices became de facto architecture pattern for every new enterprise scale application that is being implemented and many existing monolithic applications are getting migrated into Microservices. In the case of Java world, Spring Boot turned out to be the standard framework to develop Microservices. There were some other frameworks like DropWizard, Apache Karaf, and Jersey. But they were not able to give tough competition to Spring Boot and slowly their usage percentage came down and became insignificant over a period of time. If you observe the evolution of Spring Boot, initially it was not proposed as Microservices solution from Spring. It was initially proposed and implemented as the containerless web application and developer community started using it for Microservices implementation. But Spring Boot got its own limitations like:

* Fixed single language
* Lack of Inbuilt support for data accessing
* Lack of Simpler unit testing
* Lack of Inbuilt service discovery
* Lack of Inbuilt load balancing

We need explicit configuration which can be achieved through the cloud services instead of having the built-in support within the framework itself.

Here comes Micronaut which contains the aforementioned features inbuilt and designed with single and primary intent to serve as the vehicle for Microservices development.

**Comparison of Micronaut with Spring Boot**

**Ease of installation**

Both Spring Boot and Micronaut won’t be complex for installation, can be installed easily by following the installation instructions Both frameworks need below prerequisites:

* A favourite text editor or IDE
* JDK 1.8 or later
* Gradle or Maven latest versions

Code which has been generated through the CLI tool can be directly imported into your IDE:

* Spring Tool Suite (STS) — Spring Boot
* Visual Studio Code - Micronaut

**Natively Cloud Enabled**

When it comes to Spring Boot, to support above discussed Cloud-specific features, we need to depend on the third party cloud services or libraries, it doesn’t support any of the above-listed features by default, so Micronaut has an advantage here

The following list of Cloud-specific features are directly integrated into the Micronaut runtime:

* Service discovery
* Eureka, Consul or ZooKeeper service discovery servers are being supported
* Kubernetes container runtime is supported by default.
* Client side load balancing
* Netflix Ribbon can be used for load balancing
* Distributed Configuration
* Distributed Tracing
* Serverless functions

**Serverless Functions**

Serverless architecture, where Developers will deploy the function. From there onwards they are completely managed by the Cloud environment i.e. invocation, execution, control and put down. But Micronaut’s fast startup time, compile-time approach and low-memory footprint make this framework a great candidate for developing functions and in fact, Micronaut features have the dedicated support for implementing and deploying functions to the AWS Lambda and any FaaS system that supports running functions as containers.

**Application Configuration**

Micronaut inspired from both Grails and Spring Boot in integrating configurations properties from different sources directly into the core IoC container. Configurations can be provided by default in either YAML, JSON, Java properties or Groovy files. The convention is to search for a file called application.yml, application.properties, application.json or application.groovy.

* Command line arguments
* Properties from SPRING\_APPLICATION\_JSON (Only if there is any Spring dependency)
* Properties from MICRONAUT\_APPLICATION\_JSON
* Java System Properties
* OS environment variables
* Each Environment-specific properties like application-{environment}.{extension} (could be .properties, .json, .yml or .groovy)
* Application-specific properties from the application.{extension} (could be.properties, .json, .yml or .groovy)
* Special properties ( Random Values )

Spring Boot supports above all property locations, in addition to above, it also supports other property locations: Spring Boot devtools global settings properties

* @TestPropertySource annotations on your tests.
* @SpringBootTest#properties annotation attribute on your tests.
* ServletConfig init parameters.
* ServletContext init parameters.
* JNDI attributes from java:comp/env.
* @PropertySource annotations on your @Configuration classes.
* Default properties (specified by setting SpringApplication.setDefaultProperties).

“Spring Boot provided more ways to handle with properties when we compared it against Micronaut.”

**Messaging System Support**

Spring Boot supports the integration of external Messaging systems, they are:

* RabbitMQ
* Apache Kafka
* ActiveMQ
* Artemis

Micronaut also supports the popular messaging systems, such as:

* RabbitMQ
* Apache Kafka “Micronaut has the embedded support for the Apache Kafka.“ “Both frameworks have the support of the popular Messaging systems but Spring Boot supports more tools.”

**Security**

Spring Boot supports below security mechanisms by default:

* MVC Security
* WebFlux Security
* OAuth2
* Actuator Security

Micronaut supports below security mechanisms by default:

* Authentication Providers
* Security Rules
* IP Pattern Rule
* Secured Annotation
* Intercept URL Map
* Built-In Endpoints Security
* Authentication Strategies
* Basic Auth
* Session Authentication
* JSON Web Token
* Built-In Security Controllers
* Retrieve the Authenticated User
* Security Events

**Caching**

Spring Boot supports following caching providers:

* Redis
* Couchbase
* Generic
* JCache (JSR-107)
* EhCache 2.x
* Hazelcast
* Infinispan
* Caffeine

Whereas Micronaut supports following list of caching providers:

* Caffeine — by default Micronaut supports it.
* Redis

“Obviously, Spring Boot is leading in supporting caching providers”

**Management & Monitoring**

Micronaut inspired from the Grails, Spring Boot and Micronauts management dependency add support to monitor your applications via endpoints, the special URIs that returns details about the state of your application and health.

* Creating Endpoints
* Built-In Endpoints

API Portfolio :

This book will take two business domain problems and build a portfolio of APIs.

1. Online Flight

To illustrate features of Micronaut this book will take example of “online flight” application. The application will enable passengers to view flight they are traveling. You will define two component classes :

1. A service component that lets a passenger to see what flights they are booked in
2. A repository component that stores passengers for a flight. Initially you will store passengers in memory for simplicity.

| **Object** | **Field** | **Type** |  |
| --- | --- | --- | --- |
| Passenger | Name | String |  |
| Flight | Origin | String |  |
|  | Destination | String |  |
|  | Departure | Datetime |  |
|  | Flight# | int |  |

2. Message

This API will enable sending messages to the users in the system.

|  |  |  |  |
| --- | --- | --- | --- |
| Object | Fields | Type |  |
| Message | Message | String |  |
|  | From | String |  |
|  | To | String |  |
|  | Creation Date | Date |  |

Software

This book will use following software for the coding problems.

1. Micronaut

<https://micronaut.io/download/>

INSTALLING WITH SDKMAN!

This tool makes installing the Micronaut framework on any Unix based platform (Mac OSX, Linux, Cygwin, Solaris, or FreeBSD) easy.

Simply open a new terminal and enter:

$ curl -s [https://get.sdkman.io](https://get.sdkman.io/) | bash

Follow the on-screen instructions to complete installation. Open a new terminal or type the command:

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

Then install the latest stable version of the framework:

$ sdk install micronaut

If prompted, make this your default version.

After installation is complete it can be tested with:

$ mn --version

That’s all there is to it!

Now let’s create “hello from Micronaut”.

Assumption micronaut 3, gradle and jdk11 is installed.

mn create-app hello-world

2. JDK 11

<https://jdk.java.net/archive/>

3. POSTMAN

<https://www.postman.com/downloads/>

4 CURL

https://curl.se/download.html

5. You have a 2 choices to use IDE

1. Visual Studio Code

<https://code.visualstudio.com/download>

1. Intelli J

<https://www.jetbrains.com/idea/download/>

6. Maven

<https://maven.apache.org/download.cgi>

Setting up an IDE

The application created in the previous section contains a "main class" located in src/main/java that looks like the following:

package hello.world;

import io.micronaut.runtime.Micronaut;

public class Application {

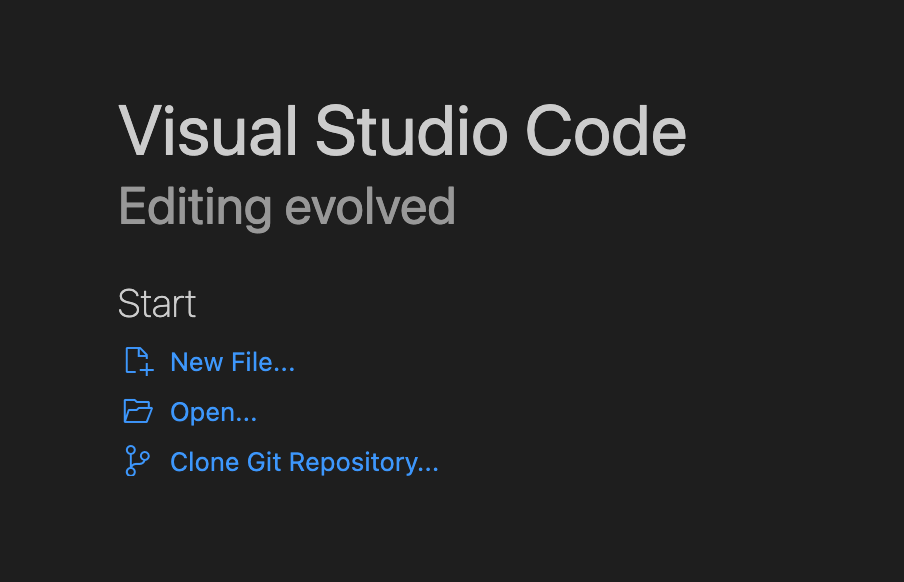
public static void main(String[] args) {

Micronaut.run(Application.class);

}

}

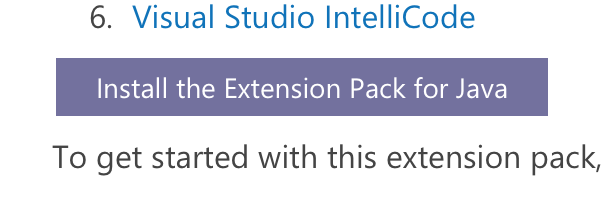
This is the class that is run when running the application via Gradle or via deployment.

Configuring Visual Studio Code

In this book will illustrate use of Visual Studio Code for editing java code.

Open code created in hello-world folder by clicking “Open” and navigating to hello-world folder.

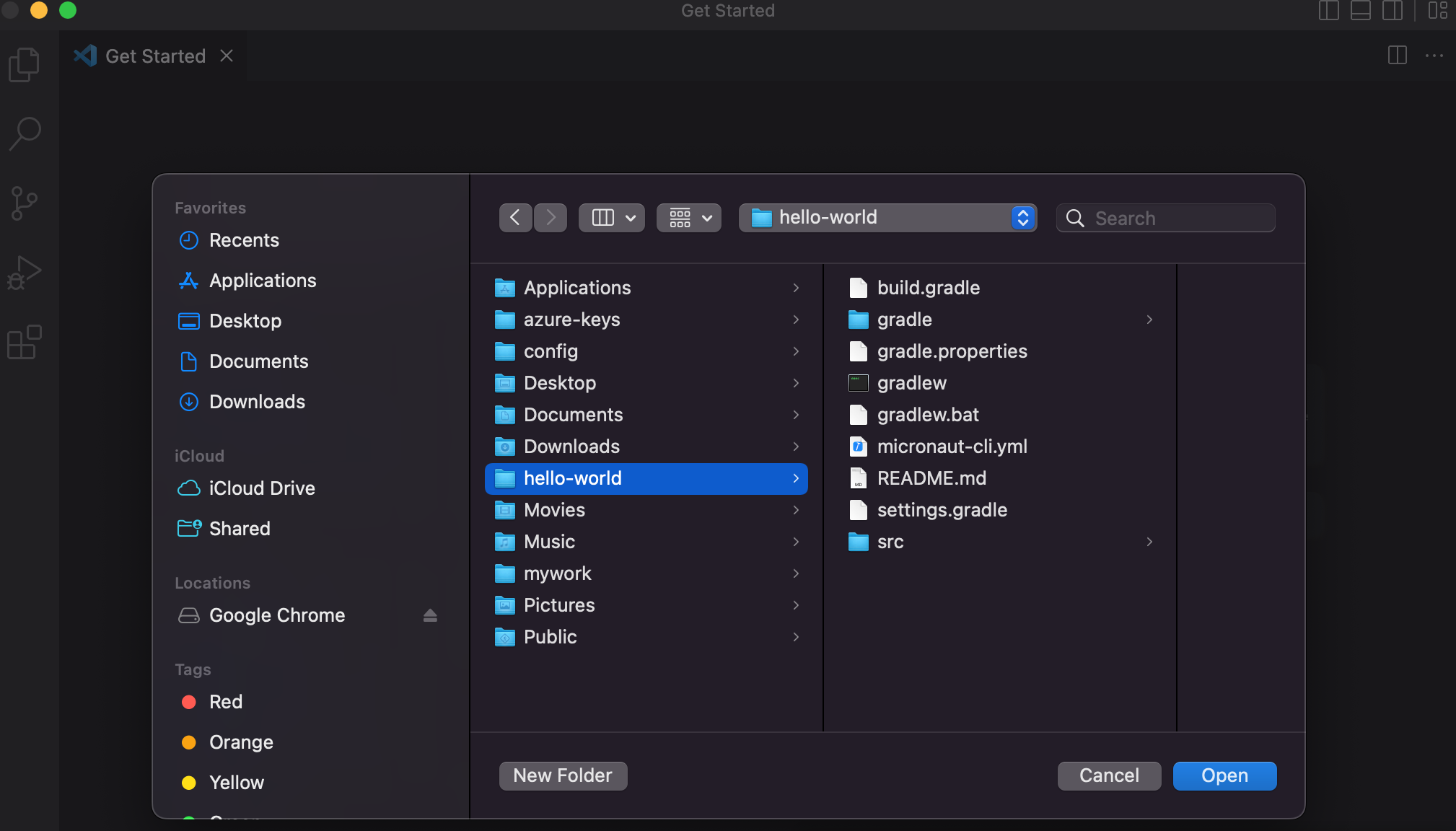
Micronaut can be setup within Visual Studio Code. You will need to first install the [Java Extension Pack](https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-pack).



Extension Pack for Java is a collection of popular extensions that can help write, test and debug Java applications in Visual Studio Code.

<https://code.visualstudio.com/docs/java/extensions>

|  |  |
| --- | --- |
|  | Visual Studio Code support currently only works for Maven builds. Hello world example will run using command line since it uses gradle. |

Once the extension pack is installed you could use IDE for editing java code.

Now create a new class in using File-> New File -> New Java Class

And paste

package hello.world;

import io.micronaut.http.MediaType;

import io.micronaut.http.annotation.Controller;

import io.micronaut.http.annotation.Get;

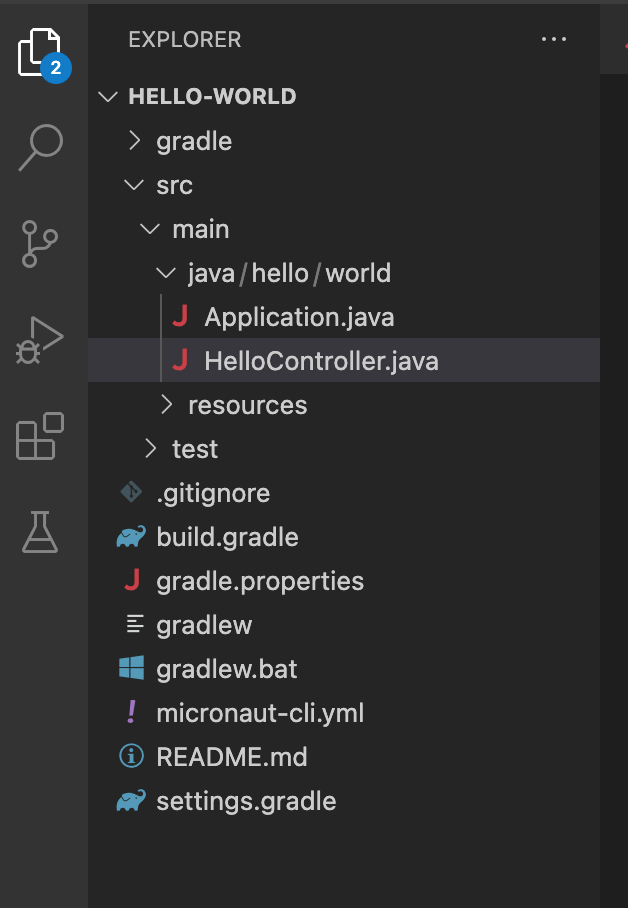
@Controller("/hello") public class HelloController {

@Get(produces = MediaType.TEXT\_PLAIN)

public String index() {

return "Hello from Micronaut"; }

}



Save file as HelloController.java. Files will look as in above. Now to run from command prompt :

cd ~/hello-world

./gradlew run

curl <http://localhost:8080/hello>

You can also run application from IDE by selecting Application.java and right clicking and running.

[1] : <https://walkingtreetech.medium.com/spring-boot-vs-micronaut-the-battle-unleashed-2682354a88e9>

To Review

In this chapter we reviewed features of Micronaut and compared those with Spring Boot. Analyzed sample domains - flight status and message to create a portfolio of APIs using Micronaut.