**Microservices**

Microservices are increasingly used in the development world as developers work to create larger, more complex applications that are better developed and managed as a combination of smaller services that work cohesively together for larger, application-wide functionality. Tools are rising to meet the need to think about and build apps using a piece-by-piece methodology that is, frankly, less mind-boggling than considering the whole of the application at once. Today, we’ll take a look at microservices, the benefits of using this capability, and a few code examples.

**What Are Microservices?**

Microservices are a form of service-oriented architecture style (one of the most [important skills for Java developers](https://stackify.com/great-java-developer-skills/)) wherein applications are built as a collection of different smaller services rather than one whole app. Instead of a monolithic app, you have several independent applications that can run on their own and may be created using different coding or programming languages. Big and complicated applications can be made up of simpler and independent programs that are executable by themselves. These smaller programs are grouped together to deliver all the functionalities of the big, monolithic app.

Microservices captures your business scenario, answering the question “*What problem are you trying to solve*?” It is usually developed by an engineering team with only a few members and can be written in any programming language as well as utilize any framework. Each of the involved programs is independently versioned, executed, and scaled. These microservices can interact with other microservices and can have unique URLs or names while being always available and consistent even when failures are experienced.

**What Are the Benefits of Microservices?**

There are several benefits to using microservices. For one, because these smaller applications are not dependent on the same coding language, the developers can use the programming language that they are most familiar with. That helps developers come up with a program faster with lower costs and fewer bugs. The agility and low costs can also come from being able to reuse these smaller programs on other projects, making it more efficient.

**Examples of Microservices Frameworks for Java**

There are several microservices frameworks that you can use for developing for Java. Some of these are:

* **Spring Boot**: This is probably the best Java microservices framework that works on top of languages for Inversion of Control, Aspect Oriented Programming, and others.
* **Jersey**: This open source framework supports JAX-RS APIs in Java is very easy to use.
* **Swagger**: Helps you in documenting API as well as gives you a development portal, which allows users to test your APIs.

Others that you can consider include: Dropwizard, Ninja Web Framework, Play Framework, RestExpress, Restlet, Restx, and Spark Framework.

**How to Create Using DropWizard**

DropWizard pulls together mature and stable Java libraries in lightweight packages that you can use for your own applications. It uses Jetty for HTTP, Jersey for REST, and Jackson for JSON, along with Metrics, Guava, Logback, Hibernate Validator, Apache HttpClient, Liquibase, Mustache, Joda Time, and Freemarker.

You can setup Dropwizard application using Maven. How?

In your POM, add in a dropwizard.version property using the latest version of DropWizard.

<properties>

<dropwizard.version>LATEST VERSION</dropwizard.version>

</properties>

<!--Then list the dropwizard-core library:-->

<dependencies>

<dependency>

<groupId>io.dropwizard</groupId>

<artifactId>dropwizard-core</artifactId>

<version>${version}</version>

</dependency>

</dependencies>

This will set up a Maven project for you. From here, you can create a configuration class, an application class, a representation class, a resource class, or a health check, and you can also build Fat JARS, then run your application.

Check out the Dropwizard user manual at [this link](http://www.dropwizard.io/1.1.0/docs/manual/index.html). The GitHub library is [here](https://github.com/dropwizard/dropwizard).

Sample code:

package com.example.helloworld;

import com.yammer.dropwizard.config.Configuration;

import com.fasterxml.jackson.annotation.JsonProperty;

import org.hibernate.validator.constraints.NotEmpty;

public class HelloWorldConfiguration extends Configuration {

@NotEmpty

@JsonProperty

private String template;

@NotEmpty

@JsonProperty

private String defaultName = "Stranger";

public String getTemplate() {

return template;

}

public String getDefaultName() {

return defaultName;

}

}

**Microservices With Spring Boot**

Spring Boot gives you Java application to use with your own apps via an embedded server. It uses Tomcat, so you do not have to use Java EE containers. A sample Spring Boot tutorial is at [this link](https://stackify.com/spring-boot-level-up/).

You can find all Spring Boot projects [here](https://spring.io/projects), and you will realize that Spring Boot has all the infrastructures that your applications need. It does not matter if you are writing apps for security, configuration, or big data; there is a Spring Boot project for it.

Spring Boot projects include:

* **Spring IO Platform**: Enterprise grade distribution for versioned applications.
* **Spring Framework**: For transaction management, dependency injection, data access, messaging, and web apps.
* **Spring Cloud**: For distributed systems and used for building or deploying your microservices.
* **Spring Data**: For microservices that are related to data access, be it map-reduce, relational or non-relational.
* **Spring Batch**: For high levels of batch operations.
* **Spring Security**: For authorization and authentication support.
* **Spring REST Docs**: For documenting RESTful services.
* **Spring Social**: For connecting to social media APIs.
* **Spring Mobile**: For mobile Web apps.

Sample code:

import org.springframework.boot.\*;

import org.springframework.boot.autoconfigure.\*;

import org.springframework.stereotype.\*;

import org.springframework.web.bind.annotation.\*;

@RestController

@EnableAutoConfiguration

public class Example {

@RequestMapping("/")

String home() {

return "Hello World!";

}

public static void main(String[] args) throws Exception {

SpringApplication.run(Example.class, args);

}

}

**Jersey**

Jersey RESTful framework is open source, and it is based on JAX-RS specification. Jersey’s applications can extend existing JAX-RS implementations and add features and utilities that would make [RESTful services](https://stackify.com/soap-vs-rest/" \t "_blank) simpler, as well as making client development easier.

The best thing about Jersey is that it has great documentation that is filled with examples. It is also fast and has extremely easy routing.

The documentation on how to get started with Jersey is at [this link](https://jersey.java.net/documentation/latest/getting-started.html), while more documentation can be found [here](https://jersey.java.net/apidocs/latest/jersey/index.html).

A sample code that you can try:

package org.glassfish.jersey.examples.helloworld;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

@Path("helloworld")

public class HelloWorldResource {

public static final String CLICHED\_MESSAGE = "Hello World!";

@GET

@Produces("text/plain")

public String getHello() {

return CLICHED\_MESSAGE;

}

}

Jersey is very easy to use with other libraries, such as Netty or Grizzly, and it supports asynchronous connections. It does not need servlet containers. It does, however, have an unpolished dependency injection implementation.

**Play Framework**

Play Framework gives you an easier way to build, create and deploy Web applications using Scala and Java. Play Framework is ideal for RESTful application that requires you to handle remote calls in parallel. It is also very modular and supports async. Play Framework also has one of the biggest communities out of all microservices frameworks.

Sample code you can try:

package controllers;

import play.mvc.\*;

public class Application extends Controller {

public static void index() {

render();

}

public static void sayHello(String myName) {

render(myName);

}

}

**Restlet**

Restlet helps developers create fast and scalable Web APIs that adheres to the RESTful architecture pattern. It has good routing and filtering, and available for Java SE/EE, OSGi, Google AppEngine (part of [Google Compute](https://stackify.com/microsoft-azure-vs-amazon-web-services-vs-google-compute-comparison/)), Android, and other major platforms.

Restlet comes with a steep learning curve that is made worse by a closed community, but you can probably get help from people at StackOverflow.

Sample code:

package firstSteps;

import org.restlet.resource.Get;

import org.restlet.resource.ServerResource;

/\*\*

\* Resource which has only one representation.

\*/

public class HelloWorldResource extends ServerResource {

@Get

public String represent() {

return "hello, world";

}

}