

With AWD RDS Backend

SPRING BOOT MICROSERVICES ON AWS
ELASTIC KUBERNETES SERVICES (EKS)

BINIT DATTA

SPRING BOOT MICROSERVICES ON AWS ELASTIC KUBERNETES SERVICES (EKS)

BINIT DATTA

Copyright © 2021 by Binit Datta

PREFACE ABOUT THE BOOK

The book will define what Containers are, how Kubernetes Container Orchestration achieves automation of running containerized applications on a massive scale. It will then show how to build Spring Boot Microservices and how to deploy that on an AWS Elastic Kubernetes Container Service (EKS). It will teach concepts behind Kubernetes Cluster, Pods, Services, and Networking. The Microservices are production-grade and not the type of Hello World APIs. Advanced Spring Boot Topics such as Aspect-Oriented Programming, Event Publishing, and Trapping will also be shown. Along the way, we will deal with AWS Elastic Container Registry, AWS eksctl command-line utility to provision an AWS EKS cluster, and AWS RDS MySQL.

SOURCE CODE

The source code for the book is available in the following two repositories

<https://github.com/binitauthor/rollingstone-ecommerce-product-catalog-k8s-api>

<https://github.com/binitauthor/rollingstone-ecommerce-category-k8s-api>

If and when you find any issues that you want to report, go to the following and create an issue and I will respond asap.

<https://github.com/binitauthor/rollingstone-ecommerce-category-k8s-api/issues>

<https://github.com/binitauthor/rollingstone-ecommerce-product-catalog-k8s-api/issues>

NOTE

- Please always treat the source code from GitHub as the source of truth, now the code from the book pages
- Please feel free to ask any Microservice Design/Migration/Monitoring and AWS/Azure Cloud related questions openly in the GitHub Issues and I will love to engage with you even if the questions are beyond the topics covered in this book. This is a huge topic and covering everything significant is not feasible in a 300 page book but we should still continue to discuss.

TABLE OF CONTENTS

[About the Author](#)

[Acknowledgements](#)

[Introduction](#)

[Caution for AWS Cost and Security](#)

[Chapter 1](#)

[Technology Concepts](#)

[1 Introduction](#)

[1.1 Containers](#)

[1.2 Container Orchestration](#)

[1.3 Microservices](#)

[1.4 Spring Boot](#)

[1.5 Spring Cloud Microservice Ecosystem](#)

[1.6 Backend Databases](#)

[1.7 AWS Cloud](#)

[Chapter 2](#)

[Tools Setup](#)

[2 Introduction](#)

[2.1 Installing JDK](#)

[2.2 Installing JDK 15 on the Mac OS](#)

[2.3 Installing Maven](#)

[2.4 Installing Gradle](#)

[2.5 Installing MySQL](#)

[2.6 Installing MySQL Database 8.x on Mac OS](#)

[2.7 Installing MySQL Workbench on Mac OS](#)

[2.8 Installing IntelliJ IDE](#)

[2.9 Installing Eclipse](#)

[2.10 Installing Git SCM](#)

[2.11 Installing Docker](#)

[Chapter 3](#)

[Kubernetes Architecture](#)

[3 Introduction: Demystifying Cloud Computing](#)

[3.1 Virtualization](#)

[3.2 Network Security](#)

[3.3 Cloud CPU, Memory, Disks, Network Bandwidth](#)

[3.4 A word on Distributed Computing](#)

[3.5 Physical Structure of Cloud Data Centers](#)

[3.6 What are Containers](#)

[3.7 Advantages of Containers](#)

[3.8 What is a Container Image?](#)

[3.9 What is Orchestration](#)

[3.10 Why Kubernetes](#)

[3.11 Kubernetes Architecture](#)

[3.12 Kubernetes Control Pane \(Master\) Components](#)

[ETCD](#)

[API Server](#)

[Scheduler](#)

[Controller Manager](#)

[Cloud Controller Manager](#)

[3.13 Kubernetes Worker Node Components](#)

[Kubelet](#)

[Kube-proxy](#)

[Container Runtime](#)

Chapter 4

Building the Category REST API

4 Introduction

4.1 Creating a New Project in IntelliJ

4.2 Adding the package Structure

4.3 Building the Model Classes

4.4 Building the Dao JPA Interface

4.5 Building Exception Classes

4.6 Building the Event Class

4.7 Building Aspects

4.8 Building Listeners

4.9 Building the Service

4.10 Building the AbstractController

4.11 Building the CategoryController

4.12 Generate the Configuration

4.13 Building the Spring Boot Main Class

4.14 Setting the Spring Config Files

4.16 Building the Jar

4.16.1 Running the Jar

4.17 Testing the Application Locally

4.18 Adding Swagger API and UI Documentation

4.18.1 Why Swagger

4.18.2 Adding Dependencies

4.18.3 Swagger Configuration

4.18.4 Swagger API Configuration

4.19 Testing Actuator

4.19.1 Health Endpoint

4.19.2 New Custom Actuator Endpoint

Chapter 5

Creating AWS Services

- [**5.1 Signing Up with AWS**](#)
- [**5.2 Creating a Bastion Host in AWS EC2**](#)
- [**5.3 Install Tools in the Bastion Host**](#)
- [**5.4 Creating AWS RDS Database**](#)
- [**5.5 Accessing AWS RDS Database Instance from Local**](#)
- [**5.6 Creating an AWS ECR**](#)
- [**5.7 Creating an AWS EKS Cluster**](#)
- [**5.8 Testing a sample deployment in AWS EKS**](#)

Chapter 6

Building Product REST API to AWS EKS

- [**6.1 Creating a New Project in IntelliJ**](#)
- [**6.2 Adding Dependencies Manually**](#)
- [**6.3 Adding the package Structure**](#)
- [**6.4 Building the Model Classes**](#)
- [**6.5 Building the Dao JPA Interface**](#)
- [**6.6 Building Exception Classes**](#)
- [**6.7 Building the Event Classes**](#)
- [**6.8 Building Aspects**](#)
- [**6.9 Building Listeners**](#)
- [**6.10 Building Swagger Configuration**](#)
- [**6.11 Actuator Metrics Configuration**](#)
- [**6.12 Building RestTemplate Configuration**](#)
- [**6.13 Configuring the RestTemplate**](#)
- [**6.14 Building the Service**](#)
- [**6.15 Building the AbstractController**](#)
- [**6.16 Building the ProductController**](#)

- [**6.17 The Dockerfile**](#)
- [**6.18 Kubernetes Deployment**](#)
- [**6.19 Building the Spring Boot Main Class**](#)
- [**6.20 Setting the Spring Config Files**](#)
- [**6.21 application.yaml for Local**](#)
- [**6.22 The AWS profile**](#)
- [**6.23 bootstrap.yaml**](#)
- [**6.24 Building the Jar**](#)
- [**6.25 Running the Jar**](#)

[**Chapter 7**](#)

[**Deploying Product REST API to AWS EKS**](#)

- [**7 Introduction**](#)
- [**7.1 AWS Database**](#)
- [**7.2 Additional Caution for AWS Security**](#)
- [**7.3 Start Bastion Host**](#)
- [**7.4 SSH into the Bastion Host**](#)
- [**7.5 Clone the Git Repositories**](#)
- [**7.6 Update Category Repository Kubernetes Deployment**](#)
- [**7.7 Update AWS Profile Property File**](#)
- [**7.8 Build the Product application**](#)
- [**7.9 Navigate to the Product App**](#)
- [**7.10 Build the Product Docker Image**](#)
- [**7.11 Get the ECR Repo Name**](#)
- [**7.12 Tag the Image**](#)
- [**7.13 Login to AWS ECR**](#)
- [**7.14 Push Image to ECR**](#)
- [**7.15 Deploy Category Microservice to EKS**](#)
- [**7.16 Expose the Category Microservice**](#)

- [7.17 View the Pods](#)
- [7.18 View the Kubernetes Log](#)
- [7.19 Check the External IP](#)
- [7.20 Get Product External IP](#)
- [7.21 Test Get All Products](#)
- [7.22 Test POST](#)
- [7.23 Test One Product](#)
- [7.24 Test PUT](#)
- [7.25 Verify PUT](#)
- [7.26 Test Delete](#)
- [7.27 Verify DB](#)
- [7.28 Test Product Actuator](#)
- [7.29 Test Actuator Health](#)
- [7.30 Test the Product Service Actuator Metrics](#)
- [7.31 Test One Custom Metric](#)
- [7.32 Terminating AWS Services](#)
- [7.33 Where to go from here](#)

ABOUT THE AUTHOR

Binit Datta has over twenty-five years of in-depth experience in business computing. He is an Enterprise Architect at home with both business and technology professionals using the latest and most remarkable cutting-edge technologies. He draws heavily growing up in the 90s, where lack of job divisions helped him understand the depth of business requirements and then design, build, and implement systems all by himself and his colleagues. His decades old experience directly interacting with end customers and stakeholders of all stripes, eliminates the disadvantage of only knowing and focusing on technology alone without knowing the relevance of their application.

Binit has spent the last ten years architecting and leading technology teams building modern high traffic eCommerce websites and scalable enterprise APIs / applications for multiple fortune 50 companies in AWS and Azure Cloud environments. Continuing from his multiple comfort zones, he spearheaded in User Interface Feasibility, Usability and Architecture, and Microservices based REST API Architecture (CRUD and CQRS), Security-related discussions, Event-Driven Streaming Architectures, among others. While his AWS Cloud Certifications (AWS Solution Architect Professional) prove his Cloud credibility's, he has led multiple real-life Cloud Migration Programs to enrich his Cloud experience.

ACKNOWLEDGEMENTS

I want to acknowledge the invaluable contribution hundreds of non-technical business users have made in shaping my thought process through the last 25 years. They have helped me believe that with all the great importance all kinds of technologies get today, they are still a means towards the common business goal at the end of the day. That mindset has helped me understand the significant similarity between competing platforms like Java and .NET, Spring Framework and Angular, Maven, Gradle, NuGet, NPM and Gulp, and the like.

I would also like to acknowledge the great effort made by Arupa in reviewing this entire book, trying the technical example independently to make sure they are accurate. Her support, silent inspiration, sacrificing weekends cannot be measured in measurable terms.

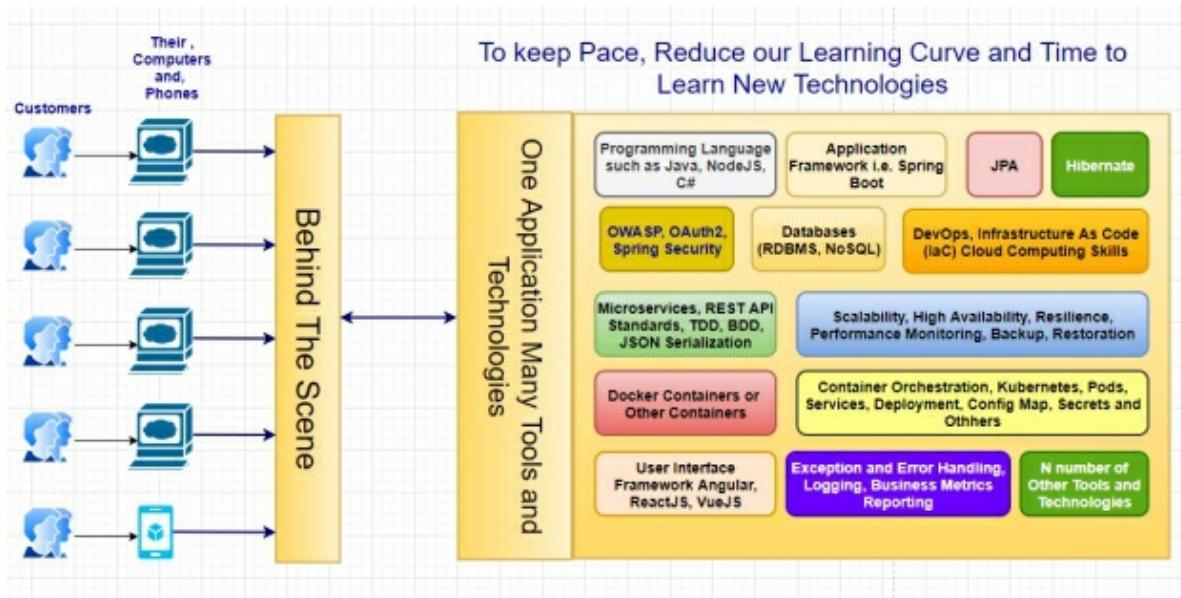
Lastly, I would like to acknowledge the rich contribution made by Mr. Biswajit Das, one of my earliest mentors. Mr. Das graduated from one of India's most prominent educational institutions, the Indian Institute of Technology, Kharagpur. Google CEO Sundar Pichai is also an alum of the same institution. Mr. Das is a Gold medalist from IIT, Kharagpur. Despite his stellar educational background, how he focused on the most humble business users/operators to the most important stakeholders alike made a deep impression on me that still motivates me to work for the users and apply the latest and greatest technology for them. The other thing that greatly validated and inspired my own passion for learning new technology relentlessly was seeing Mr. Das do the same as a number of high level positions (that could be non-technical) remained available.

INTRODUCTION

I am writing this book to be one in a series to aid rapid learning. Following are some my most critical objectives

- As millions of people across the world, joins the software engineering workforce every year, show them directly what works in high traffic application in production, rather than Hello World
- Add skills rapidly that non-technical customers are willing to pay for and use the skills as a competitive advantage for the reader

Let us take a casual look at the following diagram



The diagram above described most of the IT application development teams today. Applications are growing increasingly complex even though customers view them through their simple Browser-based (or Smartphone-based) User Interfaces. Tens of hundreds of new tools and technologies are added to the existing toolset to build applications, and we need to keep pace with that. In this scenario, the key is to learn something that works well in Production! Period. We do not have time for Hello World anymore. We may still use

Hello World types, but we certainly would not stop there without showing how to elevate them to Production grade applications.

The idea of this book is to raise the level of our knowledge quickly to build high-grade production applications in Spring Boot Microservices deployed in real Amazon Web Service (AWS) Elastic Kubernetes Service (EKS) accessing an AWS RDS Database. This is the first book in a series as all topics cannot be covered within 300 odd pages. Information Technology in general and Software Engineering is specific is very much like a cooking Recipe. A world-famous Chef knows his/her ingredients deeply, when to use what quantities at what temperature etc. However, the same Chef is not famous for his/her knowledge of the ingredients. He/she is renowned for the resulting delicious taste of the combined usage of the components. Customers who eat his/her creation make him/her famous. Customers do not care about the ingredients but the result. Similarly, our customers care about the results, which is either the website or the mobile app or the batch jobs or the analytics platform. The book's objective is to show us what to, how to, and when to use these ingredients to use a great testing (or functioning) complex Software using Spring Boot and AWS.

CAUTION FOR AWS COST AND SECURITY

This book deals with several AWS paid services like EC2, EKS, Load Balancers, RDS, and others. One can use this book on a company-controlled AWS environment for the best results. If this book is used on a personal AWS account, take special care to protect your AWS Access and Secret Keys and not reveal them accidentally in public GitHub repositories. Also, do excellent planning, understand accurately, and try the AWS dependent applications in one block of time before deleting the environment. One can take $\frac{1}{2}$ days to keep the AWS environment alive, and the cost would not be too much. However, always make sure to terminate and delete the AWS services to control the cost.

CHAPTER 1

Technology Concepts

1 INTRODUCTION

This book is not going to be an exercise in theory. It will try to deliver a potent combination of skills with high demand in the job market. As a currently practicing Solution Architect in the Cloud Microservices, I have selected various technologies to add tremendous value to the reader. Together in this journey, we will cover a series of those skills, including Spring Boot 2 and Spring Cloud Ecosystem Microservices (Service Discovery, Remote Config, Remote Client Feign, Hystrix Fault Tolerance). We will explain why Containers are great for lightweight packaging, deployment, and scalability and why we would be better off with an advanced Container Orchestration system like Kubernetes. Finally, we will show how to deploy the services in AWS Elastic Container Services or EKS. Along the way, we will also show how to create an EKS cluster. Our Microservices will represent a small subset of the eCommerce domain to demonstrate the use cases. They will persist data in an AWS RDS MySQL and AWS Document DB / MongoDB. Let us get started.

1.1 CONTAINERS

One common theme (speaking from experience) about various concepts and technologies applied as brand-new IT ideas is that these very things are massively and successfully used in life outside of IT first for decades.

Containers are not an exception. If we consider the domain of modern shipping, we will see sizeable voluminous merchant ships getting loaded and unloaded in shipping docks within 30 minutes by large cranes. All shipping containers are of standard size specifications, cranes that load and unload them, and ships can optimize the process without bothering about the containers' content. Without containers, today's efficient Sea ports would come to a crawl. Thus, using standard shipping containers about two things speed of loading and unloading and separation of concerns that free the container handlers, i.e., cranes and ships operate with ease.

The same concept is now applied to deploying software applications. If you are aware of virtual machines and how multiple virtual machines can be created on a single bare metal host, containers are much lighter versions of virtual machines with fast startup times. If we are running one instance of a specific application and under heavy traffic, we want 20 cases to run quickly; scaling that application using a Virtual machine would be slow as the VM has the full-blown OS, i.e., more than 2 GB in size. Ideally, in contrast, Containers could be 200-350 MB or lesser in size, achieving faster download, read/write IO, etc.

Containers also package everything that is needed to run the application. Imagine you purchased a large piece of furniture online, and the company is shipping the furniture in a container. The supplier would package all the components along with a user guide, and tools to assemble the furniture in one single package. Software applications packaged in container packages does exactly the same. They include the base Operating System, the language run time i.e., JRE, and all third-party libraries on that container. The container runtime would assume the container a self-sufficient application package and start/stop/run the container in very standard command lines

irrespective of the containerized application built using Java, .NET or NodeJS, or something else.

We will explore Docker as the most popular software container technology out there today and Kubernetes as the Container Orchestration System.

1.2 CONTAINER ORCHESTRATION

The word Orchestration comes I guess from the music industry. We can imagine the Orchestra Master guiding the room full of players playing different musical instruments to produce melodious music. Thus, Orchestration is about combining the various components of a complex system to create something valuable to the listener or, in this case, the System Administrator. Imagine running hundreds of containers from different applications, scaling them, monitoring their health, removing unhealthy containers, restarting some of them, scaling them to higher numbers and descaling them to a lower number in times of low traffic, protecting and securing passwords, and you as the system admin doing all of these manually! Several years back, the engineers at Google and elsewhere understood the massive value of a stable container orchestration system that can take all these tasks from the System Admins as configurations. It then stores these configurations in its database and, from there on, be on its own, freeing the System Admins doing other critically important staff.

That is precisely what Container orchestration does today. Imagine a high-traffic IT shop without Container Orchestration, like imagining a large Seaport with hundreds of Ships to be docked, loaded, and unloaded, without Cranes.

Kubernetes is one globally accepted Container Orchestration System supported by all three major Cloud providers such as AWS, Microsoft, Google, IBM, Oracle, and others.

We will cover Kubernetes architecture in its chapter.

1.3 MICROSERVICES

Let us try learning in the fast lane. The moment I heard Microservices' term about ten years back, I saw two small words: Micro and Services. I knew Services as I was dealing with a lot of REST APIs at that time. So, I understood that Microservices would still be Services, and I read that Microservice would still follow the REST protocol. About 40 percent of my learning curve disappeared when I realized how similar Microservices would be to what I already knew. The first word is then is Micro, which means small or smaller. Thus, Microservices means we would be creating smaller services than we used to. That is the concept. Please make no mistake as there are significant design challenges, I will walk you through the book, but the idea is all about designing smaller REST APIs that we used to.

1.4 SPRING BOOT

Software systems increasingly run the world. One key factor for all software engineers is how quickly we can create this software, i.e., productivity. If you are aware of the Spring MVC Framework, one challenge was to create a war file, manually download Tomcat or Jetty, start them and drop the war file on them, slowing us down. Spring Boot realized that all Java Web APIs/Applications would undoubtedly need a Servlet Container, and it freed us, software engineers, from doing those mundane manual steps. By default, Spring Boot contains a Tomcat servlet container, but if we want, we can change that to Jetty or Undertow, for example. Second Spring Boot has over 40 starter projects with hundreds of libraries for logging, monitoring, doing database programming, and others. If we remember, adding these separate dependencies in our maven or Gradle file manually, not having to do that with Spring Boot, is a massive boost to productivity. Spring Boot has hundreds of other advantages that I will show you in the coming sections. For now, remember doing way more functionality with way less manual effort or, in other words, higher productivity.

1.5 SPRING CLOUD MICROSERVICE ECOSYSTEM

What is an Ecosystem? Planet Earth has an Ecosystem providing Water to drink, Oxygen to breathe, and food to cultivate and eat. All living beings live in that Ecosystem. Microservices would also need an Ecosystem, but the question is, why? Imagine running five instances of your monolithic application behind a Load Balancer, a hardware one, or a Cloud provider one i.e., AWS LB. That monolithic application has over 140-200 APIs. As discussed in the Microservices section, we are gaining/saving a lot of time by not testing 140/200 APIs if changing one of them. But we cannot run a single instance of that Microservice, say the Product Microservice API. We need to run 5/10 or more. Can we imagine needing 140 hardware or software load balancers ahead of our Microservices instances? The management headache and the cost would be prohibitive. That is why in the Microservice Ecosystem, this load balancing is also done by another Microservice.

Another challenge is that as containerized Microservices are created and destroyed either independently or by the orchestrator like Kubernetes, they get new dynamic IPs. How does a software load balancing Microservice know where to reach the ten instances of the Product Microservice. The answer is each of the new Product Microservice instances calls the Load Balancing Microservice when they come up. They provide their IP, port, and lots of other metadata that the Load Balancer Microservice store in its storage/memory when they call. New Microservices also has a recurrently called by the same Load balancer Microservice) health URL every configurable duration to determine the healthy instances from the dead or unhealthy ones. The entire system works to support dynamic service discovery and is called the Service Discovery pattern. We will show Eureka in this book, but there could be many others.

The next pattern in the Microservice Ecosystem is Remote configuration. All applications, monolithic or Microservice based ones, need configuration to operate. The configuration includes the database credentials, among other

things. Keeping this configuration within the application configuration files, i.e., application.yaml|properties are possible, but it will need a redeployment of the Microservices when we change them. Redeployment has procedural delays. For example, senior directors vacationing in Florida may not be found in time to approve the JIRA deployment tickets. A much more flexible system if we can keep configuration securely in an external system like GitHub, and when they change, all the Microservice has to do is restart. It is even possible for some configurations to consume property/configuration changes without restarting, but that is for later.

Microservice Ecosystem has many other patterns such as remote client (Feign), client-side load balancing (Ribbon), Fault Tolerance (Hystrix), and many others. Let us start with the basic ones and show you how to use some others.

Finally, Netflix is one of the first pioneers who operationalized its monolithic services into the Microservice-based architecture. Many of these patterns and their stable implementations came from Netflix because of their Netflix Open Source (OSS) offering. Spring Cloud packaged them and made it much easier to use them. While many of the original Netflix libraries are getting replaced by Spring Cloud, the original ones are still running in production and will do for years to come.

1.6 BACKEND DATABASES

I am writing a book to help the reader developer better read world skills rather than showing him Hello World APIs. Thus, we will show you how to access Relational Databases like MySQL and NoSQL Databases like MongoDB. We will deploy both types of Databases using AWS RDS and AWS Document DB, (Mongo DB compatible). We will also show native queries, SQL mapping as the3se skills is demanded in real life.

1.7 AWS CLOUD

I would assume that you know what Cloud Computing is. However, to have a starting point what Cabs, Uber, or Lyft is to transportation without owning a vehicle, Cloud computing is to using IT hardware, software, and related services without owning them. Cloud providers like AWS have massive data centers worldwide, and we can rent computing resources (hardware/software) from them and even choose where those Cloud resources would be located. This book focuses on several AWS services starting from AWS Virtual Private Cloud (VPC), AWS Identity and Access Management (IAM), AWS RDS, AWS Document Db, AWS Elastic Container Registry, AWS Elastic Container Services for Kubernetes or EKS, and a few others. I will walk you through how to create AWS Accounts and then how to uses these services for Microservice Application development.

CHAPTER 2

Tools Setup

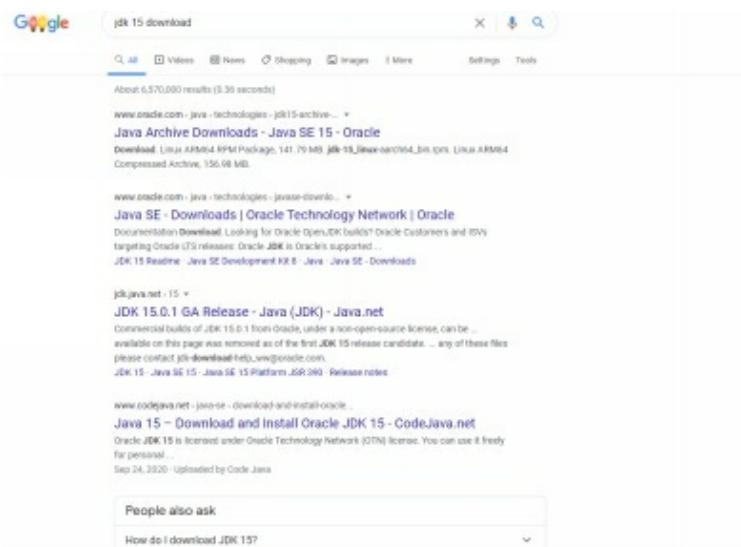
2 INTRODUCTION

If you think you can install all the software listed below, please feel free to skip. However, the following sections show screenshots of installing the software on windows and Mac OS platforms. I will show individual AWS specific software installation when we will need them.

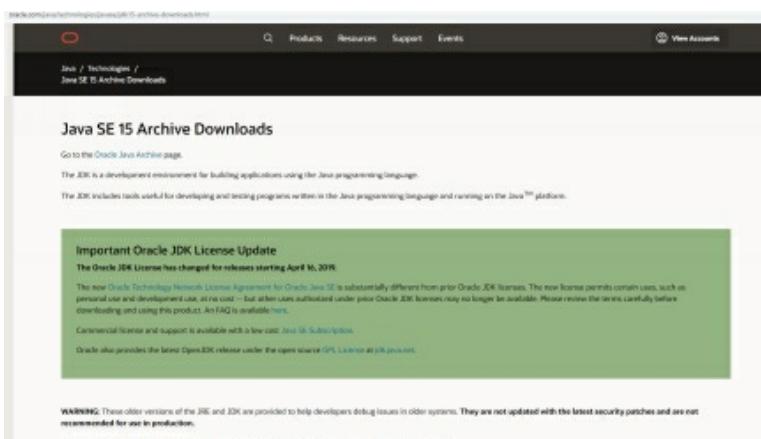
2.1 INSTALLING JDK

Windows Platform

1. Open Google and search for jdk 15 download



2. Click on the first link and scroll down



Copyrights © Oracle Corporation

3. Download the .exe file for the Windows 64-bit version

Java SE Development Kit 15		
This software is licensed under the Oracle Technology Network License Agreement for Oracle Java SE		
Product / File Description	File Size	Download
Linux AArch64 RPM Package	14179 MB	 jdk-15_linux-aarch64_bin.rpm
Linux ARM64 Compressed Archive	156.98 MB	 jdk-15_linux-aarch64_bin.tar.gz
Linux Debian Package	154.77 MB	 jdk-15_linux-x64_bin.deb
Linux RPM Package	161.99 MB	 jdk-15_linux-x64_bin.rpm
Linux Compressed Archive	179.31 MB	 jdk-15_linux-x64_bin.tar.gz
macOS Installer	175.46 MB	 jdk-15_osx-x64_bin.dmg
macOS Compressed Archive	176.07 MB	 jdk-15_osx-x64_bin.tar.gz
Windows x64 Installer	159.68 MB	 jdk-15_windows-x64_bin.exe
Windows x64 Compressed Archive	179.24 MB	 jdk-15_windows-x64_bin.zip

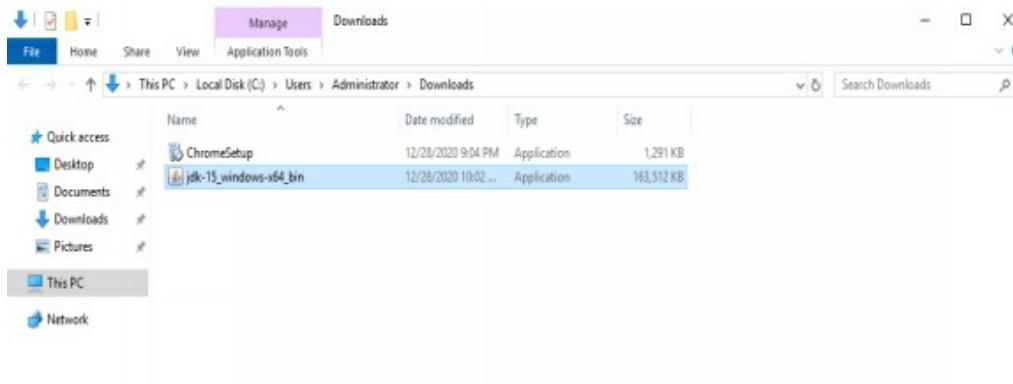
Copyrights © Oracle Corporation

4. Accept Oracle License and create an account if you need to download the file.

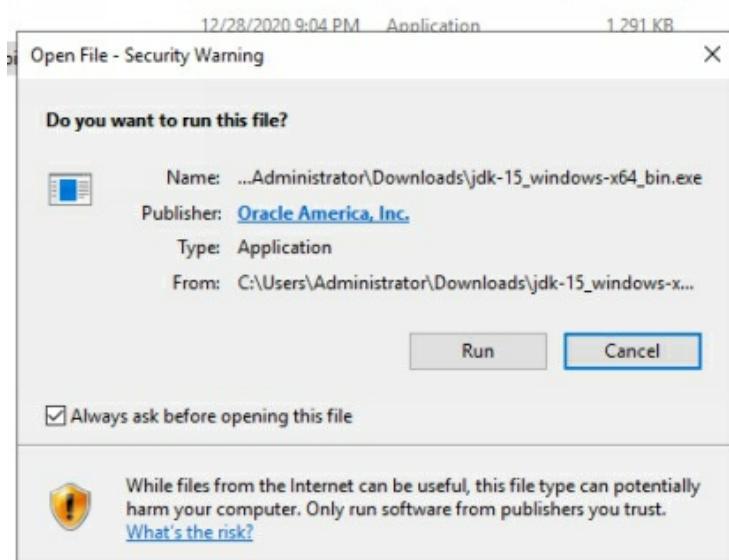


Copyrights © Oracle Corporation

5. Downloaded file shown in the Download Folder



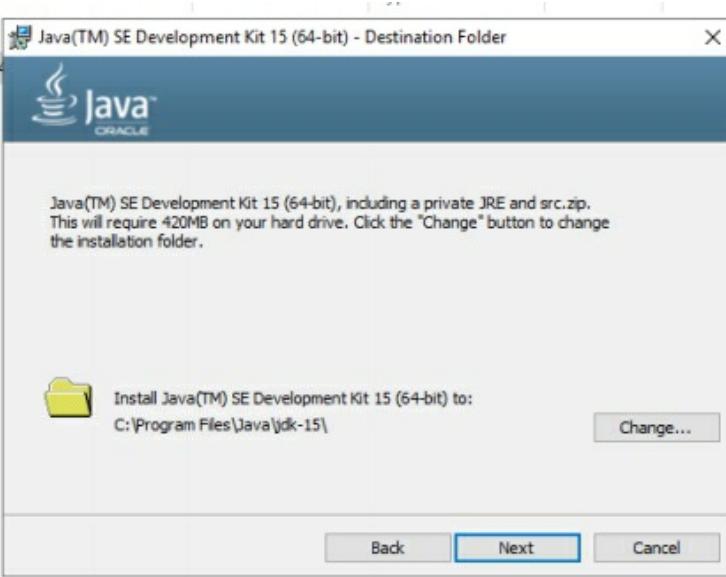
6. Double click on the exe file and click Run



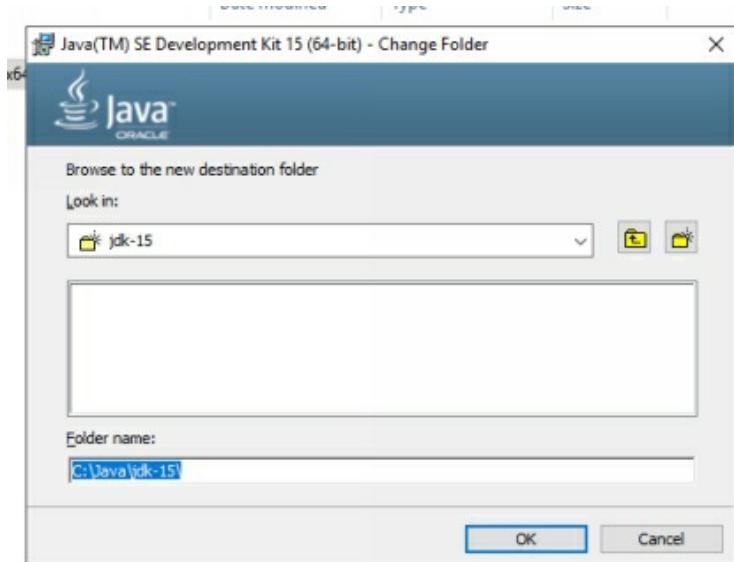
7. Click Next on the following screen



8. Click Change and remove the Program Files part to make it as shown below



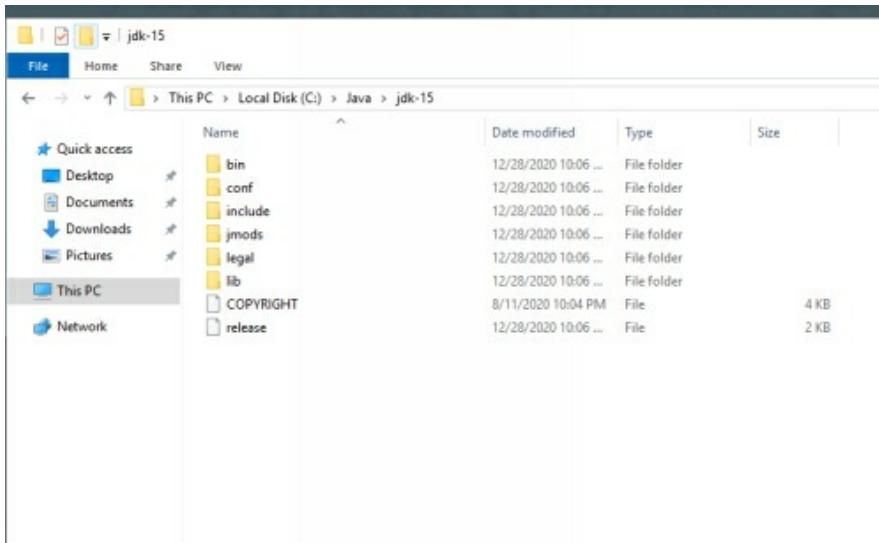
9. Dialog shown after modification. Press OK.



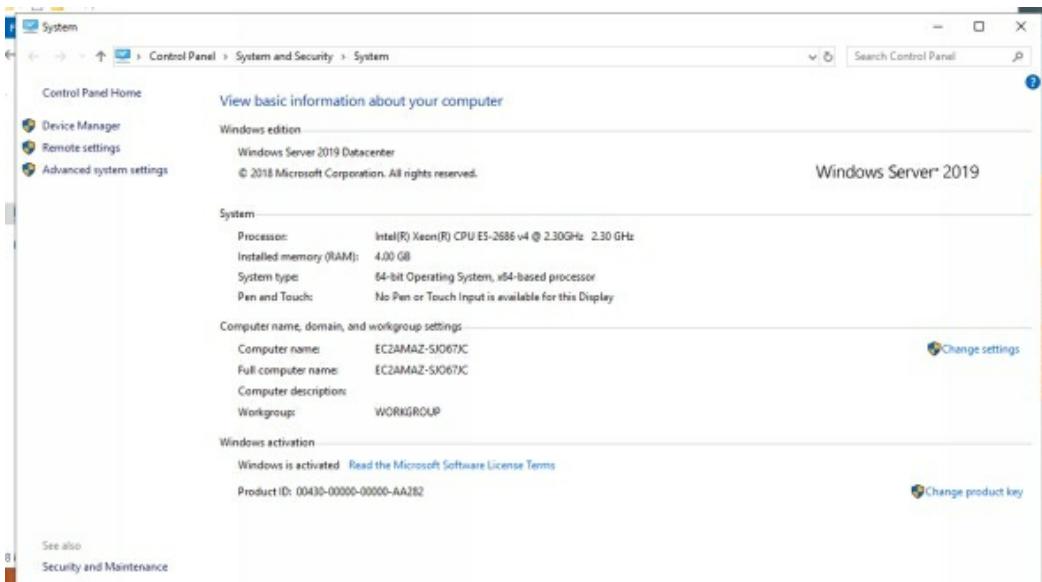
11. Click Next



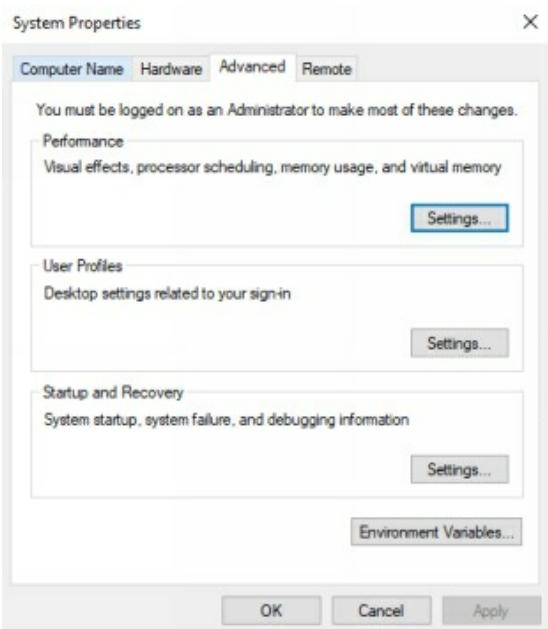
12. Directory of installation shown below



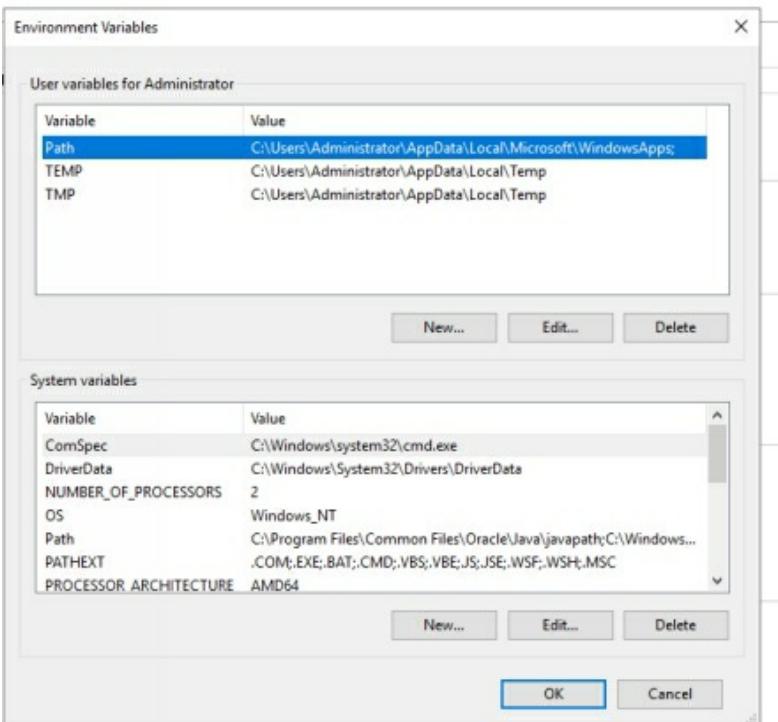
13. Right click on This PC → Advanced system settings



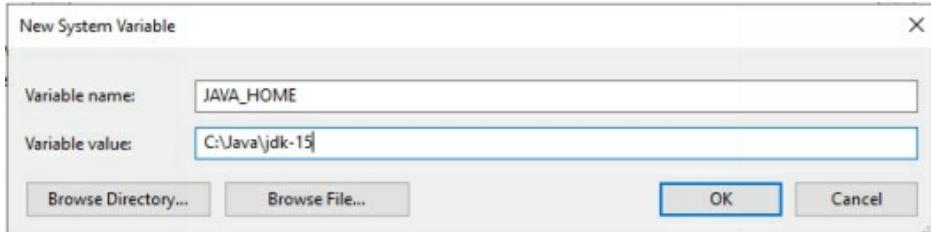
14. Click on Environment Variables



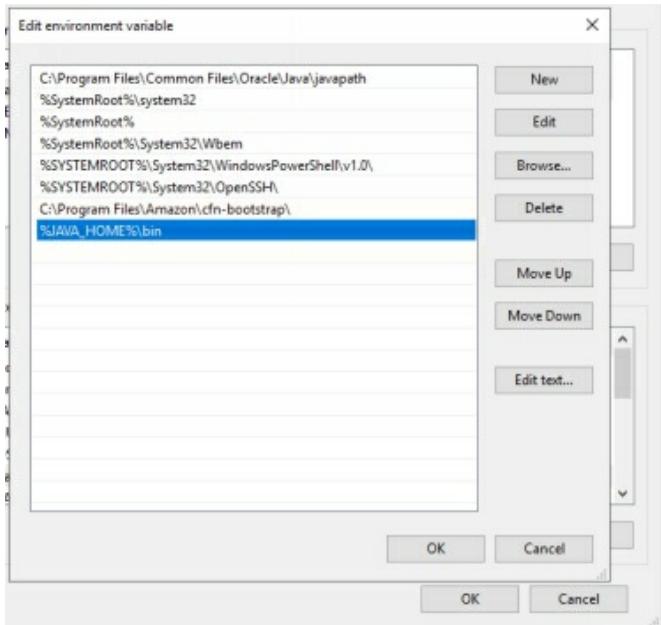
15. Click New on the System Variables Part



16. Enter as shown below. Press OK.



17. Edit the Path Environment variable to include JAVA_HOME.



18. Open a Command Prompt to verify if Java is installed correctly with java -version command

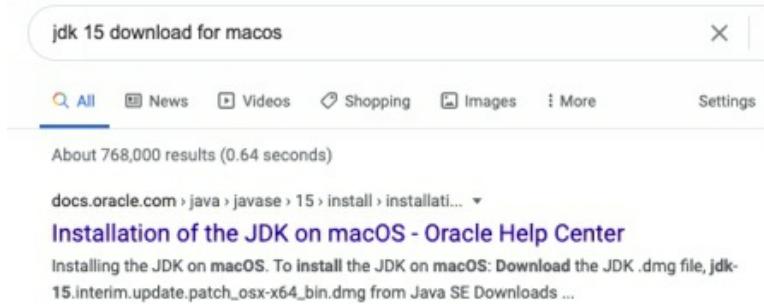
```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.17763.1637]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>java -version
java version "15" 2020-09-15
Java(TM) SE Runtime Environment (build 15+36-1562)
Java HotSpot(TM) 64-Bit Server VM (build 15+36-1562, mixed mode, sharing)

C:\Users\Administrator>
```

2.2 INSTALLING JDK 15 ON THE MAC OS

1. Start with the following google search



2. Open a terminal and verify if you already have a JDK installed on your Mac OS.

A screenshot of a terminal window titled "binitdatta — bash — 80x24". The window shows the command "java -version" being run and its output: "java version "12.0.2" 2019-07-16 Java(TM) SE Runtime Environment (build 12.0.2+10) Java HotSpot(TM) 64-Bit Server VM (build 12.0.2+10, mixed mode, sharing)". The prompt "Binitdatta\$" is visible at the bottom.

3. Visit the first link shown by Google to get the page below and click on the Java SE Downloads link

To run a different version of Java, either specify the full path, or use the `java_home` tool. For example:

```
$ /usr/libexec/java_home -v 15 --exec javac -version
```

Installing the JDK on macOS

To install the JDK on macOS:

1. Download the JDK .dmg file, `jdk-15.interim.update.patch_osx-x64_bin.dmg` from Java SE Downloads page.

Click **Accept License Agreement**.

2. From either the browser **Downloads** window or from the file browser, double-click the .dmg file to start it.

A **Finder** window appears that contains an icon of an open box and the name of the .pkg file.

3. Double-click the **JDK 15.pkg** icon to start the installation application.

The installation application displays the **Introduction** window.

4. Click **Continue**.

The **Installation Type** window appears.

5. Click **Install**.

A window appears that displays the message: **Installer is trying to install new software. Enter your password to allow this.**

6. Enter the Administrator user name and password and click **Install Software**.

The software is installed and a confirmation window is displayed.

After the software is installed, you can delete the .dmg file if you want to save disk space.

Copyrights © Oracle Corporation

4. Click again on JDK Download link on the following screen

The screenshot shows a web browser displaying the Java SE Downloads page. The URL in the address bar is `oracle.com/java/technologies/javase-downloads.html`. The page title is "Java SE Downloads". Below the title, it says "Java Platform, Standard Edition". A section titled "Java SE 15" is shown, stating "Java SE 15.0.1 is the latest release for the Java SE Platform". To the right of this section, there is a "Documentation" link and a "JDK Download" link. The "JDK Download" link is highlighted with a blue border, indicating it is the target for the next step.

Copyrights © Oracle Corporation

5. Scroll down the page below

m/java/technologies/javase-jdk15-downloads.html

Student Login - Ac... History Bookshelf Sign In Search HR Folder Bookmarks Sign In for Purdue... Sign In for Purdue... Harvard Business... Introduction to Qu... Basic EdHF Rem...

Java / Technologies / Java SE 15 - Downloads

Java SE Downloads Java SE Subscriptions

Java SE Development Kit 15 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

Important Oracle JDK License Update

The Oracle JDK License has changed for releases starting April 16, 2019.

The new [Oracle Technology Network License Agreement for Oracle Java SE](#) is substantially different from prior Oracle JDK licenses. The new license permits certain uses, such as personal use and development use, at no cost – but other uses authorized under prior Oracle JDK licenses may no longer be available. Please review the terms carefully before downloading and using this product. An FAQ is available [here](#).

Commercial license and support is available with a low cost [Java SE Subscription](#).

Oracle also provides the latest OpenJDK release under the open source [GPL License](#) at [jdk.java.net](#).

Copyrights © Oracle Corporation

6. Choose the .dmg file to download

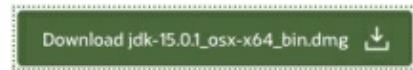
Product / File Description	File Size	Download
Linux ARM 64 RPM Package	141.81 MB	jdk-15.0.1_linux-aarch64_bin.rpm
Linux ARM 64 Compressed Archive	157.01 MB	jdk-15.0.1_linux-aarch64_bin.tar.gz
Linux x64 Debian Package	154.79 MB	jdk-15.0.1_linux-x64_bin.deb
Linux x64 RPM Package	162.02 MB	jdk-15.0.1_linux-x64_bin.rpm
Linux x64 Compressed Archive	179.33 MB	jdk-15.0.1_linux-x64_bin.tar.gz
macOS Installer	175.94 MB	jdk-15.0.1_osx-x64_bin.dmg
macOS Compressed Archive	176.53 MB	jdk-15.0.1_osx-x64_bin.tar.gz
Windows x64 Installer	159.69 MB	jdk-15.0.1_windows-x64_bin.exe
Windows x64 Compressed Archive	179.27 MB	jdk-15.0.1_windows-x64_bin.zip

Copyrights © Oracle Corporation

7. Check the box, login to your Oracle account or register and download the dmg

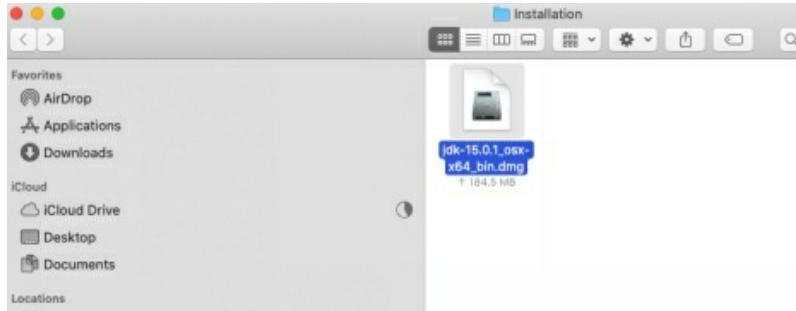
You must accept the Oracle Technology Network License Agreement for Oracle Java SE to download this software.

- I reviewed and accept the Oracle Technology Network License Agreement for Oracle Java SE



Copyrights © Oracle Corporation

8. Locate the dmg downloaded file on your hard drive



9. Double click on the dmg file to get the following pkg installer shown below



10. Follow the Mac OS JDK installer one screen after another as shown below



11. Click Install



12. Enter your Admin Password



13. Installation Succeeded



14. Open your .bash profile and create / modify the JAVA_HOME and PATH environment variables

```

binits-MacBook-Pro:~ binidata$ ls -l
total 0
drwxr-xr-x  3 root wheel  96 Dec 29 13:25 jdk-15.0.1.jdk
drwxr-xr-x  3 root wheel  96 Nov 27 2018 jdk1.8.0_192.jdk
binits-MacBook-Pro:~/Library/Java/JavaVirtualMachines binidata$ cd jdk-15.0.1.jdk/
binits-MacBook-Pro:~/Library/Java/JavaVirtualMachines/jdk-15.0.1.jdk binidata$ ls -lt
total 0
drwxr-xr-x  6 root wheel 192 Sep 15 09:11 Contents
binits-MacBook-Pro:~/Library/Java/JavaVirtualMachines/jdk-15.0.1.jdk binidata$ cd Contents/
binits-MacBook-Pro:~/Library/Java/JavaVirtualMachines/jdk-15.0.1.jdk binidata$ ls -lt
total 8
drwxr-xr-x 10 root wheel 320 Sep 15 09:11 Home
-rw-r--r--  1 root wheel 1713 Sep 15 09:11 Info.plist
drwxr-xr-x  3 root wheel  96 Sep 15 09:11 MacOS
drwxr-xr-x  3 root wheel  96 Sep 15 09:11 _CodeSignature
binits-MacBook-Pro:~/Library/Java/JavaVirtualMachines/jdk-15.0.1.jdk binidata$ cd Home
binits-MacBook-Pro:~/Library/Java/JavaVirtualMachines/jdk-15.0.1.jdk/Contents/Home
binits-MacBook-Pro:~/Library/Java/JavaVirtualMachines/jdk-15.0.1.jdk binidata$ ls -lt
total 8
drwxr-xr-x  7 root wheel 224 Sep 15 09:11 conf
drwxr-xr-x  71 root wheel 2272 Sep 15 09:11 jmod
drwxr-xr-x  73 root wheel 2272 Sep 15 09:11 legal
drwxr-xr-x  9 root wheel  284 Sep 15 09:11 lib
drwxr-xr-x  56 root wheel 1285 Sep 15 09:11 include
drwxr-xr-x  56 root wheel 1792 Sep 15 09:11 lib
drwxr-xr-x  3 root wheel  96 Sep 15 09:11 man
-rw-r--r--  1 root wheel 1285 Sep 15 09:11 release
binits-MacBook-Pro:~/Library/Java/JavaVirtualMachines/jdk-15.0.1.jdk binidata$ ls

```

15. Modified values shown below

```

binits-MacBook-Pro:~ binidata$ vi .bash_profile
export MAVEN_HOME=/Users/binidata/software/apache-maven-3.6.0
export GRADLE_HOME=/Users/binidata/software/gradle-5.0
#export JAVA_HOME=/Users/binidata/Documents/Arupa/jdk-12.0.2.jdk/Contents/Home
export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk-15.0.1.jdk/Contents/Home
#export PATH=/Users/binidata/MongoDB_Home/mongodb-macos-x86_64-4.2.1/bin:$PATH

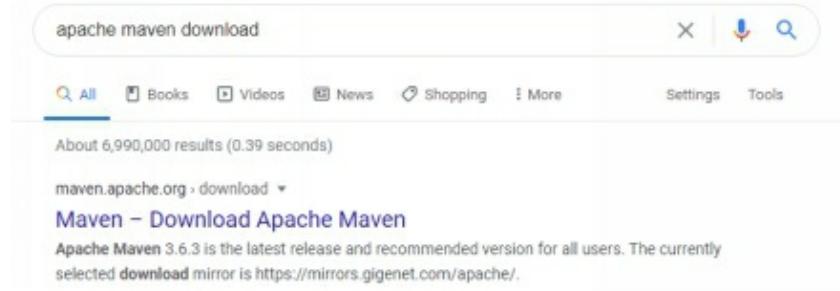
export PATH=$PATH:$MAVEN_HOME/bin:$GRADLE_HOME/bin
export MONGO_HOME=/Users/binidata/MongoDB_Home/mongodb-osx-x86_64-4.0.4
export PATH=$PATH:$MONGO_HOME/bin
export ACTIVEMQ_HOME=/Users/binidata/software/apache-activemq-5.15.0
export PATH=$PATH:$ACTIVEMQ_HOME/bin
export PATH=$PATH:/usr/local/sbin
export PATH=$PATH:$JAVA_HOME/bin
-
-
-
```

16. Verify if JDK 15 installed correctly on your Mac with java -version

```
binitdatta — bash — 80x24
Last login: Tue Dec 29 13:26:41 on ttys009
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
[Binitdatta-MacBook-Pro:~ binitdatta$ java -version
java version "15.0.1" 2020-10-20
Java(TM) SE Runtime Environment (build 15.0.1+9-18)
Java HotSpot(TM) 64-Bit Server VM (build 15.0.1+9-18, mixed mode, sharing)
Binitdatta-MacBook-Pro:~ binitdatta$ ]
```

2.3 INSTALLING MAVEN

1. Search Google for apache maven download and click on the first link



2. Download the zip archive as shown below

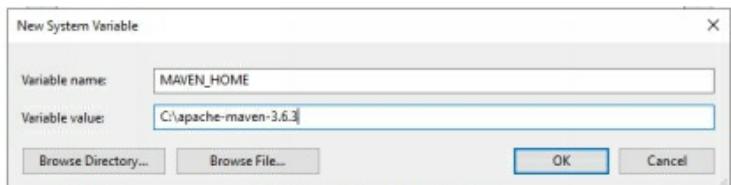
The screenshot shows the Apache Maven 3.6.3 download page. It includes sections for "System Requirements" (Java Development Kit (JDK) 8 or above, no minimum requirement for memory, disk space of approximately 10GB, and no minimum requirement for operating system), and a "Files" section listing download links for binary tar.gz, binary zip, source tar.gz, and source zip archives, along with their corresponding checksums and signatures.

Link	Checksum	Signature
Binary tar.gz archive	apache-maven-3.6.3-bin.tar.gz	apache-maven-3.6.3-bin.tar.gz.asc
Binary zip archive	apache-maven-3.6.3-bin.zip	apache-maven-3.6.3-bin.zip.asc
Source tar.gz archive	apache-maven-3.6.3-source.tar.gz	apache-maven-3.6.3-source.tar.gz.asc
Source zip archive	apache-maven-3.6.3-source.zip	apache-maven-3.6.3-source.zip.asc

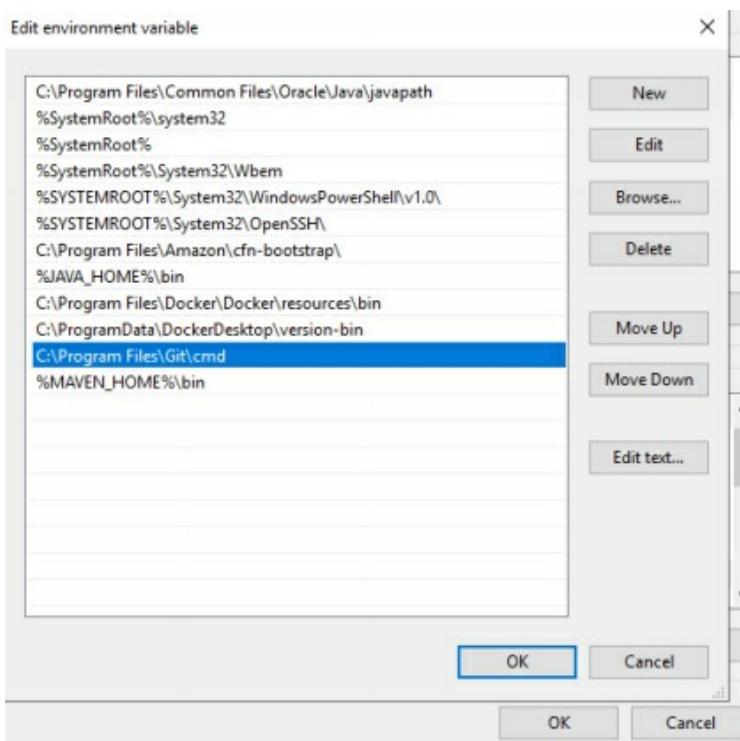
3. Copy the downloaded archive and extract in the C drive

Name	Date modified	Type	Size
bin	12/28/2020 11:06 ...	File folder	
boot	12/28/2020 11:06 ...	File folder	
conf	12/28/2020 11:06 ...	File folder	
lib	12/28/2020 11:06 ...	File folder	
LICENSE	12/28/2020 11:06 ...	File	18 KB
NOTICE	12/28/2020 11:06 ...	File	6 KB
README	12/28/2020 11:06 ...	Text Document	3 KB

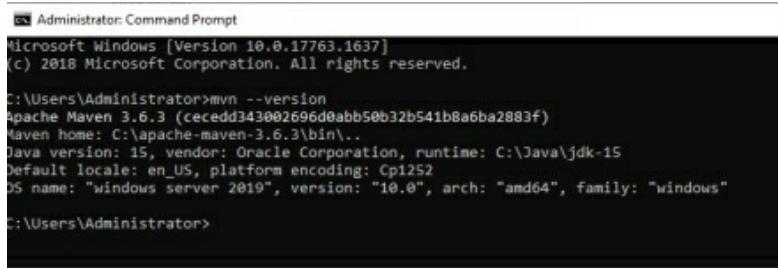
4. Define a new environment variable as show below for MAVEN_HOME



5. Edit the Path variable to include MAVEN_HOME



6. Verify Maven Installation in a command prompt



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.17763.1637]
(c) 2018 Microsoft Corporation. All rights reserved.

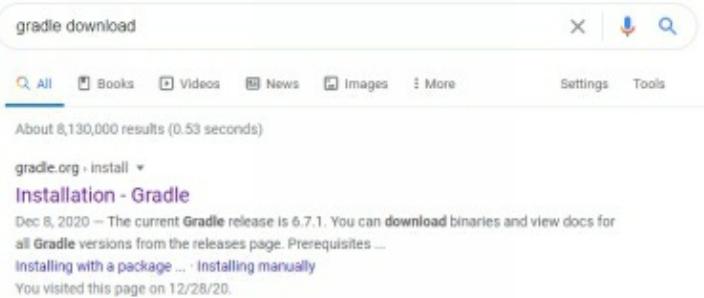
C:\Users\Administrator>mvn --version
Apache Maven 3.6.3 (cecedd343b02696d0abb50b32b541b8a6ba2883f)
Maven home: C:\apache-maven-3.6.3\bin\..
Java version: 15, vendor: Oracle Corporation, runtime: C:\Java\jdk-15
Default locale: en_US, platform encoding: Cp1252
OS name: "windows server 2019", version: "10.0", arch: "amd64", family: "windows"

C:\Users\Administrator>
```

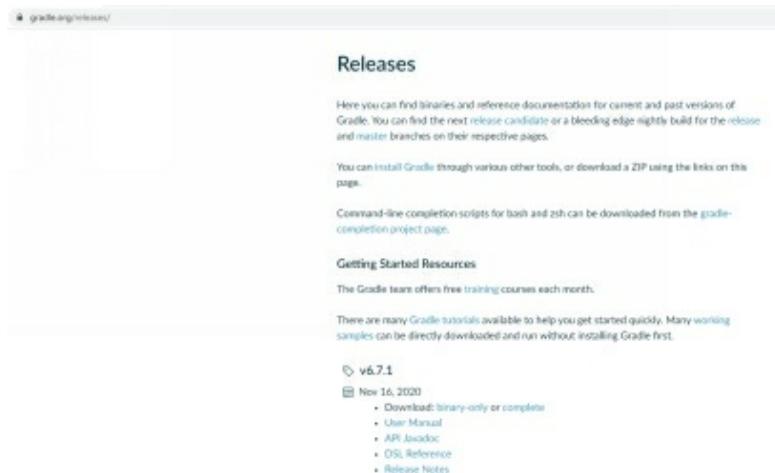
7. The process is very close to install Maven on your Mac except if may download the .gzip file, extract using gunzip and untar using tar -xvf. Once you get the extracted folder, you can edit your .bash profile file to create a MAVEN_HOME and add that to your PATH.

2.4 INSTALLING GRADLE

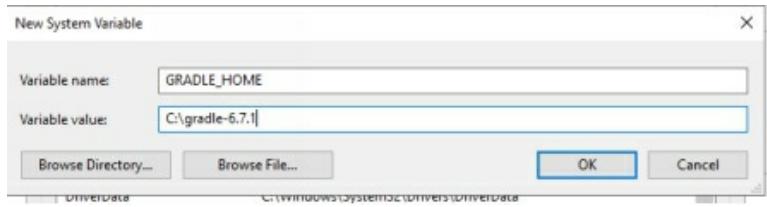
1. Search for Gradle download in google



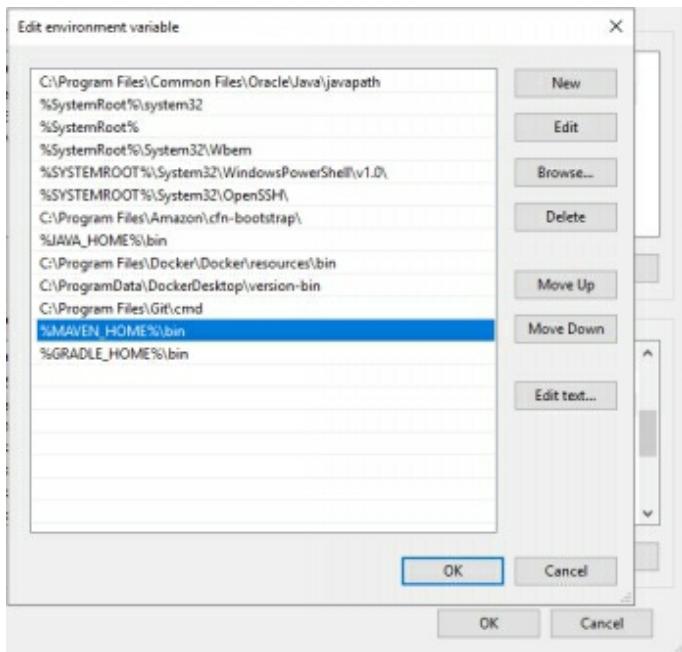
2. Visit the first link , download the binary only archive, copy to C drive and extract there



3. Create a new environment variable called GRADLE_HOME



4. Edit the Path variable to include GRADLE_HOME as shown below



5. Verify Gradle installation in a command prompt

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.17763.1637]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>gradle -v
Welcome to Gradle 6.7.1

Here are the highlights of this release:
- File system watching is ready for production use
- Declare the version of Java your build requires
- Java 15 support

For more details see https://docs.gradle.org/6.7.1/release-notes.html

Gradle 6.7.1

-----
Build time: 2020-11-16 17:09:24 UTC
Revision: 2872ff02f328d2ced2f1ea800ff026acfba5c8

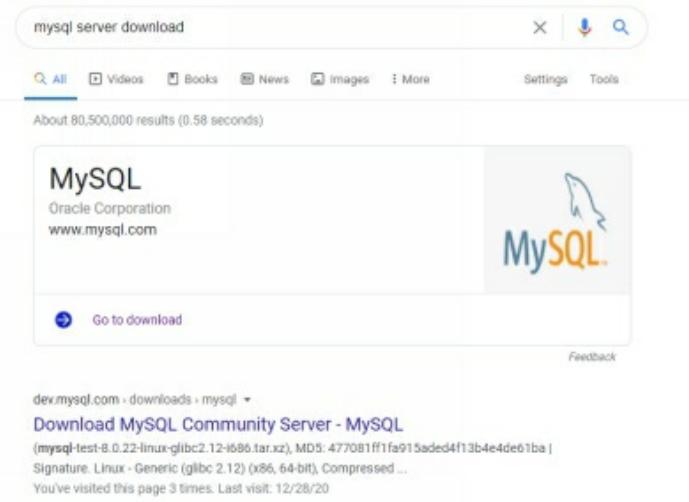
Kotlin: 1.3.72
Groovy: 2.5.12
Ant: Apache Ant(TM) version 1.10.8 compiled on May 19 2020
JVM: 15 (Oracle Corporation 15+36-1562)
OS: Windows Server 2019 10.0 am064

C:\Users\Administrator>
```

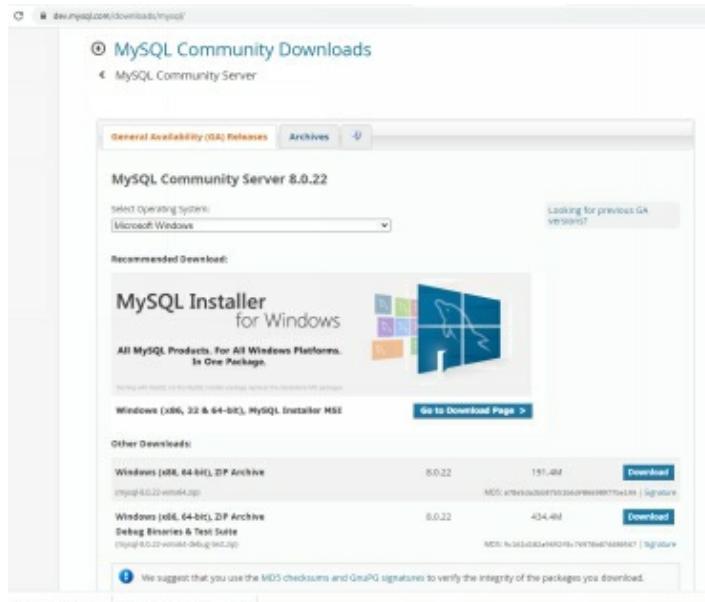
6. The process is very close to install Gradle on your Mac except if you may download the .gzip file, extract using gunzip and untar using tar -xvf. Once you get the extracted folder, you can edit your .bash profile file to create a GRADLE_HOME and add that to your PATH.

2.5 INSTALLING MYSQL

1. Search google for MySQL download and click on the first link



2. Click on All Products for Windows and then the file with size 434 MB.



3. Download the second bigger file for MySQL 8.x

④ MySQL Community Downloads

MySQL Installer

General Availability (GA) Releases Archives

MySQL Installer 8.0.22

Select Operating System: Microsoft Windows

Looking for previous GA versions?

File Type	Version	Size	Action
Windows (x86, 32-bit), MSI Installer	8.0.22	2.5M	Download
mysql-installer-web-community-8.0.22.0.msi			
Windows (x86, 32-bit), MSI Installer	8.0.22	405.2M	Download
mysql-installer-community-8.0.22.0.msi			

We suggest that you use the MD5 checksums and Oracle signatures to verify the integrity of the packages you download.

ORACLE © 2020, Oracle Corporation and/or its affiliates
Legal Policies | Your Privacy Rights | Terms of Use | Trademark Policy | Contributor Agreement | Cookie Preferences

4. Click on No Thanks, just start my download

④ MySQL Community Downloads

Login Now or Sign Up for a free account.

An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system

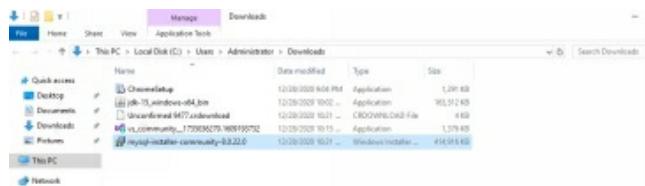
[Login » using my Oracle Web account](#) [Sign Up » for an Oracle Web account](#)

MySQL.com is using Oracle SSO for authentication. If you already have an Oracle Web account, click the Login link. Otherwise, you can signup for a free account by clicking the Sign Up link and following the instructions.

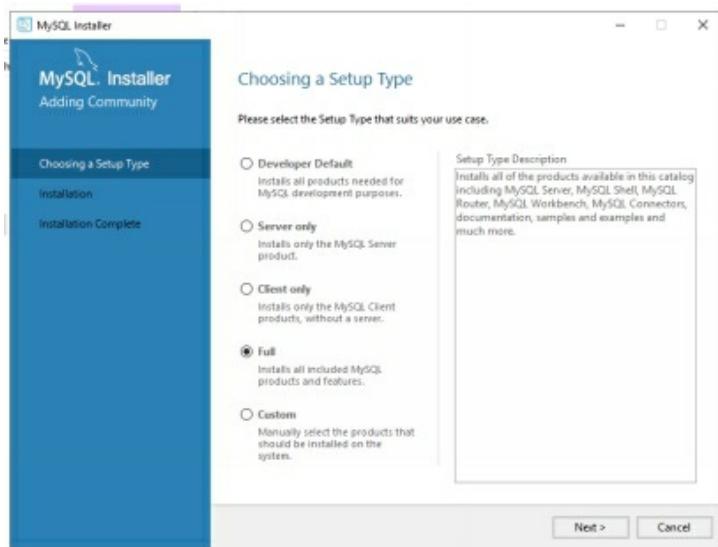
No thanks, just start my download.

ORACLE © 2020, Oracle Corporation and/or its affiliates
Legal Policies | Your Privacy Rights | Terms of Use | Trademark Policy | Contributor Agreement | Cookie Preferences

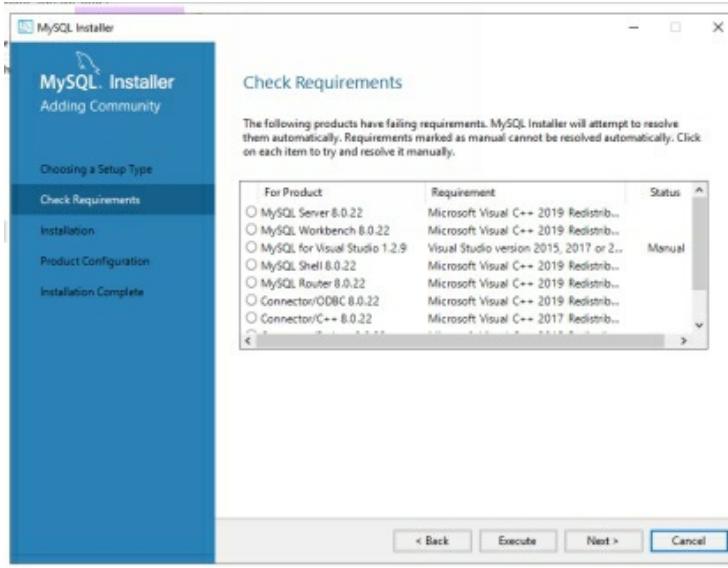
5. Downloaded file shown and double click to start installation



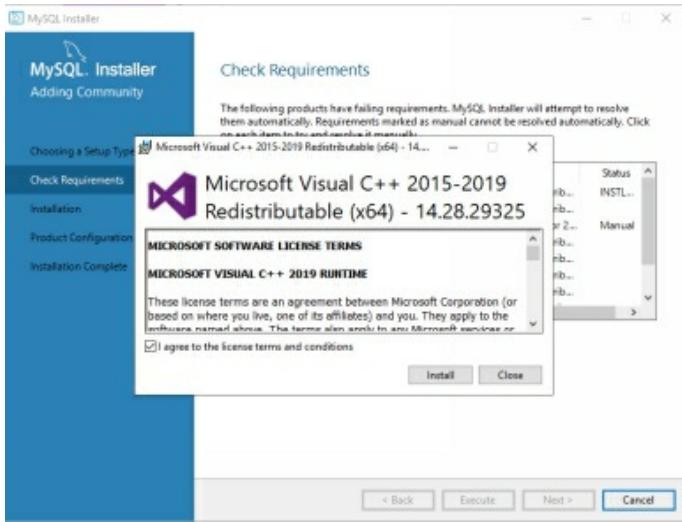
6. Choose full installation



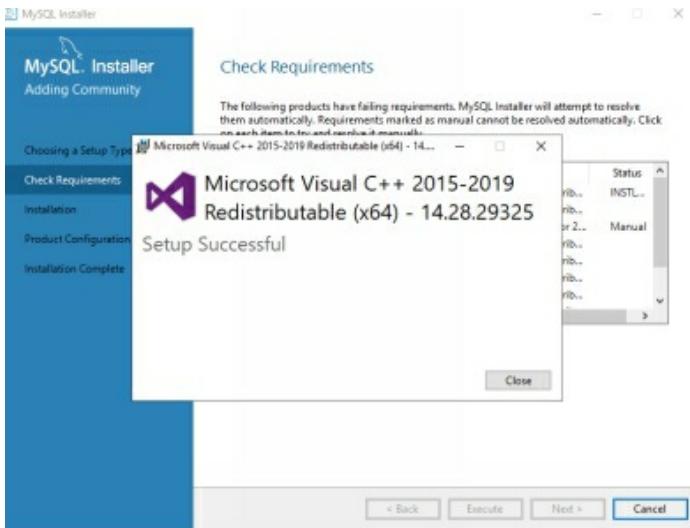
7. Choose Execute to install MySQL Windows Dependencies



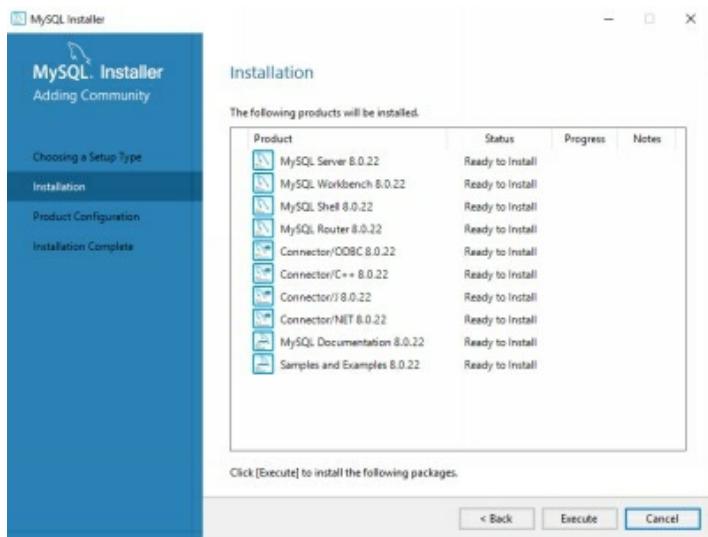
8. Choose Install



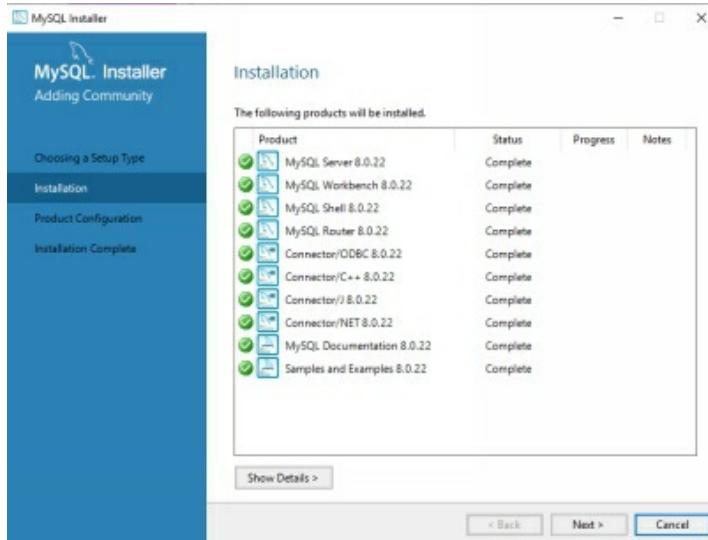
9. Choose close.



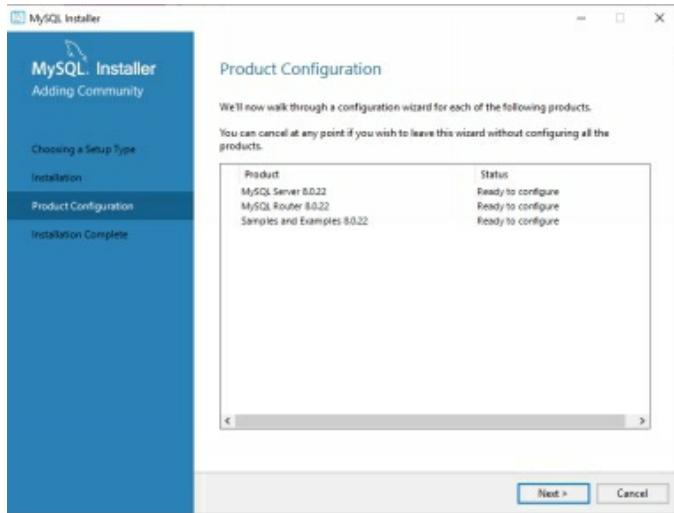
10. Choose Execute and wait



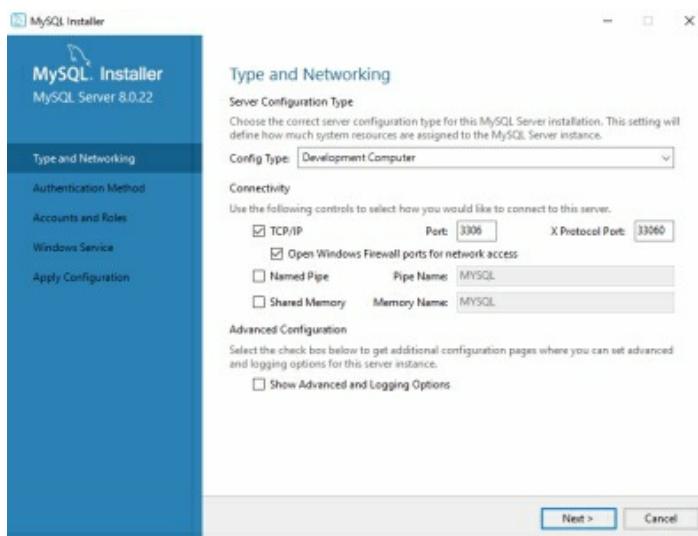
11. Till you see the following screen



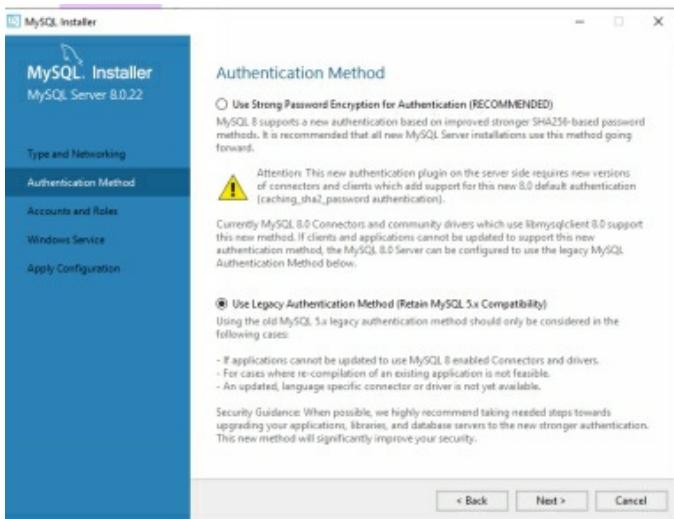
12. Installation complete now, click next to configure MySQL



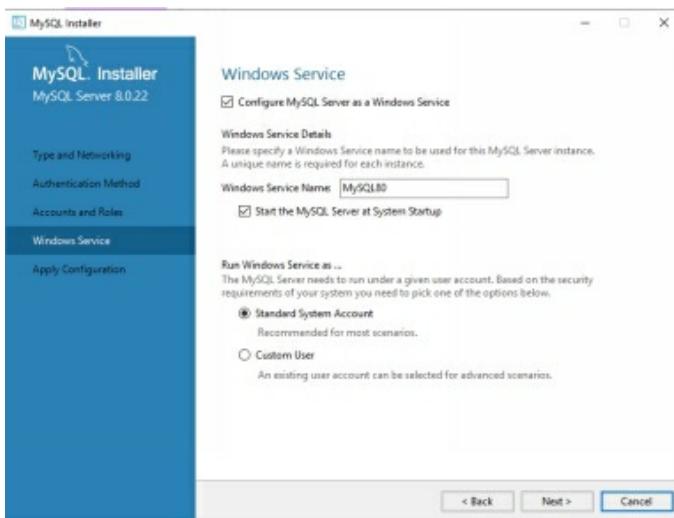
13. Accept defaults and click next



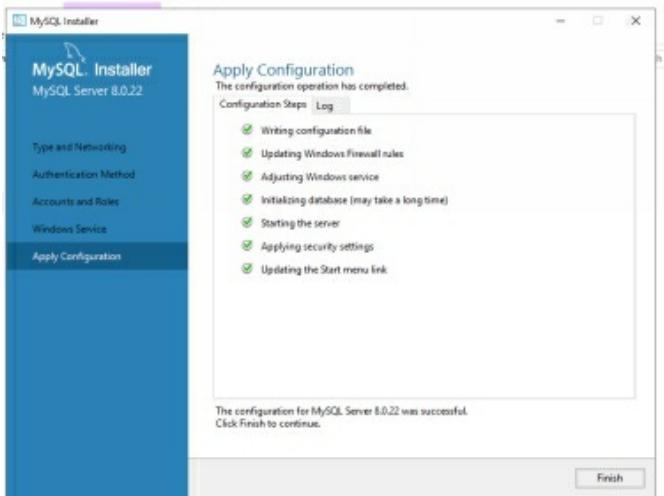
14. As this is a non-production work, choose legacy radio button



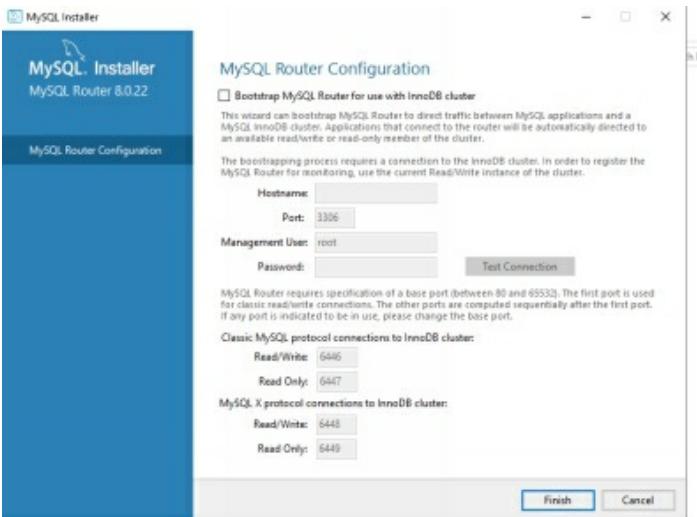
15. Choose default and click next



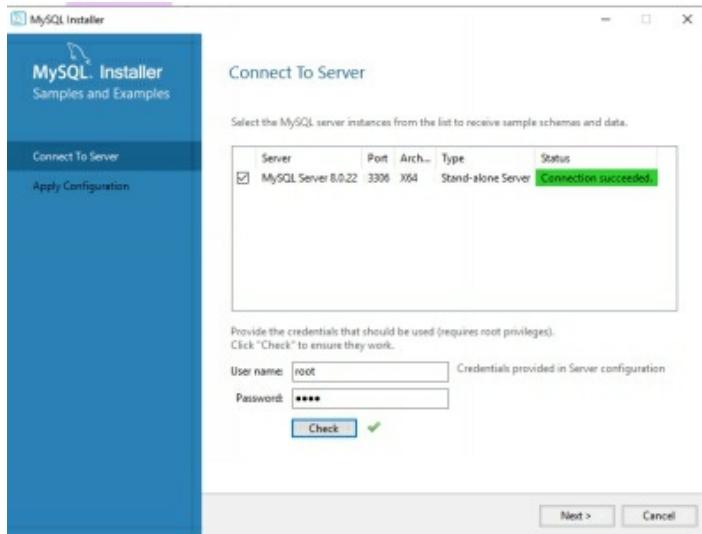
16. Choose Finish



17. Click Finish on this screen

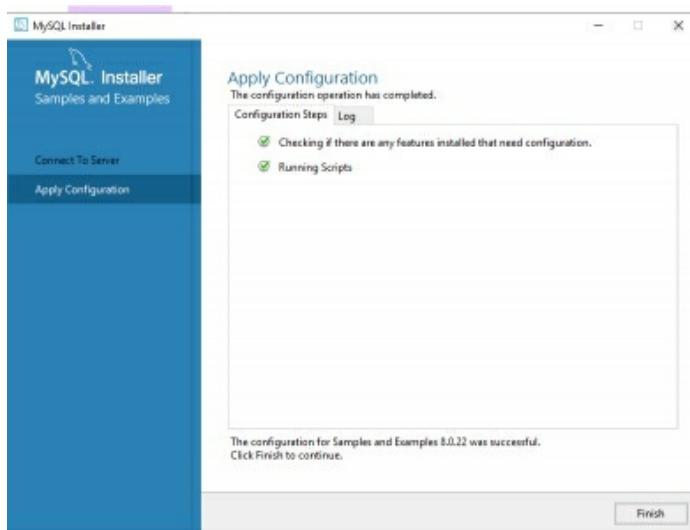


18. Enter the root account password you have chosen and click check.

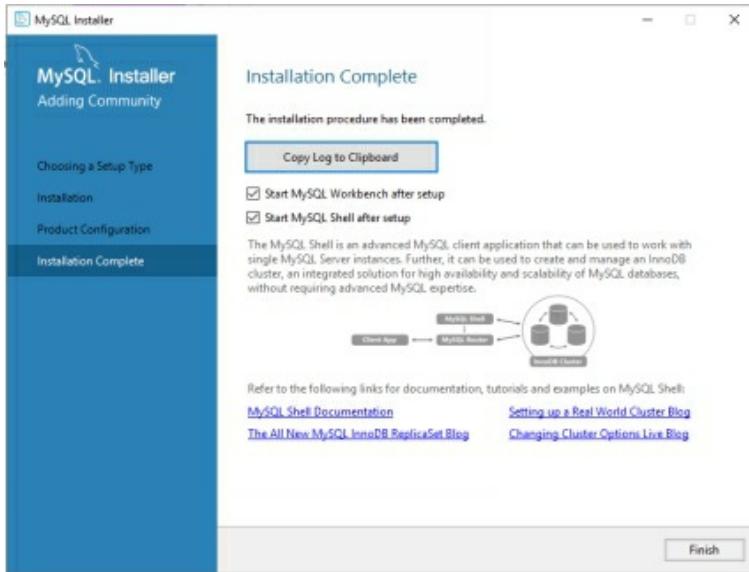


19. Click Next

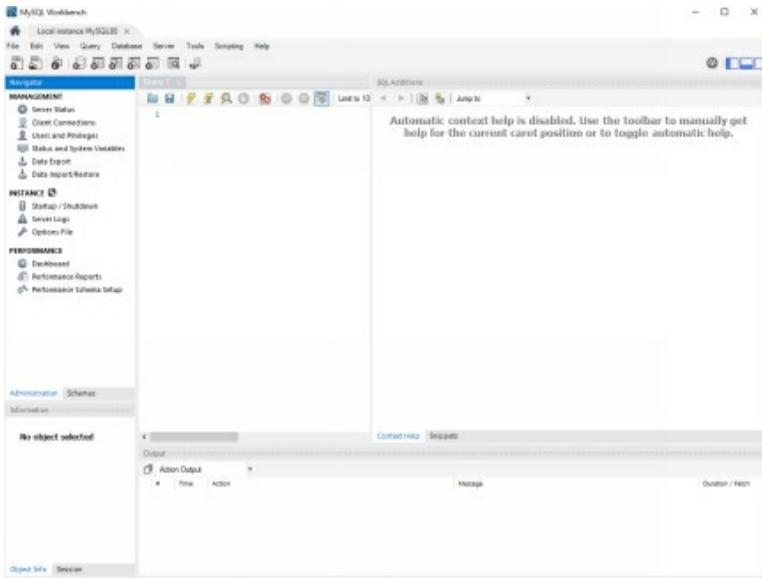
20. Click Finish on the Next



21. Click Finish on the next screen

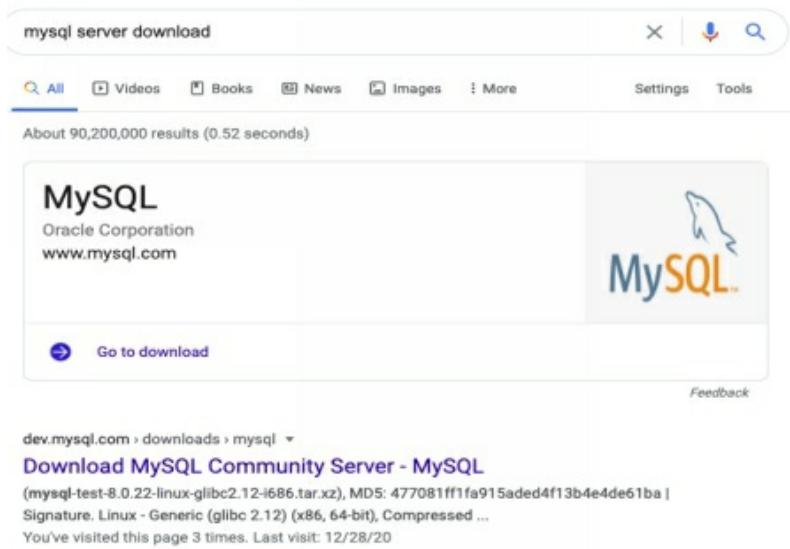


22. Check MySQL Workbench Client and close it



2.6 INSTALLING MYSQL DATABASE 8.X ON MAC OS

1. Start with the following google search



2. Visit the first link and click on the first download

The screenshot shows the MySQL Community Downloads page at dev.mysql.com/downloads/mysql/. The page title is "MySQL Community Downloads". Below it, a breadcrumb navigation shows "MySQL Community Server". A navigation bar at the top includes "General Availability (GA) Releases" (which is selected), "Archives", and a search icon.

The main content area is titled "MySQL Community Server 8.0.22". It features a dropdown menu for "Select Operating System" set to "macOS". To the right is a link "Looking for previous GA versions?".

A note indicates: "Packages for Catalina (10.15) are compatible with Mojave (10.14)".

The download section lists four packages:

- macOS 10.15 (x86, 64-bit), DMG Archive**: Version 8.0.22, size 401.5M. Download button.
- macOS 10.15 (x86, 64-bit), Compressed TAR Archive**: Version 8.0.22, size 160.6M. Download button.
- macOS 10.15 (x86, 64-bit), Compressed TAR Archive Test Suite**: Version 8.0.22, size 244.3M. Download button.
- macOS 10.15 (x86, 64-bit), TAR**: Version 8.0.22, size 420.5M. Download button.

Each package entry includes an MD5 checksum and a GnuPG signature link.

A note at the bottom suggests: "We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download."

3. Click on No thanks....

The screenshot shows the MySQL Community Downloads page again, but this time the "No thanks, just start my download." link has been clicked, bypassing the account creation step.

The page title is "MySQL Community Downloads". Below it, a note says "Login Now or Sign Up for a free account." and "An Oracle Web Account provides you with the following advantages:" followed by a bulleted list of benefits.

At the bottom, there are two buttons: "Login »" (using my Oracle Web account) and "Sign Up »" (for an Oracle Web account).

A note at the bottom of the page states: "MySQL.com is using Oracle SSO for authentication. If you already have an Oracle Web account, click the Login link. Otherwise, you can signup for a free account by clicking the Sign Up link and following the instructions."

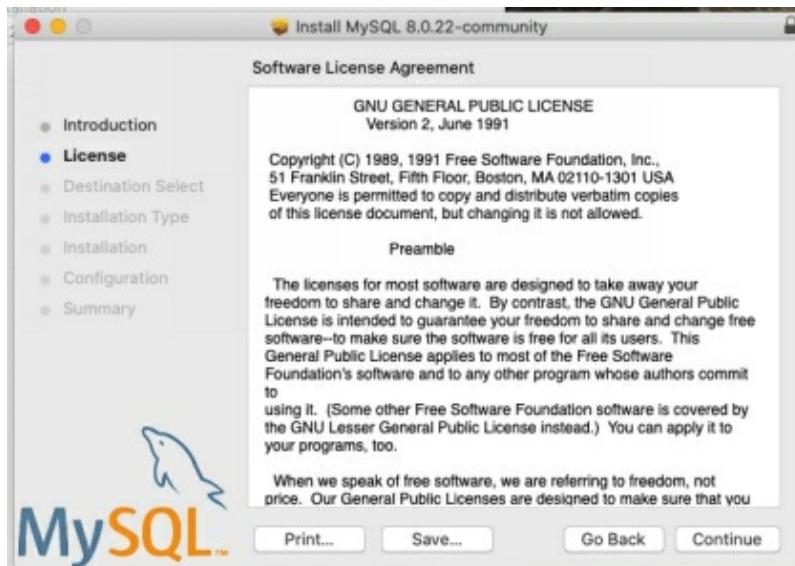
The text "No thanks, just start my download." is visible at the very bottom of the page.

4. Locate the .dmg and upon double clicking the pkg installer appears

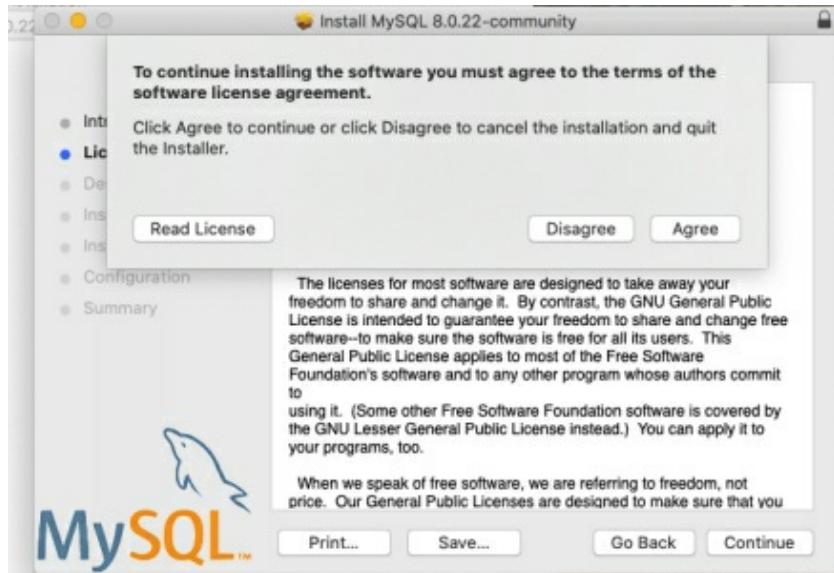
as shown below



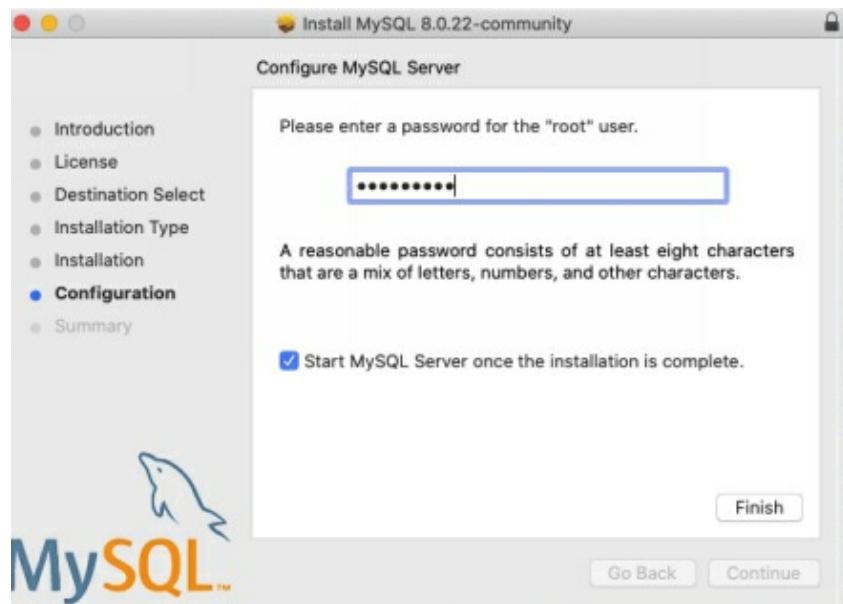
5. Click on Continue



6. Click Agree



7. Enter your Admin password



8. Click on Next and keep accepting the default to complete the MySQL configuration.



2.7 INSTALLING MYSQL WORKBENCH ON MAC OS

I did not find a full MySQL installer including the server and the workbench client for Mac OS. We will download and install the Workbench Client separately as shown below.

1. Start with the following google search and visit the first link

A screenshot of a Google search results page. The search query "download mysql workbench for mac" is entered in the search bar. Below the search bar, there are filters for "All", "Videos", "Images", "Shopping", "News", and "More". The "All" filter is selected. The search results show "About 854,000 results (0.57 seconds)". The top result is a link to "dev.mysql.com › downloads › workbench" titled "Download MySQL Workbench - MySQL". Below the link, it says "MySQL Workbench 8.0.22. Select Operating System: Select Operating System... Microsoft Windows, Ubuntu Linux, Red Hat Enterprise Linux / Oracle Linux ...". At the bottom of the snippet, it says "You visited this page on 12/29/20."

2. Download the file

④ MySQL Community Downloads

« MySQL Workbench

The screenshot shows the MySQL Community Downloads page for MySQL Workbench 8.0.22. The user has selected 'macOS' from a dropdown menu under 'Select Operating System'. A note indicates that packages for Catalina (10.15) are compatible with Mojave (10.14). Below this, there is a download link for 'macOS (x86, 64-bit), DMG Archive' with version 8.0.22, size 109.2M, and a 'Download' button. A link to the MD5 checksum and GnuPG signature is provided. A note at the bottom suggests using MD5 checksums and GnuPG signatures for integrity verification.

Copyrights © Oracle Corporation

3. Click on No thanks....

⑤ MySQL Community Downloads

Login Now or Sign Up for a free account.

An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system

The screenshot shows the MySQL login/signup page. It features two prominent buttons: a blue 'Login »' button with the subtext 'using my Oracle Web account' and a green 'Sign Up »' button with the subtext 'for an Oracle Web account'. Below these buttons, a note states: 'MySQL.com is using Oracle SSO for authentication. If you already have an Oracle Web account, click the Login link. Otherwise, you can signup for a free account by clicking the Sign Up link and following the instructions.'

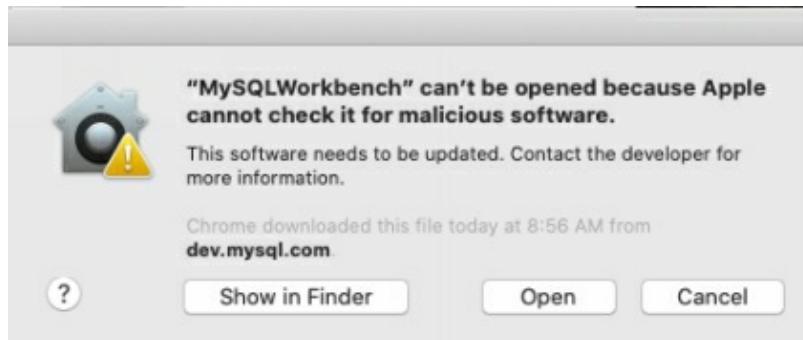
No thanks, just start my download.

Copyrights © Oracle Corporation

4. Drag the MySQL Icon to the Application folder



5. On the Mac if you double click the MySQL Workbench icon, it shows a security warning. Instead, right click and choose open to show the following screen. Choose Open again

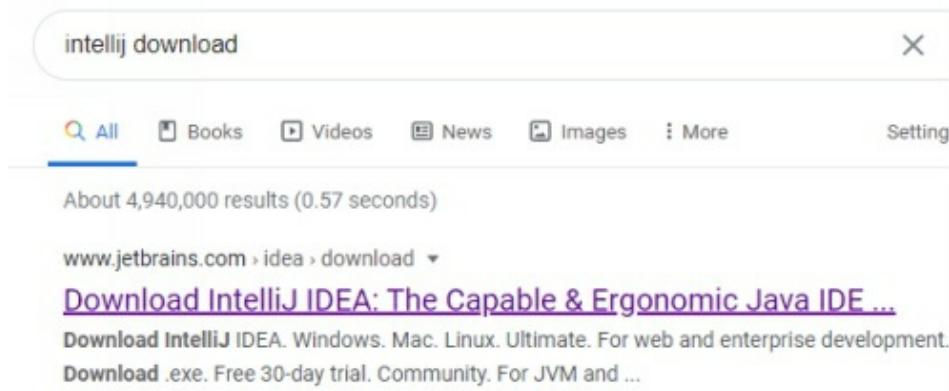


6. MySQL Workbench on your Mac OS



2.8 INSTALLING INTELLIJ IDE

1. Search google for IntelliJ download



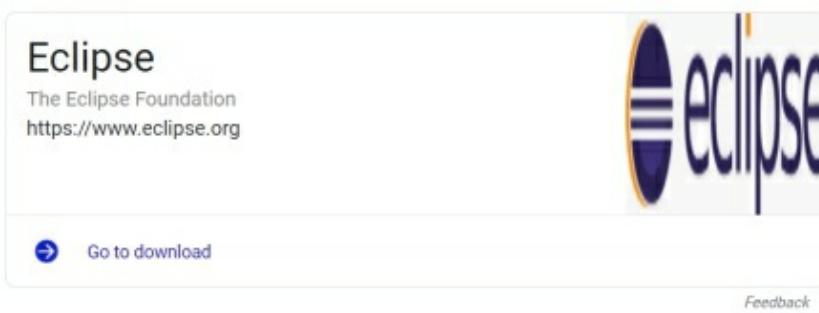
2. Click on the first link to get the following screen



3. Download the free Community edition unless you want to purchase the commercial one
4. Once the download completes you can double click the .exe file to install it

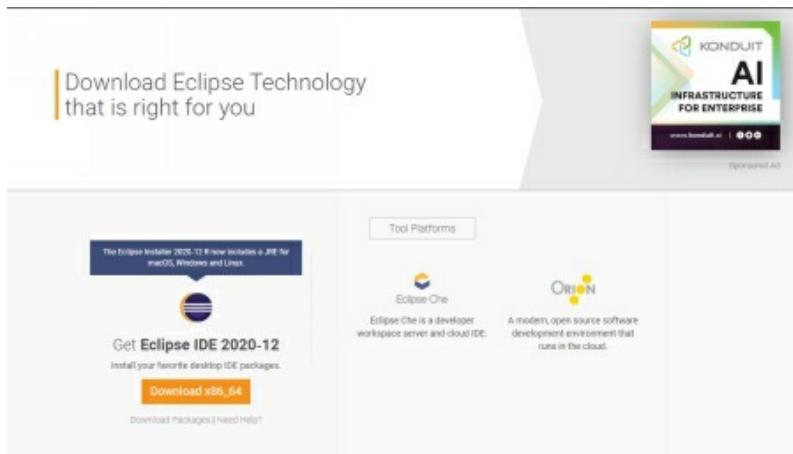
2.9 INSTALLING ECLIPSE

1. Search Google for Eclipse download



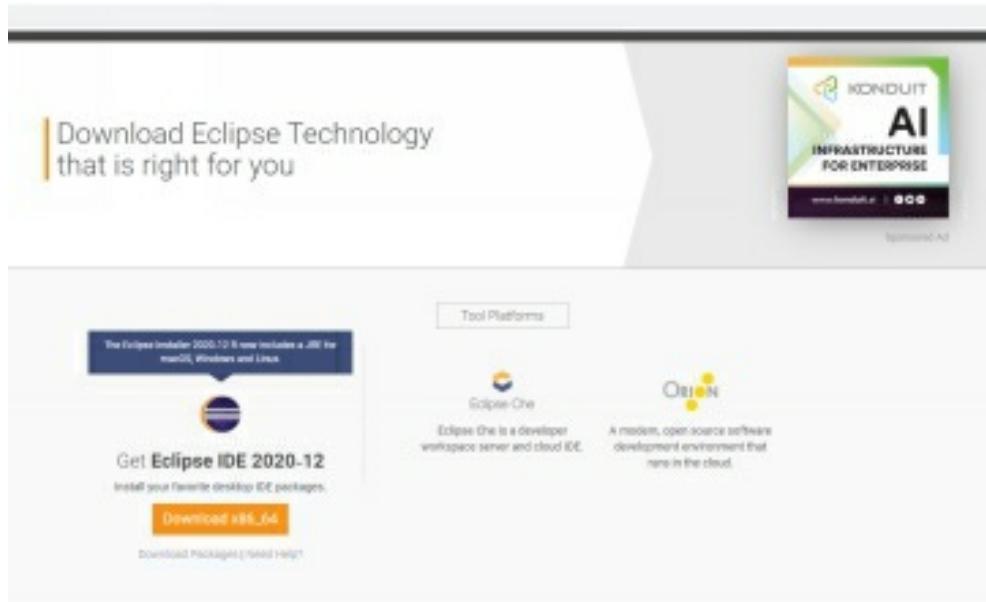
<https://www.eclipse.org> › downloads ::
Eclipse Downloads | The Eclipse Foundation
Get Eclipse IDE 2021-03. Install your favorite desktop IDE packages. Download x86_64 ·
[Download Packages](#) | Need Help? [Eclipse ...](#)

2. Click on the first link to navigate to the following



Copyrights © Eclipse Foundation

3. Download the Eclipse IDE installer on the left



4. Click on the Download button again

A screenshot of the Eclipse Foundation's Downloads page. It shows a large orange "Download" button. Below it, the download URL is listed as "File: eclipse-inst-jre-win64.exe SHA-512". There is also a link to "Select Another Mirror". A dark blue banner at the bottom says "OR Get It Faster from our Members".

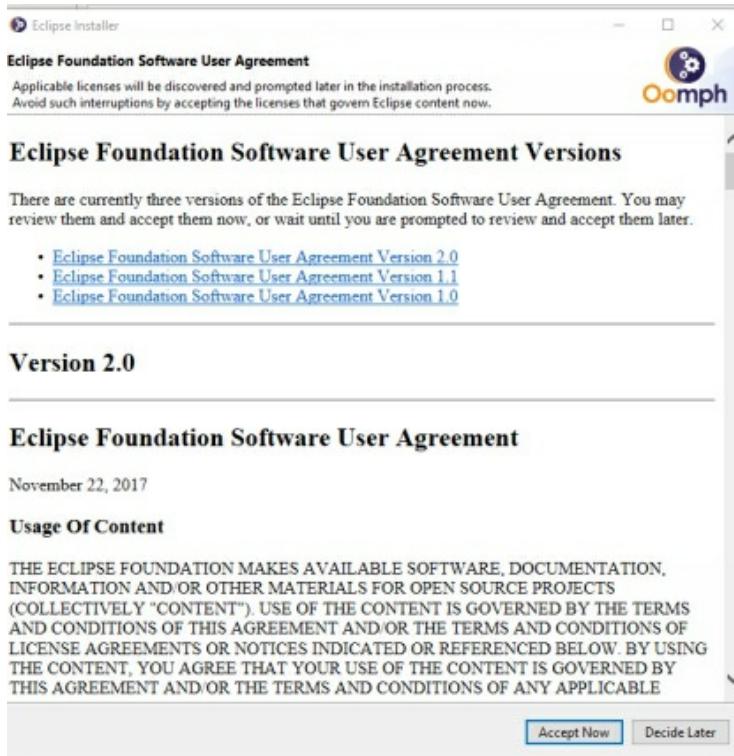
5. When the download completes, locate the file and double click on it. Choose Eclipse IDEA for Enterprise Java Developers



6. Click Install



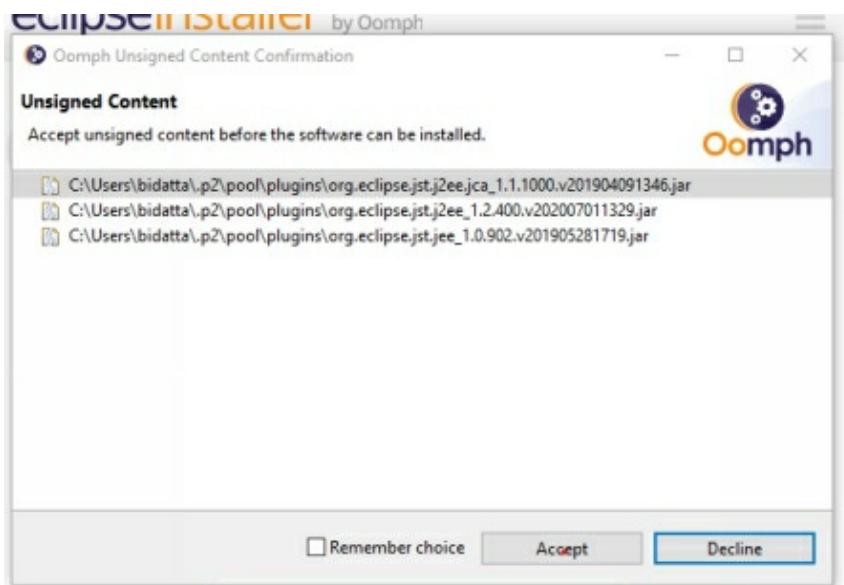
7. Click Accept



8. Installation in Progress



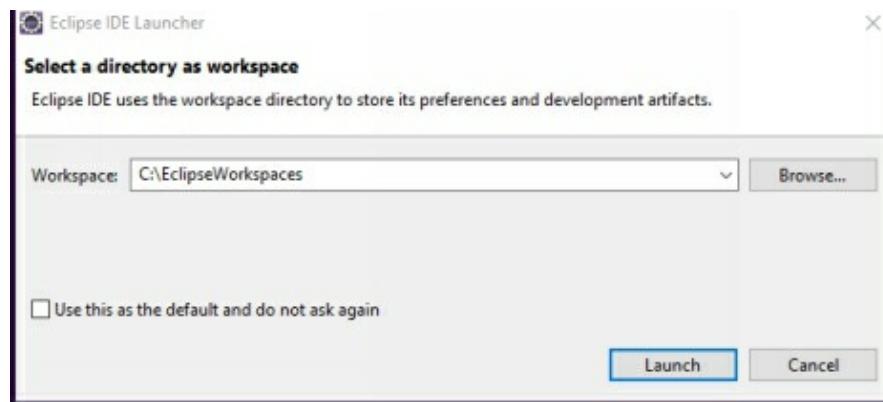
9. Click Accept again



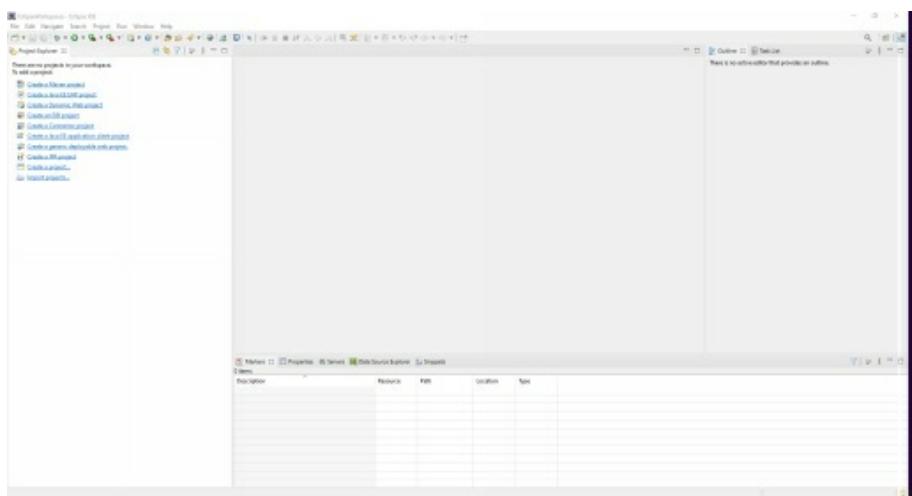
10. Click Launch



11. Create a Distinct directory and choose that as Workspace



12. Eclipse IDE opened



2.10 INSTALLING GIT SCM

Git as you might know is a widely popular source control system. We will install Git SCM client on our machines as shown in the following screens.

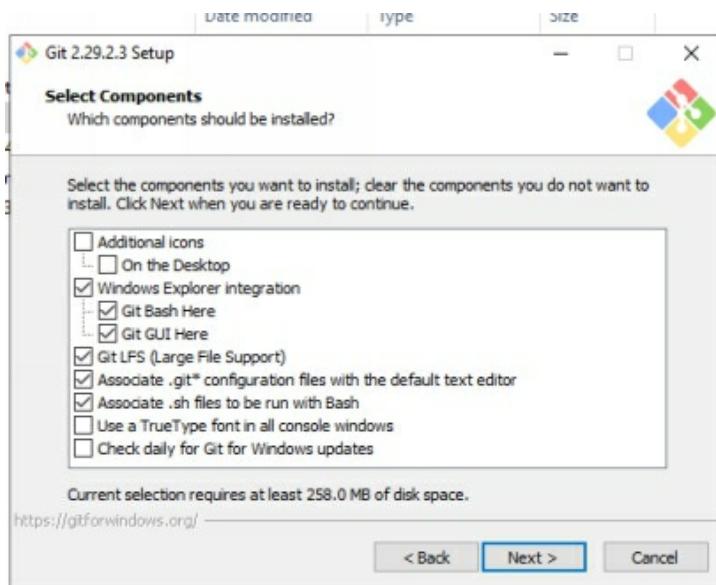
1. Visit the following site to download git scm installer



2. Download the Windows or Mac version and double click to start the installation



3. Choose all defaults in the numerous choice screen shown by the Git installer and wait till the process is complete.



2.11 INSTALLING DOCKER

We need to Docker to be installed on our Windows or Mac development machines. Following the following process to install Docker on your machine.

1. Search google for the following

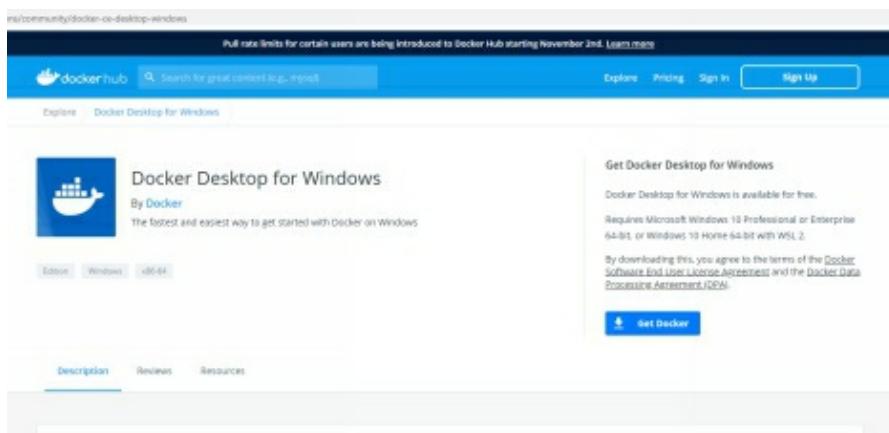


2. Visit the first link



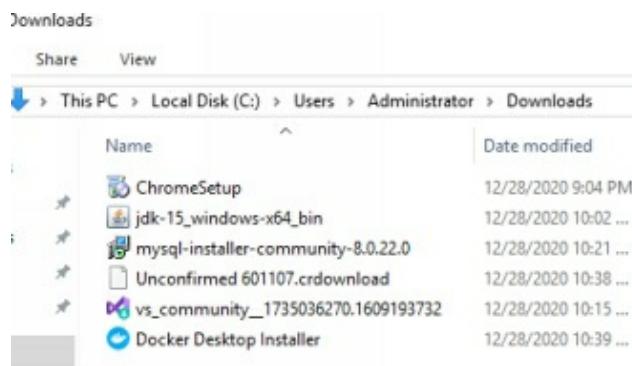
Copyrights © Docker

3. Click on the Download from Docker Hub button



Copyrights © Docker

4. Download and double click on the installer shown below



5. Installation in progress

Installing Docker Desktop 3.0.0 (50684)

Docker Desktop 3.0.0

Unpacking files...

```
Unpacking file: resources/docker-desktop.iso
Unpacking file: resources/docker
Unpacking file: resources/dvdp.ico
Unpacking file: resources/config-options.json
Unpacking file: resources/componentsVersion.json
Unpacking file: resources/bin/docker-compose
Unpacking file: resources/.gitignore
Unpacking file: InstallerCli.pdb
Unpacking file: InstallerCli.exe.config
Unpacking file: frontend/vk_swiftshader_lcd.json
Unpacking file: frontend/v8_context_snapshot.bln
Unpacking file: frontend/snapshot_blob.bin
Unpacking file: frontend/resources/app.asar.unpacked/node_modules/ssh2/util/pagent.c
Unpacking file: frontend/resources/app.asar.unpacked/node_modules/ssh2/util/build_pae
```

6. Docker Desktop on Windows Installed



CHAPTER 3

Kubernetes Architecture

3 INTRODUCTION: DEMYSTIFYING CLOUD COMPUTING

As we discussed at the beginning of the book, Cloud computing is all about renting instead of owning. As we can rent Cabs/Uber/Lyft services if we have a phone, we can rent computing services from any Cloud providers on demand within minutes. However, let us go slightly more in-depth and understand a few things better before going further.

3.1 VIRTUALIZATION

The driving force of Cloud computing is the underlying Virtualization. What is Virtualization? Imagine single-family homes standing on their own physical land. Can we have a single-family home for all the world's 8 billion-plus people? We will probably run out of physical land even if all can afford the price of such families. That is why in big cities where land is scarce, apartment buildings are quite common. An apart from building shares the same physical land but still separates the apartments among the different apartment owners. Virtualization lets us do the same with physical computer servers, which we frequently call bare metal. First, on the bare metal, we install software called Hypervisor. It is the Hypervisor that interests in the bare metal hardware. The Hypervisor can be compared with the foundation of the high-rise apartment building that stands on the physical land. On that Hypervisor, we can then install multiple Operating Systems, each representing one virtual machine. We can imagine one virtual machine as one apartment in a large building of 200 plus apartments. How does the Hypervisor secure the virtual machines from interfacing with one another? The apartment has doors and windows and soundproof walls to restrain themselves from one another. Similarly, Hypervisor has software walls to prevent one virtual machine from accessing the memory, disk, and network cards allocated to other virtual machines.

Without Virtualization, any large Cloud provider would quickly run out of physical space to offer physical machines to their millions of customers. That is why Virtualization is the founding principle of all Cloud computing. Cloud providers use large physical computers to install these virtually separated machines and offer those on rent to their customers. As they do not have to spend time buying the physical hardware and configuring the virtual machines can be done quickly in minutes, renting these VMs is so fast.

3.2 NETWORK SECURITY

Customers still need strict network isolation and security even (probably more so) in a Cloud environment. However, in the Cloud environment, the network security firewalls are written in software instead of the physical hardware in an on-premise environment. In Amazon Web Service (AWS), this software driven network security layer is called the Virtual Private Cloud or VPC. The VPC ensures that two customers network communication with their Cloud-based network and back to their on-premise network remain entirely isolated even though the communication shares the same physical infrastructure.

3.3 CLOUD CPU, MEMORY, DISKS, NETWORK BANDWIDTH

All CPU, Memory, Disks, and Network Bandwidth are shared, virtual, and implemented using the software in a Cloud environment. We have our VMs CPUs as vCPU allocated by the Hypervisor, for example. Similar memory and disks are virtually allocated by the underlying physical layer. Thus, if we can understand how a real estate company is selling different apartments to different owners and shares the same physical land, we know the base concept of Cloud computing. Let us then move from here.

3.4 A WORD ON DISTRIBUTED COMPUTING

IT in modern age is all about horizontal scalability, high ability, fault tolerance, disaster recovery, etc. After another, one commercial or open-source product has won its position in the market by scoring high in the areas mentioned. This includes a highly scalable database such as MongoDB, Cassandra, Cosmos DB, Caching software such as Redis, Batch processing software such as the Hadoop Family, etc. Kubernetes fits nicely into this family of distributed computing as well. Why is distributed computing so desirable despite the added complexity to design software in a distributed way? The answer lies in the ease of horizontal scalability. However, let define these terms a little bit more:

Horizontal Scalability: This is adding new servers to a grid of existing servers to expand capacity. Vertical scalability is adding more RAM, CPU, Disk to the same server. Let us imagine. A fast-growing business organization has one office building. Growth demands additional space, and the organization constructed two more floors on top of the ten-floor building it has its HQ. However, soon the building foundation's limits would prevent the same organization from adding more floors to the same building.

Similarly, limitations of processors in hardware would prevent adding more CPU, disks, etc. Vertical scalability has its limits in real life as in IT / software/hardware space. Now, the organization can add/build/buy/rent/lease a new office building somewhere in

the same city or even thousands of miles away. If the same organization faces capacity challenges in its high traffic website, it can add new servers. This is called horizontal scalability and is free without limits. Organizations or IT planners can add as much capacity horizontally as they want/need. As long as phone/internet connectivity among all the office buildings of business organizations, it should not matter where they are located.

Similarly, if there is internet/network connectivity between all the servers that say participative in a group, i.e., cluster of some distributed software, i.e., Kubernetes, it would not matter.

However, because servers located across a long-distance face higher network latency, general practices keep them close, say in a single cloud data center.

Adding more floors to the same office building may disrupt normal business operations. Quite similarly, adding CPU/RAM, Disks to the same server may need complex configuration, may have complex license agreement involve, may need shutdown window time, etc. In contrast, distributed computing helps us adding new capacity without disrupting existing functionality with a relatively simple configuration. However, distributed computing inherently solves the other challenges such as high ability, fault tolerance, disaster recovery. A business organization has three office buildings located miles away from each other. It may not have to shut down completely when one of the buildings is impacted by fire, flood, or other natural disasters. A business would be "highly available" with one building out of service but two still operating. The same applies to IT architecture as well.

Fault tolerance has much to do with defense programming, however. Catching exception, circuit breaking (we will see it with Hystrix), closing connections to files, databases, using connection pools instead of physical connections, and other such practices are

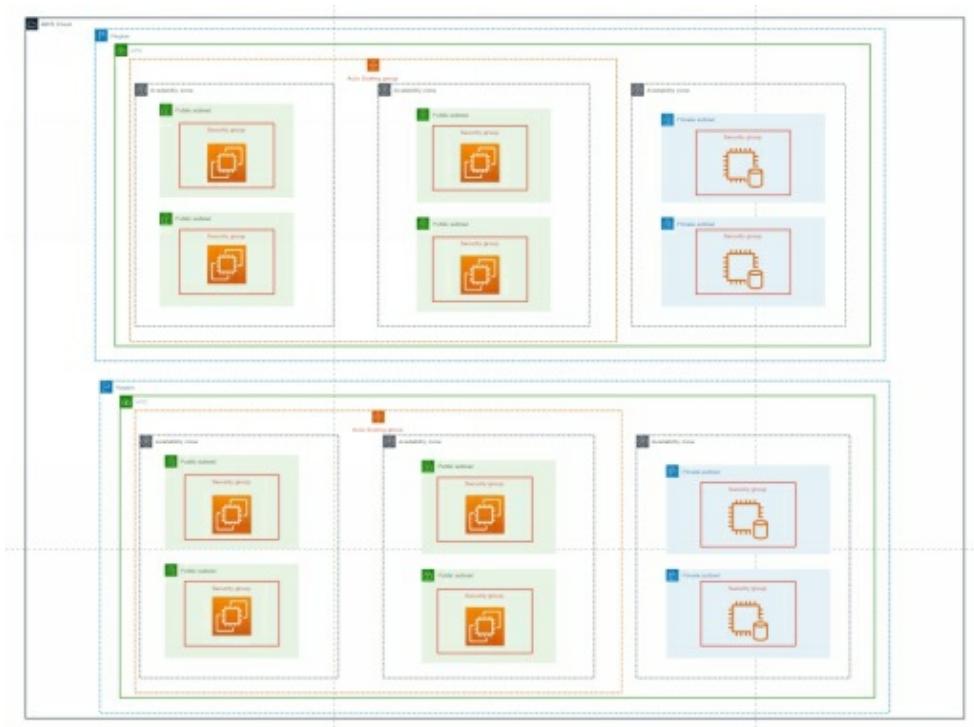
great ways to enhance our applications' faulty tolerance. However, think about connecting to a single MySQL, and when it is down, a highly fault-tolerant application is down as well. If we rearchitect the same faulty tolerant app with a MySQL that has a master/slave active/passive architecture, when the master goes down, the slave becomes the master. Our fault-tolerant app can continue to operate by catching the connection exception and reconnecting to the new master.

Disaster recovery is a more significant high availability. Companies that have massive on-premise computing capacity place them in separately located buildings. Applications that are redundantly deployed to multiple such data centers can quickly recover during a natural disaster in one set of buildings, just by changing DNS names (if we do it correctly).

When we consider the massive growth of companies such as Google, Facebook, Amazon, Apple, and others, we can understand why adding capacity on the go rapidly is so critical. We can imagine how Microsoft Teams, Zoom, or Slack may have needed an enormous capacity increase in a short time during the pandemic. The underlying architecture making it easy to add new capacity becomes a real asset during these times of high growth. Thus, we can see that distributed computing (especially in the era of Cloud computing) makes so much more sense than putting all application tiers in one hardware or all deployment platforms in a fixed number of boxes.

3.5 PHYSICAL STRUCTURE OF CLOUD DATA CENTERS

While we will delve into the AWS Cloud's some depth, it would be useful to establish some idea upfront. The image/figure below shown how, in theory, a cloud provider offers high availability, security, disaster recovery, scalability etc.



Let us understand the AWS Cloud hierarchy of compute services

- AWS Cloud (has multiple regions)
 - AWS Region (has multiple availability zones)
 - One Availability Zone has multiple subnets
 - One subnet may have multiple EC instance VMs and has a software firewall called a Network Access

Control List or NACL

- Each EC2 instance is protected by a software firewall called a Security Group

Let's try to decipher the diagram above step by step from the outer most layer to the inner most

- AWS has over 18 or more regions and we can see two of them in the diagram above
- Two regions are hundreds of thousands of miles apart from one another.
 - US West 2 is in Oregon and US East 2 is in Virginia
 - All US Regions are thousands of miles away from AWS regions in Asia, Australia and Latin America
- Thus, if applications are highly critical and are deployed to physically separate regions, chances are one full region outage may not impact the other region for us to have quick disaster recovery.
- High Availability →
 - Each Availability Zone has multiple buildings that are closely located
 - One Availability Zone is miles apart from another in the same region
 - Unless it is a huge flood, wildfire or earthquake, multiple availability zone becoming unavailable is a rare situation
 - Thus, an application that is deployed to multiple availability zones can live through smaller cloud specific outages smoothly
 - Auto scaling groups can span multiple availability zones and upon increasing CPU or memory events add new EC2 instances without manual intervention. We will talk more about this soon
- Virtual Private Cloud
 - Separate customer A's network from Customer B's

network even if they are deployed in the same physical computer in the same availability zone and in the same region

- A single VPS can span multiple availability zones in the single region
- Two VPCs in two different Regions can connect using VPC Peering, public IP addresses, NAT gateway, NAT instances, VPN Connections or Direct Connect connections

3.6 WHAT ARE CONTAINERS

The driving force of multiple brand-new concepts applied to the IT domain has its roots in business logistics' real life. Containers are one of them.

Without using many words, we can look at two pictures from an article written by Deborah Lockridge in 2017. The first image below shows how ships were loaded manually in 1937 by men working in Dockyards. The second image show how ships are loaded now using cranes.



In 1937, longshoremen at ports in New York transfer bananas from a conveyor that carries them from the hold of the ship onto the dock and then load them into freight cars.

Source: [U.S. National Archives and Records Administration](#)



Container ship at the Port of Long Beach. Photo: Jim Park

Lockridge, D. (c. 2017) Container ship at the Port of Long Beach. Photo: Jim Park. Trucking History, United States.

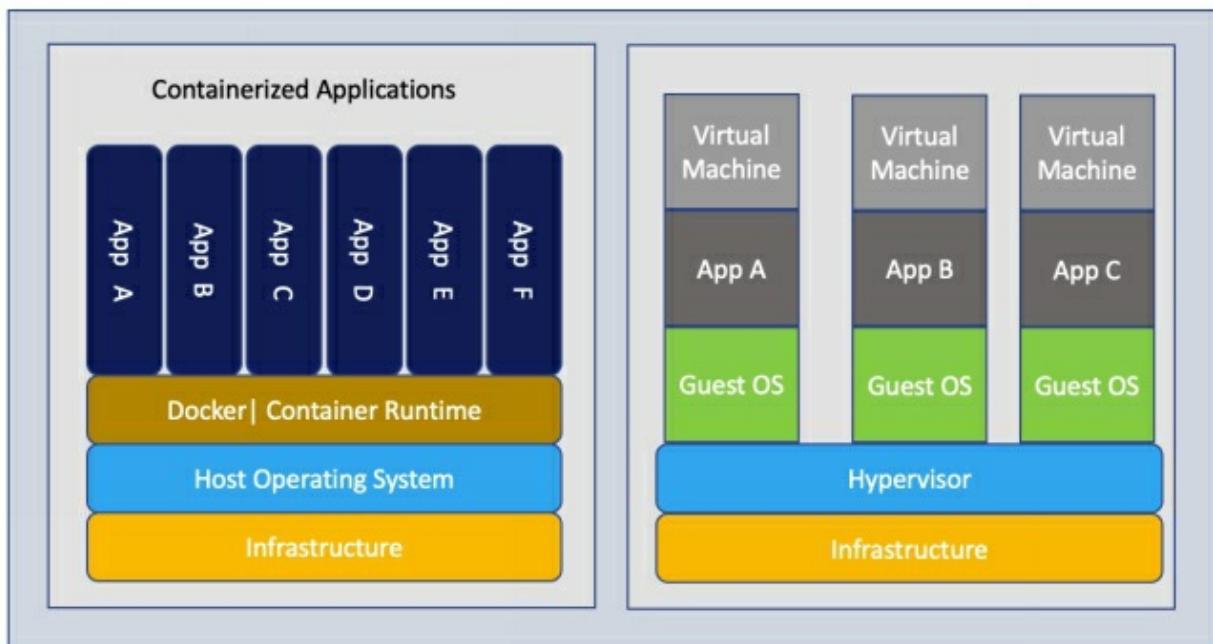
One can imagine the considerable savings in time, among other things. Dockyards loading and unloading huge ships within 30-40 minutes have their productivity grow by over 1000 percent between the 1930s and now. However, productivity growth is made possible by adopting a standard container made of steel and having all containers be of the same size. We can imagine the same containers carried by semi-trucks, or trains, put on a large ship or a cargo plane. The same container!

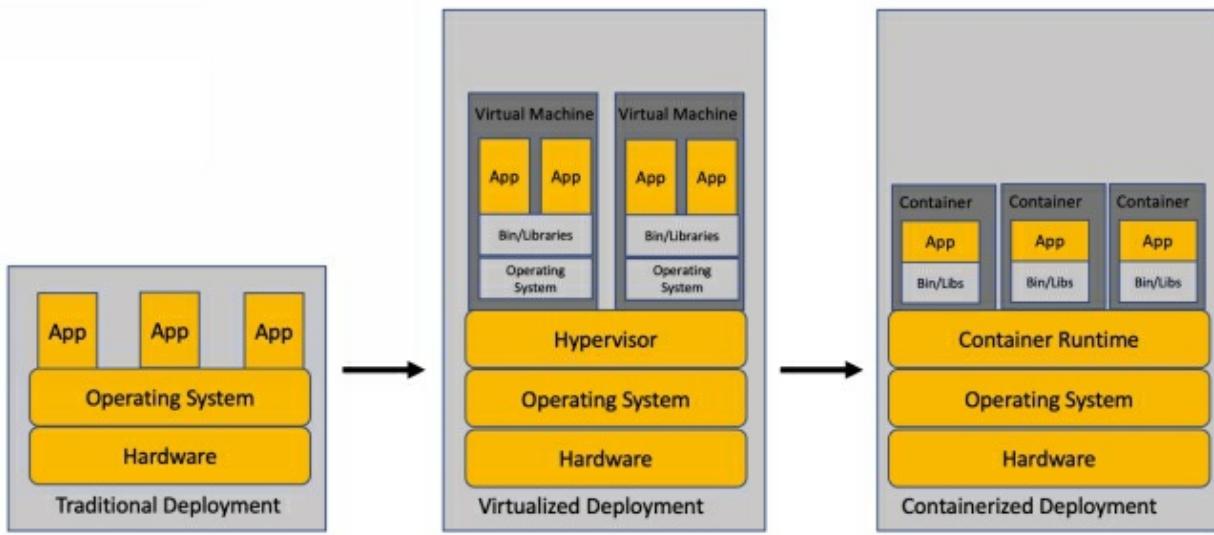
Now let us focus on IT. In IT, we have different environments like Development, QA, Staging, Performance, and Production. Manually building and configuring the same software for each environment would be like loading bananas manually on ships in 1937. If we abstract the application's details from the platform the applications run on, we can achieve the same standardization that the logistics domain achieved with steel shipping containers. The running execution platform, such as QA, Staging, or Production, would not care if the application was built using Java, NodeJS, or .NET, or something else. They would know how to download, start, stop, and update the latest version of these applications using standard containers. A

Crane can load steel containers containing food, furniture, and cosmetics without knowing/bothering about the containers' content. The underlying container runtime in QA, Staging, and Production environments would function in standard ways to download a container for running a UIU application built in NodeJS and download another REST API application container made using Java in very familiar ways. That is the massive advantage of applying some of the enormous productivity boosters in real life to IT.

3.7 ADVANTAGES OF CONTAINERS

Let's come back to one of the massive requirements of Internet based Social media, eCommerce applications. The need is dynamic and fast (with a big F) scalability. What is the dynamic scalability? It is the opposite of static scalability. What is static scalability? Static scalability is to have manual efforts involved to increase capacity. It is very much like loading bananas manually on large ships. Internet applications have fought long and hard to win their independence from human interference, and they now need to be free to deal with growth upon higher traffic and need to shrink upon lower traffic. Virtual machines (with their full-blown Operating System), the size runs into several GBs quickly. The large size impacts how fast they can be download started and stopped. Containers have Operating System, but the OS is trimmed down to a minimum number of services just to make system calls to the base OS sitting down. Following figures show the differences between traditional, virtualized and containerized deployments.

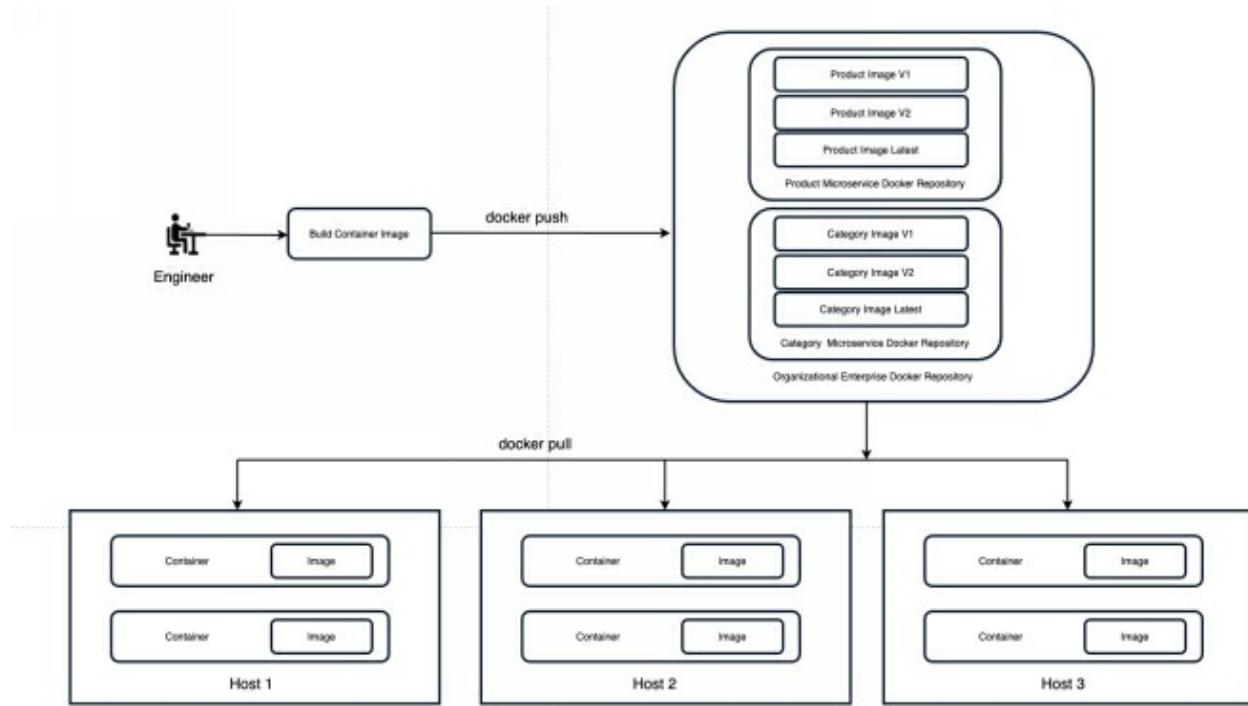




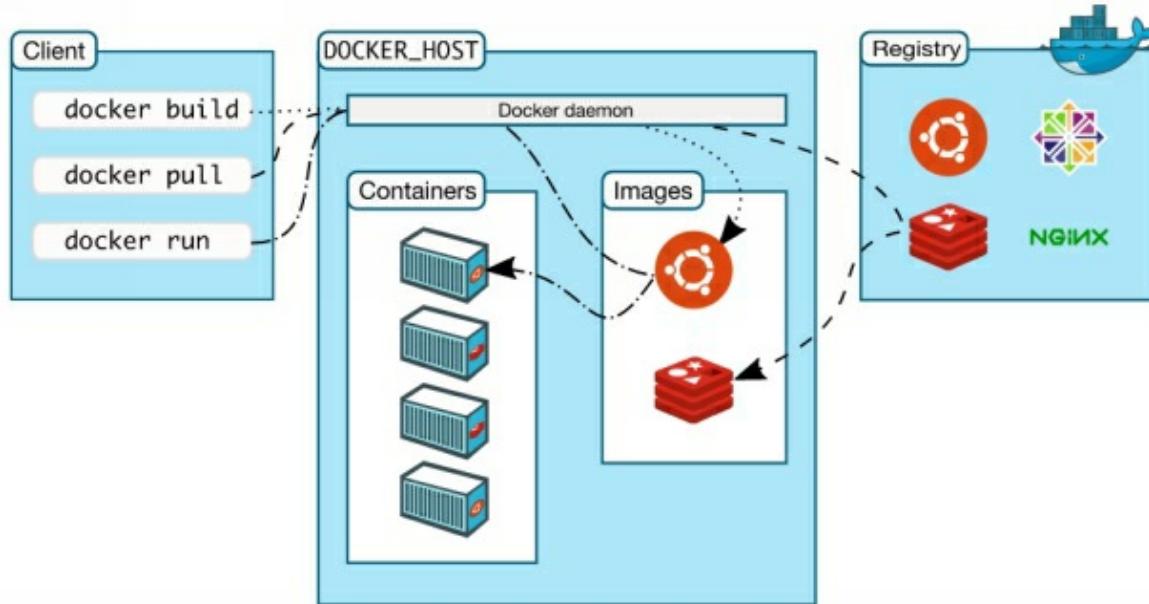
Thus, the sizes of containers are in low MBs compared to the big GB sizes of VMs. This impacts the ability to grow dynamically within seconds or low minutes. This is a big desire during Black Friday when our sites would be pounded upon by millions of users in a short duration. If we look at the side by side images comparing the container structures with the VM structure, we can see how the VMs duplicate the large OSs and give us redundancy we do not need. However, in a cloud environment, containers would be hosted by large VMs instead of physical bare-metal hardware machines. To summarize, containers are lightweight, downloaded much faster and started, and stopped much quicker than VMs. Thus, if we want our Product Catalog Microservice to grow from 5 instances to 20 within the next five minutes, we need to containerize them.

3.8 WHAT IS A CONTAINER IMAGE?

Container Images are packages made for simplifying the transportation from one environment to the other. A real-life equivalent of the container image would be the steel container transported from the shipping location to the semi-truck, the train, the ship or cargo place, etc. A software container image is a complete package to include everything needed to run the application inside the container. Shipping a large machine with all spare parts inside a single container would be quite the same. A software container includes the base operating system (Red Hat or Ubuntu OS), application libraries such as Spring Boot or NodeJS or .NET, application source code, etc. People can also include basic | all properties configuration in the container, but best practice tells us not to do so. With one package having all software code to run it, container runtime can quickly locate the container image (by the image URL), download it, and run it in a standard way without looking inside the container. All container runtimes like Docker know how to first build a container image and then tag the image with uniquely identifiable names or tags. The container runtime then knows how to upload or push the container image to a container repository. Then, some other container runtime can have the image URL to download the image and run it elsewhere. This way, we can build a single container image and run it in the Dev, QA, Staging, Performance, and Production environment provided the environment specific properties (URL to connect to the Dev/Prod MySQL Database, for example) are kept outside the container.



A more realistic execution scenario can be seen below in the image borrowed from Docker Hub



Docker, D. (c. 2021) Container Execution. Photo: Docker. Docker Hub, United States. Retrieved from <https://docs.docker.com/get-started/overview/>

3.9 WHAT IS ORCHESTRATION

Let us imagine managing containers manually in a production environment. If we do a good job, hundreds of Microservices can be carved out of an older legacy monolithic application. To achieve scalability and high availability, we may have hundreds of replicas of these Microservices. Following are some of the bare minimum tasks that we need to perform as manager of the container environment in the production

- Download container images from the container repository thousands of times per day for a large environment
- Start these containers on the best fit worker machine that we can decide.
- Monitor health of these thousands of containers that may be starting, running, stopping, or in an unreliable state
- Evict bad containers when their health is not good
- Replace bad containers with new good ones
- Monitor logs being generated from these thousands of containers
- Monitor performance metrics generated by the applications in these containers
- Provide properties or metadata to the application secure sensitive data such as passwords and supply them to the containers in a safe way
- Others

Doing so many things manually is like the Master of a Musical Orchestra having to play each different musical instrument on his/her own. That is unthinkable, and that is not how it works in real life. The Master of an entire Orchestra selects great musical instruments players and then instructs them with his/her hand on how to play that melodious music. That is precisely how Container Orchestration works. It gets powerful players (we will know them in a bit) and instructs them or coordinate their activities to work as a seamless non-nonsense system to offer peace of mind to the system admins and all

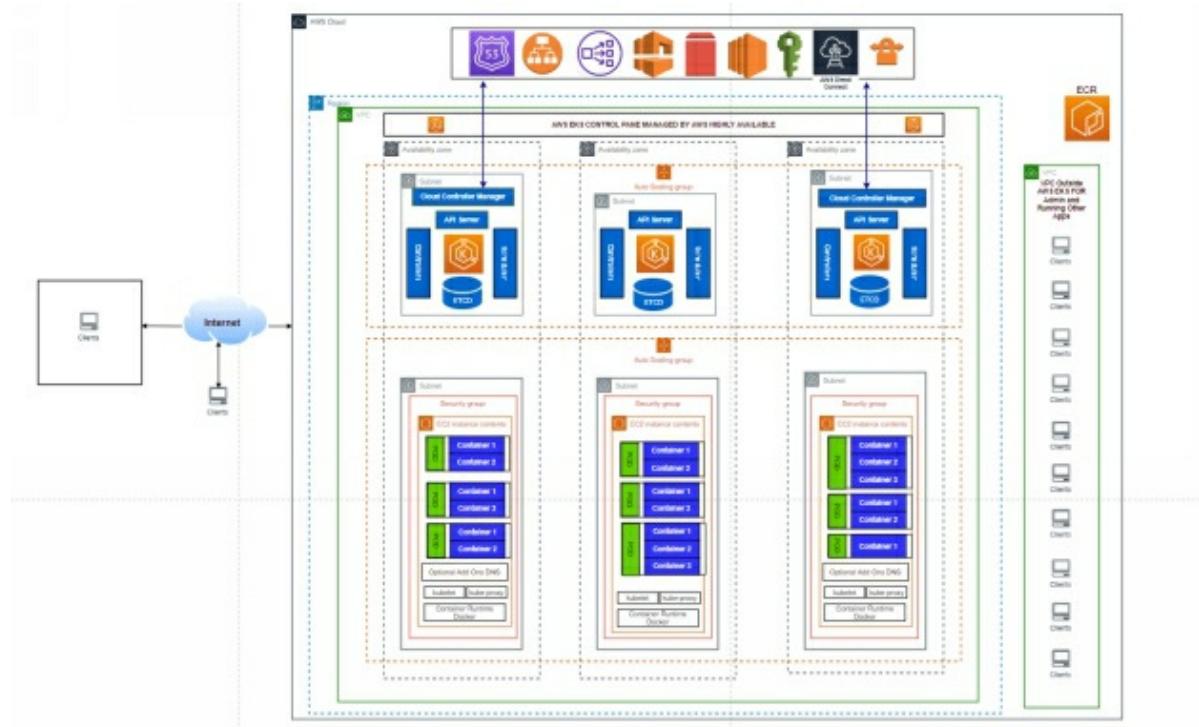
other stakeholders.

3.10 WHY KUBERNETES

While we can understand the tremendous value of a container orchestrator, let's see why we may choose Kubernetes ahead of others like Docker Swarm. The system Kubernetes is based on has been battle-tested at Google for over a long time. Google open-sourced the system called Borg in 2015 and named it Kubernetes. One can understand that Google's design choices when they created Kubernetes were made to be modular and Microservices based. The entire Kubernetes system architecture, as we will see in a little bit, is API driven and loosely coupled. Changing one component with a better replacement may not impact the other. Considering this, many significant commercial organizations such as AWS, Microsoft Azure, IBM, Oracle, VMWare, NGINX, and others have heavily involved in Kubernetes development products. The developer community is vast and growing. Tool support is excellent, and biggest of all, all large Cloud providers have a managed Kubernetes service.

3.11 KUBERNETES ARCHITECTURE

We are clear about how a cloud provider like AWS achieves high availability by placing similar components in multiple physically separate availability zones and subnets. Understanding the Kubernetes architecture would be much more comfortable. The layering of the AWS cloud is no different than any other different application/network architecture. We have the AWS Cloud at the outermost layer. Immediately after the AWS Cloud, we have our AWS Region. Inside the Region, the first AWS layer is the Virtual Private Cloud or VPC, the software layer firewall that protects one customer's traffic. A VPC, as stated, can span multiple availability zones in the same Region. We can see three of the three Availability Zones inside the VPC. Another layer that we see is the AWS AutoScaling Group that, along with AWS CloudWatch, monitors the CPU/Memory workload and, upon hitting a certain predefined percentage, would create additional capacity, more worker node Kubernetes. The same AWS Auto Scaling Zone can also watch for low traffic and eliminate extra capacity if needed



We discuss the AWS specific Kubernetes managed service called Elastic Container Service for Kubernetes or EKS in short. Kubernetes, in general, has two large components, namely the Control Pane and the Worker Nodes. In managed Kubernetes services, the control pane is managed by AWS while customers are responsible for the worker nodes. When we say managed by AWS, we mean that AWS will be accountable for ensuring security, high availability, and performance of the AWS EKS Control pane. However, I wanted to logically show how AWS may be using the same separate AZ deployment and auto-scaling groups. On the top left, we have the Elastic Container Registry or ECR, which is AWS's container registry for us to push and pull our Docker images from. We will discuss the network access in detail, explaining how clients outside the EKS cluster can access services deployed inside the cluster. However, let us first understand the sub-components that run inside the Kubernetes Control Pane/Master and the Worker Node.

3.12 Kubernetes Control Pane (Master) Components

ETCD

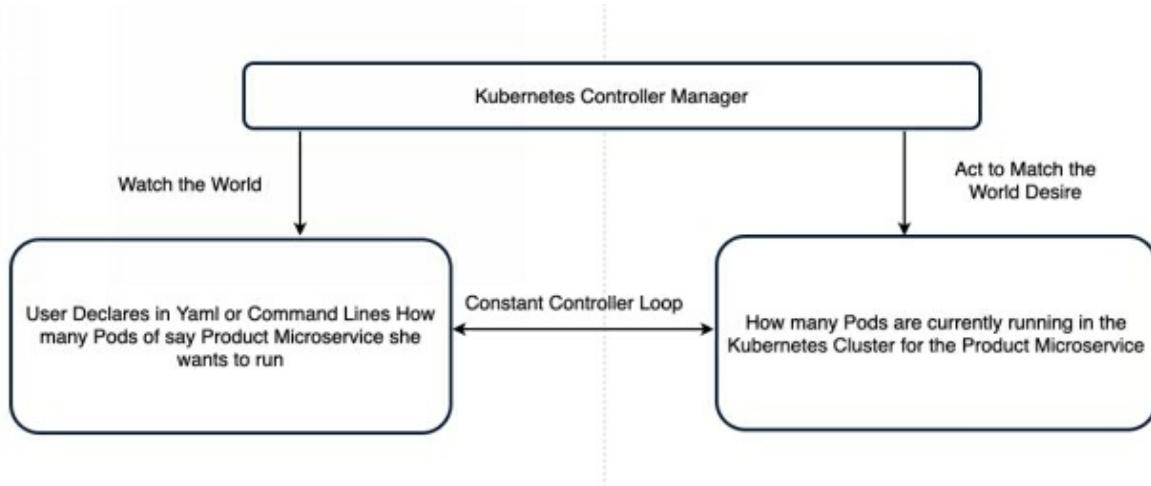
Etcd is a distributed key-value datastore used by Kubernetes to store all cluster configurations, such as how many instances of an application can run, secrets, properties, worker nodes, etc. Etcd is highly available and uses nodes located in physically separated VMs to store/replicate data. When the cluster is shut down usually or crashes for some reason, Etcd is used to restore the cluster state when Kubernetes is restarted. All commands issued using kubectl (the Kubernetes command-line utility) and configuration made through the GUI and other clients are stored in Etcd as well.

AI Server

The Kubernetes Cluster is quite like a business organization having multiple office locations (Worker Nodes). As all official communication is conducted from the Head Quarter Public Relations / Press team, Kubernetes API Server (kube-apiserver) is responsible for providing all information from anywhere in the Kubernetes Cluster. Command-line tools (kubectl)/GUIs/ other clients submitting YAML files, trying to retrieve information about

applications/secrets/configurations deployed in the cluster, all communicate with the API Server. Even internal Kubernetes components like the Controller Manager and others talk to the API Server only for doing their work. The API Server is also highly available and can scale horizontally by creating more instances and distributing the work between the instances. As one might guess, the API Server read/write information from the Etcd database.

Scheduler



The picture above is presented here on purpose. Kubernetes Scheduler (kube-scheduler) is the Kubernetes Control Pane component that constantly watches the state of the cluster. When the scheduler sees that a newly created Application in the API Server response has not yet been deployed to a worker node, it takes the unassigned application for scheduling. The scheduler runs a complex algorithm taking individual resource requirement of the application as well as consider placement constraints (more on that later), worker node current load and many other things to determine a worker node that this application can be deployed on.

3.12 CONTROLLER MANAGER

There are a few separate controllers included in the Controller Manager to ease a single binary deployment. These processes are called Controller as they watch and try to make the current cluster state same as the desired state. In other words, these processes control the cluster. The diagram shown in the Schedular section is still relevant. All the Controller tries to keep a close eye on the current state and take actions to bring the cluster to the desired state. Together they perform automation for many of the Kubernetes orchestrator's purpose of life. Following are the separate Controller includes:

- Node controller: Keep an eye on how many worker nodes were configured in the desired state and act if one worker node goes down to bring another one up to meet the desired state.
- Replication controller: Watches how many instances of applications were desired during the creation of the Kubernetes application. If the current number of replicas falls from the desired number, it creates another instance. The scheduler then watches the new instance that does not have a node assigned and assigns the node to run the new instance.
- Endpoints controller: As new application instances come and go dynamically; their IP address is very dynamic. The

Endpoints controller watched the new application instances and copy their IP address to the Kubernetes Endpoint object. Kubernetes Service, in turn, observes the Endpoint object for retrieving the dynamic IP address for load balancing.

- **Service Account & Token controllers:** Performs security actions when new namespaces (like Java packages) create default accounts and tokens for those new namespaces.

3.13 CLOUD CONTROLLER MANAGER

Cloud Controller Manager : When Google open-sourced the Kubernetes project, it was both a usable product and a specification. While Google provided a reliable implementation, the specification is used by multiple Cloud providers such as AWS, Azure, and Google Public Cloud itself to provide various Cloud specific components and services to run Kubernetes in an environment of the Cloud. The exact implementations vary from one Cloud provider to another. For example, Kubernetes uses AWS Application or Network Load Balancers for AWS EKS or AWS custom Kubernetes clusters. In Azure, Kubernetes may use the Azure Load Balancers. The Kubernetes Cloud Controller Manager (`cloud-controller-manager`) is the single inbuilt abstraction that abstracts the different Cloud specific services from the internal Kubernetes only components. The architecture diagram may show how this works.

As with the `kube-controller-manager`, the `cloud-controller-manager` consolidates numerous reasonably independent control loops into a single binary for deployment ease. The `cloud-controller-manager` can also scale horizontally.

The following controllers can have cloud provider dependencies:

- Node controller: For checking the cloud provider API to check if a node has been terminated if it stops responding
- Route controller: For setting up routes using the Cloud provider route table in the underlying cloud networking infrastructure
- Service controller: For working with the cloud provider load balancers

3.14 KUBERNETES WORKER NODE COMPONENTS

Understanding distributed computing is relatively easy if we can understand how an organization with multiple office locations (across miles in the same city or across continents). These offices use standard communication mechanisms such as Phone, video conferencing, emails, SMS, and others. The actual communication method does not matter if there is connectivity between the locations. The same applies to the Kubernetes Master or Control Plane and the Worker Nodes. All external communication goes through the master, and all work gets performed by the worker nodes. The worker node runs a few specific components to keep the communication going between the worker node and the master node components. One example of this worker node - master communication is a health check, master assigning/scheduling a new application to be run in the worker node, etc.

KUBELET

The kubelet is the local representative of the Kubernetes Master that runs in the worker node and ensures that all application assigned to that worker node runs and runs healthily.

KUBE-PROXY

Kube-proxy is the local worker node router, network rule manager that accepts external traffic, filters them and if rules match the external traffic URLs, forward them to the worker node's applications. The Kube-proxy is what is used under the hood for exposing Kubernetes applications to the external client.

CONTAINER RUNTIME

The container runtime is the worker node component responsible for pulling container images, starting them, stop them, terminating them, etc., on behalf of Kubernetes. While Docker is very popular, Kubernetes can work with other container runtimes such as containerd, CRI-O, and Kubernetes CRI.

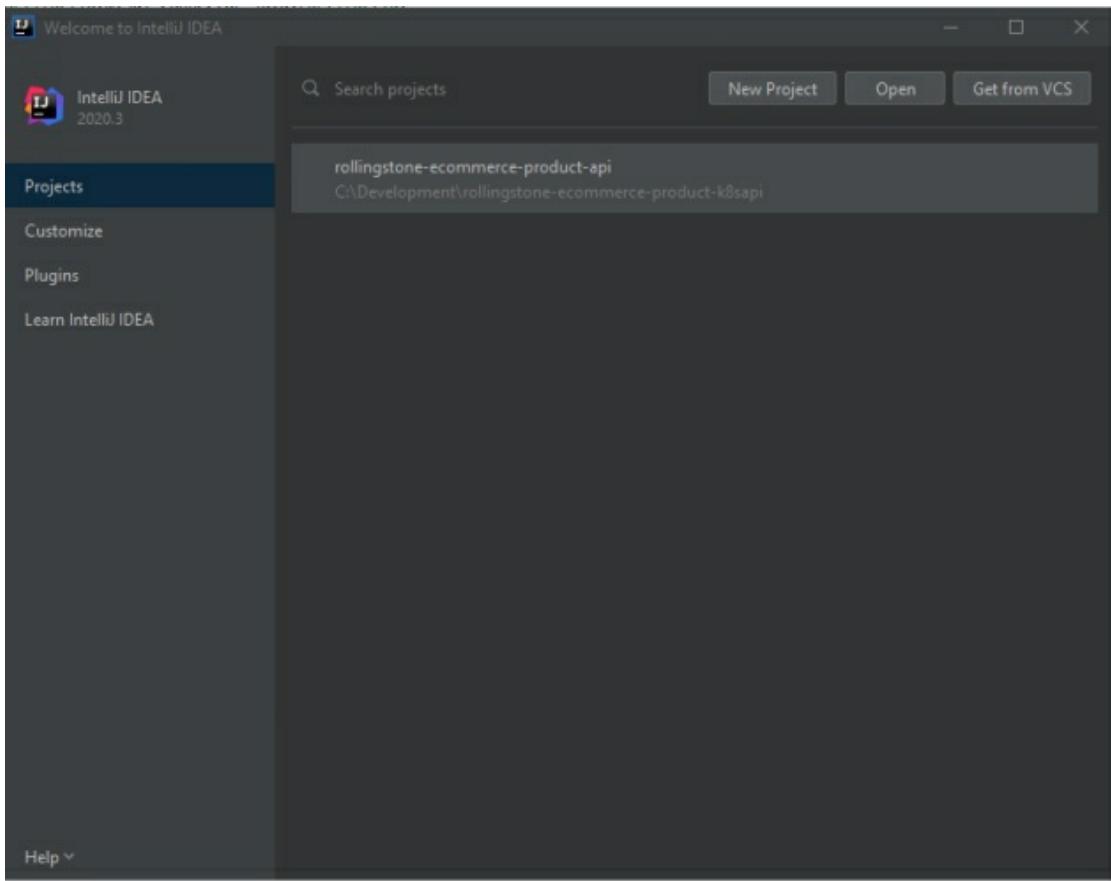
CHAPTER 4

Building the Category REST API

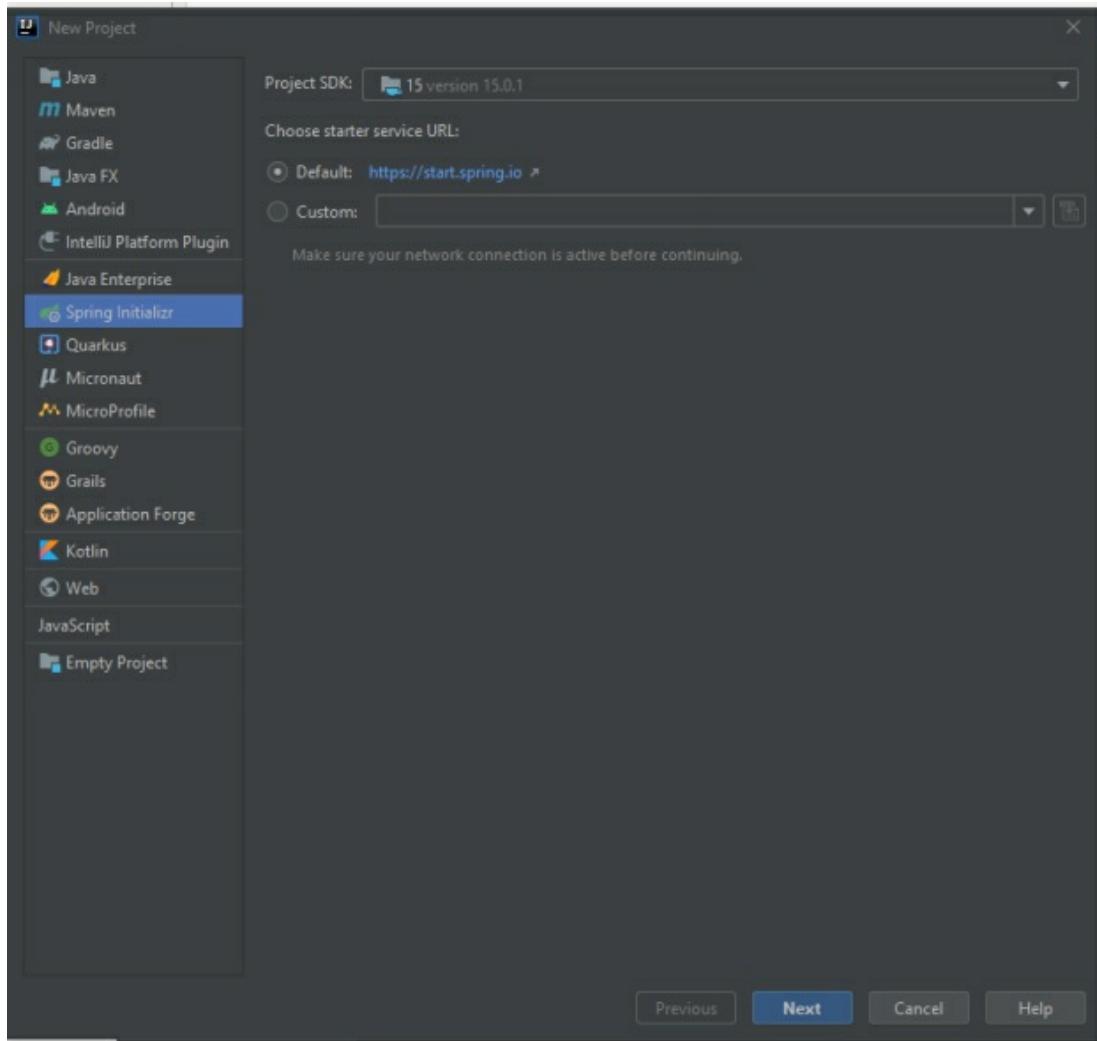
4 INTRODUCTION

4.1 CREATING A NEW PROJECT

1. Start up your IntelliJ IDE



2. Click New Project and Select Spring Initializer and click Next

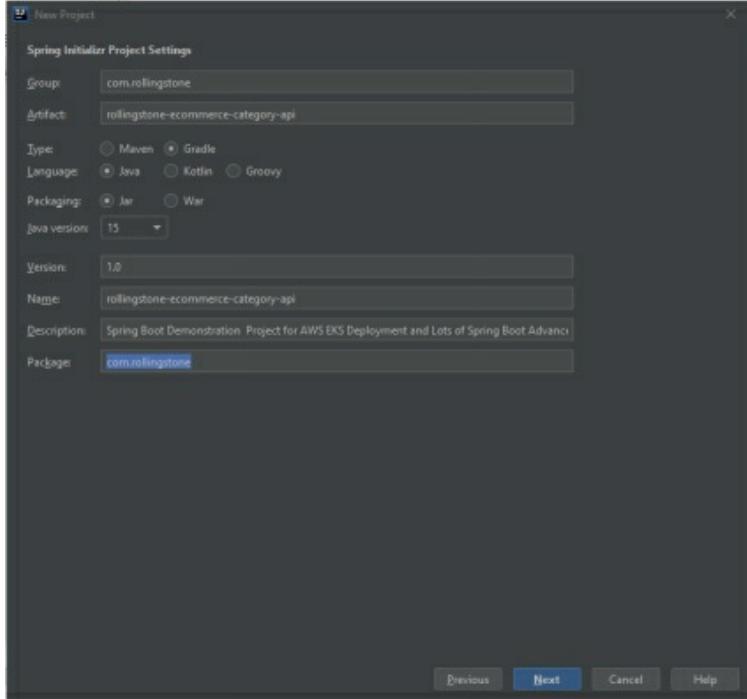


3. Enter the following in the next screen

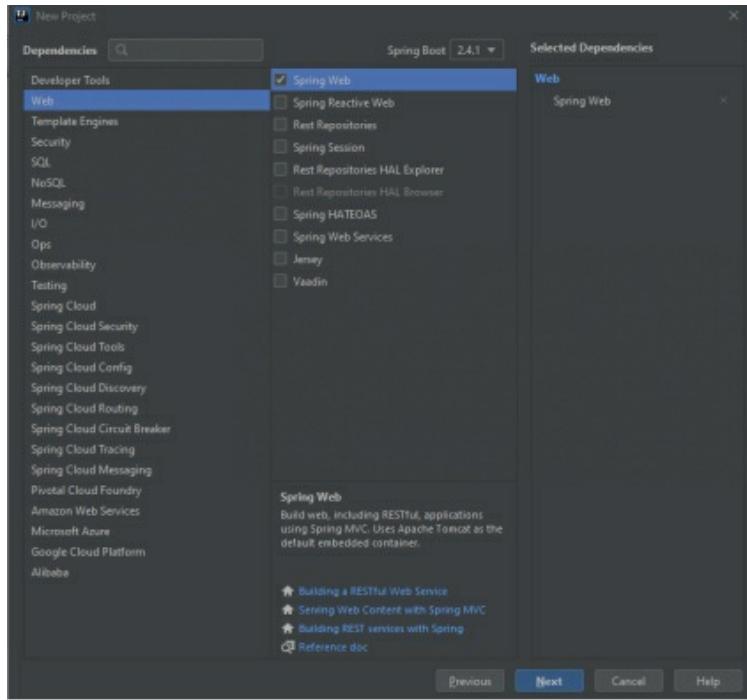
- a. Group: com.rollingstone
- b. Artifact: rollingstone-ecommerce-category-api
- c. Type: Gradle
- d. Language: Java
- e. Packaging: Jar
- f. Java Version: 15
- g. Version: 1.0
- h. Name: rollingstone-ecommerce-category-api
- i. Description: Spring Boot Demonstration Project for AWS EKS Deployment and Lots of Spring Boot

Advanced Features

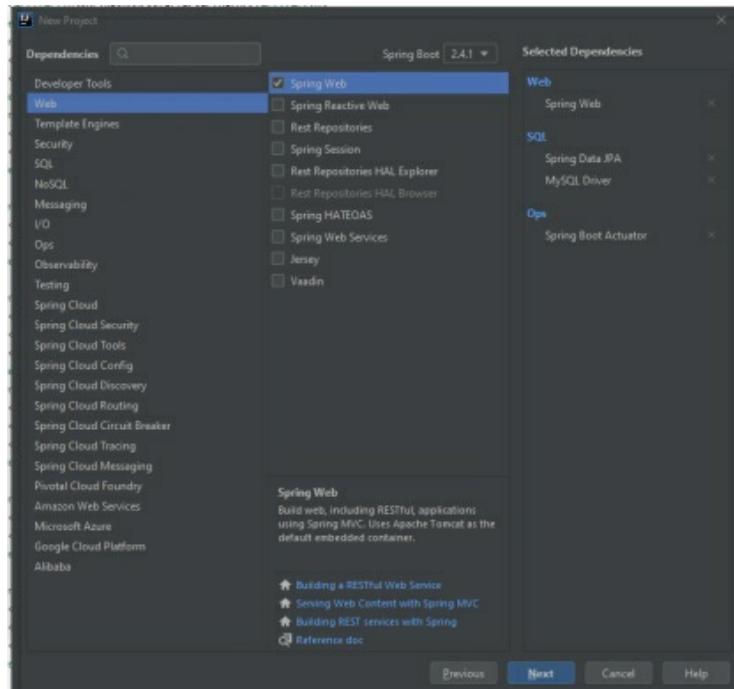
- j. Package: com.rollingstone
- k. Click Next and verify/match with the following



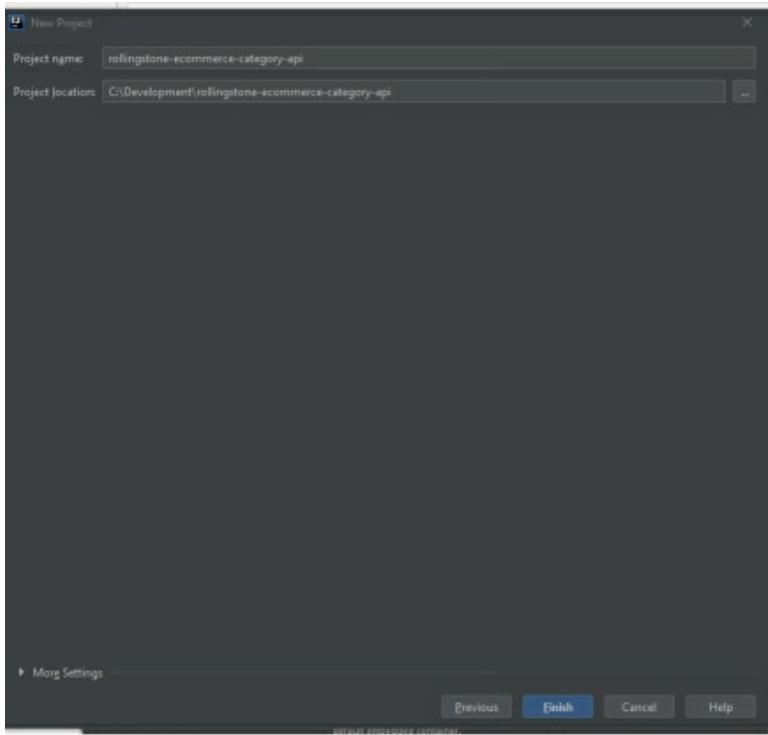
4. Select Web and Spring Web in Dependencies section and click Next



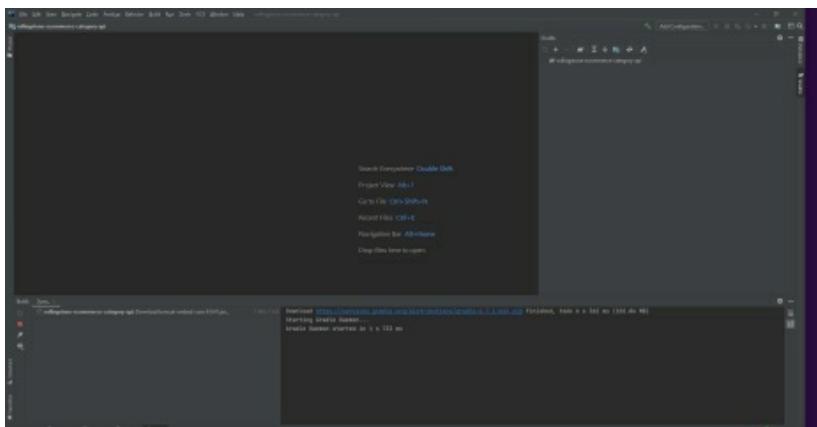
5. Search for jpa, mysql, actuator to add those as well and click next



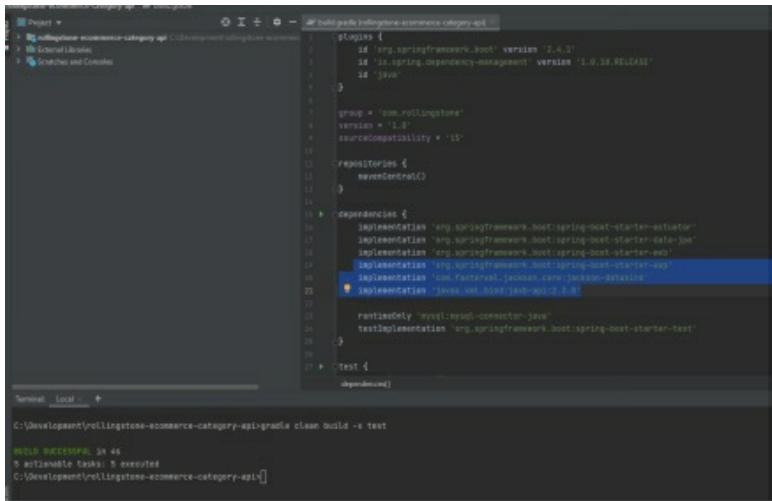
6. Click Finish



7. IntelliJ is preparing the project



8. We need a few more dependencies to be added to the build.gradle file. Open the build.gradle file to add the following
 - implementation 'org.springframework.boot:spring-boot-starter-aop'
 - implementation 'com.fasterxml.jackson.core:jackson-databind'
 - implementation 'javax.xml.bind:jaxb-api:2.3.0'
9. Open a Terminal within the IDE and enter gradle clean build -x test



```
build.gradle (rollingstone-commerce-category-api)
```

```
plugins {
    id 'org.springframework.boot' version '2.4.3'
    id 'io.spring.dependency-management' version '1.0.10.RELEASE'
    id 'java'
}

group = 'com.rollingstone'
version = '1.0.0'
sourceCompatibility = '15'

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-test'
    implementation 'org.springframework.boot:spring-boot-devtools'
    implementation 'com.fasterxml.jackson.core:jackson-databind'
    implementation 'com.mysql:mysql-connector-java'
    testImplementation 'org.mockito:mockito-core'
    testImplementation 'org.junit.jupiter:junit-jupiter-engine'
}

test {
    dependencies {
        ...
    }
}
```

```
Terminal Local: +
```

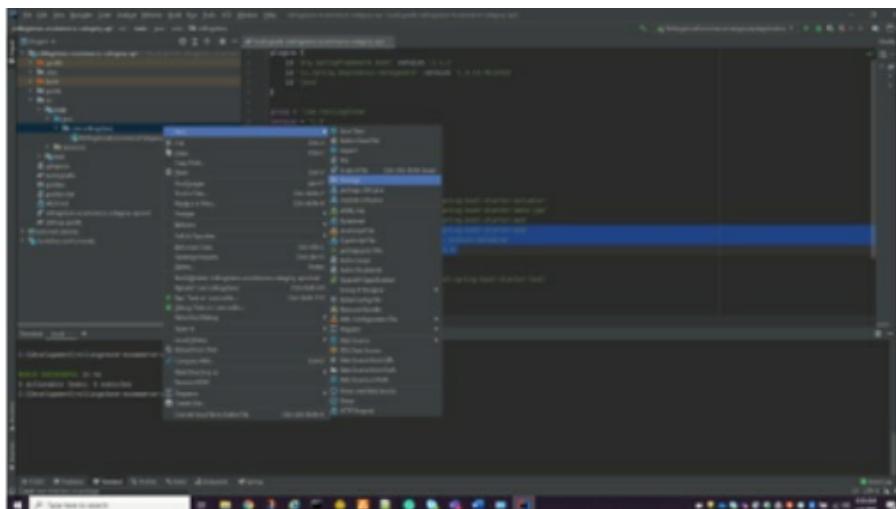
```
C:\Development\rollingstone-commerce-category-api>gradle clean build -x test
```

```
BUILD SUCCESSFUL in 4s
9 actionable tasks: 9 executed
C:\Development\rollingstone-commerce-category-api[]
```

10. Our initial setup is done and lets move to the next section

4.2 ADDING THE PACKAGE STRUCTURE

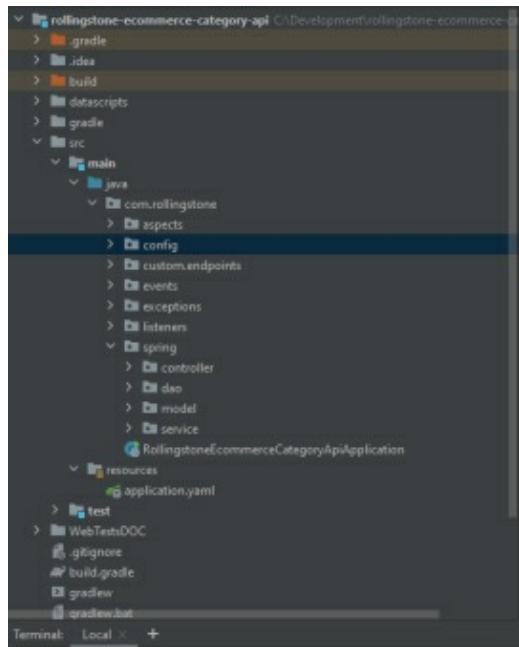
1. Right click on the package com.rollingstone and choose new → package



2. Enter aspects in the dialog
3. Repeat the same package creation process for the following packages under com.rollingstone
 - a. config
 - b. custom.endpoints
 - c. events
 - d. exceptions
 - e. listeners
 - f. spring
4. Right click on the spring sub package under com.rollingstone and create the following sub packages
 - a. controller
 - b. dao

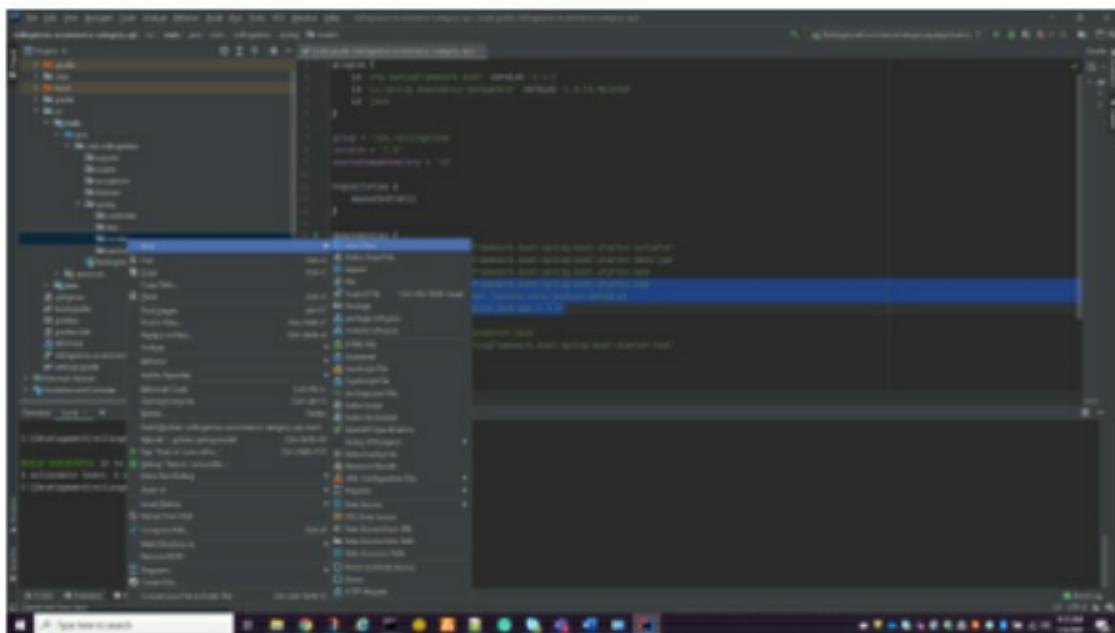
- c. model
- d. service

5. Completed package structure is shown below



4.3 BUILDING THE MODEL CLASSES

We will have one model class named Category. Let's create that now by right clicking the model package under spring sub package of com.rollingstone package



Name the class Category and it will be opened in the IDE's Editor window

Enter the following above the class name

```
@Entity(name = "rollingstone_category")
```

IntelliJ would warn to press Alt+Enter to show an Import diagram and choose javax.persistence in that import diagram to import the import javax.persistence.Entity;

The annotation is telling Java Persistence API that our corresponding table in the MySQL database would name named CATEGORY.

Enter the following in the Editor window

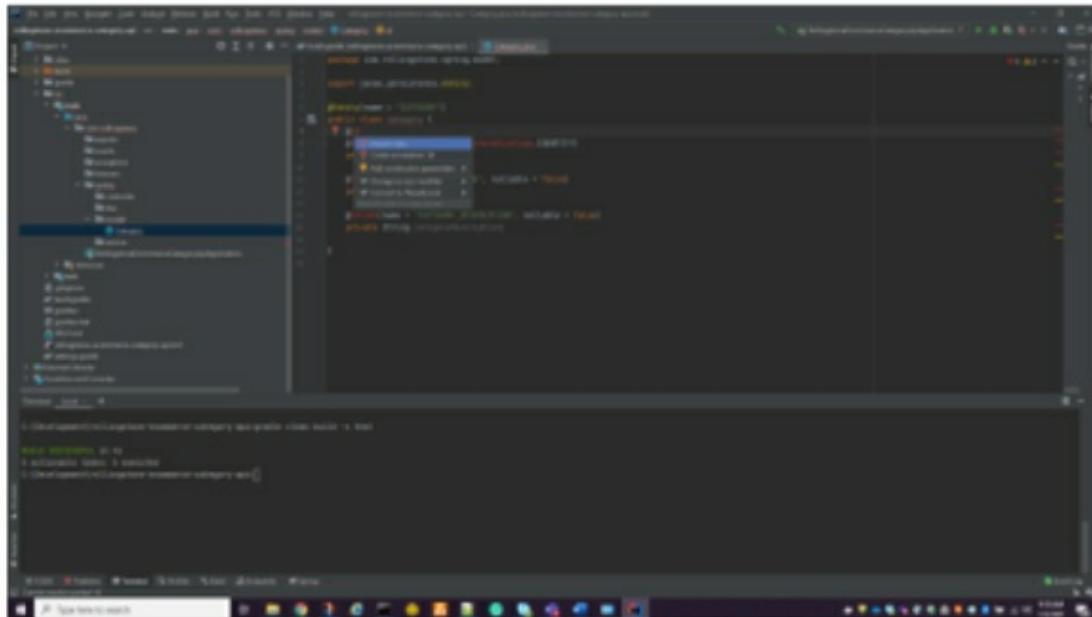
```
@Id
```

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

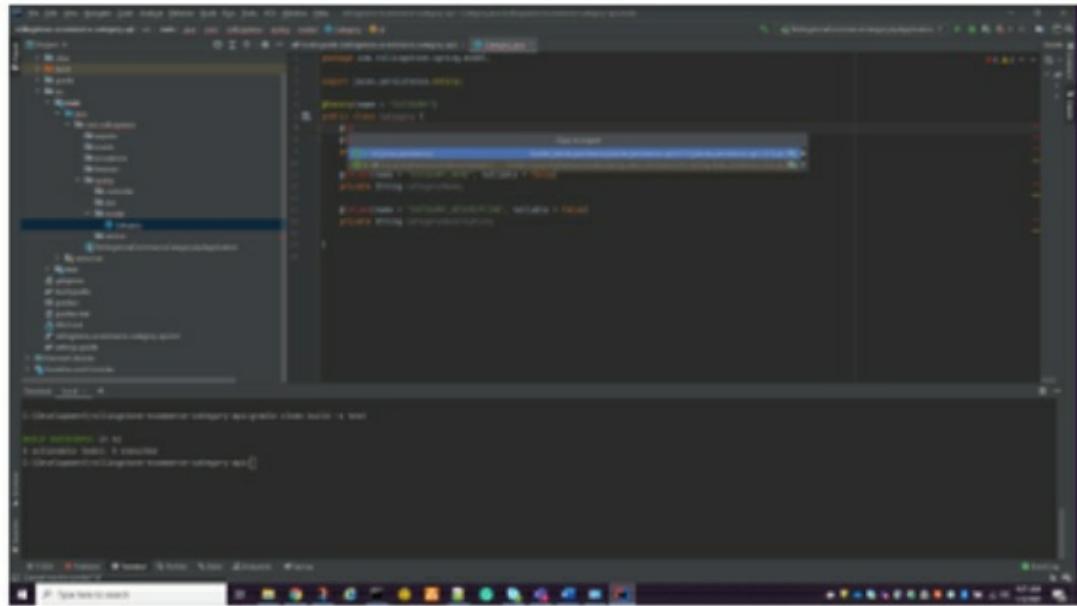
@Column(name = "CATEGORY_NAME", nullable = false)
private String categoryName;

@Column(name = "CATEGORY_DESCRIPTION", nullable =
false)
private String categoryDescription;
```

The IDE would not find many of these and would display a lot of Red. Let's resolve one by one. On the first one @Id press Alt+Enter and have the IDE display the following



Press Enter and the IDE would display the following

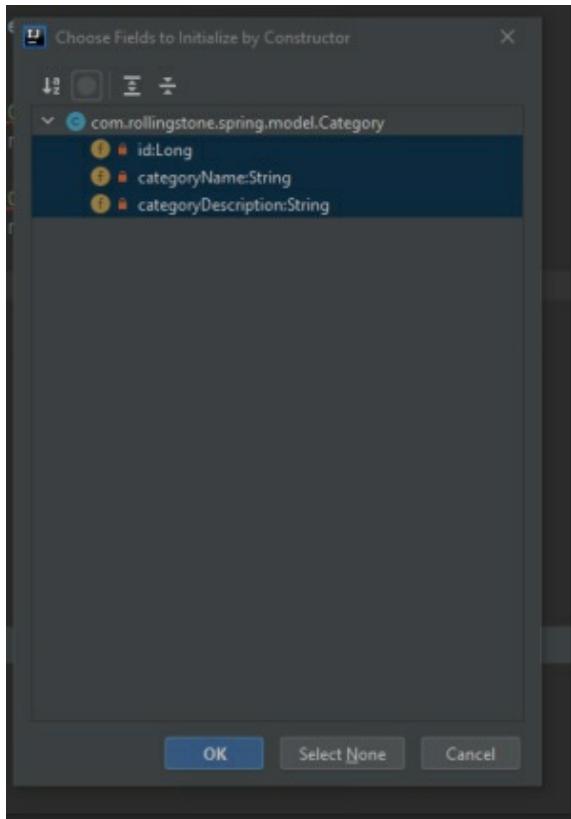


When we press Enter again, the IDE would add the import statement `import javax.persistence.Id;`

After we do this several times to resolve a few more missing imports, the IDE would optimize the code by replacing individual import statement with a global import `javax.persistence.*`; This is convenient as we are importing multiple classes from the `javax.persistence` package. Let's understand the remaining

- `@Id` → We are telling JPA that the `id` attribute would be our Primary Key in the MySQL Database
- `@GeneratedValue` →
- `@Column` → We are telling JPA that our `categoryName` java attribute would be bound to the `CATEGORY_NAME` Db column. Unless we have the `name` attribute specified JPA would assume the java attribute name is the name of the db. column

Let's generate a constructor by clicking Code → Generate → Constructor and choosing



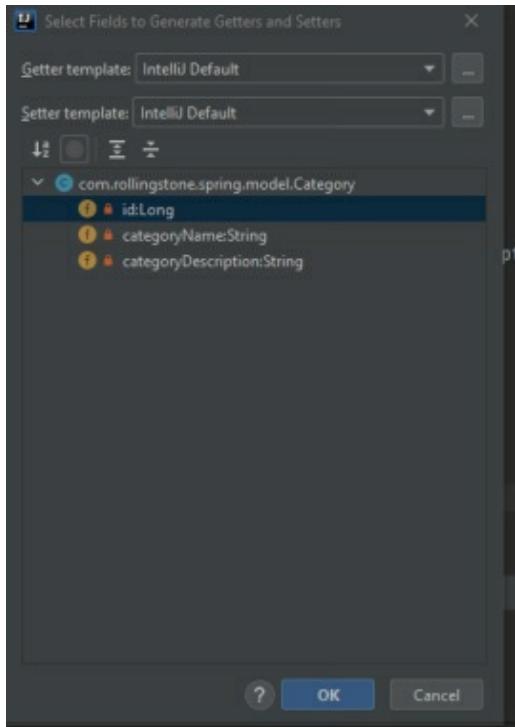
Let's follow the same Code → Generate → Constructor and this time deselect the id to generate a blank constructor.

Java Code now is below

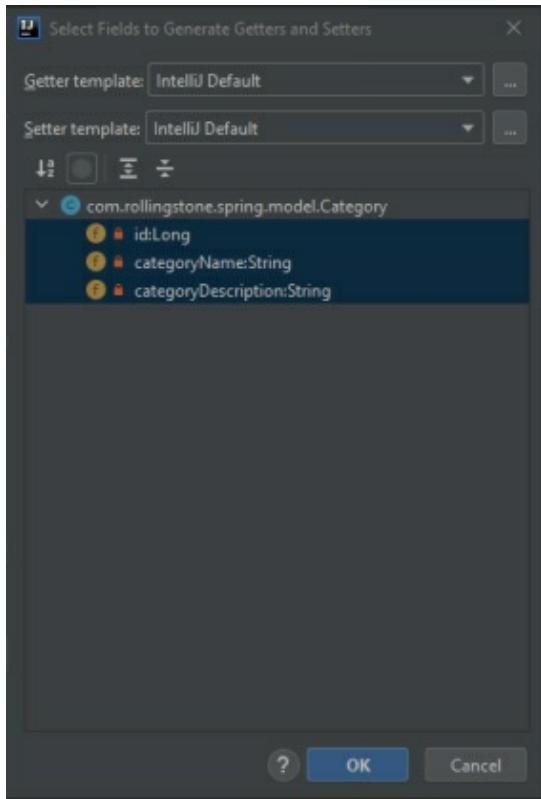
```
public Category(Long id, String categoryName, String categoryDescription)  
{  
    this.id = id;  
    this.categoryName = categoryName;  
    this.categoryDescription = categoryDescription;  
}  
  
public Category() {
```

}

Let's now choose Code → Generate → Getter and Setter to get

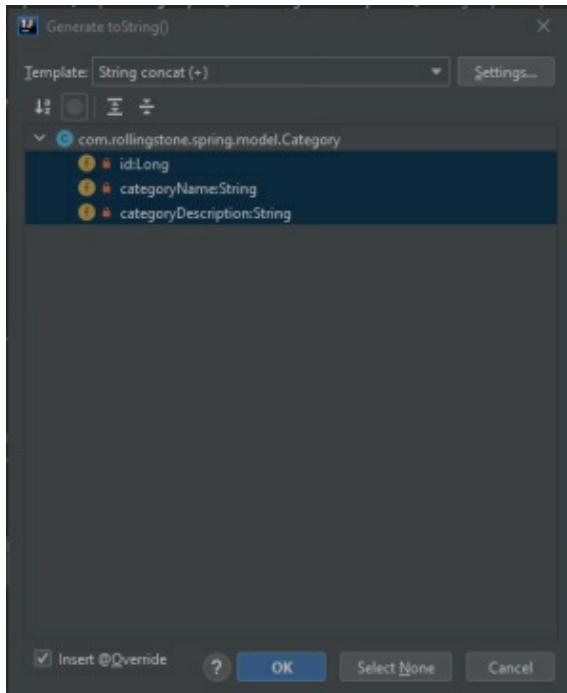


Select all three attributes and press OK



Let's now select Code → Generate → Equals and hashCode, follow the dialog , choosing default to generate the equals and hashCode methods

Finally let's choose Code → Generate → toString to generate the toString method



Following is the full source code of the class (Also available from Git)

```
package com.rollingstone.spring.model;

import javax.persistence.*;
import java.util.Objects;
@Entity(name = "rollingstone_category")
public class Category {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "CATEGORY_NAME", nullable = false)
    private String categoryName;

    @Column(name = "CATEGORY_DESCRIPTION", nullable = false)
    private String categoryDescription;
    public Category(Long id, String categoryName, String categoryDescription) {
        this.id = id;
        this.categoryName = categoryName;
```

```
    this.categoryDescription = categoryDescription;
}

public Category() {
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Category category = (Category) o;
    return Objects.equals(id, category.id) && Objects.equals(categoryName,
category.categoryName) && Objects.equals(categoryDescription, category.categoryDescription);
}

@Override
public int hashCode() {
    return Objects.hash(id, categoryName, categoryDescription);
}

@Override
public String toString() {
    return "Category{" +
        "id=" + id +
        ", categoryName='" + categoryName + '\'' +
        ", categoryDescription='" + categoryDescription + '\'' +
        '}';
}

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getCategoryName() {
```

```
    return categoryName;
}

public void setCategoryName(String categoryName) {
    this.categoryName = categoryName;
}

public String getCategoryDescription() {
    return categoryDescription;
}

public void setCategoryDescription(String categoryDescription) {
    this.categoryDescription = categoryDescription;
}

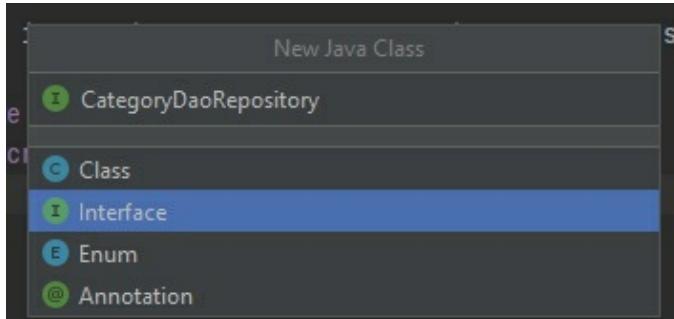
}
```

4.4 BUILDING THE DAO JPA INTERFACE

With the Model class done, lets now create the Dao Repository Interface

Right click the dao sub package under the com.rollingstone.spring and choose New → Java Class

Choose Interface and name it CategoryDaoRepository and press Enter



In the IDE window extend the Interface with
extends PagingAndSortingRepository<Category, Long>

Resolve the missing import through the Alt+Enter route. Choose our own Category model class and JPA's PagingAndSortingRepository interface.

Next create a new method as following

Page<Category> findAll(Pageable pageable);

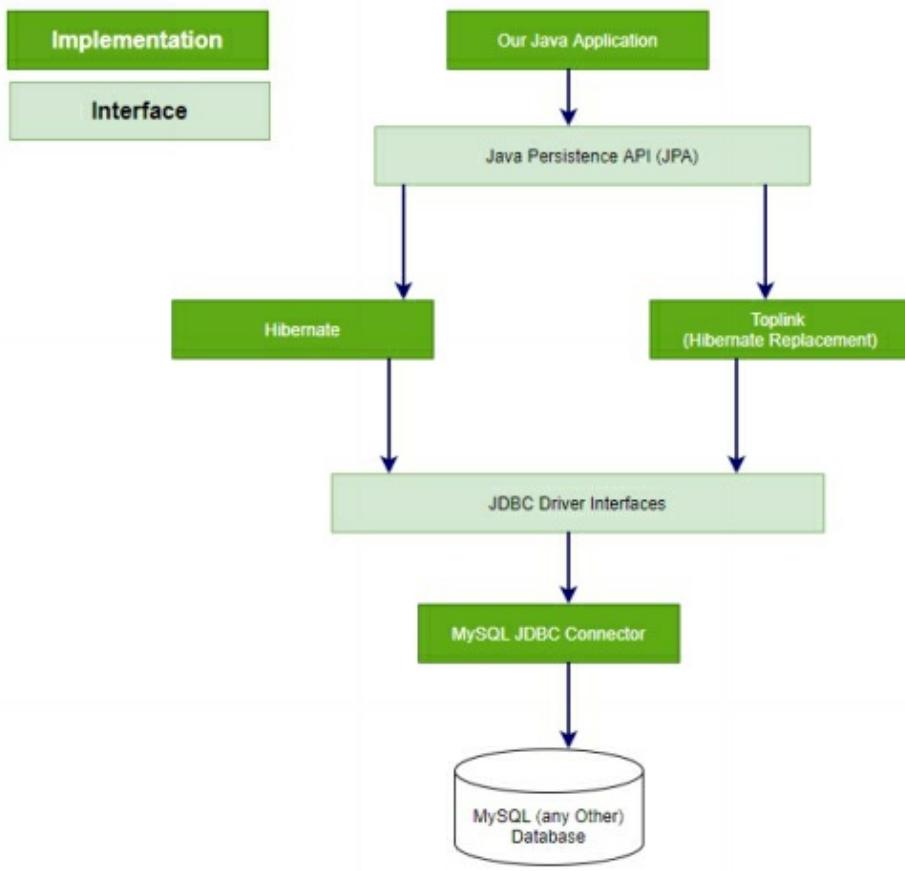
Resolve the missing imports by choosing
org.springframework.data.domain.Pageable and
org.springframework.data.domain.Page. Following is the full code

```
package com.rollingstone.spring.dao;  
import com.rollingstone.spring.model.Category;
```

```
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.repository.PagingAndSortingRepository;
public interface CategoryDaoRepository extends PagingAndSortingRepository<Category, Long>
{
    Page<Category> findAll(Pageable pageable);
}
```

A word on JPA which is interface and annotation driven. Long time back in 2001, we used to write out handwritten code to deal with SQL generated from our java code. There was no Hibernate or JPA at that time. Our classes were blotched, and it was lots of code. Now first Hibernate standardized Object Relational Mapping (ORM) and now JPA for several years make even Hibernate replaceable with any other competing JPA implementor.

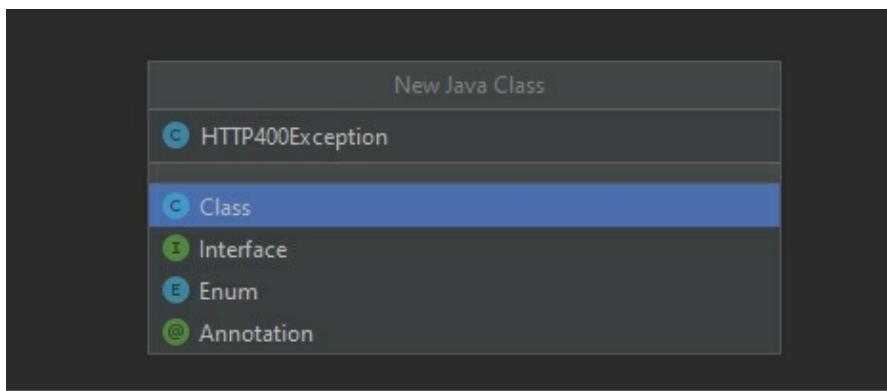
How does JPA do this? It allows us (as Hibernate also does) to annotate our model classes to identify the underlying database table, primary keys, Nullability and columns among other things. We can then just create lightweight interfaces extending various JPA CRUD interface to get Create-Retrieve-Update-Delete (CRUD) functionality out of the box free. A blank JPA interface without any method would be able to insert, update, delete and even a fidnAll method. If we need further customized finder methods, we can add them in the interface as we have done here. We can also add custom native SQL, but we will do later. How this all works can be seen in the diagram that follow



In other words, Database Code is now written a lot by JPA implementors like Hibernate and Toplink. Applications can be abstracted from the underlying JPA provider and a change in the JPA implementor would not break the rest of the application. The same existed for Databases since long as we know that we can change the underlying databases from MySQL to Oracle without also having to change our application code. This abstraction is provided by the JDBC interfaces and the respective Database JDBC Driver written by the database vendor.

4.5 BUILDING EXCEPTION CLASSES

Now let us generate a few custom Exception class we will use in our REST API application. The first one is HTTP400Exception. Right click the exceptions package under com.rollingstone and select New → Java Class. Enter HTTP400Exception



It is a simple class with no surprise. Following is the code

```
package com.rollingstone.exceptions;
public class HTTP400Exception extends RuntimeException {
    public HTTP400Exception() {
        super();
    }
    public HTTP400Exception(String message, Throwable cause) {
        super(message,cause);
    }
    public HTTP400Exception(String message) {
        super(message);
    }
    public HTTP400Exception(Throwable cause) {
        super(cause);
    }
}
```

Repeat the same process with another HTTP404Exception. Here is the full code for that one

```
package com.rollingstone.exceptions;
```

```

public class HTTP404Exception extends RuntimeException {

    public HTTP404Exception() {
        super();
    }

    public HTTP404Exception(String message, Throwable cause) {
        super(message,cause);
    }

    public HTTP404Exception(String message) {
        super(message);
    }

    public HTTP404Exception(Throwable cause) {
        super(cause);
    }

}

```

Finally, here is the full code for the last of the Exception classes **RestAPIExceptionInfo**. Generate it the same way

```

package com.rollingstone.exceptions;
public class RestAPIExceptionInfo {
    private final String message;
    private final String details;
    public RestAPIExceptionInfo() {
        message= null;
        details=null;
    }
    public RestAPIExceptionInfo(String message, String details) {
        this.message = message;
        this.details = details;
    }
    public String getMessage() {
        return message;
    }

    public String getDetails() {
        return details;
    }

}

```

4.6 BUILDING THE EVENT CLASS

An event in real life and in programming is quite similar. Celebrating New Year is an event, pressing each key on my keyboard is one, moving the mouse generates many events and Netflix asking me if I am still watching the movie after a certain time, is also an event. It is critical while learning Spring Boot that we learn how to generate, and handle or listen to programming events. We will treat creation of a new Category for example as an Event. Towards that end, let us generate our event holder class called CategoryEvent.

In the events class under com.rollingstone package, generate a new class called CategoryEvent. The class is normal java pojo except that we extended from a Spring Framework class called ApplicationEvent to tell Spring that it is our custom Event class for Category classes. The Event class is used as a data carrier when event generators generate events it instantiates this CategoryEvent class. Spring Event Handling framework carries this class instance to the Event Listener. Moral of the story is the event generator is written by us and Spring Framework ensures the instance is sent to the Event Listener which is also written by us. Following is the full code of the class

```
package com.rollingstone.events;
import org.springframework.context.ApplicationEvent;
import com.rollingstone.spring.model.Category;
public class CategoryEvent extends ApplicationEvent {
    private String eventType;
    private Category category;
    public String getEventType() {
        return eventType;
    }
    public void setEventType(String eventType) {
        this.eventType = eventType;
    }
    public Category getCategory() {
        return category;
    }
    public void setCategory(Category category) {
        this.category = category;
    }
}
```

```
public CategoryEvent(String eventType, Category category) {
    super(category);
    this.eventType = eventType;
    this.category = category;
}
@Override
public String toString() {
    return "CategoryEvent [eventType=" + eventType + ", category=" + category + "]";
}
}
```

4.7 BUILDING ASPECTS

One of the important requirements from the system/engineering department is the non-functional requirements like logging, security, maintainability, performance monitoring, and others. Let us consider logging. In our small REST APIs, if we start logging in each java method, our logging code may become scattered and tangled deeply into our application. Any slight change in the logging requirement then would have to implement in many classes/methods. That would become a maintenance nightmare. Besides, if we have written security code in each method, changing that security code would also be problematic. Imagine we want to monitor the time taken for performing a task. Writing such time measurement code exactly in hundreds of methods would also lead to code scattering and code tangling.

As these requirements are critical, we need a neat way to do all these. Aspect-Oriented Programming (AOP) is that neat way. When it first came, lots of ugly looking XML code was needed. Not anymore. Spring Boot made this a lot cleaner and annotation driven. However, before we delve into code, let's understand a few things. Aspects are possible due to the underlying event-driven framework capability of Spring Boot. Spring Boot can know/trap when a certain method is called. When we use the annotation `@Before` and identify a specific package, a particular class in that package, all methods in one Class, or all classes under a root package, Spring Boot can fire our custom Aspect method right before it calls our target method. Similarly, when we use `@AfterReturning` annotation and use similar qualifier when the annotation would be applicable, Spring calls our aspect method annotated with the `@AfterReturning` annotation, right after our target method finishes execution. There are others for throwing an exception, but let's deal with these two first.

One last thing is about byte code instrumentation. A lot of Java advanced Frameworks such as Hibernate, and Spring Boot depends on this process called byte code instrumentation. It is nothing but adding more bytecode to our custom classes and methods. Aspect-Oriented Programming in Spring

Boot uses several dependencies to gain this byte code instrumentation ability to add to our classes without us writing the code.

With that understood, let us generate our Aspect class. Right click the aspects package under com.rollingstone package and generate the RestControllerAspect class

Here is the full code and explanation right after that

```
package com.rollingstone.aspects;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import io.micrometer.core.instrument.Counter;
import io.micrometer.core.instrument.Metrics;
@Aspect
@Component
public class RestControllerAspect {
    private final Logger logger = LoggerFactory.getLogger("RestControllerAspect");
    @Autowired
    Counter createdCategoryCreationCounter;
    @Before("execution(public * com.rollingstone.spring.controller.*Controller.*(..))")
    public void generalAllMethodASpect() {
        logger.info("All Method Calls invoke this general aspect method");
    }
    @AfterReturning("execution(public *
com.rollingstone.spring.controller.*Controller.createCategory(..))")
    public void getsCalledOnCategorySave() {
        logger.info("This aspect is fired when the createCategory method of the controller is
called");
        createdCategoryCreationCounter.increment();
    }
}
```

We told Spring that it is an @Aspect and it also is a @Component. The @Aspect annotation will tell Spring Boot to instrument byte code. The @Component annotation tells Spring Boot Web Framework to load the class as part of the Spring Context.

Spring Boot 2 has added a nice monitoring framework called Micrometer to Actuator. We can use Counters and Gauges from Micrometer and we are using one of those here

@Autowired

```
Counter createdCategoryCreationCounter;
```

We named it and we can expect to see this in actuator /metrics endpoint which we will elaborate in a big.

Pay special attention to the following line

```
@Before("execution(public * com.rollingstone.spring.controller.*Controller.*(..))")
```

@Before is easy to understand as we clarified before. What is interesting is the “execution...” qualifier. We are telling Spring Boot to call the method under @Before annotation every time a method in a class that **ends** with **Controller** in the package com.rollingstone.spring.controller. We are also clarifying that the modified should only be public methods in the classes in that package with any return types. That means private and protected methods would not be impacted by the aspect method. What we are doing inside the method is also interesting. We are now having a single place to change our before logging code. If we want to add / modify / delete some logging code, we can now do that in a very limited centralize places with AOP.

The following annotation is similar but the method code executes after our target method completes

```
@AfterReturning("execution(public *  
com.rollingstone.spring.controller.*Controller.createCategory(..))")
```

We can also notice how we are using the Micrometer productCreatedCounter.increment(); to count events and report that first to Actuator and then to other monitoring systems.

4.8 BUILDING LISTENERS

We have generated our event class earlier. Now let us generate our listener class CategoryEventListener. Generate a new java class in the listener package under the com.rollingstone package and here is the full code

```
package com.rollingstone.listeners;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.Component;
import com.rollingstone.events.CategoryEvent;

@Component
public class CategoryEventListener {

    private final Logger log = LoggerFactory.getLogger(this.getClass());

    @EventListener
    public void onApplicationEvent(CategoryEvent categoryEvent) {
        log.info("Received Category Event : "+categoryEvent.getEventType());
        log.info("Received Category From Category Event
        :" +categoryEvent.getCategory().toString());
    }
}
```

Two things to notice here. First, we are identifying this class as a @Component for Spring Boot to identify and load into the Spring Context. Second with the @EventListener annotation and the specific type of the event listener method we are telling Spring Boot to call this onAppliucationEvent class with the instance of the CategoryEvent generated by the event sender / publisher.

4.9 BUILDING THE SERVICE

We would like to create an abstraction between our Customer facing Spring Web/REST Controller Class and the rest of the application. The CategoryService Java Interface would be that abstraction. We would like to replace the back-end implementation class with a different implementation of the same CategoryService interface and not have to change the Controller class at all. Let's create the java interface CategoryService in the service package under com.rollingstone.spring package. Here us the full code.

```
package com.rollingstone.spring.service;
import java.util.Optional;
import org.springframework.data.domain.Page;
import com.rollingstone.spring.model.Category;
public interface CategoryService {
    Category save(Category category);
    Optional<Category> get(long id);
    Page<Category> getCategorysByPage(Integer pageNumber, Integer pageSize);
    void update(long id, Category category);
    void delete(long id);
}
```

The Service interface will have one implementation for now. The Controller class would have a dependency of the Service interface and would call the corresponding methods through that interface.

Let's now generate the implementation class in the same package. Here is the full code

```
package com.rollingstone.spring.service;
import java.util.Optional;
import com.rollingstone.exceptions.HTTP400Exception;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;
```

```

import com.rollingstone.spring.dao.CategoryDaoRepository;
import com.rollingstone.spring.model.Category;
@Service
public class CategoryServiceImpl implements CategoryService {
    final static Logger logger = LoggerFactory.getLogger(CategoryServiceImpl.class);
    @Autowired
    private CategoryDaoRepository categoryDao;
    @Override
    public Category save(Category category) {
        try{
            return categoryDao.save(category);
        }
        catch (Exception e)
        {
            throw new HTTP400Exception(e.getMessage());
        }
    }
    @Override
    public Optional<Category> get(long id) {
        return categoryDao.findById(id);
    }
    @Override
    public Page<Category> getCategorysByPage(Integer pageNumber, Integer pageSize) {
        Pageable pageable = PageRequest.of(pageNumber, pageSize,
Sort.by("categoryName").descending());
        return categoryDao.findAll(pageable);
    }
    @Override
    public void update(long id, Category category) {
        categoryDao.save(category);
    }
    @Override
    public void delete(long id) {
        categoryDao.deleteById(id);
    }
}

```

As we can see the Service implements the CategoryService Interface and has a dependency CategoryDaoRepository which is injected by Spring Boot during startup. Please note how we are catching the Exception and throwing an HTTP400Exception. We want to implement a central exception handler in our next class. All exceptions thrown from throughout our application will be handled in a single central place giving us ease of maintenance.

4.10 BUILDING THE ABSTRACTCONTROLLER

It is good practice to keep all shared code between multiple classes in a single super class. We have a single controller now, but we will still follow the super class best practice. Let's generate an AbstractController class in the controller package of the com.rollingstone.controller package. Here is the full code with explanation right after it

```
package com.rollingstone.spring.controller;
import javax.servlet.http.HttpServletResponse;
import io.micrometer.core.instrument.MeterRegistry;
import io.micrometer.core.instrument.simple.SimpleMeterRegistry;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationEventPublisher;
import org.springframework.context.ApplicationEventPublisherAware;
import org.springframework.http.HttpStatus;
import org.springframework.http.converter.HttpMessageNotReadableException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.context.request.WebRequest;
import com.rollingstone.exceptions.HTTP400Exception;
import com.rollingstone.exceptions.HTTP404Exception;
import com.rollingstone.exceptions.RestAPIExceptionInfo;
import io.micrometer.core.instrument.Counter;
import io.micrometer.core.instrument.Metrics;

public abstract class AbstractController implements ApplicationEventPublisherAware {
    protected final Logger log = LoggerFactory.getLogger(this.getClass());
    protected ApplicationEventPublisher eventPublisher;
    protected static final String DEFAULT_PAGE_SIZE = "20";
    protected static final String DEFAULT_PAGE_NUMBER = "0";
    @Autowired
    Counter http400ExceptionCounter;
    @Autowired
    Counter http404ExceptionCounter;
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(HTTP400Exception.class)
    public @ResponseBody RestAPIExceptionInfo
```

```

handleBadRequestException(HTTP400Exception ex,
                           WebRequest request,
HttpServletResponse response)
{
    log.info("Received Bad Request Exception "+ex.getLocalizedMessage());
    http400ExceptionCounter.increment();
    return new RestAPIExceptionInfo(ex.getLocalizedMessage(), "The Request did not have
the correct parameters");
}
@ResponseStatus(HttpStatus.BAD_REQUEST)
@ExceptionHandler(HttpMessageNotReadableException.class)
public @ResponseBody RestAPIExceptionInfo
handleBadRequestExceptionForJsonBody(HttpMessageNotReadableException ex,
                                      WebRequest request,
HttpServletResponse response)
{
    log.info("Received Bad Request Exception "+ex.getLocalizedMessage());
    http400ExceptionCounter.increment();
    return new RestAPIExceptionInfo("JSON Parse Error", "The Request did not have the
correct json body");
}
@ResponseStatus(HttpStatus.NOT_FOUND)
@ExceptionHandler(HTTP404Exception.class)
public @ResponseBody RestAPIExceptionInfo
handleResourceNotFoundException(HTTP404Exception ex,
                               WebRequest request,
HttpServletResponse response)
{
    log.info("Received Resource Not Found Exception "+ex.getLocalizedMessage());
    http404ExceptionCounter.increment();
    return new RestAPIExceptionInfo(ex.getLocalizedMessage(), "The Requested Resource
was not found");
}
@Override
public void setApplicationEventPublisher(ApplicationEventPublisher eventPublisher) {
    this.eventPublisher = eventPublisher;
}
public static <T> T checkResourceFound(final T resource) {
    if (resource == null) {
        throw new HTTP404Exception("Resource Not Found");
    }
    return resource;
}
}

```

Let's see and understand the code in detail. First it implements ApplicationEventPublisherAware Interface from the Spring Framework We are telling Spring Boot by implementing this interface to call our

setApplicationEventPublisher method during Startup with an instance of the ApplicationEventPublisher class from Spring Boot internal framework. We want to capture that instance and use it later to publish events to be captured by Spring Boot and send the events to our listeners.

```
protected ApplicationEventPublisher eventPublisher;
```

This is our instance level attribute to capture the ApplicationEventPublisher. Any concrete implementation of the abstract class would inherit the attribute as it is protected.

```
protected static final String DEFAULT_PAGE_SIZE = "20";
```

```
protected static final String DEFAULT_PAGE_NUMBER = "0";
```

These are default values we would use to call our pageable JPA method

```
@Autowired  
Counter http400ExceptionCounter;  
@Autowired  
Counter http404ExceptionCounter;
```

We came to know how Micrometer works with Actuator and we are using two new counters to count different exceptions that we may have. We will configure the two instances above in a separate configuration class soon.

The following two methods are related to a concept called Central Exception Handling

```
@ResponseStatus(HttpStatus.BAD_REQUEST)
@ExceptionHandler(HTTP400Exception.class)
public @ResponseBody RestAPIExceptionInfo
handleBadRequestException(HTTP400Exception ex,
WebRequest request,
HttpServletResponse response)
{
    log.info("Received Bad Request Exception "+ex.getLocalizedMessage());
    http400ExceptionCounter.increment();
    return new RestAPIExceptionInfo(ex.getLocalizedMessage(), "The Request did not have
the correct parameters");
}
@ResponseStatus(HttpStatus.BAD_REQUEST)
@ExceptionHandler(HttpMessageNotReadableException.class)
public @ResponseBody RestAPIExceptionInfo
handleBadRequestExceptionForJsonBody(HttpMessageNotReadableException ex,
WebRequest request,
```

```

    HttpServletResponse response)
{
    log.info("Received Bad Request Exception "+ex.getLocalizedMessage());
    http400ExceptionCounter.increment();
    return new RestAPIExceptionInfo("JSON Parse Error", "The Request did not have the
correct json body");
}

@ResponseStatus(HttpStatus.NOT_FOUND)
@ExceptionHandler(HTTP404Exception.class)
public @ResponseBody RestAPIExceptionInfo
handleResourceNotFoundException(HTTP404Exception ex,
                               WebRequest request,
    HttpServletResponse response)
{
    log.info("Received Resource Not Found Exception "+ex.getLocalizedMessage());
    http404ExceptionCounter.increment();
    return new RestAPIExceptionInfo(ex.getLocalizedMessage(), "The Requested Resource
was not found");
}

```

Central Exception Handling has a lot to do with ease of maintenance, the challenges we saw while discussing Aspect-Oriented programming. We could have written them inside the Aspect class itself, but we need to generate HTTP error code/message back to our client. That is why these methods are in an HTTP specific class and have all Spring Web annotations such as `@ResponseStatus(HttpStatus.BAD_REQUEST)` `@ResponseBody`. With the `@ExceptionHandler` annotation, we tell Spring Boot to call this method central to our abstract class, if deep inside our application code, an `HTTP404Exception` exception is thrown by any code. We are also telling Spring Boot to associate a `WebRequest` and a `HttpServletResponse` instance in the method. We can get the `HttpServletRequest` instance related to this web request from the `WebRequest` class. While the `RestAPIExceptionInfo` is our formal returnable POJO to hide the exception classes' details, we are using the counters to count the number of exceptions. We can see the metrics when we investigate the actuator later.

Finally, we will use the common method from multiple places in the concrete controller

```

public static <T> T checkResourceFound(final T resource) {
    if (resource == null) {
        throw new HTTP404Exception("Resource Not Found");
    }
    return resource;
}

```

4.11 BUILDING THE CATEGORYCONTROLLER

With all our backend ready, lets create the final class that is our CategoryController. Create a new class in the same controller package. Here is the full code with explanation right after

```
package com.rollingstone.spring.controller;

import java.util.Optional;
import org.springframework.data.domain.Page;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;
import com.rollingstone.events.CategoryEvent;
import com.rollingstone.spring.model.Category;
import com.rollingstone.spring.service.CategoryService;

@RestController
public class CategoryController extends AbstractController {
    private CategoryService CategoryService;
    public CategoryController(CategoryService CategoryService) {
        this.CategoryService = CategoryService;
    }

    /*---Add new Category---*/
    @PostMapping("/category")
    public ResponseEntity<?> createCategory(@RequestBody Category Category) {
        Category savedCategory = CategoryService.save(Category);
        CategoryEvent CategoryCreatedEvent = new CategoryEvent("One Category is created",
        savedCategory);
        eventPublisher.publishEvent(CategoryCreatedEvent);
        return ResponseEntity.ok().body("New Category has been saved with ID: " +
        savedCategory.getId());
    }
}
```

```

    }
    /*---Get a Category by id---*/
    @GetMapping("/category/{id}")
    @ResponseBody
    public Category getCategory(@PathVariable("id") long id) {
        Optional<Category> returnedCategory = CategoryService.get(id);
        Category Category = returnedCategory.get();

        CategoryEvent CategoryCreatedEvent = new CategoryEvent("One Category is retrieved",
        Category);
        eventPublisher.publishEvent(CategoryCreatedEvent);
        return Category;
    }
    /*---get all Category---*/
    @GetMapping("/category")
    public @ResponseBody Page<Category> getCategoriesByPage(
        @RequestParam(value="pagenumber", required=true, defaultValue="0") Integer
    pageNumber,
        @RequestParam(value="pagesize", required=true, defaultValue="20") Integer
    pageSize) {
        Page<Category> pagedCategorys = CategoryService.getCategorysByPage(pageNumber,
    pageSize);
        return pagedCategorys;
    }
    /*---Update a Category by id---*/
    @PutMapping("/category/{id}")
    public ResponseEntity<?> updateCategory(@PathVariable("id") long id, @RequestBody
    Category Category) {
        checkResourceFound(this.CategoryService.get(id));
        CategoryService.update(id, Category);
        return ResponseEntity.ok().body("Category has been updated successfully.");
    }
    /*---Delete a Category by id---*/
    @DeleteMapping("/category/{id}")
    public ResponseEntity<?> deleteCategory(@PathVariable("id") long id) {
        checkResourceFound(this.CategoryService.get(id));
        CategoryService.delete(id);
        return ResponseEntity.ok().body("Category has been deleted successfully.");
    }
}
}

```

Explanation of the Code above:

@RestController

Tells Spring Boot that class would be reachable by HTTP REST Calls and it would have routed the Http calls to the corresponding Java methods that matches the HTTP Verbs and the request parameters and path variables

```

private CategoryService CategoryService;
public CategoryController(CategoryService CategoryService) {
    this.CategoryService = CategoryService;
}

```

```
}
```

The two above identifies the instance attribute for the Service interface we talked about earlier and the constructor to hold that. Notice that if we can create a constructor, we do not need to use @Autowired anymore.

```
/*---Add new Category---*/
@PostMapping("/category")
public ResponseEntity<?> createCategory(@RequestBody Category Category) {
    Category savedCategory = CategoryService.save(Category);
    CategoryEvent CategoryCreatedEvent = new CategoryEvent("One Category is created",
    savedCategory);
    eventPublisher.publishEvent(CategoryCreatedEvent);
    return ResponseEntity.ok().body("New Category has been saved with ID: " +
    savedCategory.getId());
}
```

@PostMapping is the Spring Boot Web annotation to identify the java method that would response when a HTTP POST Method call comes with the /category with a matching port number. @ResponseEntity and @RequestBody are standard Spring Boot Web framework annotations to deal with the response Json serialization and deserialization by jackson library we included in our build.gradle file. Spring would take the json request body and convert that into a Java pojo before calling our method. Please keep in mind that before this method gets called, we will see the aspect method called by Spring.

Inside we are calling the categoryService.save to persist the new category. We are then generating a new CategoryEvent with a proper message and the newly created instance. Mind you, the newly created instance as it is returned to use by JPA would also contain the new ID of the database record. The third line publishes the event using the eventPublisher we captured in our super class. Finally, the last line we are sending a response back with HTTP 200 status code and the new ID of the generated category

```
/*---Get a Category by id---*/
@GetMapping("/category/{id}")
@ResponseBody
public Category getCategory(@PathVariable("id") long id) {
    Optional<Category> returnedCategory = CategoryService.get(id);
    Category Category = returnedCategory.get();

    CategoryEvent CategoryCreatedEvent = new CategoryEvent("One Category is retrieved",
    Category);
```

```

        eventPublisher.publishEvent(CategoryCreatedEvent);
        return Category;
    }
}

```

The method above is for responding to the HTTP GET call with a category id. If the port number that this application listens to is 8081, then a REST call like `http://localhost:8080/category/1` would make Spring Boot to call this java method `getCategory`. Mind you the `/{id}` is a path variable not a request parameter. Request parameters comes right after the `?` mark of the HTTP URL while path variables are part of the URL itself. Both are represented by the `@PathVariable` and `@RequestParam` annotations respectively. The rest of the code is easy as we are using the service class to get the instance related to the ID received. We are also publishing an event like we did earlier.

```

/*---get all Category---*/
@GetMapping("/category")
public @ResponseBody Page<Category> getCategoriesByPage(
    @RequestParam(value="pagenumber", required=true, defaultValue="0") Integer
pageNumber,
    @RequestParam(value="pagesize", required=true, defaultValue="20") Integer
pageSize) {
    Page<Category> pagedCategorys = CategoryService.getCategorysByPage(pageNumber,
pageSize);
    return pagedCategorys;
}

```

The method above is the full version of the HTTP GET. If the port number that this application listens to is 8081, then a REST call like `http://localhost:8080/category` would make Spring Boot to call this java method `getCategoriesByPage`. We are having default values for the page size and the per page number of records. The service method we are calling would however invoke our custom method in the Dao interface. The code i.e. SQL etc. would still be written by Spring Boot Data JPA, mind you.

```

/*---Update a Category by id---*/
@PutMapping("/category/{id}")
public ResponseEntity<?> updateCategory(@PathVariable("id") long id, @RequestBody
Category Category) {
    checkResourceFound(this.CategoryService.get(id));
    CategoryService.update(id, Category);
    return ResponseEntity.ok().body("Category has been updated successfully.");
}

/*---Delete a Category by id---*/
@DeleteMapping("/category/{id}")
public ResponseEntity<?> deleteCategory(@PathVariable("id") long id) {
}

```

```
    checkResourceFound(this.CategoryService.get(id));
    CategoryService.delete(id);
    return ResponseEntity.ok().body("Category has been deleted successfully.");
}
```

The above two represent the rest of the CRUD functionality for HTTP PUT (Update) and HTTP DELETE Verbs. Both has a Path Variable and uses the method checkResourceFound we in the abstract class.

4.12 GENERATE THE CONFIGURATION

We would like to separate the Code for Spring Boot Configuration. Create a new class in the config package and name it CategoryMetricsConfiguration. Following is the full code with explanation

```
package com.rollingstone.config;
import io.micrometer.core.instrument.Counter;
import io.micrometer.core.instrument.MeterRegistry;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.client.HttpComponentsClientHttpRequestFactory;
import org.springframework.web.client.RestTemplate;
import java.time.Duration;
@Configuration
public class CategoryMetricsConfiguration {
    @Bean
    public Counter createdCategoryCreationCounter(MeterRegistry registry) {
        return Counter
            .builder("com.rollingstone.category.created")
            .description("Number of Categories Created")
            .tags("environment", "production")
            .register(registry);
    }
    @Bean
    public Counter http400ExceptionCounter(MeterRegistry registry) {
        return Counter
            .builder("com.rollingstone.CategoryController.HTTP400")
            .description("How many HTTP Bad Request HTTP 400 Requests have been
received since start time of this instance.")
            .tags("environment", "production")
            .register(registry);
    }
    @Bean
    public Counter http404ExceptionCounter(MeterRegistry registry) {
        return Counter
            .builder("com.rollingstone.CategoryController.HTTP404")
            .description("How many HTTP Resource Not Found HTTP 404 Requests have
been received since start time of this instance. ")}
```

```

        .tags("environment", "production")
        .register(registry);
    }
    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder builder) {
        return builder
            .setConnectTimeout(Duration.ofMillis(3000))
            .setReadTimeout(Duration.ofMillis(3000))
            .build();
    }
}

```

The `@Configuration` annotation tells Spring that this class is to be used only during the startup[time to create new Spring Boot Beans in the Spring Context. The name of the methods with `@Bean` annotation becomes globally available shared instance variables to be `@Autowired` elsewhere in other classes. As we can see Spring Boot Actuator would self-create an instance of the `MeterRegistry` and send it to the java method to resolve the dependency. Inside we are creating a Micrometer Counter using the builder patterns. We are giving it a name, a description, a few key value pair as tags and then registering it in the Registry that we received. These settings are deeply connected how we can collect Spring Boot Actuator metrics and release them to a host of monitoring systems through Prometheus, AWS CloudWatch, Azure Monitoring, Dynatrace, Datadog and so on. More on that later. The other two methods are similar.

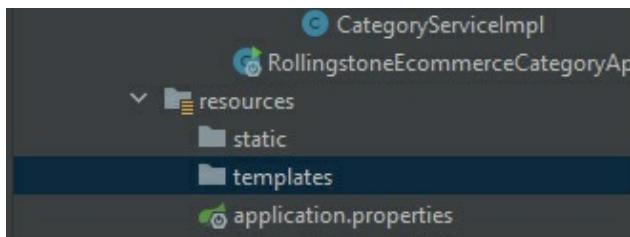
4.13 BUILDING THE SPRING BOOT MAIN CLASS

When we generated the Spring Boot App using the IDE, it generated a default Spring Boot Application starter class. This is the class that starts the inbuilt Tomcat Servlet container. We are good for now but in near future we will modify this class a little bit to add some more functionality. Here is the full code for now

```
package com.rollingstone;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class RollingstoneEcommerceCategoryApiApplication {
    public static void main(String[] args) {
        SpringApplication.run(RollingstoneEcommerceCategoryApiApplication.class, args);
    }
}
```

4.14 SETTING THE SPRING CONFIG FILES

We can delete the following static and template folders as we would not deal with HTML code in this application



Let's create a new file named as application.yaml under the resources folder and paste the following code in that file

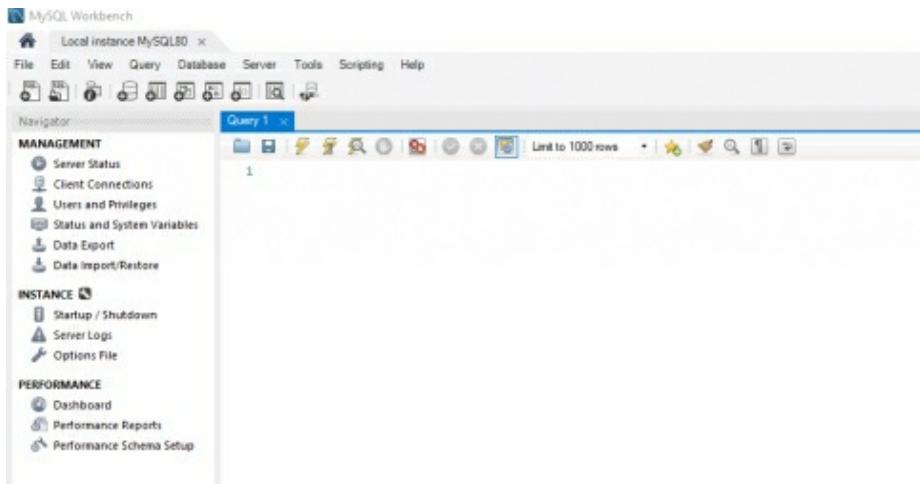
```
server:
  port: 8092
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/rs_ecommerce
    username: root
    password: root
    tomcat.max-wait: 20000
    tomcat.max-active: 50
    tomcat.max-idle: 20
    tomcat.min-idle: 15
    validationQuery: SELECT 1
  jpa:
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL5InnoDBDialect
  hibernate:
    ddl-auto: update
management:
  server:
    port: 8093
  endpoints:
    web:
      exposure:
```

```
include: "*"
endpoint:
  health:
    show-details: "always"
```

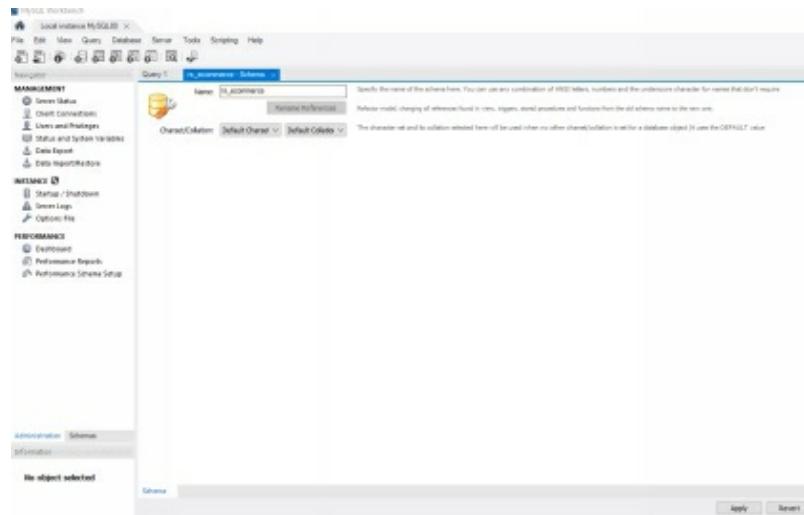
A first, we are modifying the default server port 8080 as we will run more than one Microservice and we need custom ports for them. The first part till management is standard properties to identify the MySQL connectivity parameters, the management part is the actuator configuration. We are telling Spring Boot that actuator would be available on the 8093 port. All endpoints would be exposed, and the health endpoint should provide details. We will see the result of this configuration when we test the application using a client.

4.15 CREATING THE MYSQL DATABASE AND TABLES

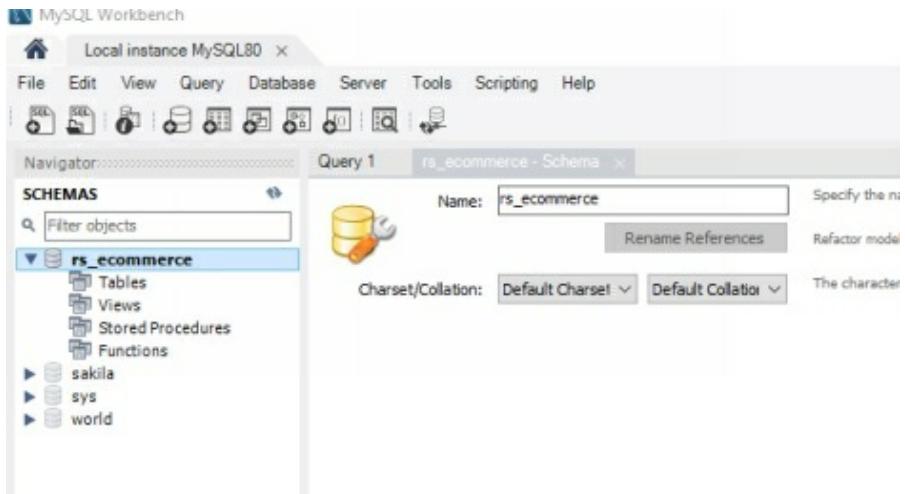
Your MySQL local database instance should be running. Start your MySQL Workbench client and connect to the local database instance with your root account and password.



Create a new schema called rs_ecommerce : CREATE SCHEMA `rs_ecommerce` ;

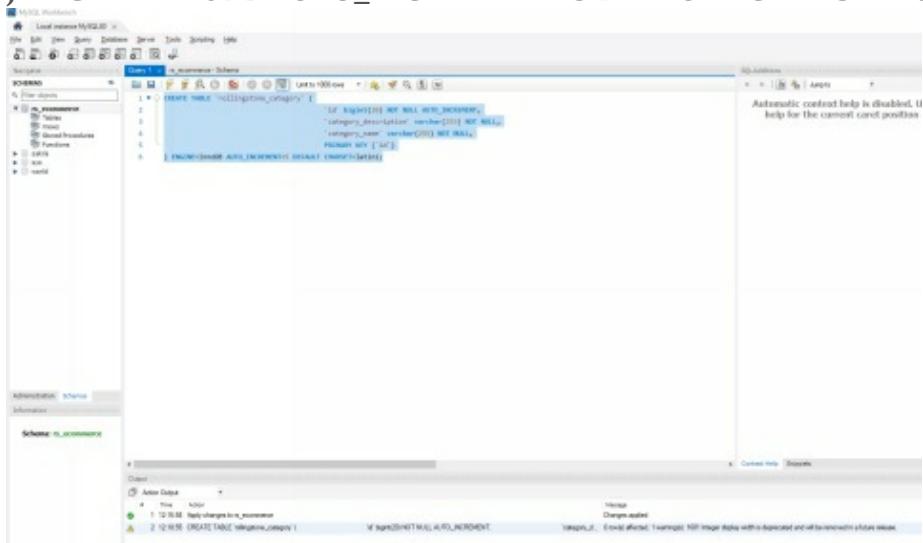


Click on the Schemas tab and Right click on the new schema to choose Set As Default



Create a new Table using the following ddl

```
CREATE TABLE `rollingstone_category` (
    `id` bigint(20) NOT NULL AUTO_INCREMENT,
    `category_description` varchar(255) NOT NULL,
    `category_name` varchar(255) NOT NULL,
    PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=latin1;
```



Create a few records in the table with the following inserts

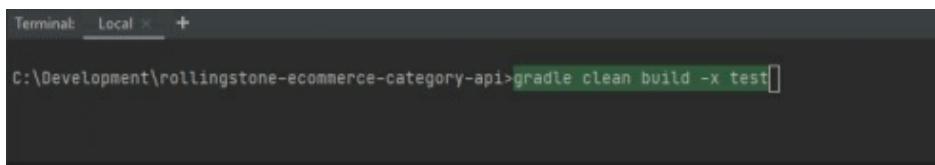
```
INSERT INTO `rollingstone_category`
(
    `category_description`,
    `category_name`)
VALUES
(
    'Food',
    'Food');
INSERT INTO `rollingstone_category`
```

```
(`category_description`,
`category_name`)
VALUES
(
  'Oranges',
  'Oranges');
INSERT INTO `rollingstone_category`
(
  `category_description`,
  `category_name`)
VALUES
(
  'Electronics',
  'Electronics');
INSERT INTO `rollingstone_category`
(
  `category_description`,
  `category_name`)
VALUES
(
  'Television',
  'Television');
```

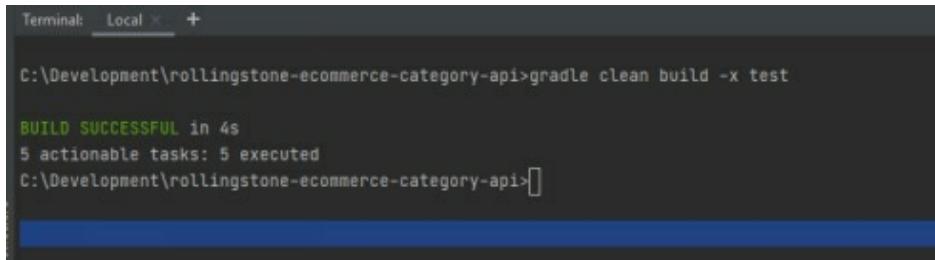
4.16 BUILDING THE JAR

Our coding is complete and time to test the application. Build the application jar with the following command in a terminal window within the IDE

gradle clean build -x test



```
Terminal: Local +  
C:\Development\rollingstone-commerce-category-api>gradle clean build -x test[]
```

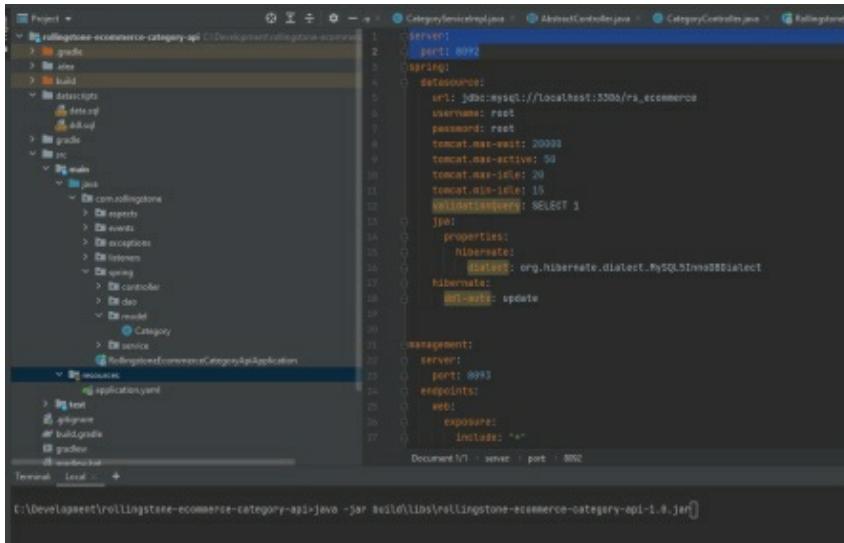


```
Terminal: Local +  
C:\Development\rollingstone-commerce-category-api>gradle clean build -x test  
  
BUILD SUCCESSFUL in 4s  
5 actionable tasks: 5 executed  
C:\Development\rollingstone-commerce-category-api>[]
```

4.16.1 RUNNING THE JAR

Gradle keeps the executable jar file under build/libs. Run the application with the following command in a terminal window (Change the slashes if you are using a Mac)

```
java -jar build\libs\rollingstone-eCommerce-category-api-1.0.jar
```



The screenshot shows a Java IDE interface with the following details:

- Project Tree:** Shows the project structure: rollingstone-eCommerce-category-api, build, gradle, build.gradle, buildscript, dependencies, and sourceSets.
- Code Editor:** Displays a configuration file (likely application.yml) with the following content:

```
server:
  port: 8992
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/r3_ecommerce
    username: root
    password: root
    tomcat.max-wait: 20000
    tomcat.max-active: 50
    tomcat.max-idle: 20
    tomcat.min-idle: 15
    validationQuery: SELECT 1
    jpa:
      properties:
        hibernate:
          dialect: org.hibernate.dialect.MySQLInnodbDialect
      hibernate:
        ddl-auto: update
  management:
    server:
      port: 8993
      endpoints:
        web:
          exposure:
            include: '*'
```
- Terminal:** Shows the command being run in the terminal: `t:\Development\rollingstone-eCommerce-category-api>java -jar build\libs\rollingstone-eCommerce-category-api-1.0.jar`.

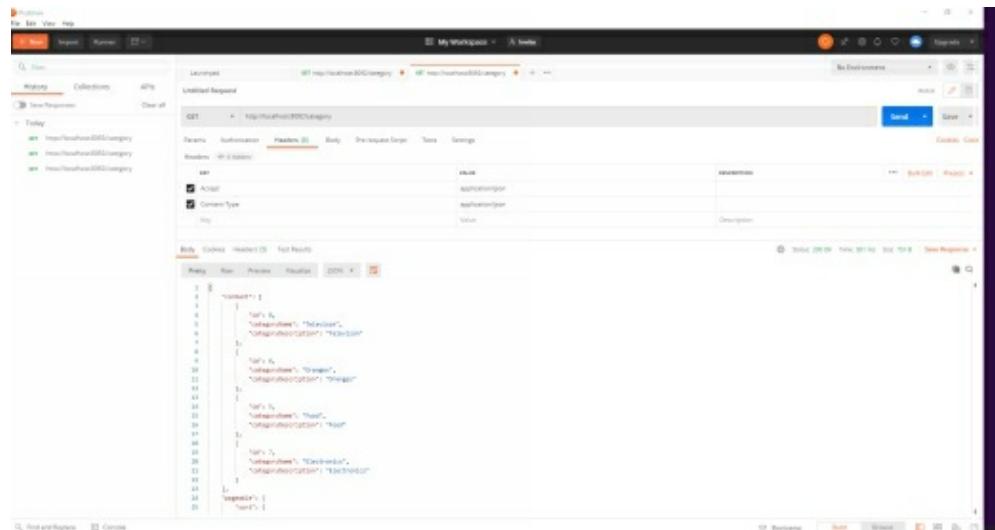
4.17 TESTING THE APPLICATION LOCALLY

In the root of the application codebase, a helper file is there with the name WebRESTTestGuide.txt under the folder WebTestsDOC

Let's open the Postman REST Client.

Here is how the GET All Category Call Looks. Details below

- Method: GET
- URL : <http://localhost:8092/category>
- Headers
 - Accept : application/json
 - Content-Type: application/json
- Click Send
- Check the Response



The screenshot shows the Postman interface with the following details:

- URL:** GET http://localhost:8092/category
- Headers:** Accept: application/json, Content-Type: application/json
- Response Body (JSON):**

```
[{"id": 1, "name": "Electronics", "description": "Electronics"}, {"id": 2, "name": "Clothing", "description": "Clothing"}, {"id": 3, "name": "Books", "description": "Books"}, {"id": 4, "name": "Sports", "description": "Sports"}]
```

Let's try a POST with the details

- Method: POST

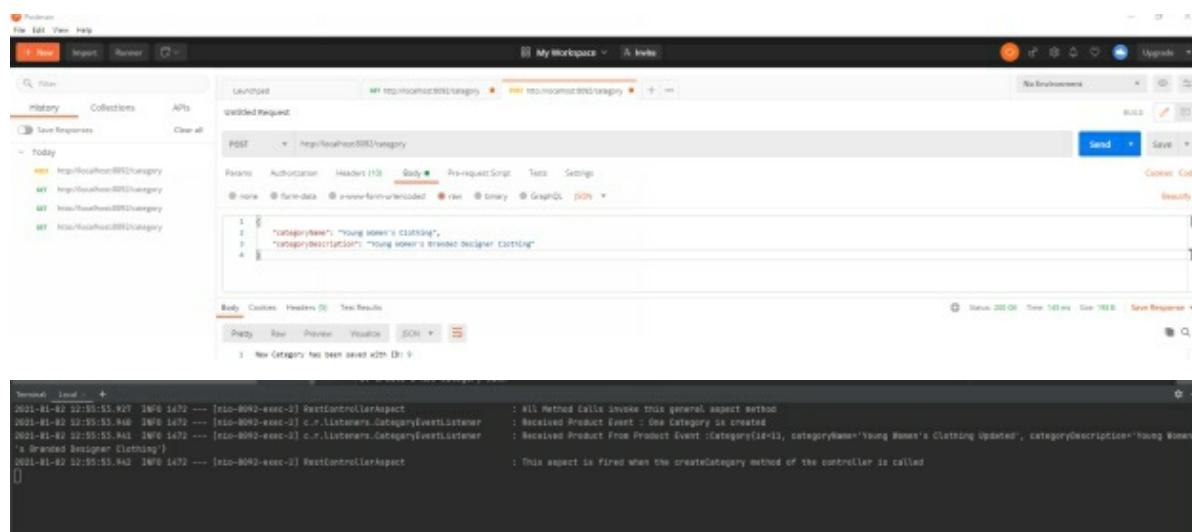
- URL : <http://localhost:8092/category>

- Headers
 - Accept : application/json
 - Content-Type: application/json

- Body

```
{
  "categoryName": "Young Women's Clothing",
  "categoryDescription": "Young Women's Branded Designer Clothing"
}
```

- Click Send
-
- Check the Response



Let's verify if our aspects, listeners etc. are working or not. Right click on your IDEA terminal window and choose clear bugger to delete the existing log messages. The first message in terminal window we see is

All Method Calls invoke this general aspect method

Check our Aspect class → `@Before` method to verify the message

```

import org.springframework.stereotype.Component;

import io.micrometer.core.instrument.Counter;
import io.micrometer.core.instrument.Metrics;

@Aspect
@Component
public class RestControllerAspect {

    private final Logger logger = LoggerFactory.getLogger( name "RestControllerAspect");

    Counter productCreatedCounter = Metrics.counter( name: "com.rollingstone.category.created");

    @Before("execution(public * com.rollingstone.spring.controller.*Controller.*(..))")
    public void generalAllMethodAspect() {
        logger.info("All Method Calls invoke this general aspect method");
    }

    @AfterReturning("execution(public * com.rollingstone.spring.controller.*Controller.createCategory(..))")
    public void getsCalledOnProductSave() {
        logger.info("This aspect is fired when the createCategory method of the controller is called");
        productCreatedCounter.increment();
    }
}

```

The second message we see in the terminal is

Received Category Event: One Category is created

First check the controller method

```

/*---Add new Category---*/
@PostMapping(@"/category")
public ResponseEntity<?> createCategory(@RequestBody Category Category) {
    Category savedCategory = CategoryService.save(Category);
    CategoryEvent CategoryCreatedEvent = new CategoryEvent( eventType: "One Category is created", savedCategory);
    eventPublisher.publishEvent(CategoryCreatedEvent);
    return ResponseEntity.ok().body("New Category has been saved with ID:" + savedCategory.getId());
}

```

Then check the CategoryEvent class

```

package com.rollingstone.events;

import org.springframework.context.ApplicationEvent;

import com.rollingstone.spring.model.Category;

public class CategoryEvent extends ApplicationEvent {

    private String eventType;
    private Category category;
    public String getEventType() {
        return eventType;
    }
    public void setEventType(String eventType) {
        this.eventType = eventType;
    }
    public Category getCategory() {
        return category;
    }
    public void setCategory(Category category) {
        this.category = category;
    }
    public CategoryEvent(String eventType, Category category) {
        super(category);
        this.eventType = eventType;
        this.category = category;
    }

    @Override
    public String toString() {
        return "CategoryEvent [eventType=" + eventType + ", category=" + category + "]";
    }
}

```

Finally check the CategoryEventListener class to match, verify

```

package com.rollingstone.listeners;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.Component;

import com.rollingstone.events.CategoryEvent;

@Component
public class CategoryEventListener {

    private final Logger log = LoggerFactory.getLogger(this.getClass());

    @EventListener
    public void onApplicationEvent(CategoryEvent categoryEvent) {
        log.info("Received Category Event : "+categoryEvent.getEventType());
        log.info("Received Category From Category Event :" +categoryEvent.getCategory().toString());
    }
}

```

This message is coming from the Event handler

Received Category From Category Event :Category{id=13,
categoryName='Young Women's Clothing Updated',
categoryDescription='Young Women's Branded Designer Clothing'}

Finally, the @AfterReturning aspect class method is printing the last line

```

@AfterReturning("execution(public * com.rollingstone.spring.controller.*Controller.createCategory(..))")
public void getsCalledOnProductSave() {
    logger.info("This aspect is fired when the createCategory method of the controller is called");
    productCreatedCounter.increment();
}

```

Let's verify with a GET

The screenshot shows the Postman interface with the following details:

- Request URL:** GET http://localhost:8092/category
- Body (JSON):**

```

1
2   "categoryName": "Young Women's Clothing",
3   "categoryDescription": "Young Women's Branded Designer Clothing"
4

```

- Response (Pretty):**

```

1
2   "content": [
3     {
4       "id": 9,
5       "categoryName": "Young Women's Clothing",
6       "categoryDescription": "Young Women's Branded Designer Clothing"
7     },
8     {
9       "id": 8,
10      "categoryName": "Televison",
11      "categoryDescription": "Televison"
12    },
13    {
14      "id": 6,
15      "categoryName": "Oranges",
16      "categoryDescription": "Oranges"
17    },
18    {
19      "id": 5,
20      "categoryName": "Food",
21      "categoryDescription": "Food"
22    },
23    {
24      "id": 7,
25      "categoryName": "Electronics",

```

Let us Update an existing category with

- Method: PUT
- URL : <http://localhost:8092/category/9>
- Headers

- Accept : application/json
- Content-Type: application/json
- Body

```
{
  "id": 9,
  "categoryName": "Young Women's Clothing",
  "categoryDescription": "Young Women's Branded Designer Clothing"
}
```

- Click Send
- Check the Response

The screenshot shows the Postman interface with the following details:

- Request URL:** http://localhost:8092/category/9
- Method:** PUT
- Headers:** Content-Type: application/json
- Body (JSON):**

```

1 {
2   "id": 9,
3   "categoryName": "Young Women's Clothing Updated",
4   "categoryDescription": "Young Women's Branded Designer Clothing"
5 }
```

- Response Status:** 200 OK
- Response Body:** Category has been updated successfully.

Let's verify the Update

- Method: GET
- URL : <http://localhost:8092/category/9>
- Headers
 - Accept: application/json
 - Content-Type: application/json
- Click Send
- Check the Response

Launchpad

http://localhost:8082/category/9

No Environment

Untitled Request

GET http://localhost:8082/category/9

Send Save

Params Authorization Headers (10) Body Pre-request Script Tests Settings

Body form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1
2
3
4
5
```

Body Cookies Headers Test Results

Pretty Raw Preview Visualize JSON

Status: 200 OK Time: 29 ms Size: 284 B Save Response

```
1
2
3
4
5
```

Let's try DELETE

Launchpad

http://localhost:8082/category/9

No Environment

Untitled Request

DELETE http://localhost:8082/category/9

Send Save

Params Authorization Headers (10) Body Pre-request Script Tests Settings

Body form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1
2
3
4
5
```

Body Cookies Headers Test Results

Pretty Raw Preview Visualize JSON

Status: 200 OK Time: 39 ms Size: 105 B Save Response

```
1. Category has been deleted successfully.
```

The Database View

Query 1 rs_ecommerce - Schema rollingstone_category

Limit to 1000 rows

```
1 • SELECT * FROM rs_ecommerce.rollingstone_category;
```

Result Grid Filter Rows: Edit Export/Import Wrap Cell Content

	id	category_description	category_name
▶	5	Food	Food
▶	6	Oranges	Oranges
▶	7	Electronics	Electronics
▶	8	Television	Television
*	NULL	NULL	NULL

4.18 ADDING SWAGGER API AND UI DOCUMENTATION

4.18.1 WHY SWAGGER

Software development is now a worldwide activity for many large business organizations. It is expected that either complete strangers may investigate code written by us today, or we might have to return to our code written many months or years ago. Keeping that in mind, conveying information in a standard and proper way is considered best practice. One old way of documenting code was to include a single static line and multiple documentation within the code itself. All languages have reliable support for such comments and ignore them during compilation. Comments also do not increase the size of our executables—all good. However, static comments tend to fall out with the code as it is maintained. Development forgets to maintain the comments as they maintain the code for bug fixes, new parameters, new methods, etc. Swagger is a new modern way of standardizing documentation and letting a proper documentation framework investigate our live / latest code to generate the documentation without relying on the software engineers' diligence to maintain the documentation. Besides, Swagger is widely and profoundly configurable and make a fully functional UI website available for testing our REST APIs

4.18.2 ADDING DEPENDENCIES

Add the following two lines of code in our build.gradle file to add Swagger to our application.

```
implementation "io.springfox:springfox-boot-starter:3.0.0"  
implementation "io.springfox:springfox-swagger-ui:3.0.0"
```

4.18.3 SWAGGER CONFIGURATION

Following is how we can configure Swagger to enhance the Documentation for our REST APIs with some static information such as Name, Email etc. Create the class named `SpringFoxConfigForCategory` in the `com.rollingstone.config` package. The full code is shown below

```
package com.rollingstone.config;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2WebMvc;

@Configuration
@EnableSwagger2WebMvc
public class SpringFoxConfigForCategory {
    public static final Contact DEFAULT_CONTACT = new Contact(
        "Binit Datta", "http://binitdatta.com", "binit-sample-email.com");
    public static final ApiInfo DEFAULT_API_INFO = new ApiInfo(
        "Category API Title", "Category API Description", "1.0",
        "urn:tos", DEFAULT_CONTACT,
        "Apache 2.0", "http://www.apache.org/licenses/LICENSE-2.0", Arrays.asList());
    private static final Set<String> DEFAULT_PRODUCES_AND_CONSUMES =
        new HashSet<String>(Arrays.asList("application/json"));

    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
            .apiInfo(DEFAULT_API_INFO)
            .produces(DEFAULT_PRODUCES_AND_CONSUMES)
            .consumes(DEFAULT_PRODUCES_AND_CONSUMES);
    }
}
```

4.18.4 SWAGGER API CONFIGURATION

This is another class to configure Swagger to provide a JSON response for the APIs. Imagine, having hundreds of APIs and the need to consolidate all JSON responses from all these APIs into a single standard UIs for external users to browser all APIs offered by us. Create another class named as CategoryApiDocumentationConfiguration in the package com.rollingstone.config. Here is the full code.

```
package com.rollingstone.config;
import io.swagger.annotations.Contact;
import io.swagger.annotations.ExternalDocs;
import io.swagger.annotations.Info;
import io.swagger.annotations.License;
import io.swagger.annotations.SwaggerDefinition;

@SwaggerDefinition(
    info = @Info(
        description = "Category REST API Resources",
        version = "V1.0",
        title = "Category REST API Full CRUD",
        contact = @Contact(
            name = "Binit Datta",
            email = "binit-sample-email@gmail.com",
            url = "http://www.binitdatta.com"
        ),
        license = @License(
            name = "Apache 2.0",
            url = "http://www.apache.org/licenses/LICENSE-2.0"
        )
    ),
    consumes = {"application/json"},
    produces = {"application/json"},
    schemes = {SwaggerDefinition.Scheme.HTTP, SwaggerDefinition.Scheme.HTTPS},
    externalDocs = @ExternalDocs(value = "For Further Information", url =
    "http://binitdatta.com")
)
public class CategoryApiDocumentationConfiguration {
```

With the configuration complete, build the application again with gradle and run it. Now Try <http://localhost:8092/v2/api-docs>



```
// 20210102135044
// http://localhost:8092/v2/api-docs

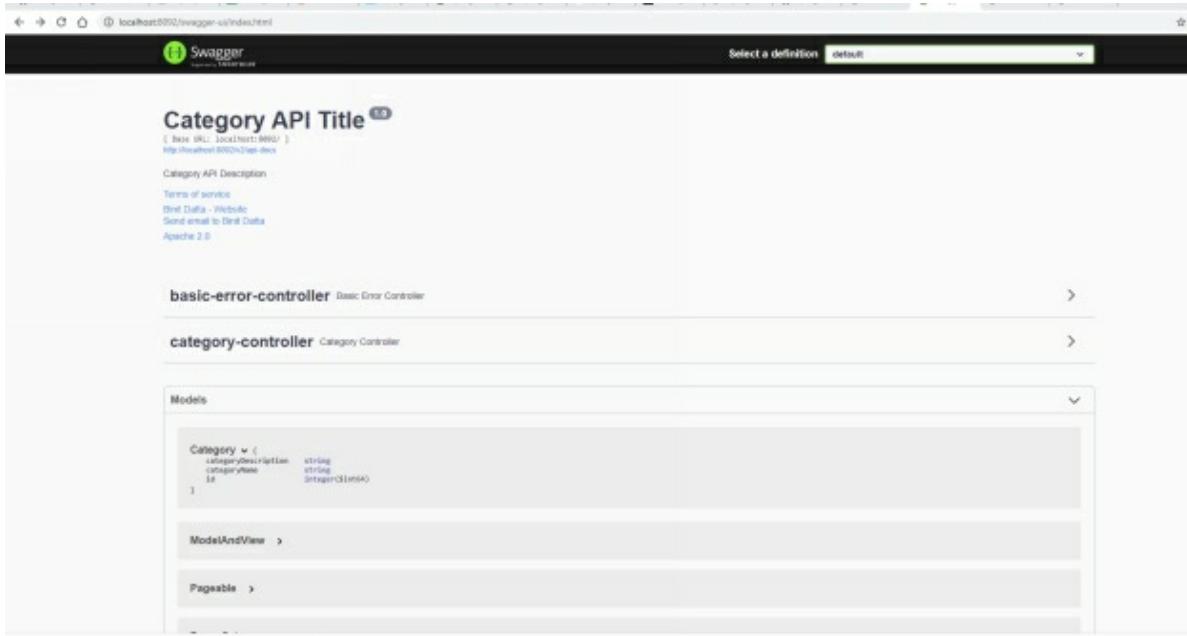
{
    "swagger": "2.0",
    "info": {
        "description": "Category API Description",
        "version": "1.0",
        "title": "Category API Title",
        "termsOfService": "urn:tos",
        "contact": {
            "name": "Binit Datta",
            "url": "http://binitdatta.com",
            "email": "binit-sample-email.com"
        },
        "license": {
            "name": "Apache 2.0",
            "url": "http://www.apache.org/licenses/LICENSE-2.0"
        }
    },
    "host": "localhost:8092",
    "basePath": "/",
    "tags": [
        {
            "name": "basic-error-controller",
            "description": "Basic Error Controller"
        },
        {
            "name": "category-controller",
            "description": "Category Controller"
        }
    ],
    "consumes": [
        "application/json"
    ],
    "produces": [
        "application/json"
    ]
},
```

localhost:8092/v2/api-docs

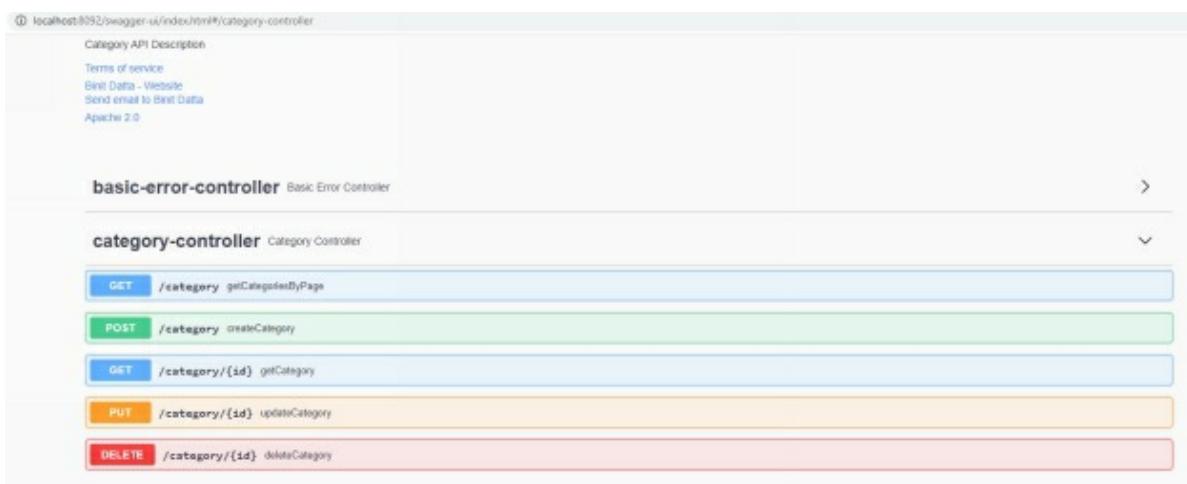
```
38     ],
39   "paths": {
40     "/category": {
41       "get": {
42         "tags": [
43           "category-controller"
44         ],
45         "summary": "getCategoriesByPage",
46         "operationId": "getCategoriesByPageUsingGET",
47         "parameters": [
48           {
49             "name": "pagenumber",
50             "in": "query",
51             "description": "pagenumber",
52             "required": false,
53             "type": "integer",
54             "default": 0,
55             "format": "int32"
56           },
57           {
58             "name": "pagesize",
59             "in": "query",
60             "description": "pagesize",
61             "required": false,
62             "type": "integer",
63             "default": 20,
64             "format": "int32"
65           }
66         ],
67         "responses": {
68           "200": {
69             "description": "OK",
70             "schema": {
71               "$ref": "#/definitions/Page«Category»"
72             }
73           }
74         }
75       }
76     }
77   }
78 }
```

Try this for the Swagger UI and see how Swagger helps us not only provide/generate dynamic documentation without writing so much docs ourselves but also lets us test the app.

<http://localhost:8092/swagger-ui/index.html>



The screenshot shows the Swagger UI homepage. At the top, it displays the title "Category API Title" with a "Basic Error Controller" link. Below this, there's a "Category API Description" section with links to "Terms of service", "Birn Data - Website", "Send email to Birn Data", and "Apache 2.0". The main content area is titled "basic-error-controller" and "Category Controller". It includes sections for "Models" (with a "Category" model definition), "ModelAndView", and "Pageable".



The screenshot shows the "category-controller" section of the Swagger UI. It lists several API endpoints:

- GET /category - getCategoriesByPage
- POST /category - createCategory
- GET /category/{id} - getCategory
- PUT /category/{id} - updateCategory
- DELETE /category/{id} - deleteCategory

Each endpoint is represented by a colored button (blue for GET, green for POST, blue for GET, orange for PUT, red for DELETE) followed by its path and description.

category-controller Category Controller

GET /category/getCategoriesByPage

Parameters

Name Description

pagenumber integer(32) pagenumber
(every)
Default value: 0

pagesize integer(32) pagesize
(every)
Default value: 20

Responses

localhost:8092/swagger-ui/index.html#/category-controller/getCategoriesByPageUsingGET

Code Description

200 OK

Example Value | Model

```
[{"content": [{"categoryDescription": "string", "categoryName": "string", "id": 0}], "empty": true, "first": true, "last": true, "number": 0, "numberOfElements": 0, "pageable": {"paged": false, "pageSize": 0, "pageNumber": 0, "sort": [{"empty": true, "sort": true, "sorted": true}, {"empty": true}], "size": 0, "sort": [{"empty": true}], "unpaged": true}, "size": 0, "sort": [{"empty": true}], "unpaged": true}
```

401 Unauthorized

403 Forbidden

404 Not Found

Response content type application/json

4.19 TESTING ACTUATOR

When we deploy our application to the production environment, these applications must run responsibly without overconsuming memory, CPU, or disks. The production environment is shared by hundreds of other microservices, and we would like to quickly identify a mal performing microservice to save others. Load balances of all kinds also ask for a health check from the targets they call. From these perspectives, Spring Boot Actuator adds a wealth of monitoring functionality to any Spring Boot Microservices. Today all language and framework stacks claim to have a robust framework for building Microservices. However, how much excellent support these Microservice development frameworks provide in the production environment is a critical factor in comparing language/framework stacks. With its numerous starters that provide substantial productivity gains, Spring Boot wins hands down compared to others. After defeating hands down merely from the starters' perspective, Spring Boot delivers a knockout punch by the Actuator and Micrometer frameworks built in the framework. In the following section, we will scratch the tip of the Actuator functionality.

Try this first <http://localhost:8093/actuator>

```
localhost:8093/actuator

1 // 20210103070108
2 // http://localhost:8093/actuator
3
4 {
5     "_links": {
6         "self": {
7             "href": "http://localhost:8093/actuator",
8             "templated": false
9         },
10        "beans": {
11            "href": "http://localhost:8093/actuator/beans",
12            "templated": false
13        },
14        "caches-cache": {
15            "href": "http://localhost:8093/actuator/caches/{cache}",
16            "templated": true
17        },
18        "caches": {
19            "href": "http://localhost:8093/actuator/caches",
20            "templated": false
21        },
22        "health": {
23            "href": "http://localhost:8093/actuator/health",
24            "templated": false
25        },
26        "health-path": {
27            "href": "http://localhost:8093/actuator/health/{*path}",
28            "templated": true
29        },
30    }
31 }
```

More Actuator Endpoints

localhost:8093/actuator

```
"info": {  
    "href": "http://localhost:8093/actuator/info",  
    "templated": false  
},  
"conditions": {  
    "href": "http://localhost:8093/actuator/conditions",  
    "templated": false  
},  
"configprops": {  
    "href": "http://localhost:8093/actuator/configprops",  
    "templated": false  
},  
"env": {  
    "href": "http://localhost:8093/actuator/env",  
    "templated": false  
},  
"env-toMatch": {  
    "href": "http://localhost:8093/actuator/env/{toMatch}",  
    "templated": true  
},  
"loggers": {  
    "href": "http://localhost:8093/actuator/loggers",  
    "templated": false  
},  
"loggers-name": {  
    "href": "http://localhost:8093/actuator/loggers/{name}",  
    "templated": true  
},  
"heapdump": {  
    "href": "http://localhost:8093/actuator/heapdump",  
    "templated": false  
},  
"threaddump": {  
    "href": "http://localhost:8093/actuator/threaddump",  
    "templated": false  
},
```

And some more

```
        },
        "metrics-requiredMetricName": {
            "href": "http://localhost:8093/actuator/metrics/{requiredMetricName}",
            "templated": true
        },
        "metrics": {
            "href": "http://localhost:8093/actuator/metrics",
            "templated": false
        },
        "scheduledtasks": {
            "href": "http://localhost:8093/actuator/scheduledtasks",
            "templated": false
        },
        "mappings": {
            "href": "http://localhost:8093/actuator/mappings",
            "templated": false
        }
    }
}
```

4.19.1 HEALTH ENDPOINT

We did not write any code to generate these details health information! We added one simple property in our config file. Spring Boot Actuator looks into our gradle/maven dependencies and figures out the external environment we are using, gets configuration information from our config file and checks connections to these external resources to report their health along with other information.

← → C ⌂ ⓘ localhost:8093/actuator/health

```
1 // 20210103070343
2 // http://localhost:8093/actuator/health
3
4 {
5     "status": "UP",
6     "components": {
7         "db": {
8             "status": "UP",
9             "details": {
10                 "database": "MySQL",
11                 "validationQuery": "isValid()"
12             }
13         },
14         "diskSpace": {
15             "status": "UP",
16             "details": {
17                 "total": 255400603648,
18                 "free": 152246095872,
19                 "threshold": 10485760,
20                 "exists": true
21             }
22         },
23         "ping": {
24             "status": "UP"
25         }
26     }
27 }
```

Metrics Endpoint

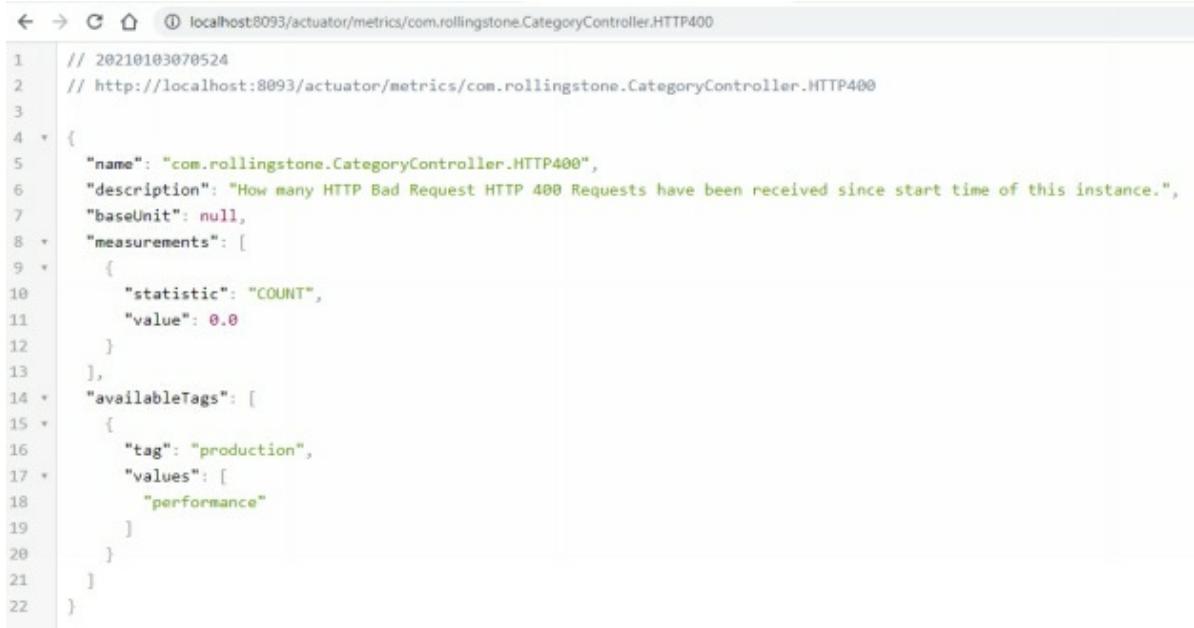


The screenshot shows a browser window with the URL `localhost:8093/actuator/metrics` in the address bar. The page content is a JSON object:

```
// 20210103070458
// http://localhost:8093/actuator/metrics

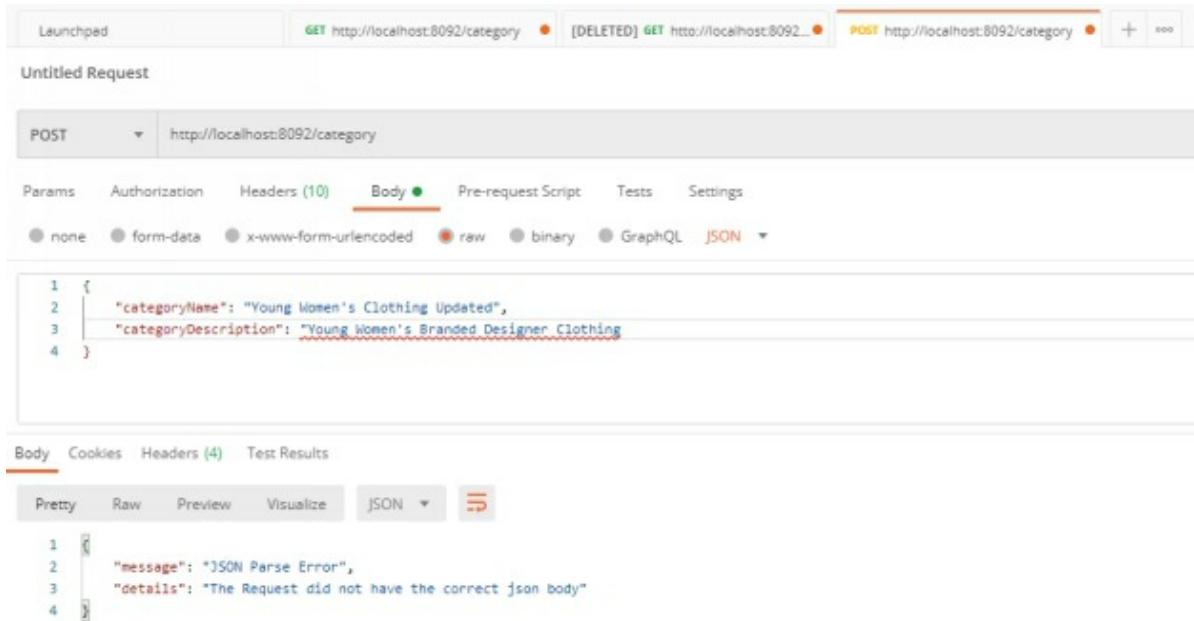
{
  "names": [
    "com.rollingstone.CategoryController.HTTP400",
    "com.rollingstone.CategoryController.HTTP404",
    "com.rollingstone.category.created",
    "hikaricp.connections",
    "hikaricp.connections.acquire",
    "hikaricp.connections.active",
    "hikaricp.connections.creation",
    "hikaricp.connections.idle",
    "hikaricp.connections.max",
    "hikaricp.connections.min",
    "hikaricp.connections.pending",
    "hikaricp.connections.timeout",
    "hikaricp.connections.usage",
    "jdbc.connections.active",
    "jdbc.connections.idle",
    "jdbc.connections.max",
    "jdbc.connections.min",
    "jvm.buffer.count",
    "jvm.buffer.memory.used",
    "jvm.buffer.total.capacity",
    "jvm.classes.loaded",
    "jvm.classes.unloaded",
    "jvm.gc.live.data.size",
    "jvm.gc.max.data.size",
    "jvm.gc.memory.allocated",
    "jvm.gc.memory.promoted",
    "jvm.gc.pause",
    "jvm.memory.committed",
    "jvm.memory.max",
    "jvm.memory.used",
    "jvm.threads.daemon",
    "jvm.threads.live",
    "jvm.threads.peak",
    "jvm.threads.states",
    "logback.events",
  ]
}
```

One specific Metrics Endpoint



```
// 20210103070524
// http://localhost:8093/actuator/metrics/com.rollingstone.CategoryController.HTTP400
{
  "name": "com.rollingstone.CategoryController.HTTP400",
  "description": "How many HTTP Bad Request HTTP 400 Requests have been received since start time of this instance.",
  "baseUnit": null,
  "measurements": [
    {
      "statistic": "COUNT",
      "value": 0.0
    }
  ],
  "availableTags": [
    {
      "tag": "production",
      "values": [
        "performance"
      ]
    }
  ]
}
```

Let's try a malformed POST Request



Launched

Untitled Request

POST http://localhost:8092/category

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
  "categoryName": "Young Women's Clothing Updated",
  "categoryDescription": "Young Women's Branded Designer Clothing"
}
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
{
  "message": "JSON Parse Error",
  "details": "The Request did not have the correct json body"
}
```

See how our exception counter is working below.

```
→ ⌂ ⌂ ⓘ localhost:8093/actuator/metrics/com.rollingstone.CategoryController.HTTP400
// 20210103072457
// http://localhost:8093/actuator/metrics/com.rollingstone.CategoryController.HTTP400

{
  "name": "com.rollingstone.CategoryController.HTTP400",
  "description": "How many HTTP Bad Request HTTP 400 Requests have been received since start time of this instance.",
  "baseUnit": null,
  "measurements": [
    {
      "statistic": "COUNT",
      "value": 1.0
    }
  ],
  "availableTags": [
    {
      "tag": "environment",
      "values": [
        "production"
      ]
    }
  ]
}
```

Let's create a few test data and try the success counter

Try the actuator metrics endpoint again to see count increased

```
← → ⌂ ⌂ ⓘ localhost:8093/actuator/metrics/com.rollingstone.category.created
1 // 20210103080719
2 // http://localhost:8093/actuator/metrics/com.rollingstone.category.created
3
4 +
5   {
6     "name": "com.rollingstone.category.created",
7     "description": "Number of Categories Created",
8     "baseUnit": null,
9     "measurements": [
10       {
11         "statistic": "COUNT",
12         "value": 4.0
13       }
14     ],
15     "availableTags": [
16       {
17         "tag": "environment",
18         "values": [
19           "production"
20         ]
21       }
22     ]
}
```

Full Code

```
package com.rollingstone.custom.endpoints;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.actuate.endpoint.annotation.*;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;

import java.util.concurrent.ConcurrentHashMap;

@Component
@Endpoint(id = "is-customer-healthy")
public class CustomerHealth {

    protected final Logger log = LoggerFactory.getLogger(this.getClass());

    @Autowired
    RestTemplate restTemplate;

    @ReadOperation
    public String IsCustomerHealthy() {
        final String uri = "http://localhost:8092/category";

        try{
            String result = restTemplate.getForObject(uri, String.class);
            return "SUCCESS";
        }
        catch(Exception e){
            log.error("Health Endpoint Failing with :" +e.getMessage());
            return "FAILURE";
        }
    }

    @WriteOperation
    public void writeOperation(@Selector String name) {
        //perform write operation
    }

    @DeleteOperation
    public void deleteOperation(@Selector String name){
        //delete operation
    }
}
```

4.19.2 NEW CUSTOM ACTUATOR ENDPOINT

While Spring Boot provides a great health endpoint out of the box, it is often necessary to test our real APIs to determine the real health of the application. For that purpose, Spring Boot Actuator lets us write a complete new custom class and get that added in the actuator list of endpoints. One added advantage (a big one) is that we can use them with various deployment orchestrators and load balancers such as AWS Application Load Balancers or Kubernetes (we will see this later). `@ReadOperation` gets called for GET and `@WriteOperation` method gets called for POST. Following is the full code which is in the `com.rollingstone.custom.endpoints` package.

```
package com.rollingstone.custom.endpoints;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.actuate.endpoint.annotation.*;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;
import java.util.concurrent.ConcurrentHashMap;

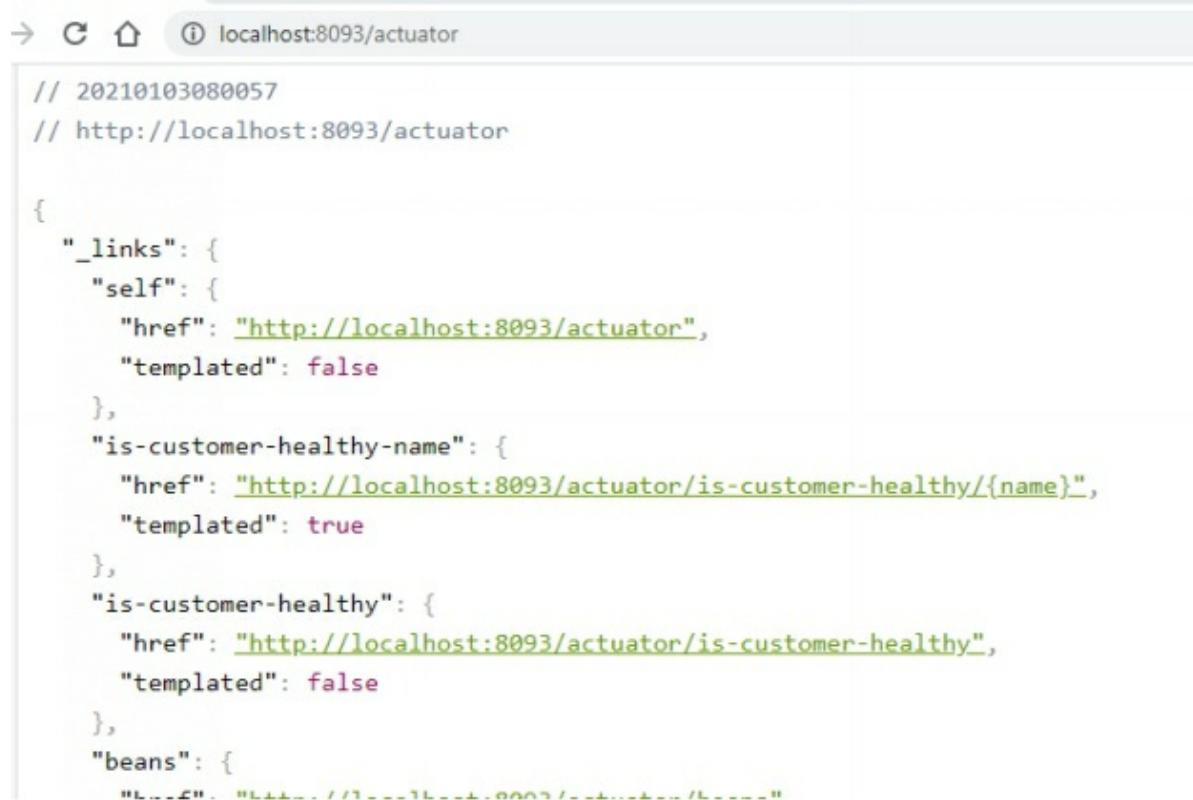
@Component
@Endpoint(id = "is-customer-healthy")
public class CustomerHealth {
    protected final Logger log = LoggerFactory.getLogger(this.getClass());
    @Autowired
    RestTemplate restTemplate;
    @ReadOperation
    public String IsCustomerHealthy() {
        final String uri = "http://localhost:8092/category";
        try{
            String result = restTemplate.getForObject(uri, String.class);
            return "SUCCESS";
        }
        catch(Exception e){
            log.error("Health Endpoint Failing with :" +e.getMessage());
            return "FAILURE";
        }
    }
}
```

```

    }
    @WriteOperation
    public void writeOperation(@Selector String name) {
        //perform write operation
    }
    @DeleteOperation
    public void deleteOperation(@Selector String name){
        //delete operation
    }
}

```

Build and run the application and try the actuator endpoint to find the new endpoint added.



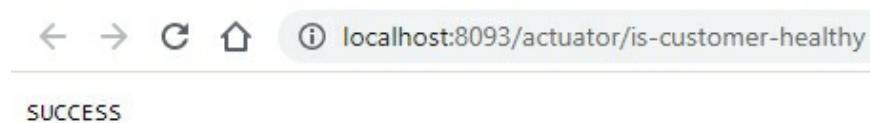
The screenshot shows a browser window with the URL `localhost:8093/actuator`. The page displays a JSON object representing the available endpoints:

```

{
  "_links": {
    "self": {
      "href": "http://localhost:8093/actuator",
      "templated": false
    },
    "is-customer-healthy-name": {
      "href": "http://localhost:8093/actuator/is-customer-healthy/{name}",
      "templated": true
    },
    "is-customer-healthy": {
      "href": "http://localhost:8093/actuator/is-customer-healthy",
      "templated": false
    },
    "beans": {
      "href": "http://localhost:8093/actuator/beans"
    }
}

```

Try it now.



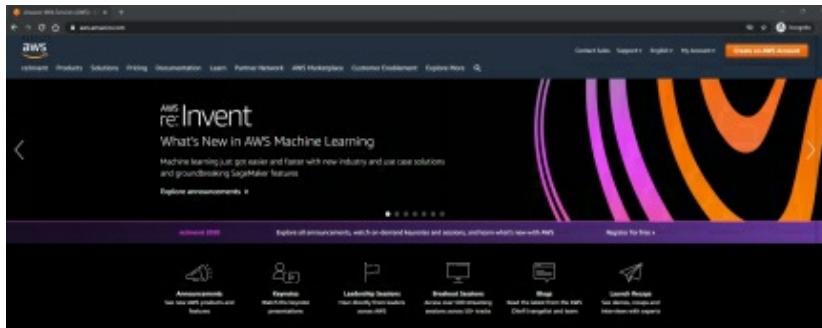
The screenshot shows a browser window with the URL `localhost:8093/actuator/is-customer-healthy`. The page displays the word "SUCCESS".

CHAPTER 5

Creating AWS Services

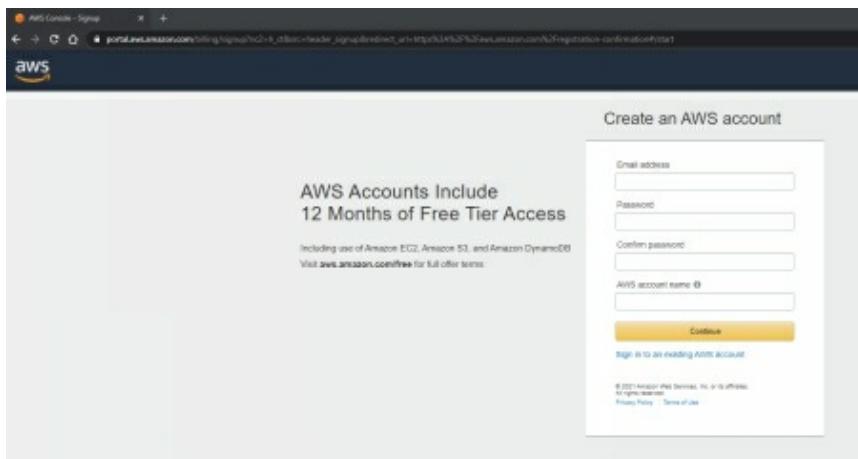
5.1 SIGNING UP WITH AWS

Open your browser and navigate to <https://aws.amazon.com/>



© Amazon Web Services

Click on Create an AWS Account Button

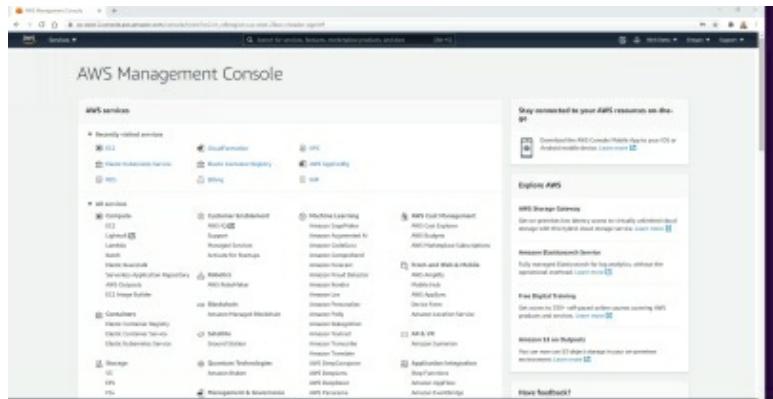


© Amazon Web Services

Please follow the rest of the easy-to-follow instructions to create an AWS Account

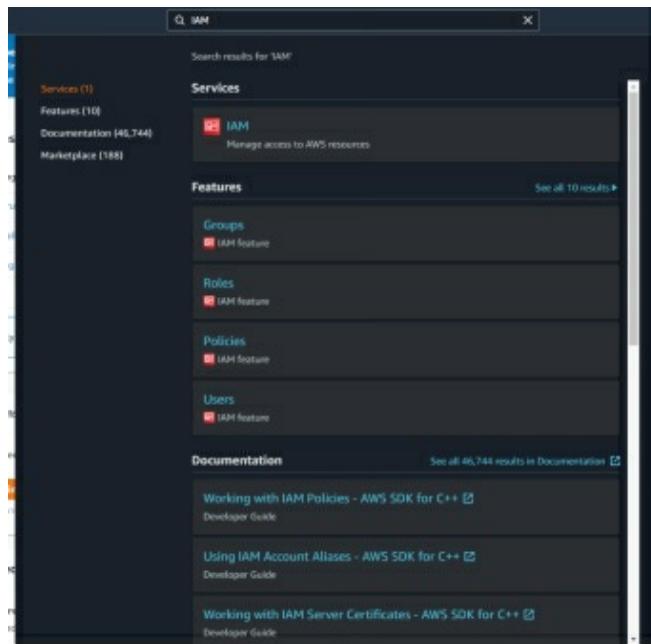
5.2 CREATING A BASTION HOST IN AWS EC2

Sign in to your New AWS Account



© Amazon Web Services

Find or Search for IAM



© Amazon Web Services

Click on IAM

The screenshot shows the AWS IAM Management Console dashboard. On the left, there's a navigation pane with options like Dashboard, Groups, Users, Roles, Policies, Identity providers, Account settings, Access reports, Access analysis, and more. The main content area displays the IAM dashboard, showing 11 users, 6 groups, 40 roles, and 2 identity providers. It includes sections for Security alerts (warning about access keys enabled) and Best practices (listing recommendations). A sidebar on the right provides additional information, tools, and related services.

© Amazon Web Services

Click on Roles on the Left Panel

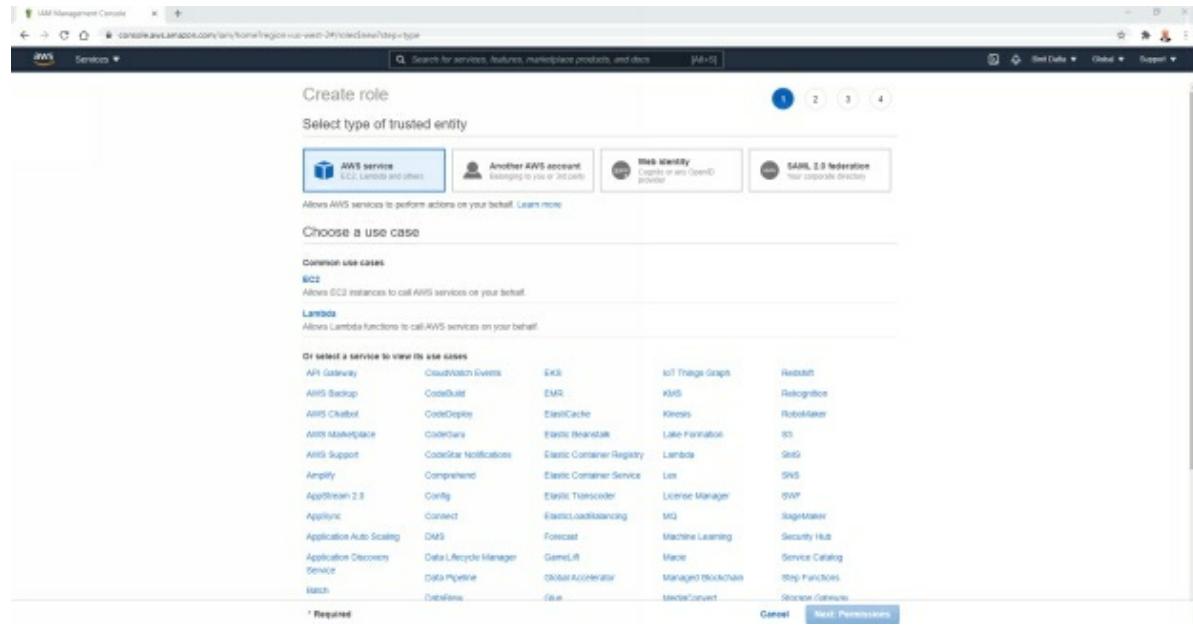
The screenshot shows the AWS IAM Management Console Roles page. The left navigation pane highlights the Roles option under Access management. The main content area shows a list of roles with columns for Role name, Trusted entities, and Last activity. A search bar at the top allows filtering by role name. The table lists various roles, such as 'AlicommCodeDeployServiceRole', 'amazon-ecs-cl-simp-bent-cogs-cl-SchedulerRole', 'aws-elasticbeanstalk-ec2-role', 'aws-elasticbeanstalk-service-role', 'AWSServiceRoleForAmazonEKS', 'AWSServiceRoleForAmazonECSNodegroup', 'AWSServiceRoleForAmazonElastiCache', and 'AWSLambdaVPCExecutionRole'. The last column indicates the last activity for each role.

Role name	Trusted entities	Last activity
AlicommCodeDeployServiceRole	Amazon service: codedeploy	None
amazon-ecs-cl-simp-bent-cogs-cl-SchedulerRole	Amazon service: ecs	None
aws-elasticbeanstalk-ec2-role	Amazon service: ec2	None
aws-elasticbeanstalk-service-role	Amazon service: elasticbeanstalk	None
AWSServiceRoleForAmazonEKS	Amazon service: eks (Service: United Grid)	Today
AWSServiceRoleForAmazonECSNodegroup	Amazon service: ecs-nodegroup (Service: Lambda)	7 days
AWSServiceRoleForAmazonElastiCache	Amazon service: ecs-application-autoscaling (Service: Lambda)	None
AWSLambdaVPCExecutionRole	Amazon service: lambda (Service: Lambda)	7 days

© Amazon Web Services

Click on Create Role

Keep AWS Service Selected and choose EC2 and click Next:Permissions



© Amazon Web Services

Click on EC2 to get to the EC2 Dashboard

The screenshot shows the AWS EC2 Management Dashboard. On the left, a sidebar navigation includes: New EC2 Experience (with a feedback link), EC2 Dashboard, Instances (with sub-options like Instances, Instance Types, Launch Templates, Spot Requests, Saving Plans, Reserved Instances, Dedicated Hosts, Scheduled Instances, Capacity Reservations), Images (AMIs), Basic Block Store (Volumes, Snapshots, Lifecycle Manager), Network & Security (Security Groups, Elastic IPs, Placement Groups, Key Pairs, Network Interfaces), and Help.

The main content area has a header "Welcome to the new EC2 console" with a note about the redesign. It displays "Resources" and "Account attributes".

Resources:

Instances (running)	0	Dedicated Hosts	0	Elastic IPs	3
Instances (all states)	6	Key pairs	27	Load balancers	8
Placement groups	0	Security groups	36	Snapshots	5
Volumes	9				

Account attributes:

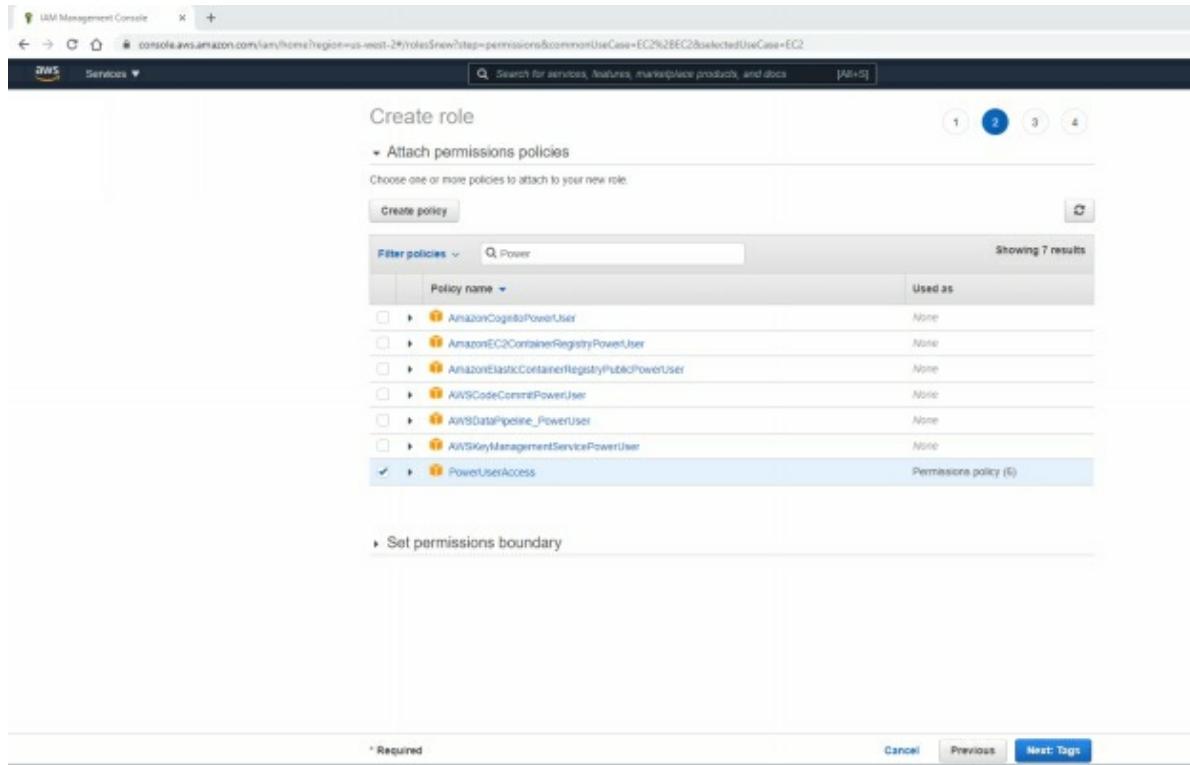
- Supported platforms: VPC (Default VPC: vpc-64d2b6c0f9)
- Settings
- SSL encryption
- Zones
- Default credit specification
- Console experiments

Explore AWS:

- Get Up to 40% Better Price Performance
- Launch Custom AMIs with Fast Snapshot Restore (PSR)
- Save up to 90% on EC2 with Spot Instances

© © Amazon Web Services

Search for Power, choose PowerUserAccess and Click Next: Tags



© Amazon Web Services

Enter the Tag and Value shown

IAM Management Console

Services ▾

Search for services, features, marketplace products, and docs [Alt+S]

Create role

Add tags (optional)

IAM tags are key-value pairs you can add to your role. Tags can include user information, such as an email address, or can be descriptive, such as a job title. You can use the tags to organize, track, or control access for this role. Learn more.

Key	Value (optional)	Remove
Name	IAMInstanceRole	X
Add new key		

You can add 49 more tags.

Cancel Previous Next: Review Create role

© Amazon Web Services

Click Review

Enter the Role Name

IAM Management Console

Services ▾

Search for services, features, marketplace products, and docs [Alt+S]

Create role

Review

Provide the required information below and review this role before you create it.

Role name: EC2PowerUserRole
Use alphanumeric and "-_, @_" characters. Maximum 64 characters.

Role description: Allows EC2 instances to call AWS services on your behalf.
Maximum 1000 characters. Use alphanumeric and "-_, @_" characters.

Trusted entities: AWS service: ec2.amazonaws.com

Policies: PowerUserAccess

Permissions boundary: Permissions boundary is not set.

The new role will receive the following tag:

Key	Value
Name	IAMInstanceRole

* Required Cancel Previous Create role Next Step

Click on Create Role

Role Created

The screenshot shows the IAM Management Console interface. On the left, there's a sidebar with navigation links like Dashboard, Access management, Policies, and Roles. The 'Roles' link is currently selected. In the main content area, a modal window titled 'Create role' is open, with the 'Create role' button highlighted in blue. Below the modal, a list of existing roles is displayed in a table. One role, 'EC2PowerUserRole', is selected and highlighted with a blue background. Other roles listed include A1EcommCodeDeployServiceRole, amazon-ecs-cl-setup-bmt-cqrs-d-EcsInstanceRole-1F6Y3SUSLVT02, aws-elasticaeastalk-ec2-role, AWSServiceRoleForApplicationAutoScaling_ECSService, AWSServiceRoleForECS, CodeDeployDemo-EC2-Instance-Profile, ec2Instances, ecsAutoscaleRole, ecsInstanceRole, ecsServiceRole, EMR_EC2_DefaultRole, lambda_basic_execution, and lambda_http_execution.

Role Details

The screenshot shows the 'Summary' tab for the 'EC2PowerUserRole'. The role ARN is listed as arn:aws:iam::107222676612:role/EC2PowerUserRole. The role description is 'Allows EC2 instances to call AWS services on your behalf.' The instance profile ARNs are listed as arn:aws:iam::107222676612:instance-profile/EC2PowerUserRole. The path is '/'. The creation time is 2021-01-09 13:52 CST. The last activity is 'Not accessed in the tracking period'. The maximum session duration is 1 hour 50M. The 'Permissions' tab is selected, showing one policy applied: 'PowerUserAccess'. The policy type is 'AWS managed policy'.

Search for EC2

The screenshot shows the AWS search interface with the query 'EC2'. The results are categorized into three main sections: Services, Features, and Documentation.

- Services (84)**:
 - EC2**: Virtual Servers in the Cloud
 - Direct Connect**: Dedicated Network Connection to AWS
 - Inspector**: Analyze Application Security
 - CodeCommit**: Store Code in Private Git Repositories
- Features (166)**:
 - Direct Connect gateways**: Direct Connect feature
 - Recorder**: Config feature
 - Detect**: IoT Core feature
 - Projects**: CodeStar feature
- Documentation (23,654)**: See all 23,654 results in Documentation

© Amazon Web Services

Click on EC2 to navigate to the following screen

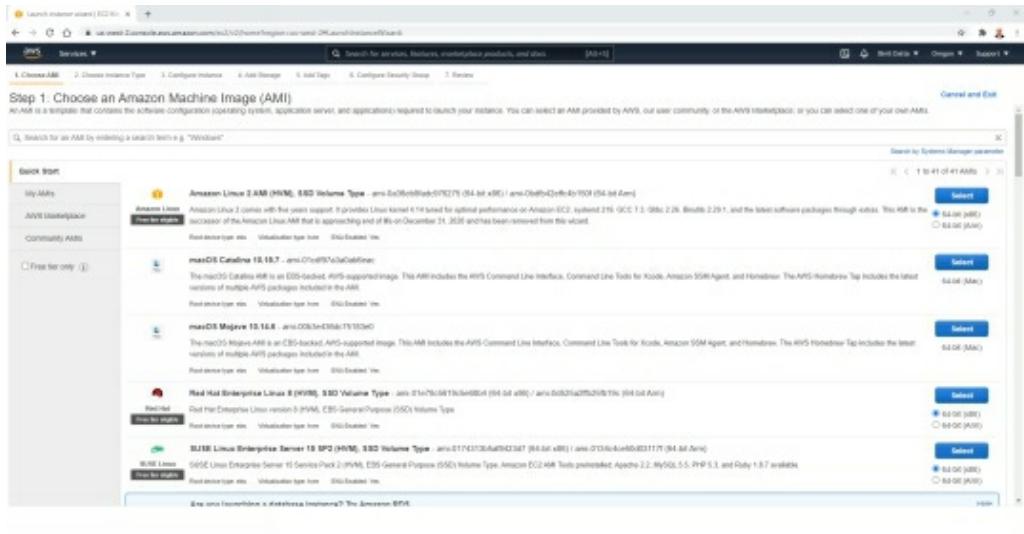
The screenshot shows the 'New EC2 Experience' dashboard. The left sidebar includes navigation links for Instances, Images, AMIs, Status Block & Share, Network & Security, and Metrics. The main content area displays the following information:

- Welcome to the new EC2 console!**: A note about the redesigned EC2 console.
- Resources**: A summary of resources in the US West (Oregon) Region:
 - Instances (Pending): 0
 - Instances (Running): 0
 - Instances (Terminated): 0
 - Placements groups: 0
 - Regions: 0
 - Dedicated Hosts: 0
 - Key pairs: 27
 - Load balancers: 0
 - Security groups: 30
 - Snapshots: 5
- Launch instance**: A button to start launching an instance.
- Service health**: Shows the status of the Service Health Dashboard.
- Zone status**: Shows the status of instances across different availability zones in the US West (Oregon) region.
- Account attributes**: A sidebar showing account settings like Support ticket backlog, IAM users, Database API, and more.
- Explore AMIs**: A sidebar with options like Launch Custom AMIs with Fast Snapshot Restore and Enable Best Price Performance with AWS Lambda.

© Amazon Web Services

Click on Launch Instance → Launch Instance

The Choose AMI Screen



© Amazon Web Services

Choose this Ubuntu 18.04 LTS



© Amazon Web Services

Click Next

Select T2.Medium as will may need this

© Amazon Web Services

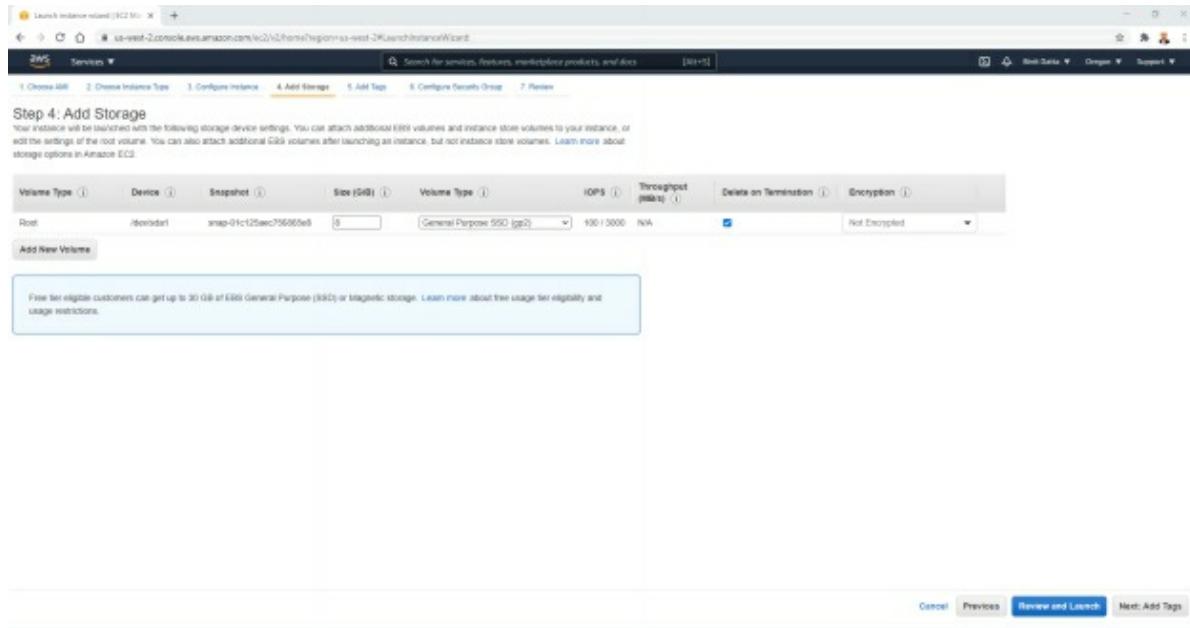
Click on Configure Instance Details

© Amazon Web Services

Provide the following Details

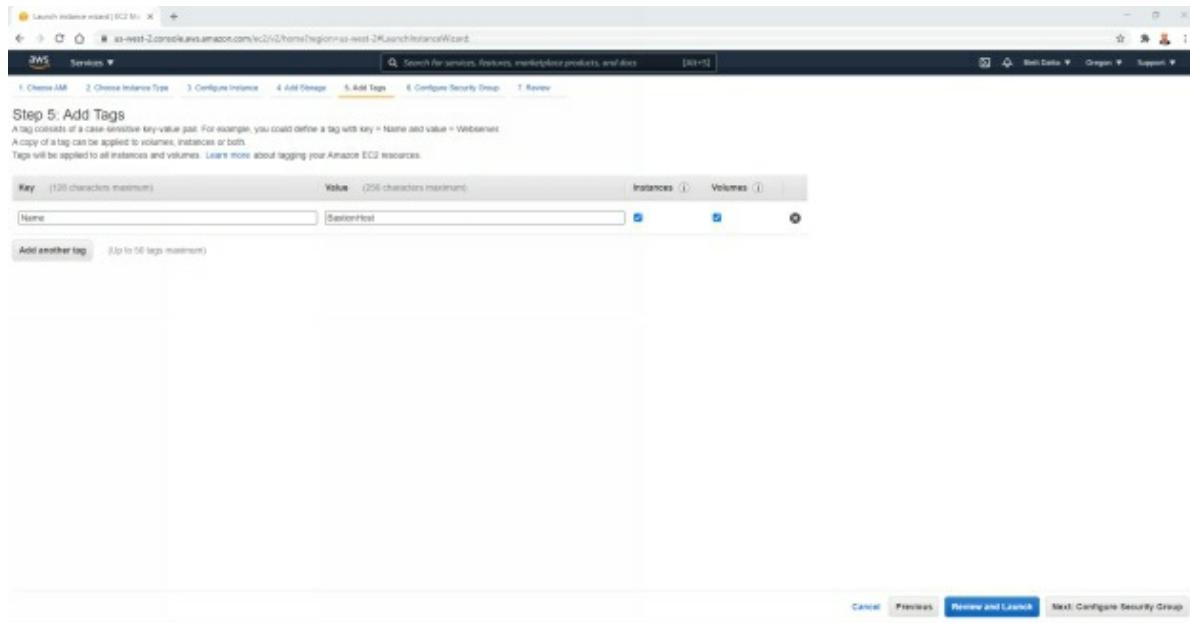
- Change Auto Assign Public IP to Enable
- Select the Role you created earlier from the drop down
- Keep the default VPC unchanged
- Keep everything else unchanged
- Click Next: Add Storage

Accept Default on this screen and click Next: Add Tags



© Amazon Web Services

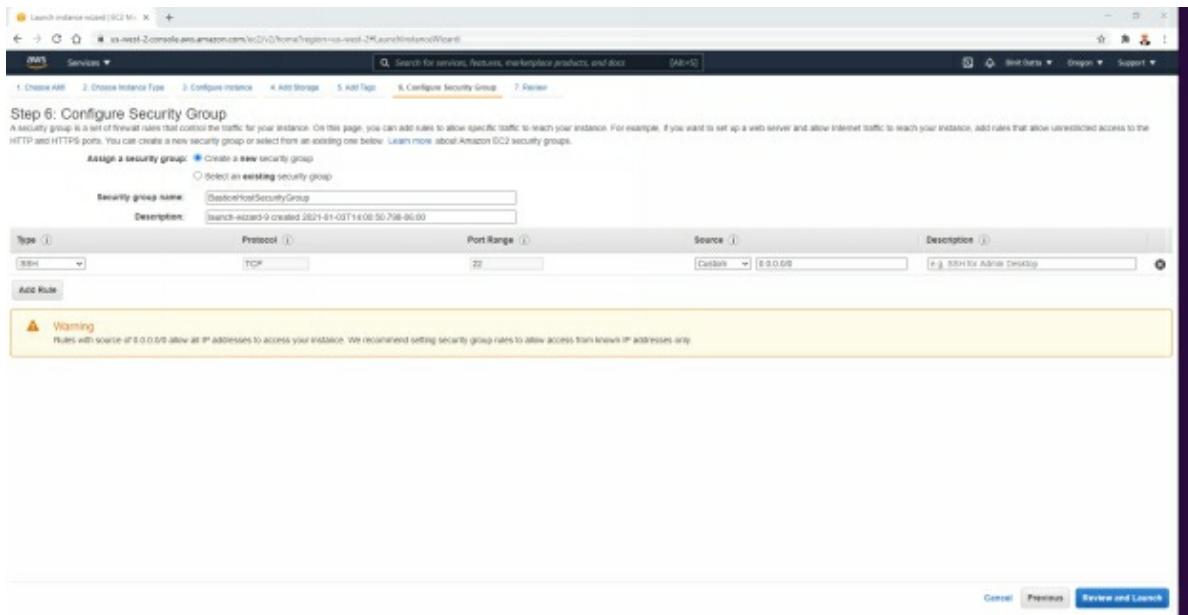
Click Add Tag and enter the Tag value



© Amazon Web Services

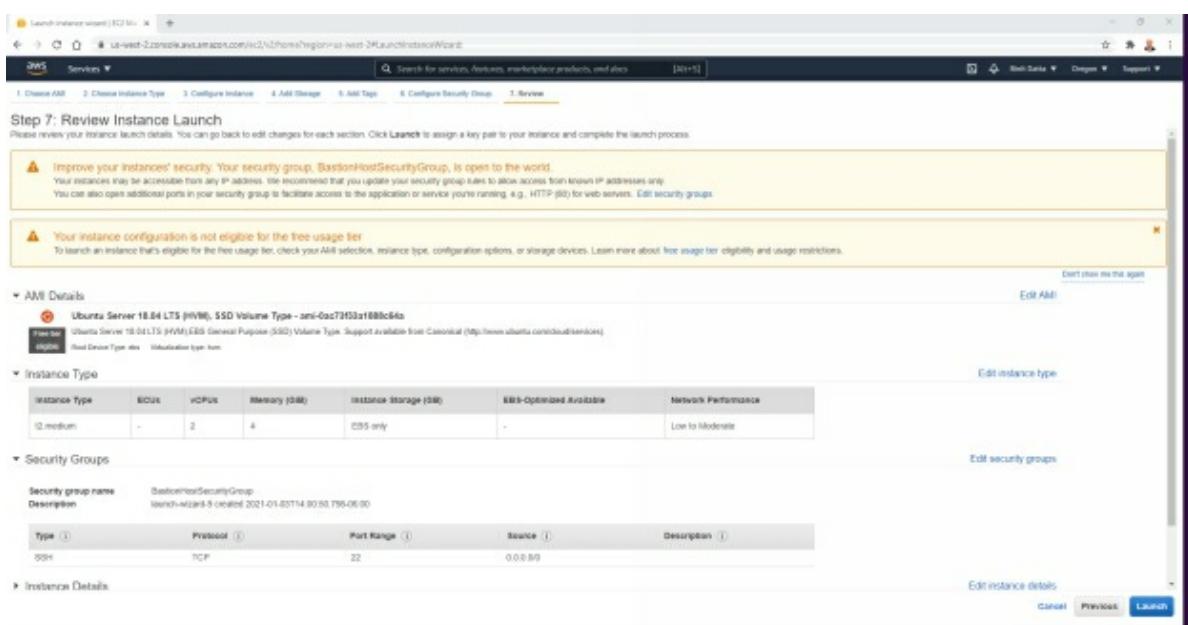
Click Next: Configure Security Group

Accept Default, name the Security Group as shown



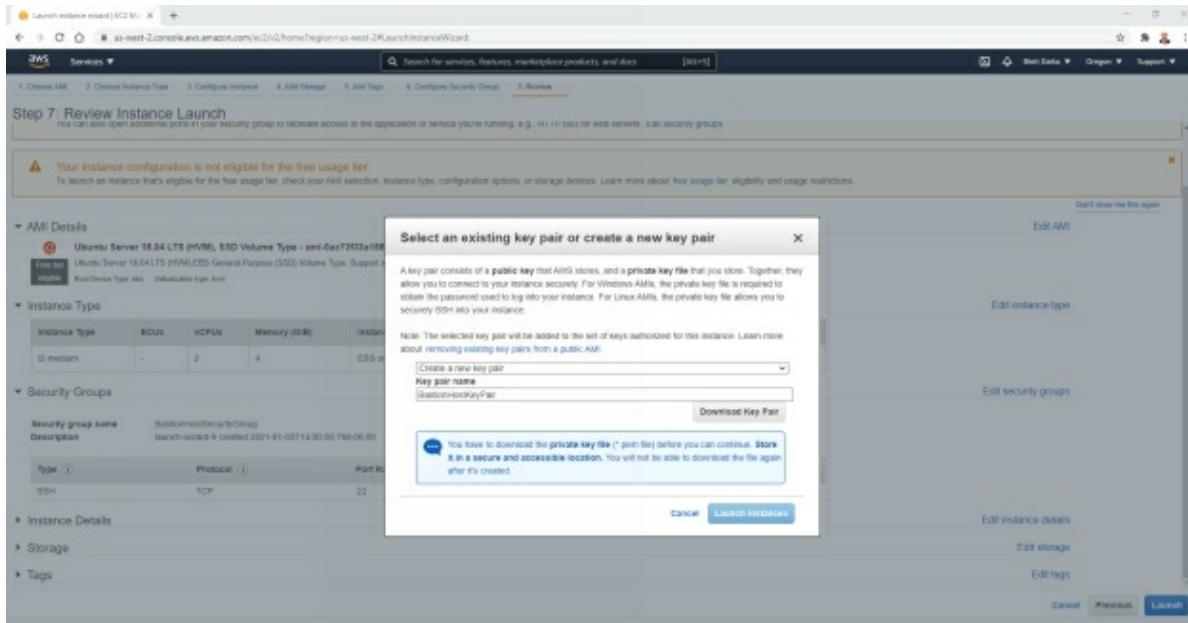
© Amazon Web Services

Click Review and Launch



© Amazon Web Services

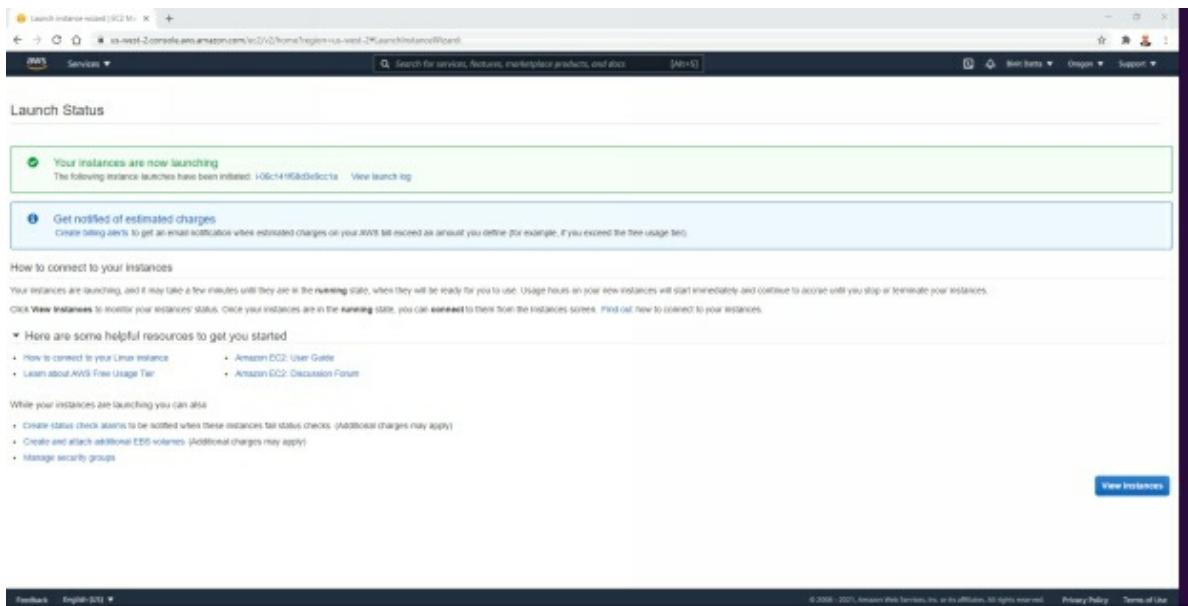
Choose Create a new Key Pair (VIP)



© Amazon Web Services

Do not forget to click on Download Key Pair to download the KP.

Click on Launch



© Amazon Web Services

Click on View Instances

Wait till the screen becomes showing Ready

The screenshot shows the AWS EC2 Instances console. On the left, there's a navigation pane with sections like Instances, Images, and Network & Security. The main area displays a table of instances with columns: Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, Public IPv4 DNS, and Public IPv4. There are 7 instances listed:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4
BentJenkinsStandard	i-0c2f124d45e72999	Stopped	t2.micro	-	No alarms	us-west-2c	-	-
JenkinsPipelineinstance	i-0210776b08448d66	Stopped	t2.micro	-	No alarms	us-west-2e	-	-
BentBdeabetesAdmin	i-0de445bae0be837	Stopped	t2.micro	-	No alarms	us-west-2e	-	-
None	i-08610f7c05035091	Stopped	t2.micro	-	No alarms	us-west-2e	-	-
AWS_TOPO_SQL_VM	i-091262306ae40ba5d	Stopped	t2.medium	-	No alarms	us-west-2e	-	-
AWS_EKS	i-0ec2b154488898c56	Stopped	t2.medium	-	No alarms	us-west-2c	-	-
BusterHost	i-0e1419b8cfef21a	Running	t2.medium	0 Initializing	No alarms	us-west-2e	ec2-56-112-75-21.us...	56.112.75.21

© Amazon Web Services

Now we are ready

This screenshot is identical to the one above, showing the AWS EC2 Instances console with 7 instances listed. The instance 'BusterHost' is still running.

© Amazon Web Services

Click on the checkbox left to the instance to get its public IP

Instances | EC2 Management Consoles | us-west-2.console.aws.amazon.com/ec2/v2/home?region=us-west-2#instances

New EC2 Experience Switch between the new and old EC2 console

Services ▾

- EC2 Dashboard new
- Events
- Tags
- Links
- Instances new
- Instance Types
- Launch Templates
- Spot Requests
- Savings Plans
- Reserved Instances
- Dedicated Hosts new
- Scheduled Instances
- Capacity Reservations
- Images AMI
- Elastic Block Store Volumes, Snapshots, Lifecycle Manager
- Network & Security Security Groups new, Elastic IPs new, Placement Groups

Welcome to the new instances experience! We're redesigning the EC2 console to make it easier to use. To switch between the old console and the new console, use the New EC2 Experience toggle above the navigation panel. We'll release updates continuously based on customer feedback.

Instances (1/7) Info

Filter instances

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4
gitlab-jenkins-standard	i-02f512445d12b899	Stopped	t2.micro	-	No alarms +	us-west-2a	-	-
JenkinsPipelineInstance	i-02cb17ab0f0f06a66	Stopped	t2.micro	-	No alarms +	us-west-2a	-	-
UbuntuServerAdmin	i-0de4cfabef0d4a7	Stopped	t2.micro	-	No alarms +	us-west-2a	-	-
None	i-08810f7668216991	Stopped	t2.micro	-	No alarms +	us-west-2a	-	-
AWS_TDDI_SQL_VH	i-093262506e408a6d	Stopped	t2.medium	-	No alarms +	us-west-2a	-	-
AWS_EKS	i-0e2615488885e58	Stopped	t2.medium	-	No alarms +	us-west-2c	-	-
BastionHost	i-06c141f68d3e0cc1a	Running	t2.medium	2/2 checks ...	No alarms +	us-west-2c	ec2-59-112-75-21.us... 59.112.75.21	59.112.75.21

Instance: i-06c141f68d3e0cc1a (BastionHost)

Details Security Networking Storage Status Checks Monitoring Tags

Instance summary Info

Instance ID i-06c141f68d3e0cc1a (BastionHost)	Public IPv4 address 59.112.75.21 [open address]	Private IPv4 addresses 172.31.5.196
Instance state Running	Public IPv4 DNS ec2-59-112-75-21.us-west-2.compute.amazonaws.com [open address]	Private IPv4 DNS ip-172-31-5-196.us-west-2.compute.internal
Instance type t2.medium	Elastic IP addresses -	VPC ID vpc-04f8f6f9

© Amazon Web Services

5.3 INSTALL TOOLS IN THE BASTION HOST

Copy your Key .pem file to a convenient location

Navigate to that folder and open a Git Bash if you are using Windows or Open a new Mac OS Terminal

Enter the command below

```
ssh -i BastionHostKeyPair.pem ubuntu@50.112.75.21
```

Note your IP would be different

```
ubuntu@ip-172-31-9-166: ~
bidatta@LNAR-SCG03294K5 MINGW64 /c/AWS/Book/SSH
$ ssh -i BastionHostKeyPair.pem ubuntu@50.112.75.21
The authenticity of host '50.112.75.21 (50.112.75.21)' can't be established.
ECDSA key fingerprint is SHA256:YvHVn1PavOlvNlIxEvzPYV2CTv29mATRut+0Mj2gLkw.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '50.112.75.21' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1029-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 System information as of Sun Jan  3 20:12:03 UTC 2021

 System load:  0.0          Processes:      98
 Usage of /:   14.6% of 7.69GB  Users logged in:  0
 Memory usage: 5%
 Swap usage:   0%
                                         IP address for eth0: 172.31.9.166

0 packages can be updated.
0 updates are security updates.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-9-166:~ |
```

sudo apt-get update

```
ubuntu@ip-172-31-9-166:~$ sudo apt-get update
Hit:1 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic InRelease
Get:2 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:3 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:4 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic/universe amd64 Packages [8570 kB]
Get:5 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Get:6 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic/universe Translation-en [4941 kB]
Get:7 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic/multiverse amd64 Packages [151 kB]
Get:8 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic/multiverse Translation-en [108 kB]
Get:9 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages [1790 kB]
Get:10 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic-updates/main Translation-en [376 kB]
Get:11 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic-updates/restricted amd64 Packages [206 kB]
Get:12 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic-updates/restricted Translation-en [27.9 kB]
Get:13 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages [1697 kB]
Get:14 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [1453 kB]
Get:15 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic-updates/universe Translation-en [357 kB]
Get:16 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic-updates/multiverse amd64 Packages [35.6 kB]
Get:17 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic-updates/multiverse Translation-en [7180 B]
Get:18 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic-backports/main amd64 Packages [10.0 kB]
Get:19 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic-backports/main Translation-en [4764 B]
Get:20 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic-backports/universe amd64 Packages [10.3 kB]
Get:21 http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic-backports/universe Translation-en [4588 B]
Get:22 http://security.ubuntu.com/ubuntu bionic-security/main Translation-en [284 kB]
Get:23 http://security.ubuntu.com/ubuntu bionic-security/restricted amd64 Packages [185 kB]
Get:24 http://security.ubuntu.com/ubuntu bionic-security/restricted Translation-en [24.3 kB]
Get:25 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages [1094 kB]
Get:26 http://security.ubuntu.com/ubuntu bionic-security/universe Translation-en [244 kB]
Get:27 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 Packages [12.8 kB]
Get:28 http://security.ubuntu.com/ubuntu bionic-security/multiverse Translation-en [2872 B]
Fetched 21.8 MB in 4s (5295 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-9-166:~$
```

Enter the following command

`sudo apt install docker.io`

Enter Y when asked

```

  etched 21.8 MB in 4s (5295 kB/s)
  reading package lists... Done
  ubuntu@ip-172-31-9-166:~$ sudo apt install docker.io
  reading package lists... Done
  building dependency tree
  reading state information... Done
  The following additional packages will be installed:
    bridge-utils cgroupfs-mount containerd pigz runc ubuntu-fan
  suggested packages:
    ifupdown aufs-tools debbootstrap docker-dhc rinse zfs-fuse | zfsutils
  The following NEW packages will be installed:
    bridge-utils cgroupfs-mount containerd docker.io pigz runc ubuntu-fan
  upgraded, 7 newly installed, 0 to remove and 50 not upgraded.
  need to get 63.7 MB of archives.
  After this operation, 319 MB of additional disk space will be used.
  Do you want to continue? [Y/n] Y
  get: http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic/universe amd64 pigz amd64 2.4-1 [57.4 kB]
  get: http://us-west-2.ec2.archive.ubuntu.com/ubuntu main amd64 bridge-utils amd64 1.5-1ubuntu1 [50.1 kB]
  get: 3: http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic/universe amd64 cgroupfs-mount all 1.4 [6320 B]
  get: 4: http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 runc amd64 1.0.0-rc10-Ubuntu1-18.04.2 [2000 kB]
  get: 5: http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 containerd amd64 1.3.3-1Ubuntu1-18.04.2 [21.7 kB]
  get: 6: http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 docker.io amd64 19.03.6-0ubuntu1-18.04.2 [39.9 kB]
  get: 7: http://us-west-2.ec2.archive.ubuntu.com/ubuntu bionic/main amd64 ubuntu-fan all 0.12.10 [34.7 kB]
  etched 63.7 MB in 2s (35.9 MB/s)
  reconfiguring packages ...
  selecting previously unselected package pigz.
  Reading database ... 57090 files and directories currently installed.
  preparing to unpack .../0-pigz_2.4-1_amd64.deb ...
  unpacking pigz (2.4-1) ...
  selecting previously unselected package bridge-utils.
  preparing to unpack .../1-bridge-utils_1.5-1ubuntu1_amd64.deb ...
  unpacking bridge-utils (1.5-1ubuntu1) ...
  selecting previously unselected package cgroupfs-mount.
  preparing to unpack .../2-cgroupfs-mount_1.4_all.deb ...
  unpacking cgroupfs-mount (1.4) ...
  selecting previously unselected package runc.
  preparing to unpack .../3-runc_1.0.0-rc10-Ubuntu1-18.04.2_amd64.deb ...
  unpacking runc (1.0.0-rc10-Ubuntu1-18.04.2) ...
  selecting previously unselected package containerd.
  preparing to unpack .../4-containerd_1.3.3-1Ubuntu1-18.04.2_amd64.deb ...
  unpacking containerd (1.3.3-1Ubuntu1-18.04.2) ...
  selecting previously unselected package docker.io.
  preparing to unpack .../5-docker.io_19.03.6-0ubuntu1-18.04.2_amd64.deb ...
  unpacking docker.io (19.03.6-0ubuntu1-18.04.2) ...
  Progress: 44% [=====>


```

Test Docker

`sudo docker run hello-world`

```

ubuntu@ip-172-31-9-166:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:1a523af650137b8accdaed439c17d684df61ee4d74feac151b5b337bd29e7eec
Status: Downloaded newer image for hello-world:latest

```

```

Hello from Docker!
This message shows that your installation appears to be working correctly.


```

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
`$ docker run -it ubuntu bash`

Share images, automate workflows, and more with a free Docker ID:
<https://hub.docker.com/>

For more examples and ideas, visit:
<https://docs.docker.com/get-started/>

```
ubuntu@ip-172-31-9-166:~$ |
```

`sudo apt install unzip`

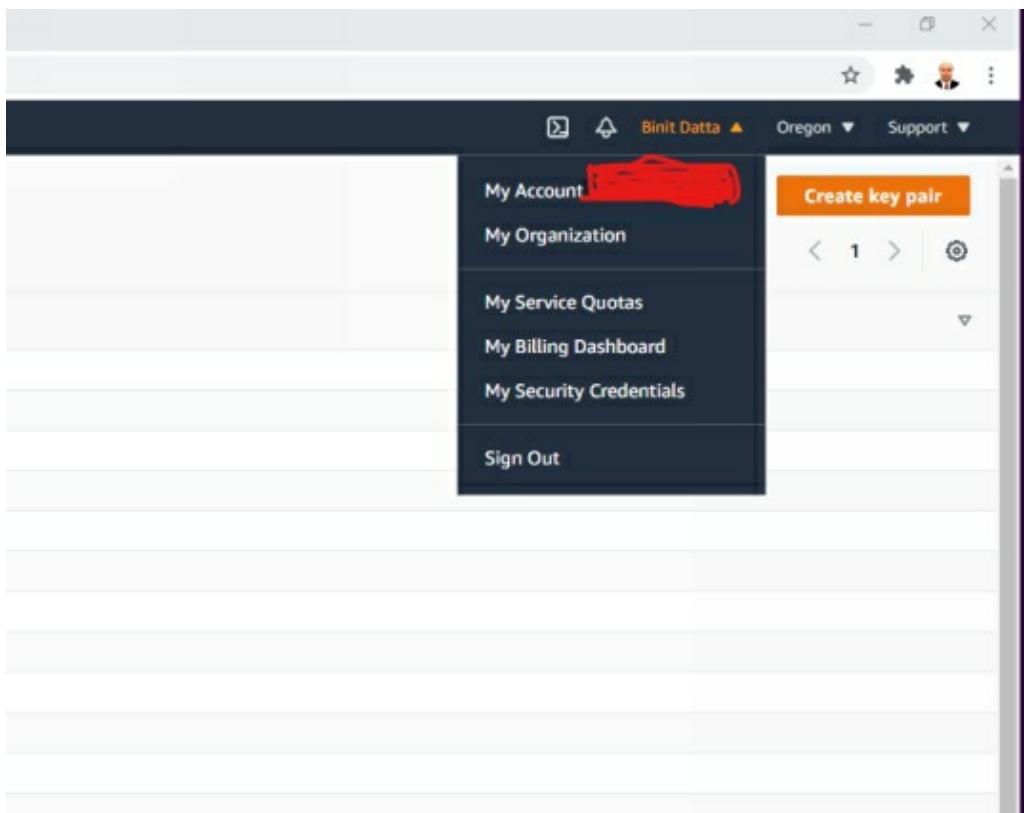
`curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"`

```
unzip awscliv2.zip
```

```
sudo ./aws/install
```

```
aws --version
```

Click on Your Name on the right side to get the image below



© Amazon Web Services

Click on My Security Credentials

© Amazon Web Services

Open the Panel Access keys (access key ID and secret access key)

Click on Create New Access Key



© Amazon Web Services

Click on Download Key File as you will need them soon. Keep the Key very secure

Go back to your SSH Sessions and enter

aws configure to enter the details

```
ubuntu@ip-172-31-9-166:~$ sudo ./aws/install
You can now run: /usr/local/bin/aws --version
ubuntu@ip-172-31-9-166:~$ aws --version
aws-cli/2.1.15 Python/3.7.3 Linux/5.4.0-1029-aws exe/x86_64/ubuntu.18 prompt/off
ubuntu@ip-172-31-9-166:~$ ^C
ubuntu@ip-172-31-9-166:~$ aws configure
AWS Access Key ID [None]: [REDACTED]
AWS Secret Access Key [None]: [REDACTED]
Default region name [None]: us-west-2
Default output format [None]: json
ubuntu@ip-172-31-9-166:~$
```

Try a command to test your configuration

aws ec2 describe-regions --output table

DescribeRegions			
Regions			
Endpoint	OptInStatus	RegionName	
ec2.eu-north-1.amazonaws.com	opt-in-not-required	eu-north-1	
ec2.ap-south-1.amazonaws.com	opt-in-not-required	ap-south-1	
ec2.eu-west-3.amazonaws.com	opt-in-not-required	eu-west-3	
ec2.eu-west-2.amazonaws.com	opt-in-not-required	eu-west-2	
ec2.eu-west-1.amazonaws.com	opt-in-not-required	eu-west-1	
ec2.ap-northeast-2.amazonaws.com	opt-in-not-required	ap-northeast-2	
ec2.ap-northeast-1.amazonaws.com	opt-in-not-required	ap-northeast-1	
ec2.sa-east-1.amazonaws.com	opt-in-not-required	sa-east-1	
ec2.ca-central-1.amazonaws.com	opt-in-not-required	ca-central-1	
ec2.ap-southeast-1.amazonaws.com	opt-in-not-required	ap-southeast-1	
ec2.ap-southeast-2.amazonaws.com	opt-in-not-required	ap-southeast-2	
ec2.eu-central-1.amazonaws.com	opt-in-not-required	eu-central-1	
ec2.us-east-1.amazonaws.com	opt-in-not-required	us-east-1	
ec2.us-east-2.amazonaws.com	opt-in-not-required	us-east-2	
ec2.us-west-1.amazonaws.com	opt-in-not-required	us-west-1	
ec2.us-west-2.amazonaws.com	opt-in-not-required	us-west-2	

Pull a Docker Image we will use later

```
sudo docker pull amazonlinux:2
```

Install Git on your Ubuntu Bastion Host

```
sudo apt-get install git
```

Install Oracle JDK 15

Sales and apt. responses

The packages in this IPA are based on the [macports-brainiac](#) Java IPA packages' https://github.com/brainiac/macports-brainiac/tree/master/java

Installation instructions (with some trial), feedback, suggestions, bug reports etc:

Oracle Java 8: <https://www.java.com/en/download/manual.jsp>

Oracle Java 6: <https://www.java.com/en/download/manual.jsp?version=6>

Enter

sudo apt update

```
sudo apt install oracle-java15-installer
```

Press Tab to be on the OK button and Press Enter

Press Tab to Navigate to Yes and Click Enter to Accept Oracle License

```
ubuntu@ip-172-31-9-166:~$ java -version
```

```
java version "15.0.1" 2020-10-20
```

```
Java(TM) SE Runtime Environment (build 15.0.1+9-18)
```

```
Java HotSpot(TM) 64-Bit Server VM (build 15.0.1+9-18, mixed mode, sharing)
```

Install Gradle on Ubuntu

Visit this page

<https://services.gradle.org/distributions/>

Enter the following command in your terminal

```
sudo wget https://services.gradle.org/distributions/gradle-6.7.1-bin.zip
```

Enter

```
unzip gradle-6.7.1-bin.zip
```

You can install sudo apt-get install unzip if unzip is not there

Edit vi .bashrc

And add the two lines at the bottom of the file

```
export GRADLE_HOME=/home/ubuntu/gradle-6.7.1
export PATH=$GRADLE_HOME/bin:$PATH
```

```
ubuntu@ip-172-31-9-166: ~
ubuntu@ip-172-31-9-166:~$ gradle -v
Welcome to Gradle 6.7.1

Here are the highlights of this release:
- File system watching is ready for production use
- Declare the version of Java your build requires
- Java 15 support

For more details see https://docs.gradle.org/6.7.1/release-notes.html

-----
Gradle 6.7.1
-----
Build time: 2020-11-16 17:09:24 UTC
Revision: 2972ff02f3210d2ceed2f1ea880f026acfbab5c0

Kotlin: 1.3.72
Groovy: 2.5.12
Ant: Apache Ant(TM) version 1.10.8 compiled on May 10 2020
JVM: 15.0.1 (Oracle Corporation 15.0.1+9-18)
OS: Linux 5.4.0-1029-aws amd64

ubuntu@ip-172-31-9-166:~$ |
```

5.4 CREATING AWS RDS DATABASE

Log back on to your AWS console

The screenshot shows the AWS Management Console homepage. At the top, there's a navigation bar with links for 'AWS Services', 'Search for services, features, marketplace products, and docs', and a user icon. Below the navigation is the 'AWS Management Console' header. On the left, there's a sidebar titled 'AWS services' with sections for 'Recently visited services' (including IAM, RDS, CloudFormation, VPC, and AWS Lambda) and 'All services' (listing Compute, Storage, and Container services like EC2, S3, EBS, Lambda, and others). The main content area displays various AWS services in a grid format, such as Customer Enablement, Machine Learning, AWS Cost Management, Front-end Web & Mobile, Amazon Redshift, Run Serverless Containers with AWS Fargate, Scalable, Durable, Secure Backup & Restore with Amazon S3, and AWS Marketplace. A sidebar on the right encourages users to stay connected on-the-go by downloading the AWS mobile app.

© Amazon Web Services

Find RDS in the Service Catalog and click on it

Amazon RDS

Amazon Aurora

Aurora Aurora is a MySQL- and PostgreSQL-compatible enterprise-class database, starting at \$0.10/day. Aurora supports up to 64TB of auto-scaling storage capacity, 6-way replication across three availability zones, and 10-second latency read replicas. Learn more.

[Create database](#)

Or, restore Aurora DB cluster from S3

Resources

You are using the following Amazon RDS resources in the US West (Oregon) region (last updated)

- DB Instances (1148)
 - Allocated storage 10.82 TB/100 TB
 - Click here to increase DB instances limit
- DB Clusters (8400)
 - Reserved Instances (5448)
 - Snapshots (33)
 - Manual (30)
 - Automated (3)
 - Recent events (4)
 - Event subscriptions (6239)
- Parameter groups (31)
 - Default (1)
 - Custom (24)
- Option groups (3)
 - Default (1)
 - Custom (2)
- Subnet groups (11758)
 - Supported platforms: MySQL
 - Default network: ipo-5cfebed9

Additional information

Getting started with RDS
Overview and Notices
Documentation
Articles and tutorials
Data import guide for MySQL
Data import guide for Oracle
Data import guide for SQL Server
New RDS feature announcements
Pricing
Forums

Database Preview Environment

Get early access to new DB engine versions, before they're generally available. The RDS preview environment lets you work with experimental database engines and preview features for MySQL and PostgreSQL engines. Preview environment instances are fully functional, so you can easily test new features and functionality with your applications.

[Preview PostgreSQL in US EAST \(Ohio\)](#)

© Amazon Web Services

Click on Create Database

us-west-2.console.aws.amazon.com/rds/home?region=us-west-2#launch-dbinstance:gdb=false:s3-import=false

RDS > Create database

Create database

Choose a database creation method

Standard create
You set all of the configuration options, including ones for availability, security, backups, and maintenance.

Easy create
Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

Engine options

Engine type

Amazon Aurora



MySQL



MariaDB



PostgreSQL



Oracle



Microsoft SQL Server



Edition

Amazon Aurora with MySQL compatibility

Amazon Aurora with PostgreSQL compatibility

ⓘ Aurora MySQL engine versions earlier than 2.09.1 don't support the newest r6g generation instance classes.

© Amazon Web Services

Choose MySQL

RDS > Create database

Create database

Choose a database creation method Info

Standard create
You set all of the configuration options, including ones for availability, security, backups, and maintenance.

Easy create
Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

Engine options

Engine type Info

Amazon Aurora 

MySQL 

MariaDB 

PostgreSQL 

Oracle 

Microsoft SQL Server 

© Amazon Web Services

Select MySQL 8.x

Edition

MySQL Community

Known Issues/Limitations
Review the [Known Issues/Limitations](#)  to learn about potential compatibility issues with specific database versions.

Version

MySQL 8.0.20 

© Amazon Web Services

Select Dev/Test

Templates
Choose a sample template to meet your use case.

Production
Use defaults for high availability and fast, consistent performance.

Dev/Test
This instance is intended for development use outside of a production environment.

Free tier
Use RDS Free Tier to develop new applications, test existing applications, or gain hands-on experience with Amazon RDS.
[Info](#)

© Amazon Web Services

DB Settings Details

- DB Identifier : rs-mortgage-aws
- Master username : admin (Remember this is only a test DB)
- Master Password : admin1973

Settings

DB instance identifier [Info](#)
Type a name for your DB instance. The name must be unique across all DB instances owned by your AWS account in the current AWS Region.

rs-mortgage-aws

The DB instance identifier is case-insensitive, but is stored as all lowercase (as in "mydbinstance"). Constraints: 1 to 60 alphanumeric characters or hyphens (1 to 15 for SQL Server). First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

Credentials Settings

Master username [Info](#)
Type a login ID for the master user of your DB instance.

admin

1 to 16 alphanumeric characters. First character must be a letter

Auto generate a password
Amazon RDS can generate a password for you, or you can specify your own password

Master password [Info](#)

Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), '(single quote), "(double quote) and @ (at sign).

Confirm password [Info](#)

© Amazon Web Services

Keep the Default for Instance Size

DB instance size

DB instance class [Info](#)

Choose a DB instance class that meets your processing power and memory requirements. The DB instance class options below are limited to those supported by the engine you selected above.

Standard classes (includes m classes)
 Memory Optimized classes (includes r and x classes)
 Burstable classes (includes t classes)

db.m5.xlarge
4 vCPUs 16 GiB RAM Network: 4,750 Mbps

ⓘ New instance classes are available for specific engine versions. [Info](#)

Include previous generation classes

© Amazon Web Services

Keep the defaults for Storage

Storage

Storage type [Info](#)

General Purpose (SSD)

Allocated storage
20 GiB
(Minimum: 20 GiB, Maximum: 65,536 GiB) Higher allocated storage [may improve](#) IOPS performance.

ⓘ Provisioning less than 100 GiB of General Purpose (SSD) storage for high throughput workloads could result in higher latencies upon exhaustion of the initial General Purpose (SSD) IO credit balance. [Learn more](#)

Storage autoscaling [Info](#)

Provides dynamic scaling support for your database's storage based on your application's needs.

Enable storage autoscaling
Enabling this feature will allow the storage to increase once the specified threshold is exceeded.

Maximum storage threshold [Info](#)

Charges will apply when your database autoscales to the specified threshold

1000 GiB
Minimum: 21 GiB, Maximum: 65,536 GiB

© Amazon Web Services

Keep the defaults for Availability and Durability

Availability & durability

Multi-AZ deployment [Info](#)

- Create a standby instance (recommended for production usage)
Creates a standby in a different Availability Zone (AZ) to provide data redundancy, eliminate I/O freezes, and minimize latency spikes during system backups.
- Do not create a standby instance

Make it publicly accessible Database but keep all other defaults such as VPC etc.

Connectivity

Virtual private cloud (VPC) [Info](#)
VPC that defines the virtual networking environment for this DB instance.

Default VPC (vpc-6cd9ec09) ▾

Only VPCs with a corresponding DB subnet group are listed.

ⓘ After a database is created, you can't change the VPC selection.

Subnet group [Info](#)
DB subnet group that defines which subnets and IP ranges the DB instance can use in the VPC you selected.

default ▾

Public access [Info](#)

Yes
Amazon EC2 instances and devices outside the VPC can connect to your database. Choose one or more VPC security groups that specify which EC2 instances and devices inside the VPC can connect to the database.

No
RDS will not assign a public IP address to the database. Only Amazon EC2 instances and devices inside the VPC can connect to your database.

VPC security group
Choose a VPC security group to allow access to your database. Ensure that the security group rules allow the appropriate incoming traffic.

Choose existing
Choose existing VPC security groups

Create new
Create new VPC security group

Existing VPC security groups

Choose VPC security groups ▾

default X

Availability Zone [Info](#)

No preference ▾

▶ Additional configuration

© Amazon Web Services

Other Details. Do not worry about the monthly cost as we will delete the DB when we are done with it.

Database authentication

Database authentication options [Info](#)

- Password authentication
Authenticates using database passwords.
- Password and IAM database authentication
Authenticates using the database password and user credentials through AWS IAM users and roles.
- Password and Kerberos authentication (not available for this version)
Choose a directory in which you want to allow authorized users to authenticate with this DB instance using Kerberos Authentication.

► Additional configuration

Database options, encryption enabled, backup enabled, backtrack disabled, Performance Insights enabled, Enhanced Monitoring enabled, maintenance, CloudWatch Logs, delete protection disabled

Estimated monthly costs

DB instance	249.66 USD
Storage	2.30 USD
Total	251.96 USD

This billing estimate is based on on-demand usage as described in [Amazon RDS Pricing](#). Estimate does not include costs for backup storage, IOs (if applicable), or data transfer.

Estimate your monthly costs for the DB Instance using the [AWS Simple Monthly Calculator](#).

ⓘ You are responsible for ensuring that you have all of the necessary rights for any third-party products or services that you use with AWS services.

[Cancel](#) [Create database](#)

© Amazon Web Services

Click on Create Database Button

RDS > Databases

Databases											
Group resources Modify Actions Restore from SS Create database											
<input type="text"/> Filter databases											
DB identifier	Role	Engine	Region & AZ	Size	Status	CPU	Current activity	Maintenance	VPC	Multi-AZ	
bintownmysql	Instance	MySQL Community	us-west-2a	db.m5.large	Available	100%	0 Sessions	none	vpc-4cf9ec09	No	
rs-mortgage-001	Instance	MySQL Community	us-west-2b	db.m5.large	Creating	-	0 Sessions	none	vpc-4cf9ec09	No	

Wait till the status says available

DB Identifier	Role	Engine	Region & AZ	Size	Status	CPU	Current activity	Maintenance	VPC	Multi-AZ
rs-mortgage-aws	Instance	MySQL Community	us-west-2a	db.m5.xlarge	Stopped	0.00%	0 Sessions	None	vpc-6cd9ec09	No
rs-mortgage-aws	Instance	MySQL Community	us-west-2b	db.m5.xlarge	Available	0.00%	0 Sessions	None	vpc-6cd9ec09	No

Let's get the connection details

Click on the DB Identifier Link to go to the DB details page

DB identifier	CPU	Status	Class
rs-mortgage-aws	0.00%	Available	db.m5.xlarge
Role	Current activity	Engine	Region & AZ
Instance	0 Sessions	MySQL Community	us-west-2b

Connectivity & security

Endpoint	Networking	Security
rs-mortgage-aws.civxewyb4pfe.us-west-2.rds.amazonaws.com	Availability zone: us-west-2b	VPC security groups: default (sg-75715811) (active)
Port: 3306	VPC: vpc-6cd9ec09	Public accessibility: Yes
	Subnet group: default	Certificate authority: rd-ca-2019
	Subnets: subnet-15752964, subnet-1a0ef4d5, subnet-1eadd37b	Certificate authority date: Aug 22nd, 2024

Endpoint

rs-mortgage-aws.civxewyb4pfe.us-west-2.rds.amazonaws.com

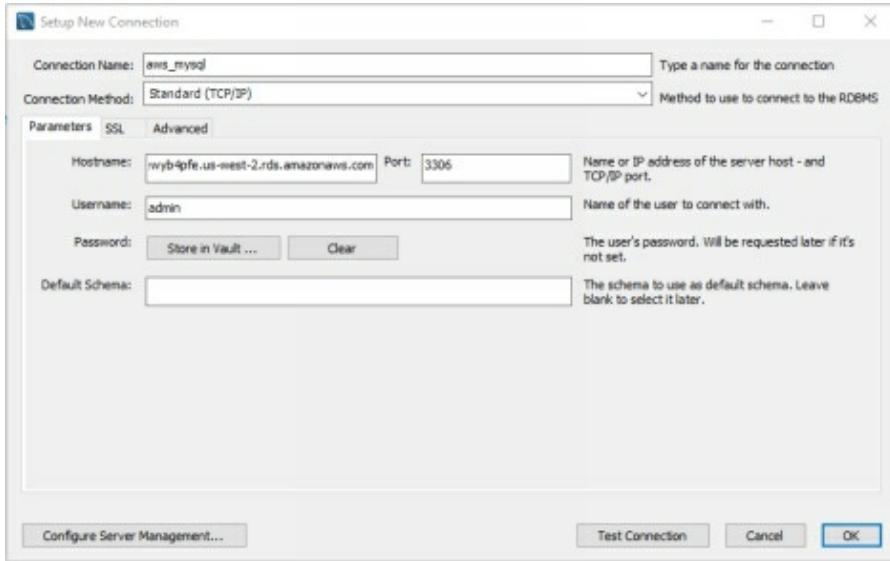
port: 3306

Start your local MySQL Workbench

Click on the + Icon to open a New Connection Dialog. Enter the endpoint URL on the Host text box.

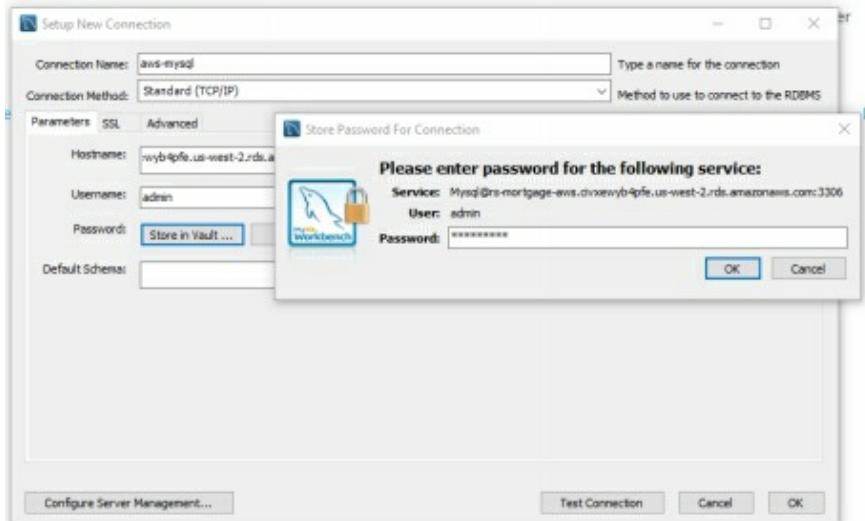
Username is admin

Password is admin1973

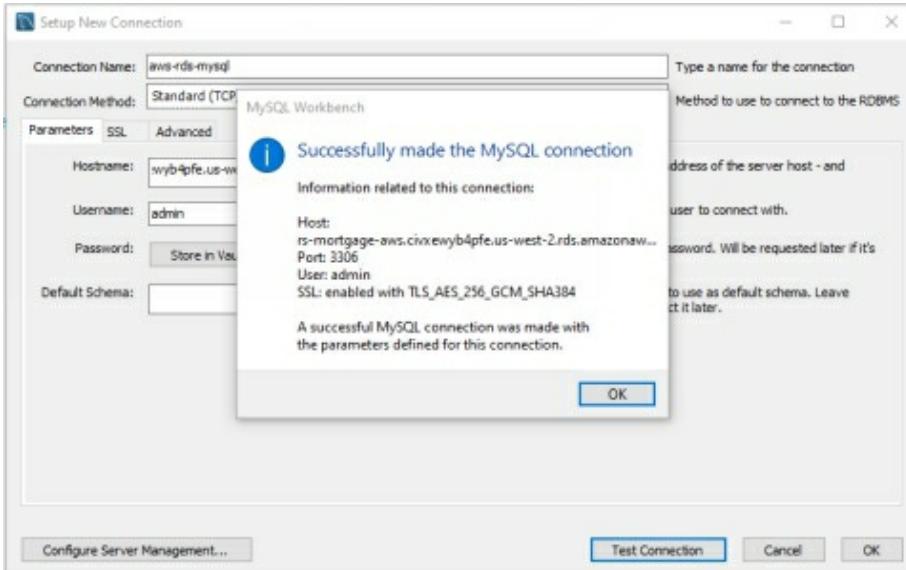


© Amazon Web Services

Click on Store In Vault to provide the password



On normal occasions you will see this if you click on Test Connection



Click OK to close the dialog.

You may notice MySQL Workbench may not be able to connect to the AWS MySQL. Let's configure the DB Security Groups

Security group rules (2)			
Security group	Type	Rule	Actions
default (sg-75715811)	EC2 Security Group - Inbound	sg-75715811	
default (sg-75715811)	ODR/IP - Outbound	0.0.0.0/0	

© Amazon Web Services

Click on the first Security Group to visit

The screenshot shows the AWS EC2 Security Groups list. There is one item in the table:

Name	Security group ID	Security group name	VPC ID	Description	Owner	Inbound rules count	Outbound rules count
-	sg-75715811	default	vpc-decided01	default VPC security gr...	1072226764	1 Permission entry	1 Permission entry

Below the table, a detailed view of the 'sg-75715811 - default' security group is shown. The 'Inbound rules' tab is selected.

© Amazon Web Services

Click on Inbound Tab

The screenshot shows the AWS EC2 Security Group details page for 'sg-75715811 - default'. The 'Inbound rules' tab is selected. It displays a single rule:

Type	Protocol	Port range	Source	Description - optional
All traffic	All	All	sg-75715811 (default)	-

© Amazon Web Services

Click on Edit Rules

The screenshot shows the 'Edit inbound rules' dialog box. It has tabs for 'Type', 'Protocol', 'Port range', 'Source', and 'Description - optional'. The 'Type' tab is selected, showing 'All traffic'. The 'Source' field contains 'sg-75715811'. A note at the bottom states: 'NOTE: Any edits made on existing rules will result in the edited rule being deleted and a new rule created with the new details. This will cause traffic that depends on that rule to be dropped for a very brief period of time until the new rule can be created.' Buttons at the bottom include 'Cancel', 'Preview changes', and 'Save rules'.

© Amazon Web Services

Click on Add Rule, enter 3306 as port and make source Anywhere

Edit inbound rules [Info](#)

Inbound rules control the incoming traffic that's allowed to reach the instance.

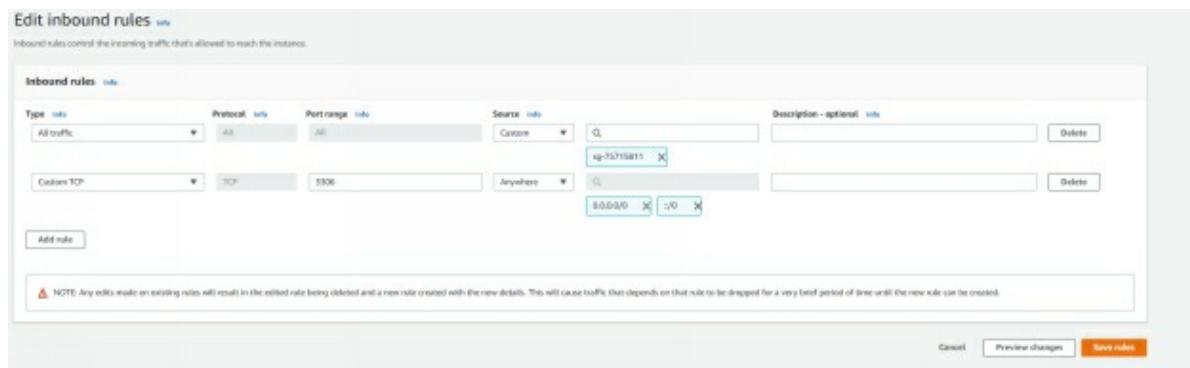
Inbound rules [Info](#)

Type	Protocol	Port range	Source	Description - optional
All traffic	All	All	Custom	<input type="text" value="sg-70210811"/> Delete
Custom TCP	TCP	3306	Anywhere	<input type="text" value="0.0.0.0/0 :3306"/> Delete

[Add rule](#)

⚠ NOTE: Any edits made on existing rules will result in the edited rule being deleted and a new rule created with the new details. This will cause traffic that depends on that rule to be dropped for a very brief period of time until the new rule can be created.

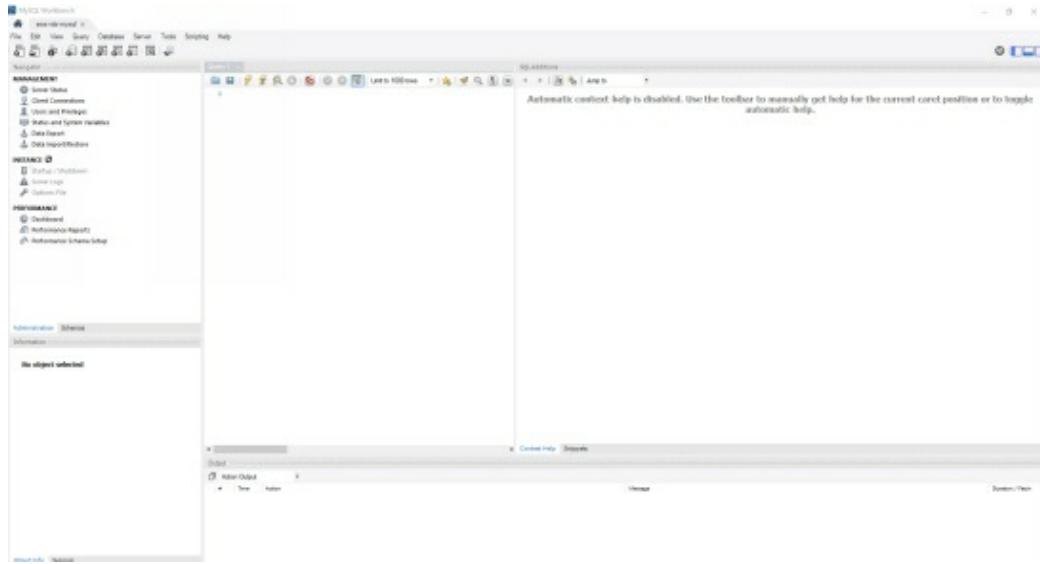
[Cancel](#) [Preview changes](#) **Save rules**



© Amazon Web Services

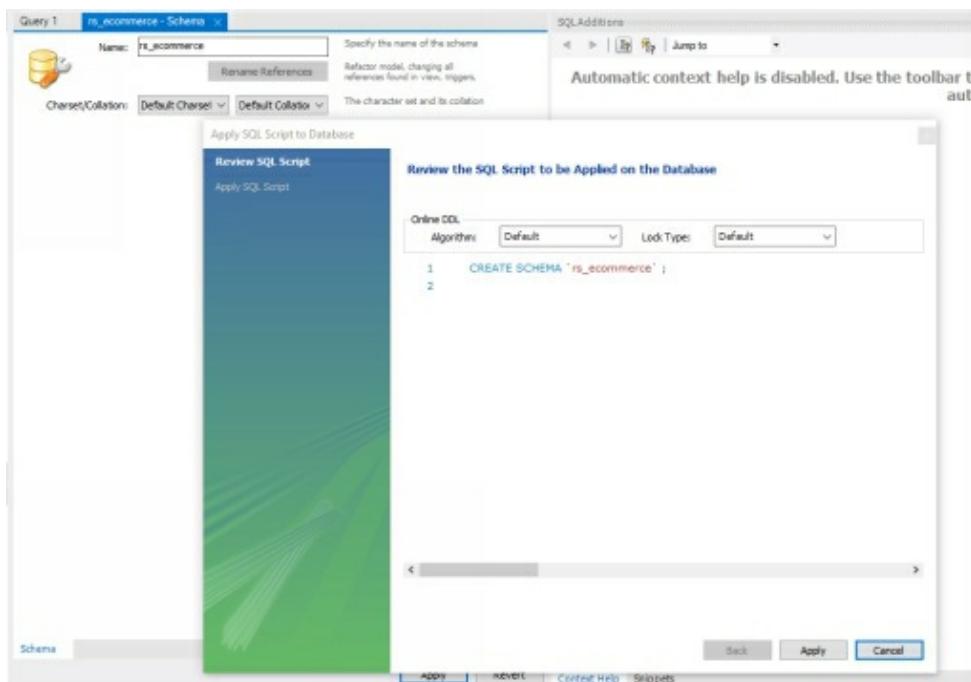
Click on Save Rules

5.5 ACCESSING AWS RDS DATABASE INSTANCE FROM LOCAL



© Amazon Web Services

Let's create the database and tables in AWS RDS MySQL

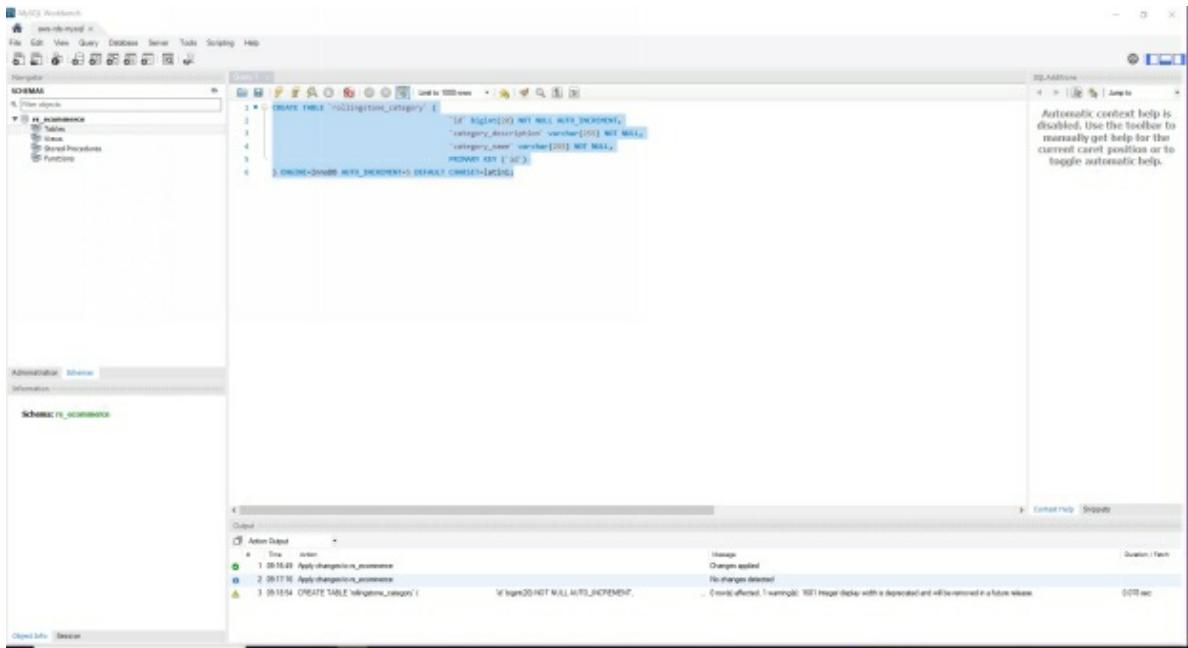


© Amazon Web Services

Click on Apply

Click on the Schemas Tab on Workbench, right click on the new Schema and Select Set As Default Schema

Find the ddl.sql file in your IDE under the datascripts folder, copy the Create Table statement and paste in your Workbench



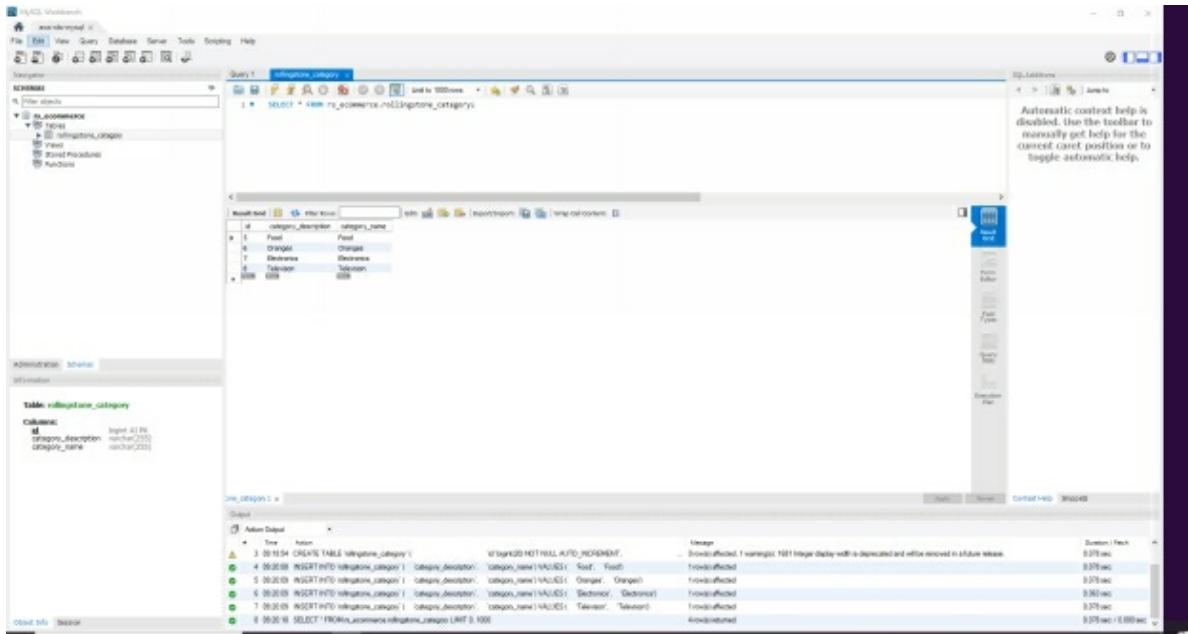
The screenshot shows the MySQL Workbench interface. The SQL Editor pane contains the following SQL code:

```
1 CREATE TABLE `voltageincategory` (
2     `id` int(10) NOT NULL AUTO_INCREMENT,
3     `category_description` varchar(100) NOT NULL,
4     `category_name` varchar(100) NOT NULL,
5     PRIMARY KEY (`id`),
6 ) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

The History tab at the bottom shows the execution details:

Action	Detailed	Message	Duration
1	Task: update	Changes applied.	0.00 sec
2	39-11-09 Apply changes in environment	No changes detected.	
3	39-11-04 CREATE TABLE `voltageincategory`	0 rows affected, 1 warning, 1027 longer display width is deprecated and will be removed in a future release.	0.00 sec

Similarly find the data.sql file in the same folder, and execute the insert statement in the AWS RDS table through Workbench



Now we have two databases, one in local machine and another in AWS RDS. The databases have two different set of endpoint, username and passwords. We have only one property file though. Spring Boot provides a nice feature called Spring profile which we will use to be able to run our application against both local and aws databases.

Copy your application-yaml to a new file called application-aws.yaml

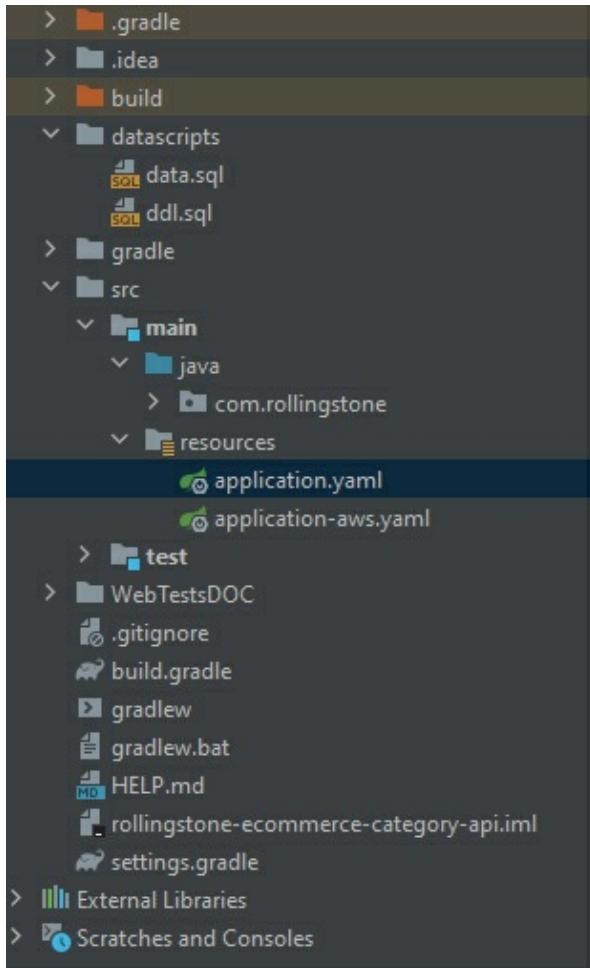
Change the following on the new application-aws.yaml

```

url: jdbc:mysql://rs-mortgage-aws.civxewyb4pfe.us-west-2.rds.amazonaws.com:3306/rs_ecommerce
username: admin
password: admin1973

```

- url should be your AWS RDS DB Endpoint
- username is admin
- password is admin1973



Build the file

```
c:\Development\rollingstone-commerce-category-api>gradle clean build -x test
Starting a Gradle Daemon (subsequent builds will be faster)

> Task :compileJava
Note: C:\Development\rollingstone-commerce-category-api\src\main\java\com\rollingstone\config\SpringFoxConfigForCategory.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

BUILD SUCCESSFUL in 14s
5 actionable tasks: 5 executed
C:\Development\rollingstone-commerce-category-api>[]
```

Run the application with the following command

```
java -jar -Dspring.profiles.active=aws build\libs\rollingstone-commerce-category-api-1.0.jar
```

Application runs properly. Look at the “The following profiles are active” line

```
Terminal Local ✘
=====
:: Spring Boot ::          (v2.4.1)

2021-01-04 09:35:53.663 INFO 22000 --- [           main] lingstonecommerceCategoryApiApplication : Starting RollingstoneCommerceCategoryApiApplication using Java 15.0.1 on iNAR-5C60329
2021-01-04 09:35:55.067 INFO 22000 --- [           main] lingstonecommerceCategoryApiApplication : The following profiles are active: aws
2021-01-04 09:35:55.840 INFO 22000 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2021-01-04 09:35:55.945 INFO 22000 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 96 ms. Found 1 JPA repository interfaces.
2021-01-04 09:35:57.256 INFO 22000 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8092 (http)
2021-01-04 09:35:57.281 INFO 22000 --- [           main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2021-01-04 09:35:57.281 INFO 22000 --- [           main] o.apache.catalina.core.StandardEngine : Starting servlet engine: [Apache Tomcat/9.0.41]
2021-01-04 09:35:57.427 INFO 22000 --- [           main] o.a.c.c.C[Tomcat].[_/] : Initializing Spring embedded WebApplicationContext
2021-01-04 09:35:57.428 INFO 22000 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 3464 ms
2021-01-04 09:35:58.045 INFO 22000 --- [           main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
```

Make a new Entry to the AWS RDS DB to distinguish it from the localo DB

INSERT INTO `rollingstone_category`

(

`category_description`,

`category_name`)

VALUES

(

'Computer',

'Laptop');

Run postman to test the AWS RDS connectivity

Unfinished

Untitled Request

GET http://localhost:8080/category [DELETED] GET http://localhost:8082... GET http://localhost:8082/category

No Environment

Send Save Cookies Code

Raw Authorization Headers (10) Body Pre-request Script Tests Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results

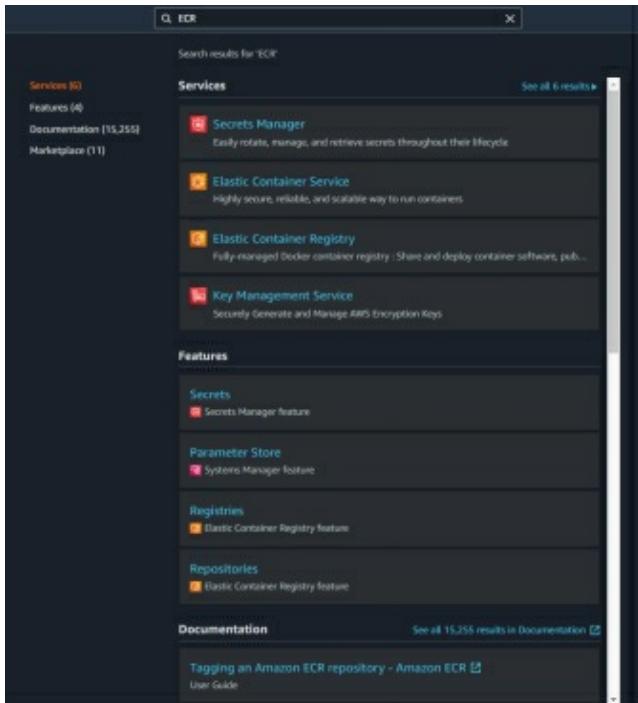
Pretty Raw Preview Visualize JSON

Status: 200 OK Time: 1036 ms Size: 317.8 Save Response

```
4   {
5     "id": 5,
6     "categoryName": "Television",
7     "categoryDescription": "Television"
8   },
9   {
10    "id": 6,
11    "categoryName": "Oranges",
12    "categoryDescription": "Oranges"
13  },
14  {
15    "id": 8,
16    "categoryName": "Laptop",
17    "categoryDescription": "Computer"
18  },
19  {
20    "id": 9,
21    "categoryName": "Food",
22    "categoryDescription": "Food"
23  },
24  {
25    "id": 7,
26    "categoryName": "Electronics",
27  }
```

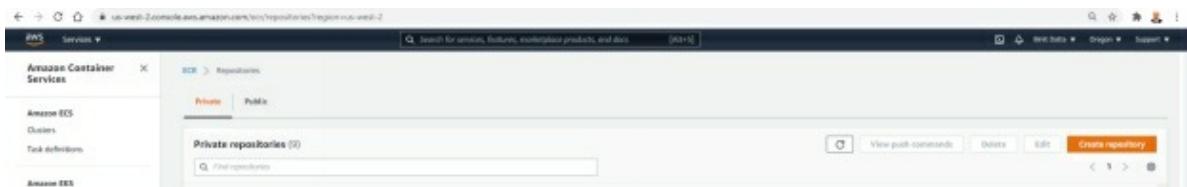
5.6 CREATING AN AWS ECR

On the top search box type ECR



© Amazon Web Services

Click on ECR to open



© Amazon Web Services

Click on Create Repository Button

Make it Private and Name it

General settings

Visibility settings [Info](#)
Choose the visibility setting for the repository.

Private
Access is managed by IAM and repository policy permissions.

Public
Publicly visible and accessible for image pulls.

Repository name
Provide a concise name. A developer should be able to identify the repository contents by the name.

107222267664.dkr.ecr.us-west-2.amazonaws.com/

28 out of 256 characters maximum (2 minimum). The name must start with a letter and can only contain lowercase letters, numbers, hyphens, underscores, and forward slashes.

Tag immutability [Info](#)
Enable tag immutability to prevent image tags from being overwritten by subsequent image pushes using the same tag. Disable tag immutability to allow image tags to be overwritten.

Disabled

Once a repository is created, the visibility setting of the repository can't be changed.

© Amazon Web Services

Leave everything else blank and click on create repository

About - optional Info

Provide a detailed description of the repository. Identify what is included in the repository, any licensing details, or other relevant information.

0 out of 10,240 characters maximum. Use GitHub Flavored Markdown format for the text. [Learn more](#)

[Preview](#)

Usage - optional Info

Provide detailed information about how to use the images in the repository. This provides context, support information, and additional usage details for users of the repository.

0 out of 10,240 characters maximum. Use GitHub Flavored Markdown format for the text. [Learn more](#)

[Preview](#)

[Cancel](#)
Create repository

© Amazon Web Services

Repository Created

Successfully created repository aws-ecr-spring-boot						
Search for services, features, marketplace products, and docs <small>(Alt+S)</small> View push commands						
ECR > Repositories						
Private Public						Create repository
Private repositories (10)	<input type="button" value="View push commands"/>	<input type="button" value="Delete"/>	<input type="button" value="Edit"/>	<input type="button" value="Create repository"/>	< 1 >	<small>1</small>
Repository name	URI	Created at	Tag immutability	Scan on push	Encryption type	
aws-ecr-spring-boot	107222267668.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot	01/04/21, 10:18:04 AM	Disabled	Disabled	AES-256	

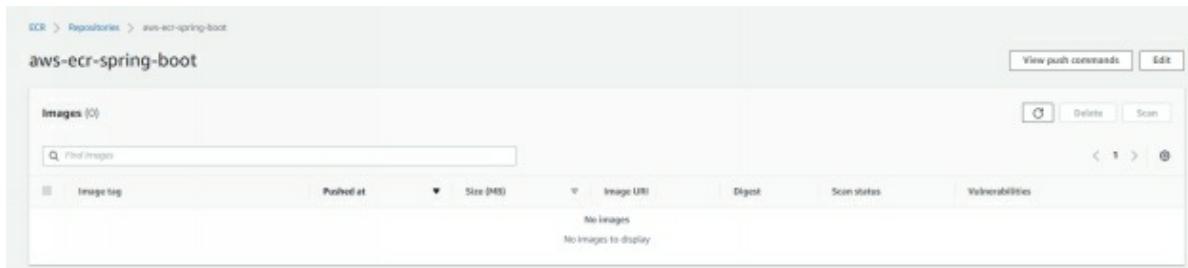
© Amazon Web Services

We will need the repository URL later

Note it somewhere in a text file

< your-aws-acct-id>.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-category

Click on the repository link to view details



The screenshot shows the AWS ECR (Amazon Elastic Container Registry) interface. At the top, the path 'ECR > Repositories > aws-ecr-spring-boot' is visible, followed by the repository name 'aws-ecr-spring-boot'. To the right are buttons for 'View push commands' and 'Edit'. Below this, the 'Images (0)' section is shown, featuring a search bar labeled 'Find images'. A table header with columns 'Image tag', 'Pushed at', 'Size (MB)', 'Image URI', 'Digest', 'Scan status', and 'Vulnerabilities' is present. A message 'No Images' and 'No Images to display' is centered in the table body. On the far right of the table are icons for 'Delete' and 'Scan'. Navigation arrows and a refresh icon are located at the bottom right of the main content area.

© Amazon Web Services

5.7 CREATING AN AWS EKS CLUSTER

It is a good practice for AWS lab sessions that take multiple days, to shut the EC2 Bastion Host at the end of the day. I did it and now I need to restart it to install the EKS cluster

The screenshot shows the AWS EC2 Instances console. At the top, a message says "Welcome to the new Instances experience! We're redesigning the EC2 console to make it easier to use. To switch between the old console and the new console, use the New EC2 Experience toggle above the navigation panel. We'll release updates continuously based on customer feedback." Below this, the "Instances (1/7) - Info" section displays a table of seven instances. One instance, "BastionHost", is selected and highlighted with a blue border. The table columns include Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, Public IPv4 DNS, Public IPv4 address, and Elastic IP. The "BastionHost" row shows the following details:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 address	Elastic IP
BentJenkinsStarted	i-062f513445d12b999	Stopped	t2.micro	-	No alarms	us-west-2c	-	-	-
JenkinsPipDependencies	i-0301138ab0f9a664	Stopped	t2.micro	-	No alarms	us-west-2a	-	-	-
BentKubernetesAdmin	i-08044c5bb0a04bc87	Stopped	t2.micro	-	No alarms	us-west-2a	-	-	-
Nginx	i-08919f1b080258091	Stopped	t2.micro	-	No alarms	us-west-2a	-	-	-
AWS_EFDD_SQL_VH	i-0f12429bae6bb1d	Stopped	t2.medium	-	No alarms	us-west-2a	-	-	-
AWS_DNS	i-06c2b154c480858c56	Stopped	t2.medium	-	No alarms	us-west-2c	-	-	-
BastionHost	i-06c141f80d5e9cc1a	Stopped	t2.medium	-	No alarms	us-west-2c	-	-	-

Below the table, a modal window titled "Instance: i-06c141f80d5e9cc1a [BastionHost]" is open. It shows the "Details" tab selected. The "Instance summary" section contains the following information:

- Instance ID: i-06c141f80d5e9cc1a (BastionHost)
- Instance state: Stopped
- Instance type: t2.medium
- AWS Compute Optimizer Finding: Opt-in to AWS Compute Optimizer for recommendations. Learn more

The "Public IPv4 address" section shows:

- Public IPv4 address: -
- Public IPv4 DNS: -
- Elastic IP addresses: -
- UVM Role: EC2PowerUserRole

The "Private IPv4 address" section shows:

- Private IPv4 address: 172.31.5.166
- Private IPv4 DNS: ip-172-31-5-166.us-west-2.compute.internal
- VPC ID: vpc-6cf9ec0f
- Subnet ID: subnet-7ef0fe40

Click on Instance State and Click on Start Instance

Welcome to the new Instances experience! We're redesigning the EC2 console to make it easier to use. To switch between the old console and the new console, use the New EC2 Experience toggle above the navigation panel. We'll release updates continuously based on customer feedback.

Instances (1/7) Info

Actions ▾ **Instance state** ▾ **Actions** ▾ **Launch instances** ▾

Start instance

Stop instance

Hibernate instance

Terminate instance

Reboot instance

Public IPv4 ▾ **Elastic IP** ▾

Instance Details

Instance: i-06c141f5d5efcc1a [BastionHost]

Details | Security | Networking | Storage | Status Checks | Monitoring | Tags

Instance summary Info

Instance ID: i-06c141f5d5efcc1a (BastionHost)

Instance state: Stopped

Instance type: t2.medium

AWS Compute Optimizer finding: Opt-in to AWS Compute Optimizer for recommendations. [Learn more]

Instance details Info

Public IPv4 address: -

Private IPv4 address: 172.31.9.166

Public IPv4 DNS: -

Private IPv4 DNS: ip-172-31-9-166.us-west-2.compute.internal

Elastic IP addresses: -

VPC ID: vpc-0cc0ec00

Subnet ID: subnet-1abfa453

IAN Role: EC2PowerUserRole

Subnet ID: subnet-1abfa453

© Amazon Web Services

Wait till the Instance shows Ready 2/2 checks done

Click on the Instance checkout and get the public Ip

BastionHost Info **i-06c141f5d5efcc1a** Running t2.medium 2/2 checks ... No alarms + us-west-2c ec2-54-202-66-40.us... 54.202.66.40 -

Instance: i-06c141f5d5efcc1a [BastionHost]

Details | Security | Networking | Storage | Status Checks | Monitoring | Tags

Instance summary Info

Instance ID: i-06c141f5d5efcc1a (BastionHost)

Instance state: Running

Instance type: t2.medium

AWS Compute Optimizer finding: Opt-in to AWS Compute Optimizer for recommendations. [Learn more]

Instance details Info

Public IPv4 address: 54.202.66.40 [Open address]

Private IPv4 address: 172.31.9.166

Public IPv4 DNS: ec2-54-202-66-40.us-west-2.compute.amazonaws.com [Open address]

Private IPv4 DNS: ip-172-31-9-166.us-west-2.compute.internal

Elastic IP addresses: -

VPC ID: vpc-0cc0ec00

Subnet ID: subnet-1abfa453

IAN Role: EC2PowerUserRole

Subnet ID: subnet-1abfa453

© Amazon Web Services

Start a new Git Bash window at the location of your .pem file that we saved earlier and

Enter (Your IP would be different)

```
ssh -i BastionHostKeyPair.pem ubuntu@54.202.66.40
```

```
$ ssh -i BastionHostKeyPair.pem ubuntu@54.202.66.40
The authenticity of host '54.202.66.40 (54.202.66.40)' can't be established.
ECDSA key fingerprint is SHA256:YvHvn1Pav0IvN1xEvzPYV2CTv29mATRut+0Mj2gLkw.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '54.202.66.40' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1029-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 System information as of Mon Jan  4 16:32:39 UTC 2021

 System load:  0.0          Processes:      105
 Usage of /:   41.3% of 7.69GB  Users logged in:   0
 Memory usage: 7%           IP address for eth0: 172.31.9.166
 Swap usage:   0%

 * Introducing self-healing high availability clusters in MicroK8s.
   Simple, hardened, Kubernetes for production, from RaspberryPi to DC.

   https://microk8s.io/high-availability

19 packages can be updated.
0 updates are security updates.

New release '20.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

*** System restart required ***
Last login: Sun Jan  3 21:15:16 2021 from 69.136.183.76
ubuntu@ip-172-31-9-166:~$ |
```

© Amazon Web Services

AWS now has a nice command line tool to abstract the gory details of creating an EKS cluster. The command line tool is called eksctl

The first thing we need to do is to download and install the eksctl tool with the following command line

Paste the command in your SSH window

```
curl --silent --location
"https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
```

Move the eksctl utility to a location that is in our path

```
sudo mv /tmp/eksctl /usr/local/bin
```

Test the eksctl utility with

We should at least see 0.35.0 or later

Next we need to install kubectl, the Kubernetes command line utility on our Bastion Host in AWS

```
curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.18.9/2020-11-02/bin/linux/amd64/kubectl
```

Apply executable permission to the utility

```
chmod +x ./kubectl
```

Include it in our Path

```
sudo mv ./kubectl /usr/local/bin
```

Test

```
kubectl version --short --client
```

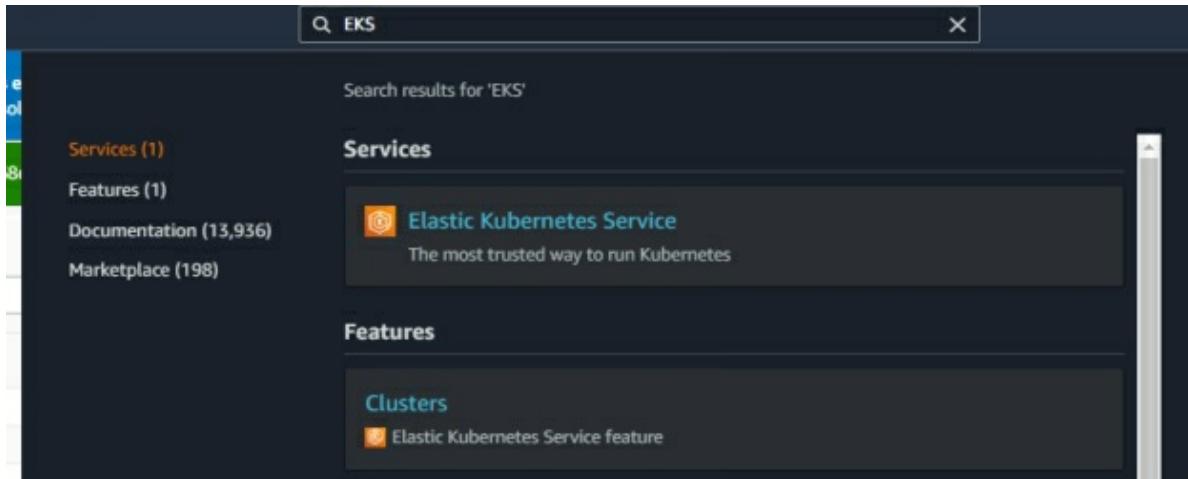
Enter the following command to create a new EKS cluster with eksctl

```
eksctl create cluster \
--name EKS-Cluster-SpringBoot \
--version 1.18 \
--region us-west-2 \
--nodegroup-name linux-nodes \
--nodes 3 \
--nodes-min 1 \
--nodes-max 4 \
--with-oidc \
--ssh-access \
--ssh-public-key Binit_AWS_GS_EKS_KP \
--managed
```

It will take 15-25 minutes, create multiple Cloudformation templates behind the scene. Please wait till it completes.

Now our new EKS cluster is ready

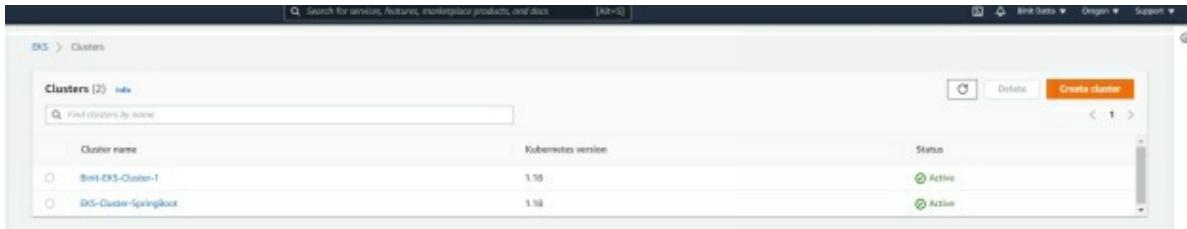
Search for EKS in the search box



© Amazon Web Services

Click on EKS

Click on Clusters on the left pane and see



© Amazon Web Services

Eksctl has created three worker nodes on our behalf



© Amazon Web Services

Eksctl configures out kubectl command line utility automatically. Enter the following command to verify

`kubectl get svc`

```
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
kubernetes  ClusterIP  10.100.0.1  <none>        443/TCP    14m
```

`ubectl get nodes`

```
NAME                  STATUS      ROLES      AGE      VERSION
ip-192-168-20-118.us-west-2.compute.internal  Ready      <none>    8m8s    v1.18.9-eks-d1db3c
ip-192-168-62-235.us-west-2.compute.internal  Ready      <none>    8m9s    v1.18.9-eks-d1db3c
ip-192-168-86-235.us-west-2.compute.internal  Ready      <none>    8m13s   v1.18.9-eks-d1db3c
```

Let's us create a new Git repository to transfer our code from local machine to the Bastion host where we will build the Docker image and work with EKS through the kubectl utility

I did not realize that I already had a git repository (without Docker/EKS) in my Git. So, I have to change the project directory name to rollingstone-ecommerce-category-k8s-api. It would have been ok to have to different repo name and project but let's change for consistency if we have to.

One change we need to do is the settings.gradle file which I did

Let's us add a new file named as Dockerfile at the root of the project

```
FROM adoptopenjdk/openjdk15:alpine-jre
VOLUME /tmp
COPY build/libs/*.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

Let's add another file named as category-kubernetes-deployment.yaml again to the root of the project

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: category-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: category-deployment
  template:
    metadata:
      labels:
        app: category-deployment
    spec:
      containers:
        - name: aws-ecr-spring-boot-category
          image: <your-aws-acct-id>.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-category
          ports:
            - containerPort: 8092
            - containerPort: 8093
        env:
          - name: spring.profiles.active
            value: aws
      imagePullPolicy: Always
```

The screenshot shows a code editor interface with a dark theme. On the left, there's a sidebar with 'Project' and 'Commit' sections. The main area displays two files: 'Dockerfile' and 'deploy-product.yaml'. The 'Dockerfile' content is:

```
FROM adoptopenjdk/openjdk15:alpine-jre
VOLUME /tmp
COPY target/*.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

The 'deploy-product.yaml' file is also visible in the sidebar. The project structure on the left includes '.idea', '.mvn', 'datascripts', 'docs', 'helpdocs', 'src', 'target', '.gitignore', 'mvnw', 'mvnw.cmd', 'pom.xml', 'README.md', and 'rollingstone-ecommerce-product-a'. There are also 'External Libraries' and 'Scratches and Consoles' sections.

Time to push my code to the Git repo

```
echo "# rollingstone-ecommerce-category-k8s-api" >> README.md
```

```
git init
```

```
git add .
```

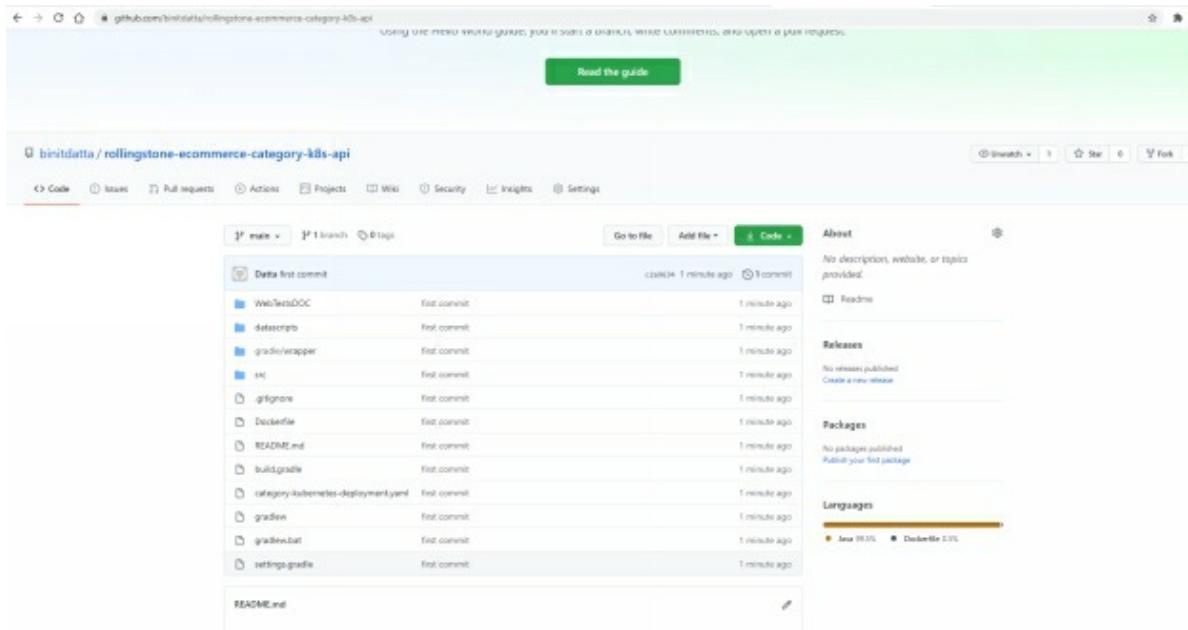
```
git commit -m "first commit"
```

```
git branch -M main
```

```
git remote add origin https://github.com/binitdatta/rollingstone-ecommerce-category-k8s-api.git
```

```
git push -u origin main
```

NOTE: Please replace my Git with your ones appropriately.



If you are not logged in the Bastion Host, please log in and run the following git clone

```
git clone https://github.com/binitdatta/rollingstone-commerce-category-k8s-api.git
```

Change current directory

```
cd rollingstone-commerce-category-k8s-api/
```

Build the java application

```
gradle clean build -x test
```

```
sudo docker build -t aws-ecr-spring-boot-category/latest .
```

Sending build context to Docker daemon 57.33MB

Step 1/4 : FROM adoptopenjdk/openjdk15:alpine-jre

alpine-jre: Pulling from adoptopenjdk/openjdk15

801bfaa63ef2: Already exists

437ac84d5ced: Pull complete

850c82d7c239: Pull complete

Digest:
sha256:15d4c683a3acae21cc49b2ed36f8b443131f8d4aa50c612d9f6465de7f2

Status: Downloaded newer image for adoptopenjdk/openjdk15:alpine-jre
---> 029fb36ffdc7

Step 2/4 : VOLUME /tmp

---> Running in f4624db9c286

Removing intermediate container f4624db9c286

---> 2cefe7ffb8b7

Step 3/4 : COPY build/libs/*.jar app.jar

---> 4bb82a99cc76

Step 4/4 : ENTRYPOINT ["java","-jar","/app.jar"]

---> Running in 96efeb0a4296

Removing intermediate container 96efeb0a4296

---> 6af696c2fa13

Successfully built 6af696c2fa13

Successfully tagged aws-ecr-spring-boot-category/latest:latest

Get the ECR repo name

< your-aws-acct-id>.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-category

Tag

sudo docker tag aws-ecr-spring-boot-category/latest < your-aws-acct-id>.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-category

Login to the AWS ECR to push our new image

aws ecr get-login-password --region us-west-2 | sudo docker login --username AWS --password-stdin < your-aws-acct-id>.dkr.ecr.us-west-2.amazonaws.com

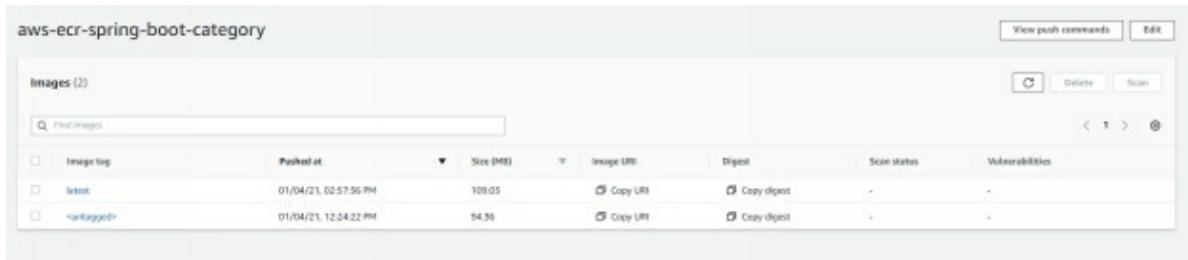
WARNING! Your password will be stored unencrypted in /home/ubuntu/.docker/config.json.

Configure a credential helper to remove this warning. See
<https://docs.docker.com/engine/reference/commandline/login/#credentials-store>

Login Succeeded

Now push the Docker image to ECR

```
sudo docker push < your-aws-acct-id>.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-category
```



© Amazon Web Services

Create the Kubernetes Deployment

```
kubectl apply -f ./category-kubernetes-deployment.yaml
deployment.apps/category-deployment created
```

Create the Kubernetes Service

```
kubectl expose deployment category-deployment --type=LoadBalancer --
name=category-service
```

service/category-service exposed

Let's get the service external dns name with

```
kubectl get svc
```

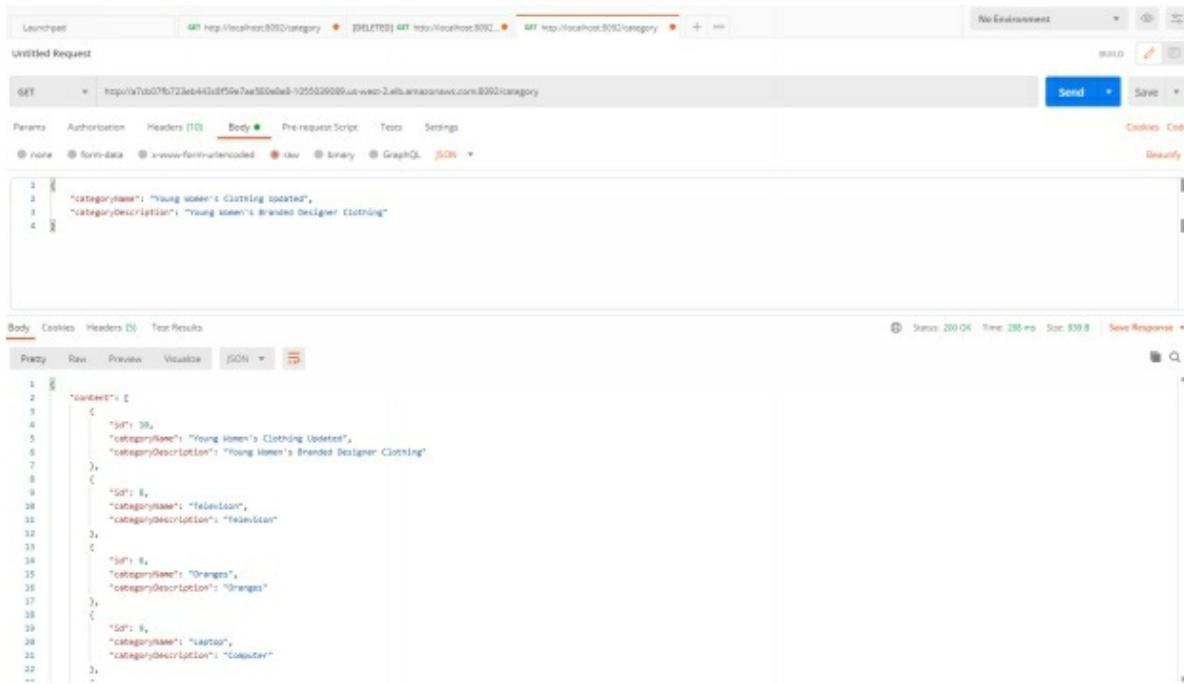
```
ubuntu@ip-172-31-9-166:~/rollingstone-ecommerce-category-k8s-api$ kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP
category-service LoadBalancer 10.100.89.52  a7cb07fb723eb443c8f59e7ae580e8e8-1055039089.us-west-2.elb.amazonaws.com
kubernetes     ClusterIP  10.100.0.1   <none>
ubuntu@ip-172-31-9-166:~/rollingstone-ecommerce-category-k8s-api$
```

a7cb07fb723eb443c8f59e7ae580e8e8-1055039089.us-west-2.elb.amazonaws.com

5.8 TESTING A SAMPLE DEPLOYMENT IN AWS EKS

Let's test our Docker Kubernetes Deployed Container

GET



The screenshot shows a Postman interface with the following details:

- URL:** https://a7d207fb723eb42c2f50e7ae5f0e5e8-1255236989.us-west-2.eks.amazonaws.com:8092/category
- Method:** GET
- Body:** JSON (empty)
- Response Status:** 200 OK
- Response Time:** 286 ms
- Response Size:** 339.8

The response body is a JSON array of categories:

```
1 | [ { "id": 1, "categoryName": "Young women's Clothing updated", "categoryDescription": "Young women's Branded designer Clothing" }, { "id": 2, "categoryName": "Televisions", "categoryDescription": "Televisions" }, { "id": 3, "categoryName": "Oranges", "categoryDescription": "Oranges" }, { "id": 4, "categoryName": "Laptops", "categoryDescription": "Computer" } ]
```

POST

The screenshot shows the Postman interface with a 'Launched' tab selected. There are three tabs at the top: 'GET http://localhost:8002/category' (status 200 OK), '[DELETE] GET http://localhost:8002...' (status 200 OK), and 'PUT http://localhost:8002/category' (status 200 OK). The bottom tab is 'POST http://a7cd07b6723eb443cf59e7ee50e8b-1055039089.us-west-2.elb.amazonaws.com:8002/category'. The 'Body' tab is selected, showing JSON input:

```
1 {"categoryName": "Young Men's Clothing",  
2 "categoryDescription": "Young Men's Branded Designer Clothing"}  
3  
4
```

The response status is 200 OK, time 330 ms, size 154 B. The response body is: '1 New Category has been saved with ID: 11'

GET one

The screenshot shows the Postman interface with a 'Launched' tab selected. There are three tabs at the top: 'GET http://localhost:8002/category' (status 200 OK), '[DELETE] GET http://localhost:8002...' (status 200 OK), and 'PUT http://localhost:8002/category' (status 200 OK). The bottom tab is 'GET http://a7cd07b6723eb443cf59e7ee50e8b-1055039089.us-west-2.elb.amazonaws.com:8002/category/11'. The 'Body' tab is selected, showing JSON input:

```
1 {"categoryName": "Young Men's Clothing",  
2 "categoryDescription": "Young Men's Branded Designer Clothing"}  
3  
4
```

The response status is 200 OK, time 171 ms, size 273 B. The response body is: '1 {
2 "id": 11,
3 "categoryName": "Young Men's Clothing",
4 "categoryDescription": "Young Men's Branded Designer Clothing"}
5
6'

Update

The screenshot shows the Postman interface with a 'Launched' tab selected. There are three tabs at the top: 'PUT http://localhost:8002/category' (status 200 OK), '[DELETE] GET http://localhost:8002...' (status 200 OK), and 'PUT http://localhost:8002/category' (status 200 OK). The bottom tab is 'PUT http://a7cd07b6723eb443cf59e7ee50e8b-1055039089.us-west-2.elb.amazonaws.com:8002/category/11'. The 'Body' tab is selected, showing JSON input:

```
1 {"id": 11,  
2 "categoryName": "Young Men's Clothing Updated Docker",  
3 "categoryDescription": "Young Men's Branded Designer Clothing"}  
4  
5
```

The response status is 200 OK, time 106 ms, size 185 B. The response body is: '1 Category has been updated successfully.'

Verify Update

The screenshot shows the Postman interface with an 'Untitled Request' tab. The request method is 'PUT', and the URL is <http://a7cb07fb723eb443c8f59e7ae580e8e8-1055039089.us-west-2.elb.amazonaws.com:8092/category/11>. The 'Body' tab is selected, showing JSON data:

```
1 {
2   "id": 11,
3   "categoryName": "Young Men's Clothing Updated Docker",
4   "categoryDescription": "Young Men's Branded Designer Clothing"
5 }
```

The response status is 200 OK, Time: 127 ms, Size: 291 B.

Delete

The screenshot shows the Postman interface with an 'Untitled Request' tab. The request method is 'DELETE', and the URL is <http://a7cb07fb723eb443c8f59e7ae580e8e8-1055039089.us-west-2.elb.amazonaws.com:8092/category/11>. The 'Body' tab is selected, showing JSON data:

```
1 {
2   "id": 11,
3   "categoryName": "Young Men's Clothing Updated Docker",
4   "categoryDescription": "Young Men's Branded Designer Clothing"
5 }
```

The response message is: Category has been deleted successfully.

To test the Actuator Endpoint One by One please follow each of the several screenshots shown below to test.

Default Actuator Endpoint of the AWS EKS Deployment Service can be found at

host:port/actuator

```
← → C ⌂ ▲ Not secure | a7cb07fb723eb443c8f59e7ae580e8e8-1055039089.us-west-2.elb.amazonaws.com:8093/actuator
1 // 20210104150443
2 // http://a7cb07fb723eb443c8f59e7ae580e8e8-1055039089.us-west-2.elb.amazonaws.com:8093/actuator
3
4 +
5   "_links": {
6     "self": {
7       "href": "http://a7cb07fb723eb443c8f59e7ae580e8e8-1055039089.us-west-2.elb.amazonaws.com:8093/actuator",
8       "templated": false
9     },
10    "is-customer-healthy": {
11      "href": "http://a7cb07fb723eb443c8f59e7ae580e8e8-1055039089.us-west-2.elb.amazonaws.com:8093/actuator/is-customer-healthy",
12      "templated": false
13    },
14    "is-customer-healthy-name": {
15      "href": "http://a7cb07fb723eb443c8f59e7ae580e8e8-1055039089.us-west-2.elb.amazonaws.com:8093/actuator/is-customer-healthy/{name}",
16      "templated": true
17    }
18  }
```

Custom Health Check can be found at host:port/actuator/is-customer-healthy

```
← → C ⌂ ▲ Not secure | a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/is-customer-healthy
SUCCESS
```

Standard Actuator Health endpoint can be found at host:port/actuator/health

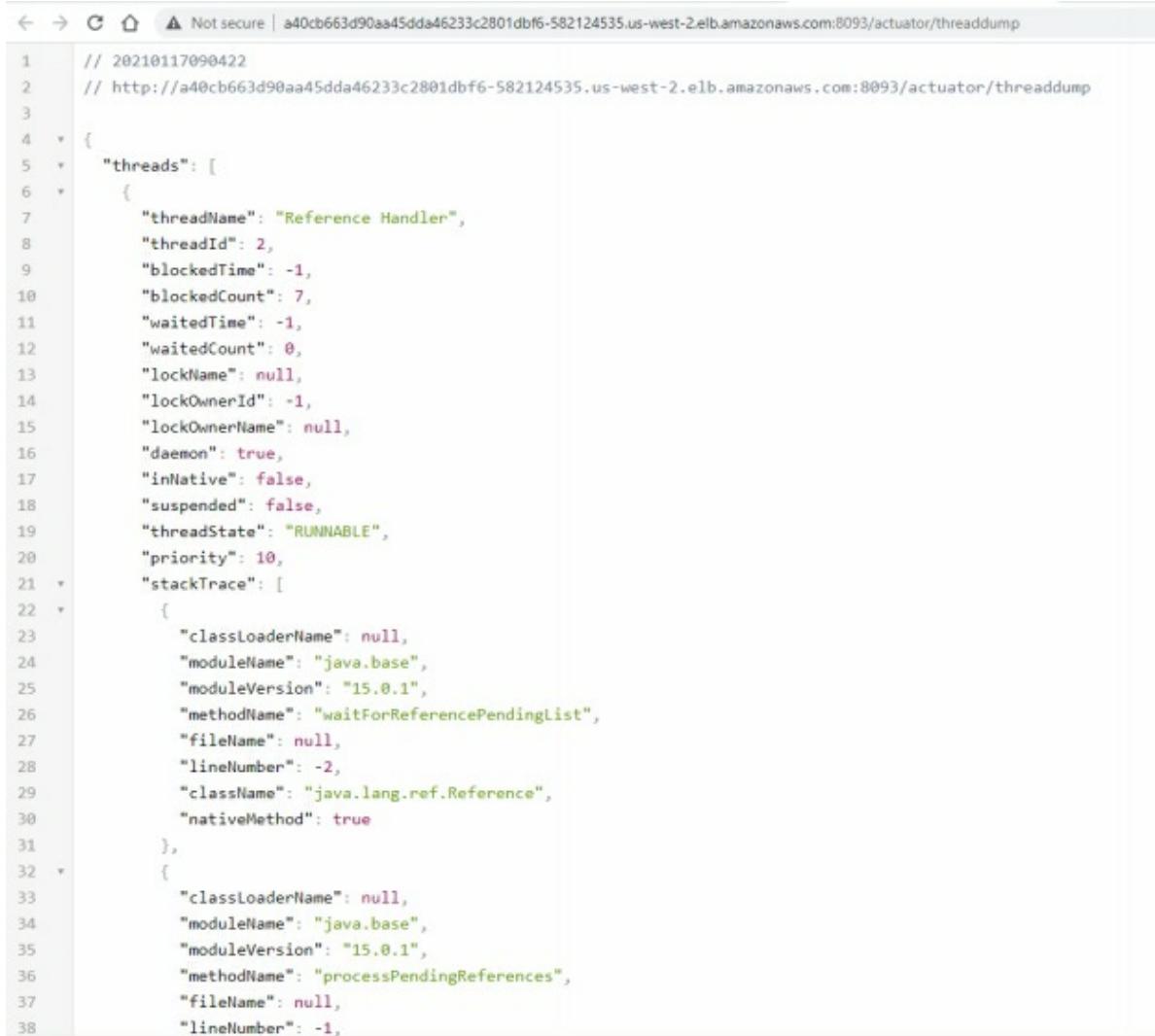
```
← → C ⌂ ▲ Not secure | a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/health
1 // 20210117090202
2 // http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/health
3
4 {
5     "status": "UP",
6     "components": {
7         "db": {
8             "status": "UP",
9             "details": {
10                 "database": "MySQL",
11                 "validationQuery": "isValid()"
12             }
13         },
14         "diskSpace": {
15             "status": "UP",
16             "details": {
17                 "total": 85886742528,
18                 "free": 83256610816,
19                 "threshold": 10485760,
20                 "exists": true
21             }
22         },
23         "livenessState": {
24             "status": "UP"
25         },
26         "ping": {
27             "status": "UP"
28         },
29         "readinessState": {
30             "status": "UP"
31         }
32     },
33     "groups": [
34         "liveness",
35         "readiness"
36     ]
37 }
```

Environment Variables can be found at host:port/actuator/env

```
← → C ⌂ Not secure | a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/env
1 // 20210117090245
2 // http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/env
3
4 {
5     "activeProfiles": [
6         "aws"
7     ],
8     "propertySources": [
9         {
10             "name": "server.ports",
11             "properties": {
12                 "local.server.port": {
13                     "value": 8092
14                 },
15                 "local.management.port": {
16                     "value": 8093
17                 }
18             }
19         },
20         {
21             "name": "servletContextInitParams",
22             "properties": {
23
24             },
25         },
26         {
27             "name": "systemProperties",
28             "properties": {
29                 "java.specification.version": {
30                     "value": "15"
31                 },
32                 "sun.jnu.encoding": {
33                     "value": "UTF-8"
34                 },
35                 "java.class.path": {
36                     "value": "/app.jar"
37                 },
38                 "java.vm.vendor": {
39                     "value": "AdoptOpenJDK"
40                 }
41             }
42         }
43     ]
44 }
```

```
Not secure | a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator
CONTINUATION - 1
43     "href": "http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/conditions",
44     "templated": false
45   },
46   "configprops": {
47     "href": "http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/configprops",
48     "templated": false
49   },
50   "env": {
51     "href": "http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/env",
52     "templated": false
53   },
54   "env-toMatch": {
55     "href": "http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/env/{toMatch}",
56     "templated": true
57   },
58   "loggers": {
59     "href": "http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/loggers",
60     "templated": false
61   },
62   "loggers-name": {
63     "href": "http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/loggers/{name}",
64     "templated": true
65   },
66   "heapdump": {
67     "href": "http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/heapdump",
68     "templated": false
69   },
70   "threaddump": {
71     "href": "http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/threaddump",
72     "templated": false
73   },
74   "metrics": {
75     "href": "http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/metrics",
76     "templated": false
77   },
78   "metrics-requiredMetricName": {
79     "href": "http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/metrics/{requiredMetricName}",
80     "templated": true
81 }
headdump
```

Thread dump can be found at host:port/actuator/threaddump



The screenshot shows a browser window displaying a JSON object representing a thread dump. The URL in the address bar is `a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/threaddump`. The JSON structure is as follows:

```
// 20210117090422
// http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/threaddump
{
  "threads": [
    {
      "threadName": "Reference Handler",
      "threadId": 2,
      "blockedTime": -1,
      "blockedCount": 7,
      "waitedTime": -1,
      "waitedCount": 0,
      "lockName": null,
      "lockOwnerId": -1,
      "lockOwnerName": null,
      "daemon": true,
      "inNative": false,
      "suspended": false,
      "threadState": "RUNNABLE",
      "priority": 10,
      "stackTrace": [
        {
          "classLoaderName": null,
          "moduleName": "java.base",
          "moduleVersion": "15.0.1",
          "methodName": "waitForReferencePendingList",
          "fileName": null,
          "lineNumber": -2,
          "className": "java.lang.ref.Reference",
          "nativeMethod": true
        },
        {
          "classLoaderName": null,
          "moduleName": "java.base",
          "moduleVersion": "15.0.1",
          "methodName": "processPendingReferences",
          "fileName": null,
          "lineNumber": -1,
        }
      ]
    }
  ]
}
```

Metrics can be found at host:port/actuator/metrics

```
← → C ⌂ Not secure | a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/metrics
1 // 20210117090458
2 // http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/metrics
3
4 +
5 {
6     "names": [
7         "com.rollingstone.CategoryController.HTTP400",
8         "com.rollingstone.CategoryController.HTTP404",
9         "com.rollingstone.category.created",
10        "hikaricp.connections",
11        "hikaricp.connections.acquire",
12        "hikaricp.connections.active",
13        "hikaricp.connections.creation",
14        "hikaricp.connections.idle",
15        "hikaricp.connections.max",
16        "hikaricp.connections.min",
17        "hikaricp.connections.pending",
18        "hikaricp.connections.timeout",
19        "hikaricp.connections.usage",
20        "http.client.requests",
21        "http.server.requests",
22        "jdbc.connections.active",
23        "jdbc.connections.idle",
24        "jdbc.connections.max",
25        "jdbc.connections.min",
26        "jvm.buffer.count",
27        "jvm.buffer.memory.used",
28        "jvm.buffer.total.capacity",
29        "jvm.classes.loaded",
30        "jvm.classes.unloaded",
31        "jvm.gc.live.data.size",
32        "jvm.gc.max.data.size",
33        "jvm.gc.memory.allocated",
34        "jvm.gc.memory.promoted",
35        "jvm.gc.pause",
36        "jvm.memory.committed",
37        "jvm.memory.max",
38        "jvm.memory.used",
39        "jvm.threads.daemon",
40    ]
41 }
```

Specific metrics can be found at host:port/actuator/metrics/<metric_name>

```
← → C ⌂ ▲ Not secure | a7cb07fb723eb443c8f59e7ae580e8e8-1055039089.us-west-2.elb.amazonaws.com:8093/actuator/metrics/com.rollingstone.category.created
1 // 20210104152400
2 // http://a7cb07fb723eb443c8f59e7ae580e8e8-1055039089.us-west-2.elb.amazonaws.com:8093/actuator/metrics/com.rollingstone.category.created
3
4 +
5   "name": "com.rollingstone.category.created",
6   "description": "Number of Categories Created",
7   "baseUnit": null,
8   "measurements": [
9     {
10       "statistic": "COUNT",
11       "value": 2.0
12     }
13   ],
14   "availableTags": [
15     {
16       "tag": "environment",
17       "values": [
18         "production"
19       ]
20     }
21   ]
22 }
```

← → C ⌂ Not secure | a7cb07fb723eb443c8f59e7ae580e8e8-1055039089.us-west-2.elb.amazonaws.com:8093/actuator/health

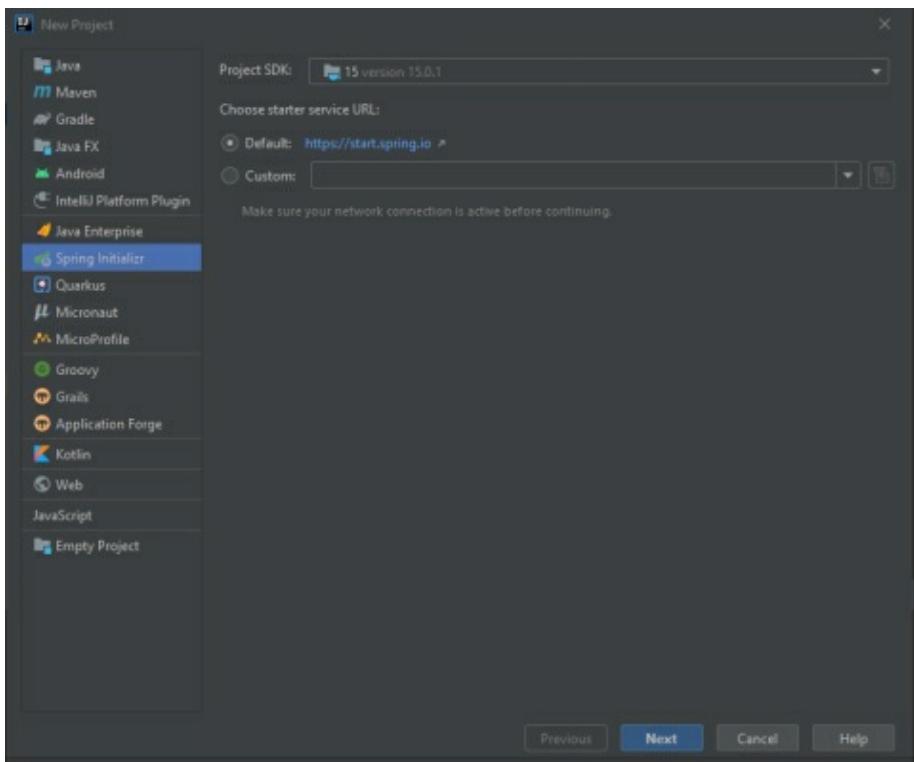
```
1 // 20210104150525
2 // http://a7cb07fb723eb443c8f59e7ae580e8e8-1055039089.us-west-2.elb.amazonaws.com:8093/actuator/health
3
4 +
5     "status": "UP",
6     "components": {
7         "db": {
8             "status": "UP",
9             "details": {
10                 "database": "MySQL",
11                 "validationQuery": "isValid()"
12             }
13         },
14         "diskSpace": {
15             "status": "UP",
16             "details": {
17                 "total": 85886742528,
18                 "free": 82958454784,
19                 "threshold": 10485760,
20                 "exists": true
21             }
22         },
23         "livenessState": {
24             "status": "UP"
25         },
26         "ping": {
27             "status": "UP"
28         },
29         "readinessState": {
30             "status": "UP"
31         }
32     },
33     "groups": [
34         "liveness",
35         "readiness"
36     ]
37 }
```

CHAPTER 6

Building Product REST API to AWS EKS

6.1 CREATING A NEW PROJECT IN INTELLIJ

We built the Category application already. We also deployed it to a local machine and AWS EKS. Now is the time to build the Product Microservice, which is the second of the two applications we said we would build to demonstrate Spring Boot 2 Microservices deployed in AWS EKS. Start your IDE (IntelliJ in my case), click File New and choose Spring Initializer to create a new project.

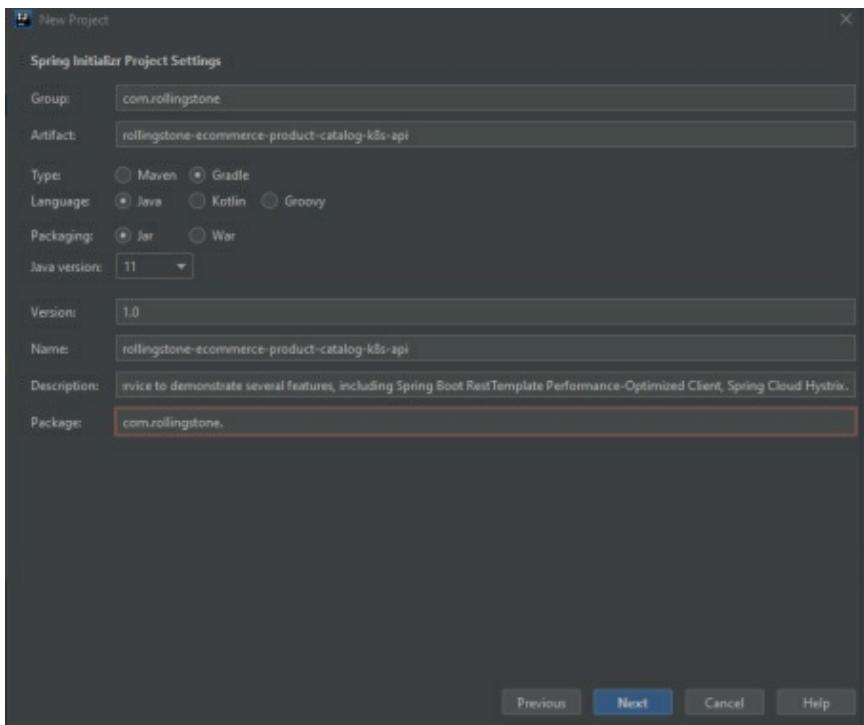


Click Next

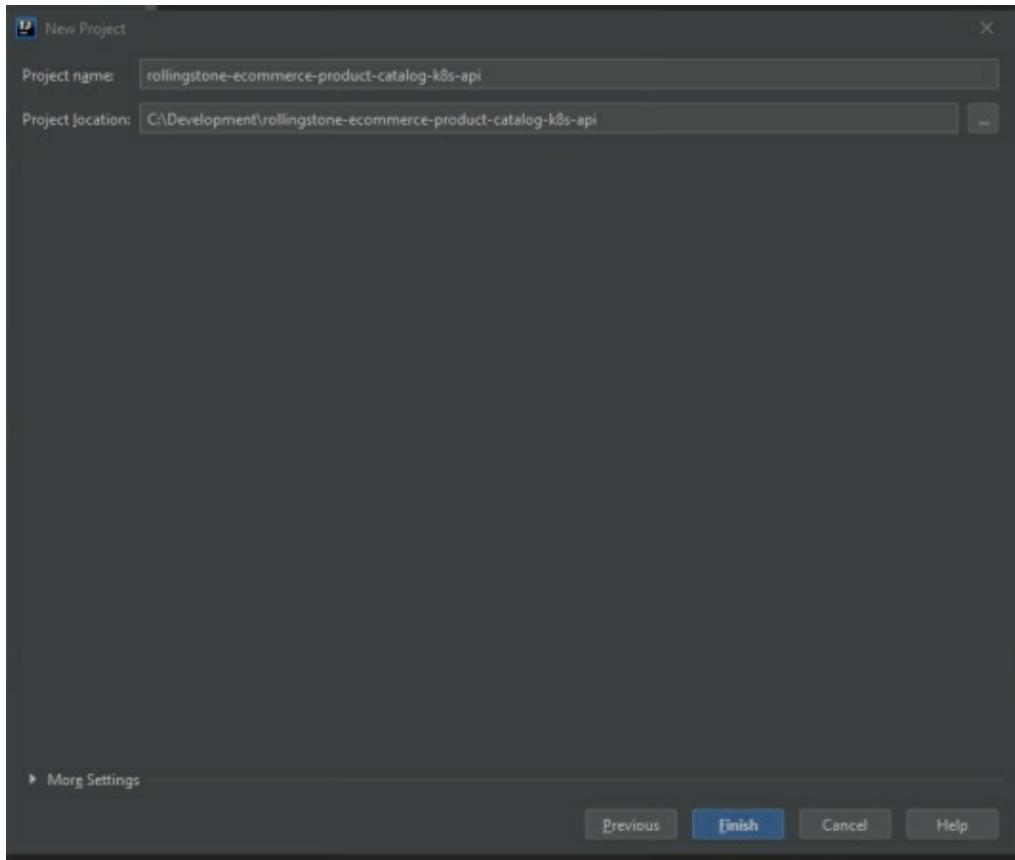
Enter the following Details in this screen and click Next

- Group : com.rollingstone
- Artifact : rollingstone-ecommerce-product-catalog-k8s-api

- Type : Gradle
- Language : Java
- Packaging: Jar
- Java Version: 15
- Version: 1.0
- Name : rollingstone-ecommerce-product-catalog-k8s-api
- Description : New Product Catalog API Spring Boot Microservice to demonstrate several features, including Spring Boot Performance-Optimized RestTemplate Client to validate the Category received as part of the request body.
- Package: com.rollingstone

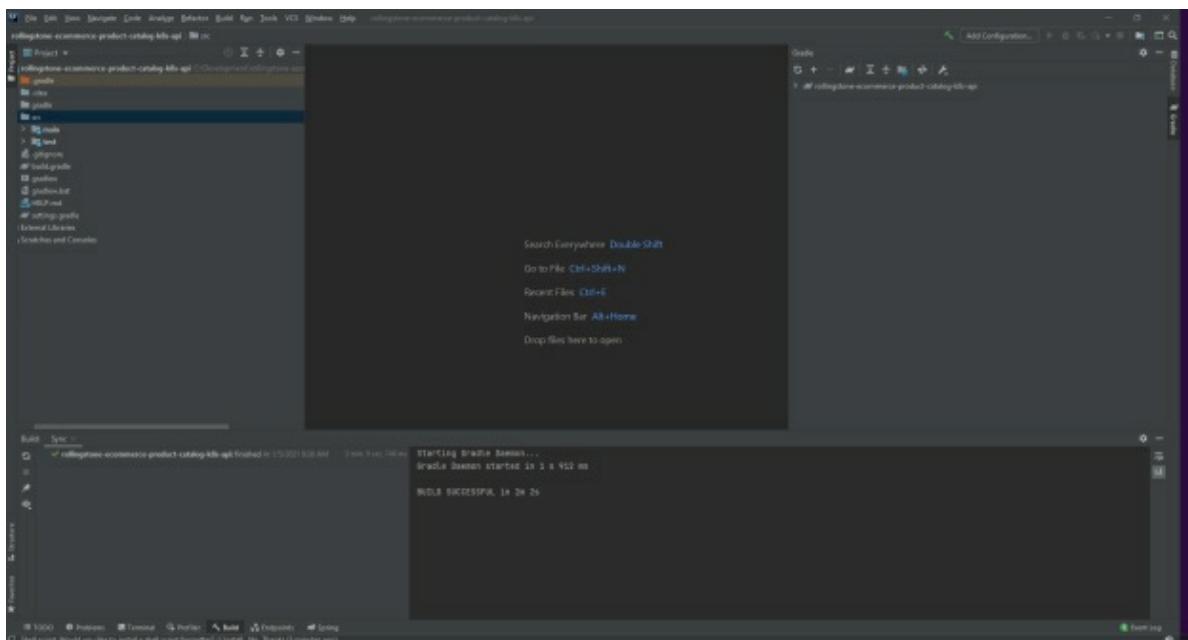


Click Next



Click Finish

Project Build complete is shown below



6.2 ADDING DEPENDENCIES MANUALLY

Not all the dependencies we would need could be added from the Spring Boot Initializer. Let us open our build.gradle file and add the following dependencies manually.

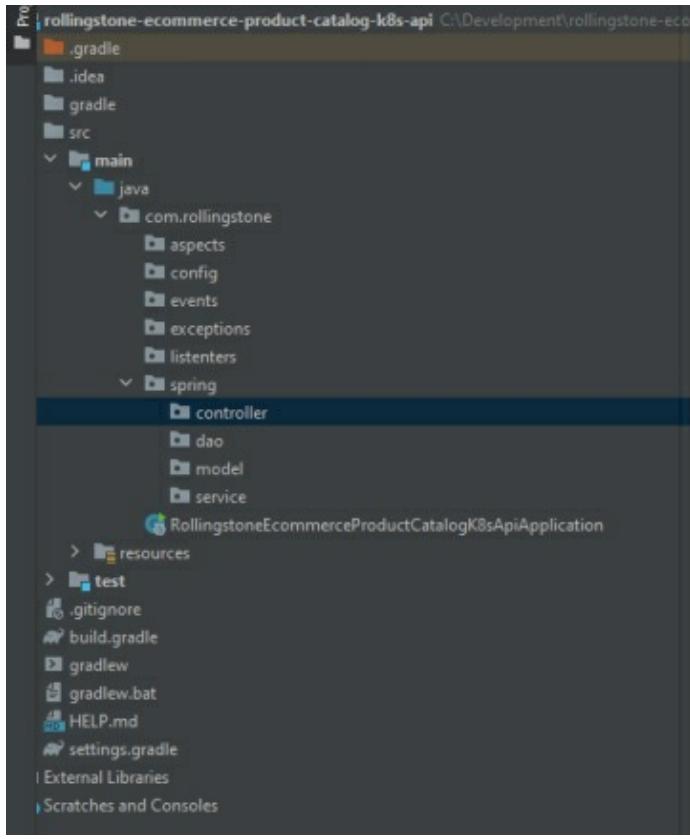
```
implementation 'org.springframework.boot:spring-boot-starter-aop'  
implementation 'com.fasterxml.jackson.core:jackson-databind'  
implementation "io.springfox:springfox-boot-starter:3.0.0"  
implementation "io.springfox:springfox-swagger-ui:3.0.0"  
implementation 'org.apache.httpcomponents:httpclient'  
  
implementation 'javax.xml.bind:jaxb-api:2.3.0'
```

6.3 ADDING THE PACKAGE STRUCTURE

Add the following java packages under the root package com.rollingstone

1. Right click the package com.rollingstone, choose New → Package
2. Enter aspects and press enter
3. Repeat step 1 and 2 for the following
 - a. config
 - b. events
 - c. exceptions
 - d. listeners
 - e. spring
 - i. controller
 - ii. dao
 - iii. model
 - iv. service

After the package creation is complete, the structure should like the following



6.4 BUILDING THE MODEL CLASSES

We would add two model classes here in this application. We did not need two of them in the Category application. The JSON payload was simple (for the Category) and straightforward key-value pairs. However, here in this Product Microservice, our JSON payload would also specify two complex JSON attributes. When a JSON attribute in a JSON payload has its critical own value pair, we call the structure either nested or complex JSON. To help Spring Boot Jackson easily unmarshal the JSON payload into java POJO, we would need both the Product and the nested Category model classes. First, copy the Category model class from your Category application to the com.Rollingstone.spring.model package

The Category class is already known to us, and the Product model class would be closely similar. Following is the full code for the Product Model class.

Now right click on the above package, choose New → Java Class and name the class as Product. Add the code to the editor

```
package com.rollingstone.spring.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToOne;

@Entity(name = "ROLLINGSTONE_PRODUCT")
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "PCODE", nullable = false)
```

```
private String productCode;

    @Column(name = "NAME", nullable = false)
private String productName;

    @Column(name = "SHORT_DESCRIPTION", nullable = false)
private String shortDescription;

    @Column(name = "LONG_DESCRIPTION", nullable = false)
private String longDescription;

    @Column(name = "CANDISPLAY", nullable = false)
private boolean canDisplay;

    @Column(name = "ISDELETED", nullable = false)
private boolean isDeleted;

    @Column(name = "ISAUTOMOTIVE", nullable = false)
private boolean isAutomotive;

    @Column(name = "ISINTERNATIONAL", nullable = false)
private boolean isInternational;

    @OneToOne
@JoinColumn(name = "parent_category_id")
private Category parentCategory;

    @OneToOne
@JoinColumn(name = "category_id")
private Category category;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getProductCode() {
        return productCode;
    }

    public void setProductCode(String productCode) {
        this.productCode = productCode;
    }

    public String getProductName() {
```

```
        return productName;
    }

    public void setProductName(String productName) {
        this.productName = productName;
    }

    public String getShortDescription() {
        return shortDescription;
    }

    public void setShortDescription(String shortDescription) {
        this.shortDescription = shortDescription;
    }

    public String getLongDescription() {
        return longDescription;
    }

    public void setLongDescription(String longDescription) {
        this.longDescription = longDescription;
    }

    public boolean isCanDisplay() {
        return canDisplay;
    }

    public void setCanDisplay(boolean canDisplay) {
        this.canDisplay = canDisplay;
    }

    public boolean isDeleted() {
        return isDeleted;
    }

    public void setDeleted(boolean isDeleted) {
        this.isDeleted = isDeleted;
    }

    public boolean isAutomotive() {
        return isAutomotive;
    }

    public void setAutomotive(boolean isAutomotive) {
        this.isAutomotive = isAutomotive;
    }

    public boolean isInternational() {
```

```
    return isInternational;
}

public void setInternational(boolean isInternational) {
    this.isInternational = isInternational;
}

public Category getParentCategory() {
    return parentCategory;
}

public void setParentCategory(Category parentCategory) {
    this.parentCategory = parentCategory;
}

public Category getCategory() {
    return category;
}

public void setCategory(Category category) {
    this.category = category;
}

@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + (canDisplay ? 1231 : 1237);
    result = prime * result + ((category == null) ? 0 : category.hashCode());
    result = prime * result + ((id == null) ? 0 : id.hashCode());
    result = prime * result + (isAutomotive ? 1231 : 1237);
    result = prime * result + (isDeleted ? 1231 : 1237);
    result = prime * result + (isInternational ? 1231 : 1237);
    result = prime * result + ((longDescription == null) ? 0 : longDescription.hashCode());
    result = prime * result + ((parentCategory == null) ? 0 : parentCategory.hashCode());
    result = prime * result + ((productCode == null) ? 0 : productCode.hashCode());
    result = prime * result + ((productName == null) ? 0 : productName.hashCode());
    result = prime * result + ((shortDescription == null) ? 0 : shortDescription.hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())

```

```
        return false;
    Product other = (Product) obj;
    if (canDisplay != other.canDisplay)
        return false;
    if (category == null) {
        if (other.category != null)
            return false;
    } else if (!category.equals(other.category))
        return false;
    if (id == null) {
        if (other.id != null)
            return false;
    } else if (!id.equals(other.id))
        return false;
    if (isAutomotive != other.isAutomotive)
        return false;
    if (isDeleted != other.isDeleted)
        return false;
    if (isInternational != other.isInternational)
        return false;
    if (longDescription == null) {
        if (other.longDescription != null)
            return false;
    } else if (!longDescription.equals(other.longDescription))
        return false;
    if (parentCategory == null) {
        if (other.parentCategory != null)
            return false;
    } else if (!parentCategory.equals(other.parentCategory))
        return false;
    if (productCode == null) {
        if (other.productCode != null)
            return false;
    } else if (!productCode.equals(other.productCode))
        return false;
    if (productName == null) {
        if (other.productName != null)
            return false;
    } else if (!productName.equals(other.productName))
        return false;
    if (shortDescription == null) {
        if (other.shortDescription != null)
            return false;
    } else if (!shortDescription.equals(other.shortDescription))
        return false;
    return true;
}
```

```
public Product() {
    super();
}

@Override
public String toString() {
    return "Product [id=" + id + ", productCode=" + productCode + ", productName=" + productName
           + ", shortDescription=" + shortDescription + ", longDescription=" + longDescription + ",
    canDisplay="
           + canDisplay + ", isDeleted=" + isDeleted + ", isAutomotive=" + isAutomotive + ",
    isInternational="
           + isInternational + ", parentCategory=" + parentCategory + ", category=" + category + "]";
}

}
```

6.5 BUILDING THE DAO JPA INTERFACE

The Dao interface would be very similar to what we developed in the Category application as well as shown in the code below. Everything we learned about how Spring Boot JPA write SQL code behind the scene relieving us to focus on business logic still stands. Add the Dao interface to the com.rollingstone.spring.dao package

```
package com.rollingstone.spring.dao;

import com.rollingstone.spring.model.Product;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.repository.PagingAndSortingRepository;

public interface ProductDaoRepository extends PagingAndSortingRepository<Product, Long> {
    Page<Product> findAll(Pageable pageable);
}
```

6.6 BUILDING EXCEPTION CLASSES

We would need the same exception classes we had in the Category Microservices. Just copy them from that application to the Product Catalog Microservice com.rollingstone.exceptions package. There are three classes that we need to copy to the same com.rollingstone.exceptions package in the Product Microservice

- HTTP400Exception.java
- HTTP404Exception.java
- RestAPIExceptionInfo.java

6.7 BUILDING THE EVENT CLASSES

We talked about Events while building the Category Microservice. A lot of big Java shops build a generic starter project adding all dependencies and classes that should exists in all the Microservices, to eliminate development time. If your organization are developing 100 plus Microservices in Spring Boot or NodeJS or C#, building a suitable starter project would be a huge savings. However, for now, add the ProductEvent class to the com.rollingstone.events package with the code very similar to the CategoryEvent class

```
package com.rollingstone.events;

import com.rollingstone.spring.model.Product;
import org.springframework.context.ApplicationEvent;

public class ProductEvent extends ApplicationEvent {

    private String eventType;
    private Product product;
    public String getEventType() {
        return eventType;
    }
    public void setEventType(String eventType) {
        this.eventType = eventType;
    }
    public Product getProduct() {
        return product;
    }
    public void setProduct(Product product) {
        this.product = product;
    }
    public ProductEvent(String eventType, Product product) {
        super(product);
        this.eventType = eventType;
        this.product = product;
    }
    @Override
```

```
public String toString() {
    return "ProductEvent [eventType=" + eventType + ", product=" + product + "]";
}

}
```

6.8 BUILDING ASPECTS

We decided to create the Spring Boot Actuator MicroMeter Counters in their own class sharing them throughout the application. We will see the configuration a little later. For now, the Aspect class also is the nearly same from the Category application. Add the class to the com.rollingstone.aspects package. Code is shown below

```
package com.rollingstone.aspects;
import io.micrometer.core.instrument.Counter;
import io.micrometer.core.instrument.Metrics;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class RestControllerAspect {

    private final Logger logger = LoggerFactory.getLogger("RestControllerAspect");
    @Autowired
    Counter createdProductCreationCounter;
    @Before("execution(public * com.rollingstone.spring.controller.*Controller.*(..))")
    public void generalAllMethodASpect() {
        logger.info("All Method Calls invoke this general aspect method");
    }

    @AfterReturning("execution(public *
com.rollingstone.spring.controller.*Controller.createProduct(..))")
    public void getsCalledOnProductSave() {
        logger.info("This aspect is fired when the save method of the controller is called");
        createdProductCreationCounter.increment();
    }
}
```

6.9 BUILDING LISTENERS

We would need the Listener as well for the Product Catalog Microservice. Add the ProductEventListener to the com.rollingstone.listener package. The technical discussion we had during building the Category Microservice applies here as well.

```
package com.rollingstone.listeners;

import com.rollingstone.events.ProductEvent;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.Component;

@Component
public class ProductEventListener {

    private final Logger log = LoggerFactory.getLogger(this.getClass());

    @EventListener
    public void onApplicationEvent(ProductEvent productEvent) {
        log.info("Received Product Event : "+productEvent.getEventType());
        log.info("Received Product From Product Event :" +productEvent.getProduct().toString());
    }
}
```

6.10 BUILDING SWAGGER CONFIGURATION

We explained what Swagger is, how it benefits us and how it dynamically generates documentation and a live test website from our Spring Boot code. Now it is quite easy to migrate Swagger configuration code from one Microservice to another. We already added the swagger dependencies to our build.gradle file. Some of the information are sample only as the code will openly live within the Git public repository. However, in a real project, we can or should update the contact information of a real contact person or group email.

Let us add a class named as SpringFoxConfigForProduct in the com.rollingstone.config package. Here is the full code

```
package com.rollingstone.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2WebMvc;

import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

@Configuration
@EnableSwagger2WebMvc
public class SpringFoxConfigForProduct {

    public static final Contact DEFAULT_CONTACT = new Contact(
        "Binit Datta", "http://binitdatta.com", "binit-sample-email.com");

    public static final ApiInfo DEFAULT_PRODUCT_API_INFO = new ApiInfo(
        "Product API Title", "Product API Description", "1.0",
        "urn:tos", DEFAULT_CONTACT,
        "Apache 2.0", "http://www.apache.org/licenses/LICENSE-2.0", Arrays.asList());
}
```

```

private static final Set<String> DEFAULT_PRODUCT_API_PRODUCES_AND_CONSUMES =
    new HashSet<String>(Arrays.asList("application/json"));

    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
            .apiInfo(DEFAULT_PRODUCT_API_INFO)
            .produces(DEFAULT_PRODUCT_API_PRODUCES_AND_CONSUMES)
            .consumes(DEFAULT_PRODUCT_API_PRODUCES_AND_CONSUMES);
    }
}

```

Let's now add the second Swagger config class ProductApiDocumentationConfiguration in the same com.rollingstone.config package. Below is the full code

```

package com.rollingstone.config;

import io.swagger.annotations.*;

@SwaggerDefinition(
    info = @Info(
        description = "Product REST API Resources",
        version = "V1.0",
        title = "Product REST API for Demonstrating Full CRUD APIs either in a local environment or in AWS EKS Kubernetes",
        contact = @Contact(
            name = "Binit Datta",
            email = "binit-sample-email@gmail.com",
            url = "http://www.binitdatta.com"
        ),
        license = @License(
            name = "Apache 2.0",
            url = "http://www.apache.org/licenses/LICENSE-2.0"
        )
    ),
    consumes = {"application/json"},
    produces = {"application/json"},
    schemes = {SwaggerDefinition.Scheme.HTTP, SwaggerDefinition.Scheme.HTTPS},
    externalDocs = @ExternalDocs(value = "For Further Information", url = "http://binitdatta.com")
)
public class ProductApiDocumentationConfiguration {
}

```

6.11 ACTUATOR METRICS CONFIGURATION

A word on design patterns that you may have heard about. We know many kinds of design patterns, such as Java Design Patterns and J2EE Design Patterns. Design patterns, simply put, are best practice guidance of making software/or Planes, Trains, Ships, Cars, Television, smartphones, or anything else. We are using a popular Java Creational Design Patterns called the Builder design pattern. You can read a lot about this design pattern and a whole other from Joshua Bloch's legendary book Effective Java. Besides, as you can see, our `@Bean` annotated methods tell Spring Boot Context Loaded to create Spring Beans and name them with the name of the methods themselves. Thus, we can expect a Bean called `createdProductCreationCounter`, for example. Spring Boot would also notice that our method `createdProductCreationCounter` has a dependency on the `MeterRegistry`. It would create an object (singleton in this case) and pass that instance of the `MeterRegistry` to this and other methods. All Spring Boot Actuator beans would then register themselves to the common `MeterRegistry` helping the metrics exposed to an external metrics collector such as Prometheus.

Add a new class named `ProductMetricsConfiguration` to the package `com.rollingstone.config` package Here is the code

```
package com.rollingstone.config;

import io.micrometer.core.instrument.Counter;
import io.micrometer.core.instrument.MeterRegistry;
import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

import java.time.Duration;

@Configuration
```

```
public class ProductMetricsConfiguration {  
  
    @Bean  
    public Counter createdProductCreationCounter(MeterRegistry registry) {  
        return Counter  
            .builder("com.rollingstone.product.created")  
            .description("Number of Products Created")  
            .tags("environment", "production")  
            .register(registry);  
    }  
  
    @Bean  
    public Counter http400ExceptionCounter(MeterRegistry registry) {  
        return Counter  
            .builder("com.rollingstone.ProductController.HTTP400")  
            .description("How many HTTP Bad Request HTTP 400 Requests have been received since  
start time of this instance.")  
            .tags("environment", "production")  
            .register(registry);  
    }  
  
    @Bean  
    public Counter http404ExceptionCounter(MeterRegistry registry) {  
        return Counter  
            .builder("com.rollingstone.ProductController.HTTP404")  
            .description("How many HTTP Resource Not Found HTTP 404 Requests have been received  
since start time of this instance. ")  
            .tags("environment", "production")  
            .register(registry);  
    }  
}
```

6.12 BUILDING RESTTEMPLATE CONFIGURATION

It is essential to learn not just how to build Spring Boot Microservice APIs but also how to consume/call them with super-fast performance. This book aims to show the reader some of the challenges that we face in high traffic application in production. Of course, Spring Boot has an effortless way to call other Spring Boot REST APIs using the reliable RestTemplate. However, the default RestTemplate uses the HttpURLConnection, which opens a physical new TCP Connection every time we call it. Opening a Physical connection takes time and slows us down. If we are to go to the physical store on each one of the five working days to get the breakfast cereal bar, it will take a lot of time for us. What we instead buy a box of breakfast cereal bar when we visit the grocery store. The same concept is called pooling in software engineering. We have come across Database Connection Pooling, Redis Connection pooling, and now we will deal with Spring Boot RestTemplate HTTP Connection pooling. As usual, Spring's default RestTemplate may not perform well kept unchanged in high traffic production. That is why we would show you how to configure a more performing RestTemplate

Configuring the Spring Boot RestTemplate connection pool is getting under the hood, and for good reasons. At first, let us define a few terms to help us understand the rest of the details.

CONNECT_TIMEOUT - We have two ways to open a connection to the server that hosts our REST API. Either to open a physical TCP connection (default) or establish a connection from a previously established pool of connections. Getting a serial breakfast bar from our home pantry or going to the physical grocery store, parking our car etc. would be a real-life example of establishing a connection. How long (in milliseconds) we want to wait during the connection establishment process is called connection timeout. We can say 2000 milliseconds and if we are using an HTTP connection pool of 20 and all the 18 are busy handing requests, we would quickly get one of the

two available ones.

If, however, all pooled connections are busy, our connection requesting client will have to wait. This is very similar to us waiting in line if all eight checkout counters are busy in the grocery store. We can decide to stay a little for one of the cash counter / busy TCP connections to be free, or if it takes too long, we can determine that we have waited for long, abort our cart, and drop the attempt to process the request. How long we are willing to wait when all the pooled connections are busy is called CONNECTION_REQUEST_TIMEOUT

The final one is called SOCKET_TIMEOUT. When we can get a pooled connection, and our connection could send the request to the server, Socket Timeout is the amount of time we are ready to wait before the server finally responds with our data. Socket Timeout deals with the real call rather than trying to get a connection from the pool.

Here is the most basic configuration from our ApacheHttpClientConfiguration class.

Step 1 → Define Connection Timeouts

```
// Timeouts
int CONNECTION_TIMEOUT = 10 * 1000; // We will wait for 10 seconds until a (physical)
connection is established
int REQUEST_TIMEOUT = 20 * 1000; // We will wait 20 seconds for getting a connection from the
connection pool
int SOCKET_TIMEOUT = 10 * 1000; // We will wait 10 seconds, to receive the data from the
external call
```

Let's us see where these are used

We have a method annotated with @Bean to create one of the components for configuring our Apache HttpClient. Here is the RequestConfig

Step 2 → Configure RequestConfig

```
@Bean RequestConfig configureRequestTimeouts(){
    RequestConfig requestConfig = RequestConfig.custom()
        .setConnectTimeout(CONNECT_TIMEOUT)
        .setConnectionRequestTimeout(REQUEST_TIMEOUT)
        .setSocketTimeout(SOCKET_TIMEOUT)
        .build();
}
```

Step 3 → Define Max Connection Numbers

We can say, our client needs to call two downstream Microservice API Category (/category) and say User (/user). Let's say we want to limit that for each route i.e. /category and /user we want to set a max of 4 connections per route. We also want to set a total max connection for all routes to 8. The validate property is a safety check for the pooling manager to make sure the connection is healthy and useable before leasing it to the request client thread.

```
int MAX_ROUTE_CONNECTIONS = 15; // route would be like /product
int MAX_TOTAL_CONNECTIONS = 50; // All routes together should not exceed 50
int VALIDATE_AFTER_INACTIVITY = 15 * 1000; // After 30 seconds of a connection being unused, the pool
// would validate the connection before lending it to a requesting thread
```

Step 4 → Set the Connection Pool

```
@Bean
public PoolingHttpClientConnectionManager poolingHttpConnectionManager() {
    PoolingHttpClientConnectionManager poolingConnectionManager = new
    PoolingHttpClientConnectionManager();

    // set total amount of connections across all HTTP routes
    poolingConnectionManager.setMaxTotal(MAX_TOTAL_CONNECTIONS);

    // set maximum amount of connections for each http route in pool
    poolingConnectionManager.setDefaultMaxPerRoute(MAX_ROUTE_CONNECTIONS);

    //Validate before leasing
    poolingConnectionManager.setValidateAfterInactivity(VALIDATE_AFTER_INACTIVITY);
    return poolingConnectionManager;
}
```

Step 5 → Define Keep Alive time

```
// Keep alive
int DEFAULT_KEEP_ALIVE_TIME = 10 * 1000; // One connection would be kept alive for 10 seconds
```

Step 6 → Define a Connection Keep Alive Strategy

```
@Bean
public ConnectionKeepAliveStrategy connectionKeepAliveStrategy() {
```

```

return (httpResponse, httpContext) -> {
    HeaderIterator headerIterator = httpResponse.headerIterator(HTTP.CONN_KEEP_ALIVE);
    HeaderElementIterator elementIterator = new BasicHeaderElementIterator(headerIterator);

    while (elementIterator.hasNext()) {
        HeaderElement element = elementIterator.nextElement();
        String param = element.getName();
        String value = element.getValue();
        if (value != null && param.equalsIgnoreCase("timeout")) {
            return Long.parseLong(value) * 1000; // convert to milliseconds
        }
    }

    return DEFAULT_KEEP_ALIVE_TIME;
};
}

```

Step 7 →

Your application may need all 20 or 40 polled connections active and alive during high peak time. What about during the night. You would want to release unused resources. The keep alive strategy is basically telling the pool manager, that after certain milliseconds of idleness, we would like to release the physical connection and reduce the number of pooled connections, to save resources.

```

@Bean
public Runnable idleConnectionWatcher(PoolingHttpClientConnectionManager pool) {
    return new Runnable() {
        @Override
        public void run() {
            // only if connection pool is initialised
            if (pool != null) {
                pool.closeExpiredConnections();
                pool.closeIdleConnections(IDLE_CONNECTION_WAIT_TIME,
                TimeUnit.MILLISECONDS);

                logger.info("Idle connection Watcher / Guard: Closing expired and idle connections
that not used or long waiting ones");
            }
        }
    };
}

```

Step 8 → Set the we are ready to set the HttpClient with all the nested and

dependent components getting used below

```
@Bean
public CloseableHttpClient httpClient() {

    return HttpClients.custom()
        .setDefaultRequestConfig(configureRequestTimeouts())
        .setConnectionManager(poolingHttpConnectionManager())
        .setKeepAliveStrategy(connectionKeepAliveStrategy())
        .build();
}
```

Here is the full Code together in the ApacheHttpClientConfiguration.java

```
package com.rollingstone.config;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.apache.http.HttpHost;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.conn.ConnectionKeepAliveStrategy;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.HeaderIterator;
import org.apache.http.protocol.HTTP;
import org.apache.http.HeaderElementIterator;
import org.apache.http.message.BasicHeaderElementIterator;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.HeaderElement;

import java.util.concurrent.TimeUnit;

@Configuration
public class ApacheHttpClientConfiguration {

    private final Logger logger = LoggerFactory.getLogger(this.getClass());

    // Connection pool
    int MAX_ROUTE_CONNECTIONS = 15; // route would be like /product
    int MAX_TOTAL_CONNECTIONS = 50; // All routes together should not exceed 50
    int VALIDATE_AFTER_INACTIVITY = 15 * 1000; // After 30 seconds of a connection being unused, the pool
    // would validate the connection before lending it to a requesting thread
}
```

```

// Timeouts
int CONNECTION_TIMEOUT = 10 * 1000; // We will wait for 10 seconds until a (physical)
connection is established
int CONNECTION_REQUEST_TIMEOUT = 20 * 1000; // We will wait 20 seconds for getting a
connection from the connection pool
int SOCKET_TIMEOUT = 10 * 1000; // We will wait 10 seconds, to receive the data from the
external call

// Keep alive
int DEFAULT_KEEP_ALIVE_TIME = 10 * 1000; // One connection would be kept alive for 10
seconds

// Idle connection monitor
int IDLE_CONNECTION_WAIT_TIME = 30 * 1000; // If a physical connection is idling, hanging
for 30 seconds or more, it will be terminated or cleaned up

@Bean
public PoolingHttpClientConnectionManager poolingHttpConnectionManager() {
    PoolingHttpClientConnectionManager poolingConnectionManager = new
    PoolingHttpClientConnectionManager();

    // set total amount of connections across all HTTP routes
    poolingConnectionManager.setMaxTotal(MAX_TOTAL_CONNECTIONS);

    // set maximum amount of connections for each http route in pool
    poolingConnectionManager.setDefaultMaxPerRoute(MAX_ROUTE_CONNECTIONS);

    //Validate before leasing
    poolingConnectionManager.setValidateAfterInactivity(VALIDATE_AFTER_INACTIVITY);
    return poolingConnectionManager;
}

@Bean
public ConnectionKeepAliveStrategy connectionKeepAliveStrategy() {
    return (httpResponse, httpContext) -> {
        HeaderIterator headerIterator = httpResponse.headerIterator(HTTP.CONN_KEEP_ALIVE);
        HeaderElementIterator elementIterator = new BasicHeaderElementIterator(headerIterator);

        while (elementIterator.hasNext()) {
            HeaderElement element = elementIterator.nextElement();
            String param = element.getName();
            String value = element.getValue();
            if (value != null && param.equalsIgnoreCase("timeout")) {
                return Long.parseLong(value) * 1000; // convert to milliseconds
            }
        }

        return DEFAULT_KEEP_ALIVE_TIME;
    }
}

```

```

    };

}

@Bean
public Runnable idleConnectionWatcher(PoolingHttpClientConnectionManager pool) {
    return new Runnable() {
        @Override
        public void run() {
            // only if connection pool is initialised
            if (pool != null) {
                pool.closeExpiredConnections();
                pool.closeIdleConnections(IDLE_CONNECTION_WAIT_TIME,
TimeUnit.MILLISECONDS);

                    logger.info("Idle connection Watcher / Guard: Closing expired and idle
connections that not used or long waiting ones");
            }
        }
    };
}

@Bean
RequestConfig configureRequestTimeouts(){
    RequestConfig requestConfig = RequestConfig.custom()
        .setConnectTimeout(CONNECTION_TIMEOUT)
        .setConnectionRequestTimeout(CONNECTION_REQUEST_TIMEOUT)
        .setSocketTimeout(SOCKET_TIMEOUT)
        .build();

    return requestConfig;
}

@Bean
public CloseableHttpClient httpClient() {

    return HttpClients.custom()
        .setDefaultRequestConfig(configureRequestTimeouts())
        .setConnectionManager(poolingHttpConnectionManager())
        .setKeepAliveStrategy(connectionKeepAliveStrategy())
        .build();
}
}

```

6.13 CONFIGURING THE RESTTEMPLATE

Step 1 → Setting up the Error Handler

```
package com.rollingstone.config;

import com.rollingstone.exceptions.HTTP404Exception;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.client.ClientHttpResponse;
import org.springframework.web.client.ResponseErrorHandler;

import java.io.IOException;

public class CustomClientErrorInterceptor implements ResponseErrorHandler {

    final Logger log = LoggerFactory.getLogger(this.getClass());

    @Override
    public boolean hasError (ClientHttpResponse clientHttpResponse) throws IOException {
        return clientHttpResponse.getStatusCode().is4xxClientError();
    }

    @Override
    public void handleError (ClientHttpResponse clientHttpResponse) throws IOException {
        log.error("CustomClientErrorHandler | HTTP Status Code: " +
clientHttpResponse.getStatusCode().value());
        throw new HTTP404Exception(("Resource Not Found"));
    }
}
```

Step 2 → Setting up the Request Interceptor

```
package com.rollingstone.config;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpRequest;
```

```

import org.springframework.http.client.ClientHttpRequestExecution;
import org.springframework.http.client.ClientHttpRequestInterceptor;
import org.springframework.http.client.ClientHttpResponse;

import java.io.IOException;

public class CustomClientHttpRequestInterceptor implements ClientHttpRequestInterceptor {

    private Logger log = LoggerFactory.getLogger(this.getClass());

    @Override
    public ClientHttpResponse intercept(HttpRequest request, byte[] bytes, ClientHttpRequestExecution
execution) throws IOException {
        log.info("URI: {}", request.getURI());
        log.info("HTTP Method: {}", request.getMethodValue());
        log.info("HTTP Headers: {}", request.getHeaders());

        return execution.execute(request, bytes);
    }
}

```

Step 3 → Setting up the Rest Template

```

package com.rollingstone.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.apache.http.impl.client.CloseableHttpClient;
import org.springframework.http.client.HttpComponentsClientHttpRequestFactory;
import org.springframework.web.client.RestTemplate;

@Configuration
public class RestTemplateConfiguration {

    private final CloseableHttpClient httpClient;

    @Autowired
    public RestTemplateConfiguration(CloseableHttpClient httpClient) {
        this.httpClient = httpClient;
    }

    @Bean
    public HttpComponentsClientHttpRequestFactory clientHttpRequestFactory() {
        HttpComponentsClientHttpRequestFactory clientHttpRequestFactory = new
HttpComponentsClientHttpRequestFactory();

```

```
clientHttpRequestFactory.setHttpClient(httpClient);
    return clientHttpRequestFactory;
}

@Bean
public RestTemplate restTemplate() {
    return new RestTemplateBuilder()
        .requestFactory(this::clientHttpRequestFactory)
        .errorHandler(new CustomClientErrorInterceptor())
        .interceptors(new CustomClientHttpRequestInterceptor())
        .build();
}
```

6.14 BUILDING THE SERVICE

```
package com.rollingstone.spring.service;

import com.rollingstone.spring.model.Product;
import org.springframework.data.domain.Page;

import java.util.Optional;

public interface ProductService {

    Product save(Product product);
    Optional<Product> get(long id);
    Page<Product> getProductsByPage(Integer pageNumber, Integer pageSize);
    void update(long id, Product product);
    void delete(long id);
}
```

```
package com.rollingstone.spring.service;

import com.rollingstone.exceptions.HTTP400Exception;
import com.rollingstone.spring.dao.ProductDaoRepository;
import com.rollingstone.spring.model.Category;
import com.rollingstone.spring.model.Product;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;
import java.util.Optional;

@Service
public class ProductServiceImpl implements ProductService {

    final static Logger logger = LoggerFactory.getLogger(ProductServiceImpl.class);
```

```

    @Value("${category.request.path}")
private String CATEGORY_REQUEST_PATH = "";

    @Value("${category.port}")
private Integer CATEGORY_SERVICE_PORT = 0;

    @Value("${category.service.host}")
    // @Value("${value.from.file}")
private String CATEGORY_SERVICE_HOST = "";

private String REQUEST_URI = CATEGORY_SERVICE_HOST + ":" +
CATEGORY_SERVICE_PORT + CATEGORY_REQUEST_PATH;

    @Autowired
private ProductDaoRepository productDao;

    @Autowired
private RestTemplate restTemplate;

    @Override
public Product save(Product product) {

    Category category = null;
Category parentCategory = null;
String URI = CATEGORY_SERVICE_HOST + ":" + CATEGORY_SERVICE_PORT +
CATEGORY_REQUEST_PATH;

    if (product.getCategory() == null) {
logger.info("Product Category is null :");
throw new HTTP400Exception("Bad Request as Category Cannot be empty");
} else {
    logger.info("Product Category is not null :" + product.getCategory());
    logger.info("Product Category is not null ID :" + product.getCategory().getId());
}

if (product.getParentCategory() == null) {
    logger.info("Product Parent Category is null :");
    throw new HTTP400Exception("Bad Request as Parent Category Cannot be empty");
} else {
    logger.info("Product Parent Category is not null :" + product.getParentCategory());
    logger.info("Product Parent Category is not null Id :" + product.getParentCategory().getId());
}

    logger.info("request port :" + CATEGORY_SERVICE_PORT);
logger.info("request host :" + CATEGORY_SERVICE_HOST);
logger.info("request path :" + CATEGORY_REQUEST_PATH);
}

```

```

    logger.info("request uri :" + REQUEST_URI);
    logger.info(" uri :" + URI);

    logger.info("request uri modified :" + CATEGORY_SERVICE_HOST + ":" +
CATEGORY_SERVICE_PORT + CATEGORY_REQUEST_PATH);
try{
    ResponseEntity<Category> categoryEntity = restTemplate.getForEntity(URI + "/{id}",
        Category.class,
        Long.toString(product.getCategory().getId()));
    if (categoryEntity != null) {
        Category validCategory = categoryEntity.getBody();

        if (validCategory == null){
            logger.info("Product Category is invalid :");
            throw new HTTP400Exception("Bad Request as Category Can not be invalid");
        }
    }
}
catch(Exception e){
    logger.info("Product Category is invalid :");
    throw new HTTP400Exception("Bad Request as Category Can not be invalid");
}
return productDao.save(product);
}

public Product saveProductWithoutValidation(Product product) {
logger.info("Hystrix Circuit Breaker Enabled and called fallback method");

    return productDao.save(product);
}

@Override
public Optional<Product> get(long id) {
    return productDao.findById(id);
}

@Override
public Page<Product> getProductsByPage(Integer pageNumber, Integer pageSize) {
    Pageable pageable = PageRequest.of(pageNumber, pageSize,
Sort.by("productCode").descending());
    return productDao.findAll(pageable);
}

@Override
public void update(long id, Product product) {
    productDao.save(product);
}

```

```
@Override  
public void delete(long id) {  
    productDao.deleteById(id);  
}  
}
```

6.15 BUILDING THE ABSTRACTCONTROLLER

```
package com.rollingstone.spring.controller;

import com.rollingstone.exceptions.HTTP400Exception;
import com.rollingstone.exceptions.HTTP404Exception;
import com.rollingstone.exceptions.RestAPIExceptionInfo;
import io.micrometer.core.instrument.Counter;
import io.micrometer.core.instrument.Metrics;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationEventPublisher;
import org.springframework.context.ApplicationEventPublisherAware;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotationResponseStatus;
import org.springframework.web.context.request.WebRequest;

import javax.servlet.http.HttpServletResponse;

public abstract class AbstractController implements ApplicationEventPublisher {

    protected final Logger log = LoggerFactory.getLogger(this.getClass());
    protected ApplicationEventPublisher eventPublisher;
    protected static final String DEFAULT_PAGE_SIZE = "20";
    protected static final String DEFAULT_PAGE_NUMBER = "0";

    @Autowired
    Counter http400ExceptionCounter;

    @Autowired
    Counter http404ExceptionCounter;

    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(HTTP400Exception.class)
```

```

public @ResponseBody RestAPIExceptionInfo handleBadRequestException(HTTP400Exception ex,
    WebRequest request, HttpServletResponse response)
{
    log.info("Received Bad Request Exception"+ex.getLocalizedMessage());
    http400ExceptionCounter.increment();
    return new RestAPIExceptionInfo(ex.getLocalizedMessage(), "The Request did not have the
correct parameters");
}

    @ResponseStatus(HttpStatus.NOT_FOUND)
    @ExceptionHandler(HTTP404Exception.class)
    public @ResponseBody RestAPIExceptionInfo
handleResourceNotFoundException(HTTP404Exception ex,
    WebRequest request, HttpServletResponse response)
{
    log.info("Received Resource Not Found Exception"+ex.getLocalizedMessage());
    http404ExceptionCounter.increment();
    return new RestAPIExceptionInfo(ex.getLocalizedMessage(), "The Requested Resource was not
found");
}

    @Override
    public void setApplicationEventPublisher(ApplicationEventPublisher eventPublisher) {
        this.eventPublisher = eventPublisher;
    }

    public static <T> T checkResourceFound(final T resource) {
        if (resource == null) {
            throw new HTTP404Exception("Resource Not Found");
        }
        return resource;
    }

}

```

6.16 BUILDING THE PRODUCTCONTROLLER

```
package com.rollingstone.spring.controller;

import com.rollingstone.events.ProductEvent;
import com.rollingstone.spring.model.Product;
import com.rollingstone.spring.service.ProductService;
import org.springframework.data.domain.Page;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Optional;

@RestController
public class ProductController extends AbstractController {

    private ProductService productService;

    public ProductController(ProductService productService) {
        this.productService = productService;
    }

    /*---Add new Product---*/
    @PostMapping("/product")
    public ResponseEntity<?> createProduct(@RequestBody Product product) {
        Product savedProduct = productService.save(product);
        ProductEvent productCreatedEvent = new ProductEvent("One Product is created", savedProduct);
        eventPublisher.publishEvent(productCreatedEvent);
        return ResponseEntity.ok().body("New Product has been saved with ID:" + savedProduct.getId());
    }

    /*---Get a Product by id---*/
    @GetMapping("/product/{id}")
    public ResponseEntity<Product> getProduct(@PathVariable("id") long id) {
        Optional<Product> returnedProduct = productService.get(id);
        Product product = returnedProduct.get();
```

```

        ProductEvent productCreatedEvent = new ProductEvent("One Product is retrieved", product);
        eventPublisher.publishEvent(productCreatedEvent);
        return ResponseEntity.ok().body(product);
    }

    /*---get all Product---*/
    @GetMapping("/product")
    public @ResponseBody Page<Product> getProductsByPage(
        @RequestParam(value="pagenumber", required=true, defaultValue="0") Integer pageNumber,
        @RequestParam(value="pagesize", required=true, defaultValue="20") Integer pageSize) {
        Page<Product> pagedProducts = productService.getProductsByPage(pageNumber, pageSize);
        return pagedProducts;
    }

    /*---Update a Product by id---*/
    @PutMapping("/product/{id}")
    public ResponseEntity<?> updateProduct(@PathVariable("id") long id, @RequestBody Product
product) {
        checkResourceFound(this.productService.get(id));
        productService.update(id, product);
        return ResponseEntity.ok().body("Product has been updated successfully.");
    }

    /*---Delete a Product by id---*/
    @DeleteMapping("/product/{id}")
    public ResponseEntity<?> deleteProduct(@PathVariable("id") long id) {
        checkResourceFound(this.productService.get(id));
        productService.delete(id);
        return ResponseEntity.ok().body("Product has been deleted successfully.");
    }
}

```

6.17 THE DOCKERFILE

```
FROM adoptopenjdk/openjdk15:alpine-jre
VOLUME /tmp
COPY build/libs/*.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

6.18 KUBERNETES DEPLOYMENT

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: category-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: category-deployment
  template:
    metadata:
      labels:
        app: category-deployment
    spec:
      containers:
        - name: aws-ecr-spring-boot-category
          image: <act-id>.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-product
          ports:
            - containerPort: 8081
            - containerPort: 8091
        env:
          - name: spring.profiles.active
            value: aws
      imagePullPolicy: Always
```

6.19 BUILDING THE SPRING BOOT MAIN CLASS

```
package com.rollingstone;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class RollingstoneEcommerceProductCatalogK8sApiApplication {

    public static void main(String[] args) {
        SpringApplication.run(RollingstoneEcommerceProductCatalogK8sApiApplication.class, args);
    }
}
```

6.20 SETTING THE SPRING CONFIG FILES

```
category.request.path = /category/  
category.port = 8092  
category.service.host = http://localhost
```

6.21 APPLICATION.YAML FOR LOCAL

```
server:
  port: 8081
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/rs_ecommerce?useSSL=false
    username: root
    password: root
    tomcat.max-wait: 20000
    tomcat.max-active: 50
    tomcat.max-idle: 20
    tomcat.min-idle: 15
    validationQuery: SELECT 1
  jpa:
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL5InnoDBDialect
    hibernate:
      ddl-auto: update

management:
  server:
    port: 8091
  endpoints:
    web:
      exposure:
        include: "*"
  endpoint:
    health:
      show-details: "always"
```

6.22 THE AWS PROFILE

```
server:
  port: 8081
spring:
  datasource:
    url: jdbc:mysql://rs-mortgage-aws.civxewyb4pfe.us-west-2.rds.amazonaws.com:3306/rs_ecommerce
    username: admin
    password: admin1973
    tomcat.max-wait: 20000
    tomcat.max-active: 50
    tomcat.max-idle: 20
    tomcat.min-idle: 15
    validationQuery: SELECT 1
  jpa:
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL5InnoDBDialect
    hibernate:
      ddl-auto: update

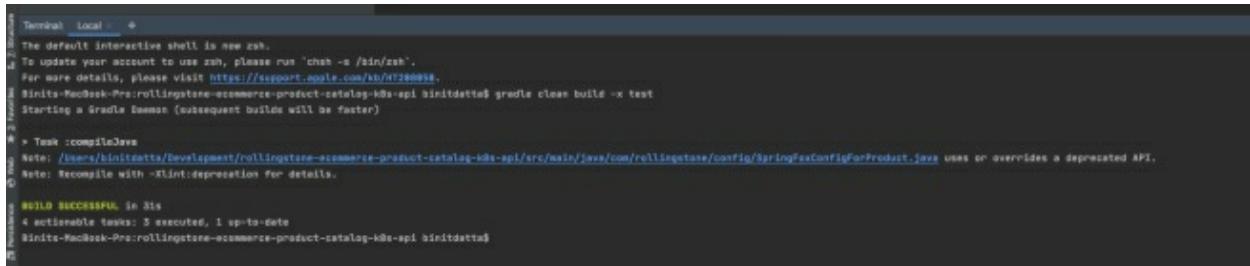
management:
  server:
    port: 8091
  endpoints:
    web:
      exposure:
        include: "*"
  endpoint:
    health:
      show-details: "always"
```

6.23 BOOTSTRAP.YAML

```
spring:  
  application:  
    name: rollingstone-ecommerce-product-catalog-k8s-api
```

6.24 BUILDING THE JAR

Open a command prompt / terminal, say from within your IntelliJ IDE and run the command gradle clean build -x test to build the jar



```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT248098.
Binitd-MacBook-Pro:rollingstone-ecommerce-product-catalog-k8s-api binitdette$ gradle clean build -x test
Starting a Gradle Daemon (subsequent builds will be faster)

+ Task :compileJava
note: /Users/binitdette/Development/rollingstone-ecommerce-product-catalog-k8s-api/src/main/java/com/rollingstone/config/SpringExtConfigForProduct.java uses or overrides a deprecated API.
note: Recompile with -Xlint:deprecation for details.

BUILD SUCCESSFUL in 35s
4 actionable tasks: 3 executed, 1 up-to-date
Binitd-MacBook-Pro:rollingstone-ecommerce-product-catalog-k8s-api binitdette$
```

6.25 RUNNING THE JAR

Run the jar using the command

```
java -jar build/libs/rollingstone-eCommerce-product-catalog-k8s-api-1.0.jar
```

As we have shown how to test the category Microservice locally, we will avoid that for the Product Microservice to save time and space. However, in real development, always test your service locally first to save AWS costs. The next chapter shows how to deploy the service to AWS EKS and test together.

CHAPTER 7

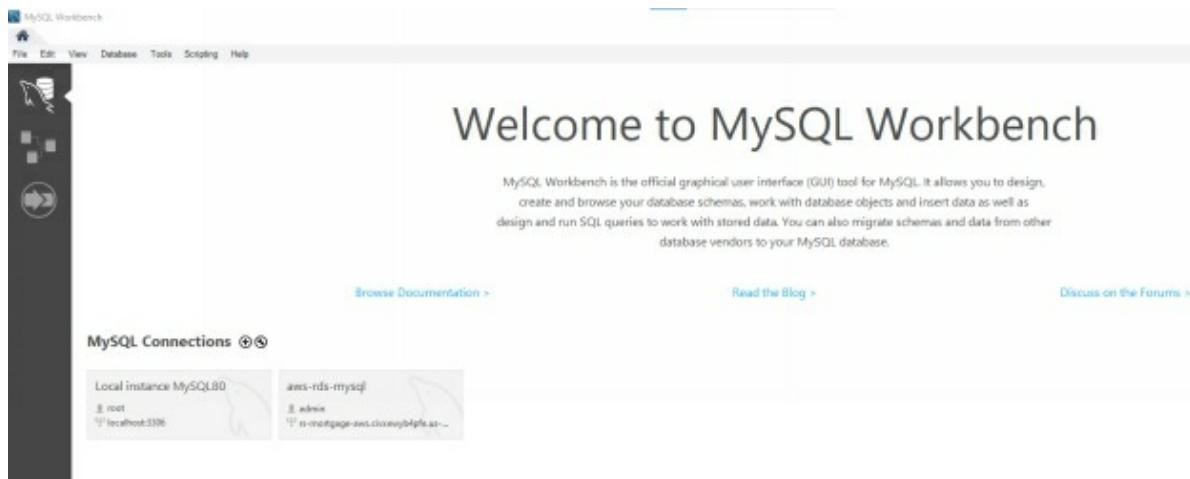
**Deploying Product REST API to
AWS EKS**

7 INTRODUCTION

Previously in the last chapter, we have deployed a single Category Microservice to the EKS Cluster. This chapter will deploy another Microservice, the Product catalog Microservice, to the same EKS Cluster. We want to demonstrate how a Microservice Client built-in Spring Boot calls another Microservice in the same cluster using the Kubernetes Service. We will also show how to configure a RestTemplate using the Apache HttpClient for a sustainable and performant Microservice. Along the way, we will also provide reasons why the RestTemplate default may not perform well in a Production environment under heavy load. Let us get started.

7.1 AWS DATABASE

First things first. Let's start at the bottom, which is our MySQL Database in AWS RDS. We would need the same database tables to be created in the AWS RDs MySQL Database as we did in our local MySQL. Start your MySQL Workbench Client application and double click on the AWS connection.

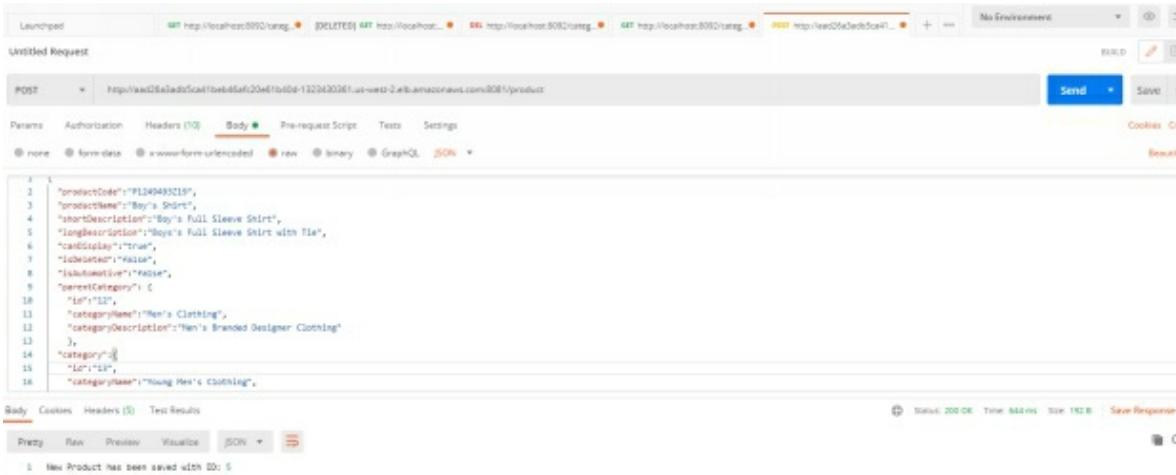


Locate the ddl.sql file in the project datascripts folder, copy and paste the create table statement in your MySQL Script within the MySQL Workbench.



```
53
54
55 * CREATE TABLE `rollingstone_product` (
56     `id` bigint(20) NOT NULL AUTO_INCREMENT,
57     `candidisplay` bit(1) NOT NULL,
58     `isautomotive` bit(1) NOT NULL,
59     `isdeleted` bit(1) NOT NULL,
60     `isinternational` bit(1) NOT NULL,
61     `long_description` varchar(255) NOT NULL,
62     `pcodes` varchar(255) NOT NULL,
63     `name` varchar(255) NOT NULL,
64     `short_description` varchar(255) NOT NULL,
65     `category_id` bigint(20) DEFAULT NULL,
66     `parent_category_id` bigint(20) DEFAULT NULL,
67     PRIMARY KEY (`id`),
68     KEY `FK1273qdxsgf8Erz216bbtj99t7` (`category_id`),
69     KEY `FKltvgl6yjjn6lpqgawmkE35c` (`parent_category_id`),
70     CONSTRAINT `FK1273qdxsgf8Erz216bbtj99t7` FOREIGN KEY (`category_id`) REFERENCES `rollingstone_category` (`id`),
71     CONSTRAINT `FKltvgl6yjjn6lpqgawmkE35c` FOREIGN KEY (`parent_category_id`) REFERENCES `rollingstone_category` (`id`)
72 ) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=latin1;
```

To also create a few seed data, locate the data.sql; in the same datascripts folder, copy the insert statement and paste them in your MySQL Workbench script tab. Once pasted, you can select them and click the execute icon.



POST http://localhost:2000/api/v1/product

Params Authorization Headers (10) Body **JSON** Pre-request Script Tests Settings

Body (raw JSON)

```
2 {"productCode": "PL1249493219",
3 "productName": "Boys Shirt",
4 "shortDescription": "Boys Full Sleeve Shirt",
5 "longDescription": "Boys Full Sleeve Shirt with Tie",
6 "candidisplay": "true",
7 "isDeleted": "false",
8 "isInternational": "false",
9 "parentCategory": 1
10 "id": 12,
11 "categoryName": "Men's Clothing",
12 "categoryDescription": "Men's Branded Designer Clothing"
13 },
14 "category": []
15 "id": 12,
16 "categoryName": "Young Men's Clothing",
```

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 844 ms Size: 192 B Save Response

7.2 ADDITIONAL CAUTION FOR AWS SECURITY

Under normal circumstances, Microservices would be residing in their private Git repositories, be built by CI/CD Pipelines, and deployed to the Kubernetes AWS EKS cluster protected by Enterprise Active Directory credentials, for example. Here, in this case, our code is in a public Git repository. Thus, we cannot expose our AWS account (root or IAM user account) to the public domain. Hackers will get those sensitive data and may misuse them, causing us financial damage.

What we will do instead of removing the sensitive part of the Microservice Kubernetes Service URLs from the Git repository themselves. Here is what we will do to complete the full cycle

- We will clone them in our bastion host,
- First deploy the Category Microservice to the EKS cluster,
- Get the Kubernetes Service external DNS name/IP for the Category Microservice
- Update the Product Microservice properties file for the AWS Spring profile property file
- Build the Product Microservice using gradle
- Build the Docker Image for the Product Microservice on our bastion host
- Deploy the Product Microservice to Kubernetes using kubectl apply command

Please remember editing application property files will be unnecessary in an internal production deployment environment.

Let's get started

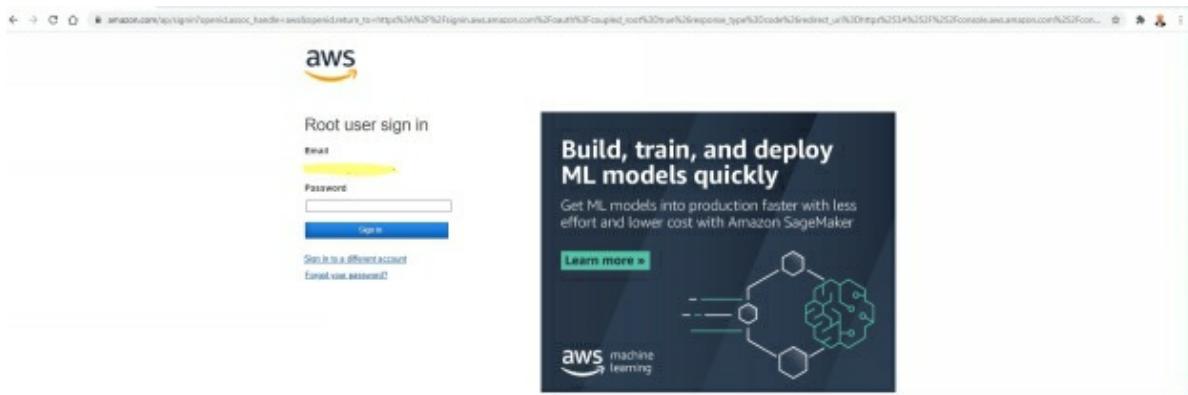
7.3 START BASTION HOST

First navigate to the AWS Management Console



© Amazon Web Services

Click on the MyAccount on the Right Top Corner



© Amazon Web Services

Enter your password to login to the AWS Management Console

The screenshot shows the AWS Management Console homepage. The left sidebar lists "AWS services" under "Recently visited services" and "All services". Under "Compute", "Containers", and "Storage", there are several service links. The right sidebar includes sections for "Stay connected to your AWS resources on-the-go", "Explore AWS" (with links to "Free Digital Training", "Build Apps Faster with GraphQL", "AWS Business Support", and "UI, micro API"), and "Have feedback?".

© Amazon Web Services

Find EC2 if it is not visible to you already. You can search EC2 on the top search bar

Click on EC2 when you see the link, which I can see on my screen

Shown below is the EC2 Console

The screenshot shows the EC2 Management Console. The left sidebar has sections for "EC2 Dashboard", "Instances", "Images", "Elastic Block Store", "Network & Security", and "Load Balancing". The main area displays "Resources" (Instances, Instances (Pending), Key pairs, Placement groups, Volumes) and "Launch instance" (with a "Launch Instance" button). It also shows "Scheduled events" (US West (Oregon)) and "Migrate a machine". The right sidebar includes "Account attributes" (Supported platforms: VPC, Default VPC, vpc-peering), "Explore AWS" (links to "Block Level Storage for EC2 Instances", "Get Up to 40% Better Price Performance", and "Launch Container AMIs with Fast Snapshot Restore (FSR)"), and "Additional information" (Getting started guide, Documentation).

© Amazon Web Services

Click on the Instances Link below the Instances Running Link

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 IP	Elastic IP
aws_7080_sqsl_vm	i-0f12621056eef8bfaf	Stopped	t2.medium	-	No alarms	us-west-2a	-	-	-
aws_rds	i-06c2b15464895bc36	Stopped	t2.medium	-	No alarms	us-west-2c	-	-	-
BastionHost	i-06c14798b35e9ecfa	Stopped	t2.medium	-	No alarms	us-west-2c	-	-	-

© Amazon Web Services

Check the checkbox on the left of the Bastion Host EC2 instance

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 IP	Elastic IP
aws_7080_sqsl_vm	i-0f12621056eef8bfaf	Stopped	t2.medium	-	No alarms	us-west-2a	-	-	-
aws_rds	i-06c2b15464895bc36	Stopped	t2.medium	-	No alarms	us-west-2c	-	-	-
BastionHost	i-06c14798b35e9ecfa	Stopped	t2.medium	-	No alarms	us-west-2c	-	-	-

© Amazon Web Services

Click On Instance State to open the Drop Down Menu

Welcome to the new instance experience! We're redesigning the EC2 console to make it easier to use. To switch between the old console and the new console, use the New EC2 Experience toggle above the navigation panel. We'll release updates continuously based on customer feedback.

Instances (1/3) info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
AWS_TODO_SQL_VH	i-093262506ae48ba5d	Stopped	t2.medium	-	No alarms +	us-west-2a	-	-	-
AWS_DNS	i-00e20154068808c56	Stopped	t2.medium	-	No alarms +	us-west-2c	-	-	-
BastionHost	i-06c141f68d5e9c1a	Stopped	t2.medium	-	No alarms +	us-west-2c	-	-	-

Actions

- Stop instance
- Start instance
- Reboot instance
- Hibernate instance
- Terminate instance

Launch instances

Instance i-06c141f68d5e9c1a [BastionHost]

Details Security Networking Storage Status Checks Monitoring Tags

Instance summary - info

Instance ID	i-06c141f68d5e9c1a [BastionHost]	Public IPv4 address	Private IPv4 address
Instance state	Stopped	Public IPv4 DNS	Private IPv4 DNS
Instance type	t2.medium	Elastic IP addresses	VPC ID
Amazon Compute Optimizer findings	(Opt-in to AWS Compute Optimizer for recommendations.) Learn more	VM Role	Subnet ID

Instance details - info

© Amazon Web Services

Click on Start Instance to see the below

Welcome to the new instance experience! We're redesigning the EC2 console to make it easier to use. To switch between the old console and the new console, use the New EC2 Experience toggle above the navigation panel. We'll release updates continuously based on customer feedback.

Successfully started i-06c141f68d5e9c1a

Instances (1/3) info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
AWS_TODO_SQL_VH	i-093262506ae48ba5d	Stopped	t2.medium	-	No alarms +	us-west-2a	-	-	-
AWS_DNS	i-00e20154068808c56	Stopped	t2.medium	-	No alarms +	us-west-2c	-	-	-
BastionHost	i-06c141f68d5e9c1a	Stopped	t2.medium	-	No alarms +	us-west-2c	-	-	-

© Amazon Web Services

The EC2 Bastion Host instance status is now Initializing.

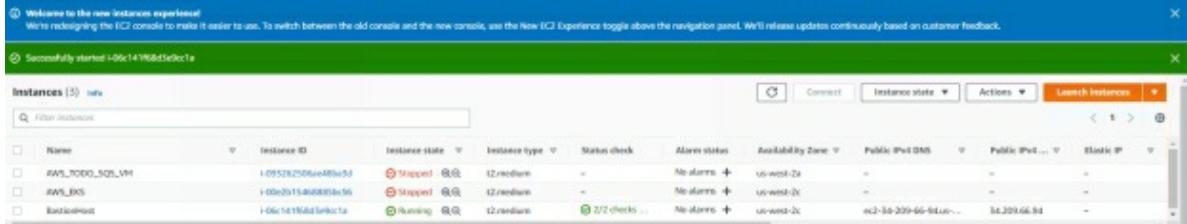
Welcome to the new instance experience! We're redesigning the EC2 console to make it easier to use. To switch between the old console and the new console, use the New EC2 Experience toggle above the navigation panel. We'll release updates continuously based on customer feedback.

Successfully started i-06c141f68d5e9c1a

Instances (3) info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
AWS_TODO_SQL_VH	i-093262506ae48ba5d	Stopped	t2.medium	-	No alarms +	us-west-2a	-	-	-
AWS_DNS	i-00e20154068808c56	Stopped	t2.medium	-	No alarms +	us-west-2c	-	-	-
BastionHost	i-06c141f68d5e9c1a	Running	t2.medium	Initializing	No alarms +	us-west-2c	ec2-54-209-66-94.us... 34.209.66.54	-	-

Wait till we see the following 2/2 Checked Status



Name	Instance ID	Instance State	Instance type	Status check	Alarm status	Availability zone	Public IP(s) SNS	Public IP(s)	Elastic IP
AWS_7000_505_VH	i-0952b2506e48fb5d	Stopped	t2.medium	-	No alarm	us-west-2a	-	-	-
AWS_EBS	i-09e2015460889bc56	Stopped	t2.medium	-	No alarm	us-west-2c	-	-	-
localhost	i-06c14119d1efc1ca	Running	t2.medium	2/2 checks	No alarm	us-west-2c	ec2-50-209-66-84.us.../	50.209.66.84	-

7.4 SSH INTO THE BASTION HOST

Find the local directory folder where you have kept your .pem file, the Key Pair file for ssh ing into our Bastion Host. We can also use a path to the pem file but that is errorprone.

Name	Date modified	Type	Size
AWSEKS_KP.pem	12/23/2020 9:45 AM	PEM File	2 KB
BastionHostKeyPair.pem	1/3/2021 2:04 PM	PEM File	2 KB
[REDACTED]	1/3/2021 2:39 PM	Microsoft Excel C...	1 KB

© Amazon Web Services

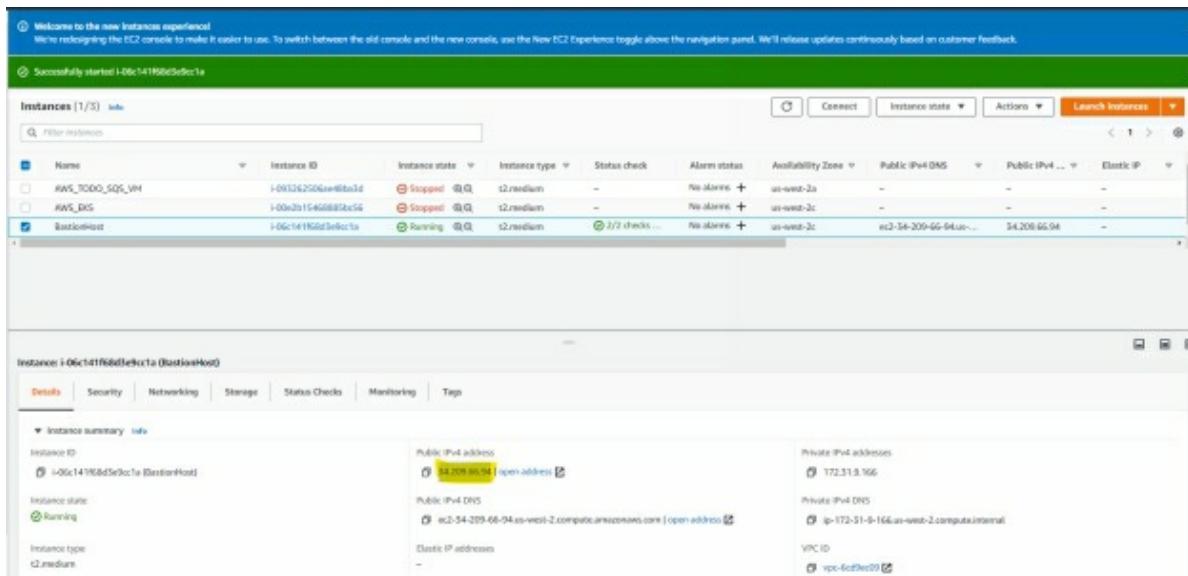
Right click on this folder and choose Open Git Bash



```
MINGW64:/c/AWS/Book/SSH
bidatta@LNAR-5CG03294K5 MINGW64 /c/AWS/Book/SSH
$
```

© Amazon Web Services

Click the checkbox on the left of the Bastion Host, locate the public IP and copy it



Welcome to the new Instances experience! We're redesigning the EC2 console to make it easier to use. To switch between the old console and the new console, use the New EC2 Experience toggle above the navigation pane. We'll release updates continuously based on customer feedback.

Successfully started i-06c141f0d8d3e9cta

Instances [1/3] [Info](#)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
AWS_TODO_SQL_VH	i-003262506ee4fb3d	Stopped	t2.medium	-	No alarm	+ us-west-2a	-	-	-
AWS_DNS	i-00e3b15468883bc5	Stopped	t2.medium	-	No alarm	+ us-west-2c	-	-	-
BastionHost	i-06c141f0d8d3e9cta	Running	t2.medium	2/2 checks ...	No alarm	+ us-west-2c	ec2-34-209-66-94.us... 34.209.66.94	-	-

Instance: i-06c141f0d8d3e9cta (BastionHost)

[Details](#) [Security](#) [Networking](#) [Storage](#) [Status Checks](#) [Monitoring](#) [Tags](#)

Instance summary

Instance ID: i-06c141f0d8d3e9cta (BastionHost)

Instance state: **Running**

Instance type: t2.medium

Public IPv4 address: [34.209.66.94](#) [open address]

Public IPv4 DNS: [ec2-34-209-66-94.us-west-2.compute.amazonaws.com](#) [open address]

Private IPv4 addresses:

- 172.31.3.166

Private IPv4 DNS:

- ip-172-31-3-166.us-west-2.compute.internal

VPC ID: [vpc-6cf9ec09](#)

© Amazon Web Services

Now enter the command to ssh into our Bastion Host

ssh -i BastionHostKeyPair.pem ubuntu@34.209.66.94

NOTE: Your IP and Key name would be different, though

```
ubuntu@ip-172-31-9-166: ~
bidatta@LN4R-SCG03294KS MINGW64 /c/AWS/Book/SSH
$ ssh -i BastionHostKeyPair.pem ubuntu@34.209.66.94
The authenticity of host '34.209.66.94 (34.209.66.94)' can't be established.
ECDSA key fingerprint is SHA256:YvHVn1Pav0lvNllxEvzPYV2CTv29mATRut+OMj2gLkw.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '34.209.66.94' (ECDSA) to the list of known hosts.
welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1035-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Sat Jan 23 15:18:04 UTC 2021

System load: 0.0          Processes:      105
Usage of /: 63.0% of 7.69GB  Users logged in:   0
Memory usage: 5%           IP address for eth0: 172.31.9.166
Swap usage:  0%

* Introducing self-healing high availability clusters in MicroK8s.
  Simple, hardened, Kubernetes for production, from RaspberryPi to DC.

  https://microk8s.io/high-availability

29 packages can be updated.
0 updates are security updates.

Last login: Sun Jan 17 14:13:18 2021 from 69.136.183.76
ubuntu@ip-172-31-9-166:~$ |
```

7.5 CLONE THE GIT REPOSITORIES

We need to get the code of the two Spring Boot Microservices first. Let's make a new folder and clone them in that folder. Run the following git clone command one after another.

NOTE: The accompanying image shows a different Git repo but use the one below

```
git clone https://github.com/binitauthor/rollingstone-ecommerce-product-catalog-k8s-api.git
```

```
ubuntu@ip-172-31-9-166:~/test$ git clone https://github.com/binitdatta/rollingstone-ecommerce-product-catalog-k8s-api.git
Cloning into 'rollingstone-ecommerce-product-catalog-k8s-api'...
remote: Enumerating objects: 75, done.
remote: Counting objects: 100% (75/75), done.
remote: Compressing objects: 100% (56/56), done.
remote: Total 75 (delta 9), reused 75 (delta 9), pack-reused 0
Unpacking objects: 100% (75/75), done.
ubuntu@ip-172-31-9-166:~/test$
```

```
git clone https://github.com/binitauthor/rollingstone-ecommerce-category-k8s-api.git
```

NOTE: The accompanying image shows a different Git repo but use the one below

```
ubuntu@ip-172-31-9-166:~/test$ git clone https://github.com/binitdatta/rollingstone-ecommerce-category-k8s-api.git
Cloning into 'rollingstone-ecommerce-category-k8s-api'...
remote: Enumerating objects: 69, done.
remote: Counting objects: 100% (69/69), done.
remote: Compressing objects: 100% (50/50), done.
remote: Total 69 (delta 5), reused 66 (delta 5), pack-reused 0
Unpacking objects: 100% (69/69), done.
ubuntu@ip-172-31-9-166:~/test$
```

One clarification here. We already deployed and tested the Category Microservice in the last chapter. For cost savings, I deleted my EKS cluster, and my Category service disappeared. Besides, my Category service in the public git repository is not deployable unless we clone it and provide the accurate AWS Elastic Container Repository URL to push the new image. Thus we will have to repeat a few steps here for testing the Product Microservice calling the Category Microservice for validating the Category

in the JSON payload.

7.6 UPDATE CATEGORY REPOSITORY KUBERNETES DEPLOYMENT

Change directory to the Category Microservice application using

```
cd rollingstone-eCommerce-category-k8s-api/
```

```
ubuntu@ip-172-31-9-166:~/test/rollingstone-eCommerce-category-k8s-api$ ls -lt
total 8
drwxrwxr-x 7 ubuntu ubuntu 4096 Jan 23 15:22 rollingstone-eCommerce-category-k8s-api
drwxrwxr-x 6 ubuntu ubuntu 4096 Jan 23 15:21 rollingstone-eCommerce-product-catalog-k8s-api
ubuntu@ip-172-31-9-166:~/test$ cd rollingstone-eCommerce-category-k8s-api/
ubuntu@ip-172-31-9-166:~/test/rollingstone-eCommerce-category-k8s-api$ |
```

Locate the file named category-kubernetes-deployment.yaml at the root of the application cloned git repository folder

```
ubuntu@ip-172-31-9-166:~/test/rollingstone-eCommerce-category-k8s-api$ ls -lt
total 8
drwxrwxr-x 7 ubuntu ubuntu 4096 Jan 23 15:22 rollingstone-eCommerce-category-k8s-api
drwxrwxr-x 6 ubuntu ubuntu 4096 Jan 23 15:21 rollingstone-eCommerce-product-catalog-k8s-api
ubuntu@ip-172-31-9-166:~/test$ cd rollingstone-eCommerce-category-k8s-api/
ubuntu@ip-172-31-9-166:~/test/rollingstone-eCommerce-category-k8s-api$ ls -lt
total 48
-rw-rw-r-- 1 ubuntu ubuntu 118 Jan 23 15:22 Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu 43 Jan 23 15:22 README.md
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 23 15:22 WebTestsDOC
-rw-rw-r-- 1 ubuntu ubuntu 953 Jan 23 15:22 build.gradle
-rw-rw-r-- 1 ubuntu ubuntu 589 Jan 23 15:22 category-kubernetes-deployment.yaml
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 23 15:22 datascripts
drwxrwxr-x 3 ubuntu ubuntu 4096 Jan 23 15:22 gradle
-rw-rw-r-- 1 ubuntu ubuntu 5766 Jan 23 15:22 gradlew
-rw-rw-r-- 1 ubuntu ubuntu 2674 Jan 23 15:22 gradlew.bat
-rw-rw-r-- 1 ubuntu ubuntu 61 Jan 23 15:22 settings.gradle
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 23 15:22 src
ubuntu@ip-172-31-9-166:~/test/rollingstone-eCommerce-category-k8s-api$ |
```

Open the file using vi editor

```
vi category-kubernetes-deployment.yaml
```

Locate the following file

image: <act-id>.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-category

```

ubuntu@ip-172-31-9-166:~/test/rollingstone-e-commerce-category-k8s-api
apiVersion: apps/v1
kind: Deployment
metadata:
  name: category-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: category-deployment
  template:
    metadata:
      labels:
        app: category-deployment
    spec:
      containers:
        - name: aws-ecr-spring-boot-category
          image: <act-id>.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-category
          ports:
            - containerPort: 8092
            - containerPort: 8093
          env:
            - name: spring.profiles.active
              value: aws
          imagePullPolicy: Always

```

We need to replace the <act-id> with our AWS account id and save the file with wq! To come out of the vi editor

Next, let's verify the AWS Spring Profile property file.

Navigate to the following directory in the Category application

```
cd src/main/resources/
```

List the files on that directory

```

ubuntu@ip-172-31-9-166:~/test/rollingstone-e-commerce-category-k8s-api$ vi category-kubernetes-deployment.yaml
ubuntu@ip-172-31-9-166:~/test/rollingstone-e-commerce-category-k8s-api$ pwd
/home/ubuntu/test/rollingstone-e-commerce-category-k8s-api
ubuntu@ip-172-31-9-166:~/test/rollingstone-e-commerce-category-k8s-api$ cd src/main/resources/
ubuntu@ip-172-31-9-166:~/test/rollingstone-e-commerce-category-k8s-api/src/main/resources$ ls -lt
total 8
-rw-rw-r-- 1 ubuntu ubuntu 611 Jan 23 15:22 application-aws.yaml
-rw-rw-r-- 1 ubuntu ubuntu 558 Jan 23 15:22 application.yaml
ubuntu@ip-172-31-9-166:~/test/rollingstone-e-commerce-category-k8s-api/src/main/resources$ :

```

Make sure that the application-aws.yaml file Database properties, URL, username and password etc are matching with the AWS RDS URL, username and password etc.

```

ubuntu@ip-172-31-9-166: ~/test/rollingstone-ecommerce-category-k8s-api/src/main/resources
server:
  port: 8092
spring:
  datasource:
    url: jdbc:mysql://rs-mortgage-aws.civxewyb4pfe.us-west-2.rds.amazonaws.com:3306/rs_ecommerce
    username: admin
    password: [REDACTED]
    tomcat.max-wait: 20000
    tomcat.max-active: 50
    tomcat.max-idle: 20
    tomcat.min-idle: 15
    validationQuery: SELECT 1
  jpa:
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL5InnoDBDialect
  hibernate:
    ddl-auto: update

management:
  server:
    port: 8093
  endpoints:
    web:
      exposure:
        include: "*"
  endpoint:
    health:
      show-details: "always"
~
```

Edit if needed. Otherwise save the file with !wq and exit the VI editor.

Follow the steps below the deploy the Category Service to EKS

1. Build the application
 - a. gradle clean build -x test
2. Build the Docker image for the Category application
 - a. sudo docker build -t aws-ecr-spring-boot-category/latest .
3. Navigate to the AWS ECR Repository and find the AWS ECR Repo URL
 - a. <your-aws-acct-id-here>.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-category
4. Tag the newly built Docker image for the Category application with
 - a. sudo docker tag aws-ecr-spring-boot-category/latest <your-aws-acct-id-here>.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-category
5. Now Login to the AWS ECR with
 - a. aws ecr get-login-password --region us-west-2 | sudo docker login --username AWS --password-stdin <your-

aws-act-id-here>.dkr.ecr.us-west-2.amazonaws.com

6. Push the Docker Tagged image to the AWS ECR Repository with
 - a. sudo docker push <your-aws-act-id-here>.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-category
7. Navigate back to the root of the category application directory with
 - a. Cd ../../..
8. View the category-kubernetes-deployment.yaml with for a last check before we deploy
 - a. vi category-kubernetes-deployment.yaml
9. Deploy the Category Service to EKS with
 - a. kubectl apply -f ./category-kubernetes-deployment.yaml
10. Expose the Category Service Pods to the external world with
 - a. kubectl expose deployment category-deployment --type=LoadBalancer --name=category-service

Remember we talked about why Kubernetes Pods (all Kubernetes, On Prem, Minikube, Google, AWS and Azure Cloud, IBM, Oracle Alibaba included) do not have public IP addresses. Pods are expected to die any time and Kubernetes can create them and destroy them as it sees fit. Every time a new Pod comes up, it gets a dynamic new IP address assigned by Kubernetes cluster. Thus, external clients cannot rely on the Pods IP address.

The solution is to have a static domain name / IP address that can even survive Kubernetes cluster restarts. Mind you the static Kubernetes Service URL is part of codebase / property file and it cannot change dynamically without difficulties in engineering. A Kubernetes Service is basically a one to many logical load balancing abstractions. One static and always reachable Service URL load balances the traffic coming its way, among the currently running Pods that the Kubernetes Deployment has created. Remember the we have reviewed the Deployment yaml file and one of the elements was replicas. If we say, replicas would be 5, Kubernetes Deployment and the Scheduler and Controllers running in the Master Control Plane would ensure that at least 5 Pods are always running for the application Microservice.

Back to services again. Now, these 5 pods have their own private IP address. And the Service will have to know their private IP addresses. But how?

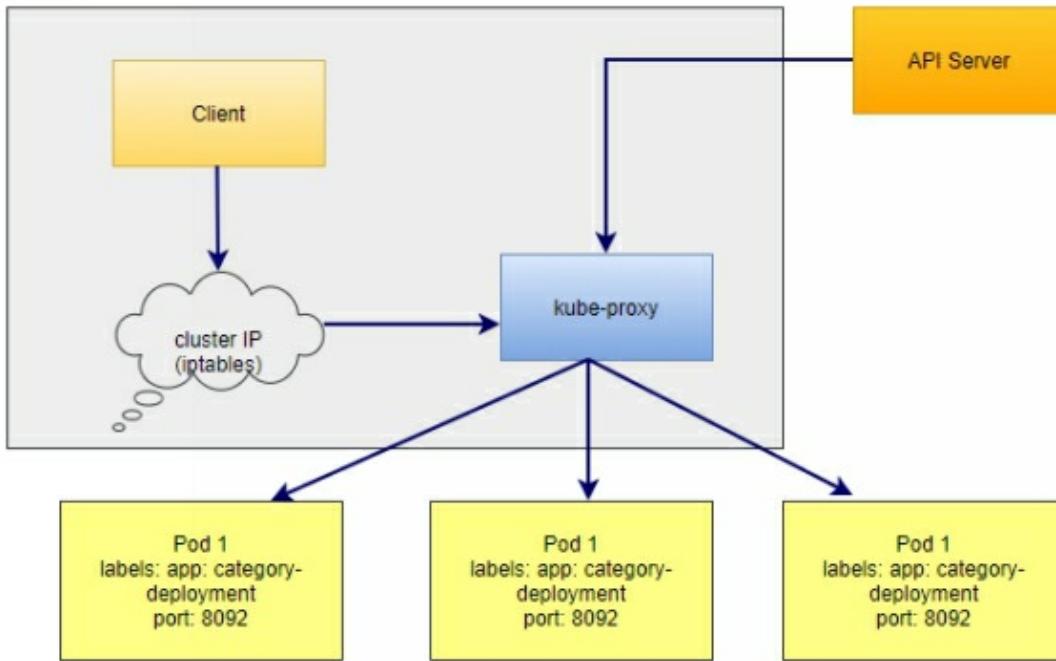
Remember, in the same Deployment yaml file, we included

```
spec:  
  replicas: 1  
  selector:  
    matchLabels:  
      app: category-deployment
```

Thus, we gave each Pod a label and told Kubernetes to make sure all Pods running for the category Microservice have this “category-deployment” label. What are Labels by the way? They are key value pairs that are attached to the Kubernetes objects, any object be it Pods, Deployment, Service, or anything else, can have labels.

When we discussed, the Kubernetes architecture, remember we talked about a Etcd database that is part of the Kubernetes control plane. This database itself is replicated, distributed and highly available and self-healing like Kubernetes itself. When Kubernetes cluster is created and starts up for first time, it creates a lot of data in its own ETCD database. A lot of Kubernetes own services runs as Pods themselves. Now, whenever we deploy a Kubernetes objects like Pods (through the Deployment file we just reviewed) through the kubectl command or through the GUI interface, Kubernetes created fresh data in the ETCD database. The labels we just talked about in the Category service Deployment file are also part of the ETCD database. The object that holds these Pods definition is called an Endpoint Object inside Kubernetes.

Kubernetes Service Controller constantly Queries this Endpoint Object with the label we gave to the Deployment. When one of the Pod dies and Kubernetes creates a new Pod, the Endpoint object gets a POST from the Kubernetes Controller and updates its state. This is how the Kubernetes Service (available statically to the external clients) make sure that it always has latest IP addresses of the running Pods. Following is a logical diagram of showing a representative workflow



© Kubernetes Docs

The next interesting and very critical topic is the service type. Like there are many different types of services in real life (financial services, legal services, to name a few), Kubernetes also has different types of services. We may want some services not to be accessible outside the cluster, but they still need the static Service IP/DNS. For other type of services, we may want external exposed service IP/DNS. Following are some of the most used Service types

- **Cluster IP:** In this case (the default) the Service IP will be an internal one if we do not want the service to be exposed outside the cluster. Services using cluster IP can only be accessed from inside the cluster i.e., from other pods running in the same cluster.
- **NodePort:** This type of service exposes the service on each of the Kubernetes worker Nodes IP with a static port. Clients from outside the cluster can reach the NodePort service using Node IP: Port pattern. NodePort automatically also creates a ClusterIP behind the scene.
- **LoadBalancer:** This type of service will create a Cloud Provider

i.e., AWS Load Balancer and configure the load balance to route traffic to the NodePort:Port IP/Port combination. This type automatically creates a NodePort and a Cluster IP.

Enough background on Kubernetes networking and let's get back to testing the Category Service

1. Run the following command to get the Category Service DNS name
 - a. `kubectl get svc`
2. Test the GET Endpoint (Your Endpoint URL will be different than mine)
 - a. Open PostMan or a Chrome / Safari browser
 - b. `http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8092/category`
3. Test PostMapping
 - a. `http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8092/category`
 - b. {

 "categoryName": "Young Men's Clothing",
 "categoryDescription": "Young Men's Branded Designer
Clothing"
}
4. Test GET One Category (Your ID could be different)
 - a. `http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8092/category/11`
5. Test PutMapping
 - a. `http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8092/category/11`
 - b. {

 "id":11,
 "categoryName": "Young Men's Clothing",

```
        "categoryDescription": "Young Men's Branded Designer  
Clothing"  
    }  
}
```

6. Test Update Verify
 - a. http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8092/category/11
7. Test Deleting One Category in PostMan
 - a. http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8092/category/11
8. Test some Actuator Endpoints for the Category Service (See the Actuator port we exposed earlier)
 - a. http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator
 - b. http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/metrics/com.rollings
 - c. http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/is-customer-healthy
 - d. http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/health
 - e. http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/env
 - f. http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/heapdump
 - g. http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8093/actuator/threaddump
 - h. http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-

2.elb.amazonaws.com:8093/actuator/metrics

With the Category Service deployed and working, lets us now deploy the Products Microservice.

7.7 UPDATE AWS PROFILE PROPERTY FILE

Like we have made sure we have the accurate AWS RDS details in our Spring Profile file in the category Microservice, we need to do the same for the Product Catalog Service as well.

Navigate to the top root directly of the Product Catalog Microservice in our Bastion Host

Run the following command

```
cd src/main/resources
```

Run the following command to view the Spring AWS profile file.

```
vi application-aws.yaml
```

This time, we also have another AWS specific properties file named as application-aws.properties. After reviewing and making sure AWS RDS credentials are accurate in the application-aws.yaml file, open the following file in the same folder

```
vi application-aws.properties
```

Update the aws specific application-aws.properties file to have the accurate /category service AWS EKS Service host DNS Name.

```
category.request.path = /category/
```

```
category.port = 8092
```

```
category.service.host = http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com
```

NOTE : Your URL will be different, and you can get that with

```
kubectl get svc command
```

7.8 BUILD THE PRODUCT APPLICATION

Now let us deploy the Product Microservice which depends on the Category Microservice

7.9 NAVIGATE TO THE PRODUCT APP

cd ..

cd rollingstone-ecommerce-product-catalog-k8s-api/

Let us now build the application with

gradle clean build -x test

```
ubuntu@ip-172-31-9-166:~/rollingstone-ecommerce-product-catalog-k8s-api$ gradle clean build -x test
> Task :compileJava
Note: /home/ubuntu/rollingstone-ecommerce-product-catalog-k8s-api/src/main/java/com/rollingstone/config/SpringFoxConfigForProduct.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

BUILD SUCCESSFUL in 2s
4 actionable tasks: 4 executed
ubuntu@ip-172-31-9-166:~/rollingstone-ecommerce-product-catalog-k8s-api$ |
```

7.10 BUILD THE PRODUCT DOCKER IMAGE

```
sudo docker build -t aws-ecr-spring-boot-product-catalog/latest .
```

```
ubuntu@ip-172-31-9-166:~/rollingstone-e-commerce-product-catalog-k8s-api$ sudo docker build -t aws-ecr-spring-boot-category/latest .
Sending build context to Docker daemon 68.09MB
Step 1/4 : FROM adoptopenjdk/openjdk15:alpine-jre
--> 029fb36fffdc7
Step 2/4 : VOLUME /tmp
--> Using cache
--> 2cefe7ffb8b7
Step 3/4 : COPY build/libs/*.jar app.jar
--> 7efc39be414
Step 4/4 : ENTRYPOINT ["java","-jar","/app.jar"]
--> Running in ff0b482293c6
Removing intermediate container ff0b482293c6
--> f3c60337fdc8
Successfully built f3c60337fdc8
Successfully tagged aws-ecr-spring-boot-category/latest:latest
ubuntu@ip-172-31-9-166:~/rollingstone-e-commerce-product-catalog-k8s-api$ |
```

7.11 GET THE ECR REPO NAME

Login to your AWS Account, navigate to the Elastic Container Repository (ECR) and get the URL of the Category Repository we created earlier. We need that to first tag and then push the docker image to ECR

<aws-account-id>.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-product-catalog

NOTE: Your Account ID needs to be used.

7.12 TAG THE IMAGE

Now, tag the Product Service Docker image with the following command. Apply and replace your account id.

```
sudo docker tag aws-ecr-spring-boot-product-catalog/latest:latest <aws-account-id>.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-product-catalog
```

7.13 LOGIN TO AWS ECR

Like we did for the Category Microservice, log in to AWS ECR with the following command

```
aws ecr get-login-password --region us-west-2 | sudo docker login --username AWS --password-stdin <aws-account-id>.dkr.ecr.us-west-2.amazonaws.com
```

7.14 PUSH IMAGE TO ECR

Now push the Product Catalog

```
sudo docker push <aws-account-id>.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-product-catalog
```

```
[root@ip-172-31-9-166 ~]# rollingstone-ecommerce-product-catalog-k8s-api$ sudo docker push 10722267664.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-product-catalog
The push refers to repository [10722267664.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-product-catalog]
cfa51f404930: Pushed
33fe87ac3623: Layer already exists
b47eb67ec3be: Layer already exists
777b2cd48970: Layer already exists
latency: digest: sha256:10b9b56fb26e48119e174e68f4ed0980cdf46c0cd54b4e3d4de4c47c98e13b64 size: 1363
ubuntu@ip-172-31-9-166:~/rollingstone-ecommerce-product-catalog-k8s-api$
```

7.15 Deploy Category Microservice to EKS

Following is the file we will use

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-catalog-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product-catalog-deployment
  template:
    metadata:
      labels:
        app: product-catalog-deployment
    spec:
      containers:
        - name: aws-ecr-spring-boot-category
          image: <act-id>.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-product
          ports:
            - containerPort: 8081
            - containerPort: 8091
          env:
            - name: spring.profiles.active
              value: aws
      imagePullPolicy: Always
```

Deploy it from your Bastion Host

```
kubectl apply -f ./category-kubernetes-deployment.yaml
```

```
deployment.apps/category-deployment created
```

7.16 EXPOSE THE CATEGORY MICROSERVICE

```
kubectl expose deployment product-catalog-deployment --  
type=LoadBalancer --name=product-catalog-servicesservice/category-service  
exposed
```

7.17 VIEW THE PODS

We can get the Pods by running the following command

```
kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
category-deployment-7b675894f5-nlmgh	1/1	Running	0	37m
product-catalog-deployment-56bbcb9987-pn268	1/1	Running	0	5s

7.18 VIEW THE KUBERNETES LOG

At times we may want to check the logs of a specific Pod. Please run the following command for doing that

```
kubectl logs category-deployment-7b675894f5-pd2nt
```

7.19 CHECK THE EXTERNAL IP

We need the external IP to test the Product Service. Run the following command

```
kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-PORT(S)	AGE
category-				
service	LoadBalancer	10.100.89.52	a7cb07fb723eb443c8f59e7ae58c1055039089.us-west-2.elb.amazonaws.com	8092:31081/TCP,8093:30052/TCP 2d1h
product-catalog-				
service	LoadBalancer	10.100.230.109	aad26a3adb5ca41beb46afc20e61b41323430361.us-west-2.elb.amazonaws.com	8081:31256/TCP,8091:31186/TCP 2m44s

7.20 GET PRODUCT EXTERNAL IP

```
kubectl get svc
```

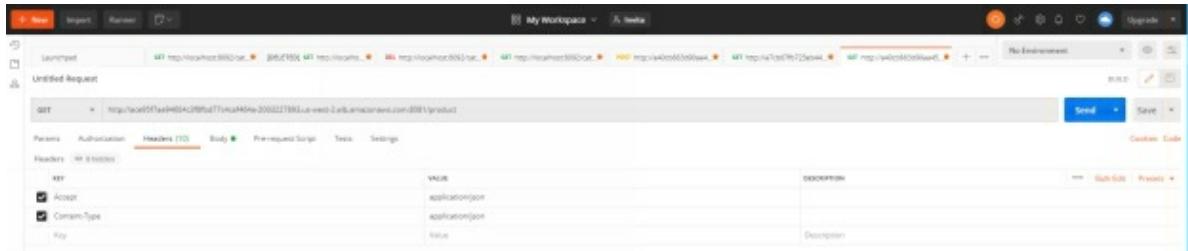
```
ubuntu@ip-172-31-9-166:~/rollingstone-ecommerce-product-catalog-k8s-api$ kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
category-service   LoadBalancer   10.100.217.80    a40cb663d90aa45dd84623c2801dbf6-582124535.us-west-2.elb.amazonaws.com   8092:30994/TCP,8093:30338/TCP   38m
kubernetes       ClusterIP     10.100.0.1    <none>        443/TCP          55m
product-catalog-service   LoadBalancer   10.100.23.232   ace95f7ae44604c3f8fbfd77c4caf464a-2003227893.us-west-2.elb.amazonaws.com   8081:30233/TCP,8091:32505/TCP   88s
ubuntu@ip-172-31-9-166:~/rollingstone-ecommerce-product-catalog-k8s-api$ |
```

7.21 TEST GET ALL PRODUCTS

Let us open our PostMan or Chrome Advanced REST Client and tey the following one by one.

<http://ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8081/product> (NOTE: Your URL would be different)

Make sure that we have the following two headers



The screenshot shows the PostMan interface with a single request listed:

- Method:** GET
- URL:** <http://ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8081/product>
- Headers Tab:** The Headers tab is selected, showing the following configuration:
 - Key:** Accept, **Value:** application/json
 - Key:** Content-Type, **Value:** application/json
- Buttons:** Send, Save, and a blue highlighted button.

Hit Send and the watch the results panel below. It should bring back results with the HTTP status code as 200

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'File', 'Edit', 'Run', 'Help', and a 'Send' button. Below the navigation is a toolbar with icons for 'Import', 'Runner', and 'Logout'. The main area is titled 'My Workspace' with a 'Create New' button. A list of recent requests is shown, including one for 'http://localhost:3001/api/products'. A modal window titled 'Untitled Request' is open, showing an 'HTTP' method and the URL 'http://localhost:3001/api/products'. Below the URL, there are tabs for 'Params', 'Authorization', 'Headers', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. Under the 'Body' tab, there's a 'Raw' section with a key-value pair: 'key' and 'value'. To the right of the body, there are 'Description' and 'HTTP Body' buttons. At the bottom of the modal, there are 'Send' and 'Save' buttons. The results panel at the bottom shows a JSON response with the following content:

```
1: { "id": 4,
2:   "name": "SAMSUNG 4K",
3:   "price": 1000000,
4:   "description": "Samsung 4K Smart TV",
5:   "category": "Electronics",
6:   "subcategory": "TVs",
7:   "dimensions": "32 inch Samsung television with remote included",
8:   "weight": "50kg",
9:   "productType": "TV",
10:  "productID": "P4",
11:  "categoryID": "C1",
12:  "subcategoryID": "S1",
13:  "categoryName": "Electronics",
14:  "subcategoryName": "Electronics"
15: },
16: {
17:   "category": "Electronics"
18: }
```

The results panel also includes tabs for 'Body', 'Cookies', 'Headers', 'Tests', and 'Results'. The status bar at the bottom right indicates 'Status: 200 OK', 'Time: 570 ms', 'Size: 1.24 kB', and a 'Save Response' button.

7.22 TEST POST

Now let us try to create a new Product through the same Service Endpoint but with the HTTP POST Method. Keep the same headers used in the GET. The request body is below

```
{  
    "productCode": "P1249493Z19",  
    "productName": "Boy's Shirt",  
    "shortDescription": "Boy's Full Sleeve Shirt",  
    "longDescription": "Boys's Full Sleeve Shirt with Tie",  
    "canDisplay": "true",  
    "isDeleted": "false",  
    "isAutomotive": "false",  
    "parentCategory": {  
        "id": "6",  
        "categoryName": "Men's Clothing",  
        "categoryDescription": "Men's Branded Designer Clothing"  
    },  
    "category": {  
        "id": "7",  
        "categoryName": "Young Men's Clothing",  
        "categoryDescription": "Young Men's Branded Designer Clothing"  
    }  
}
```

It seems our request failed. The first check is to make sure the Category IDs are accurate with our database. Your Category ids will be different than mine.

The screenshot shows a Postman request to a Lambda function at `https://ecc69f7ee5460a398fbfd77cdec464e-2003227893.us-west-2.amazonaws.com:8081/product`. The request body is a JSON object with various fields like shortDescription, longDescription, canDisplay, isDeleted, isAutomatic, parentCategory, and category. The response status is 400 Bad Request, with a message and details indicating that category IDs 12 and 13 are invalid.

```

4 "shortDescription": "Boy's Full Sleeve Shirt",
5 "longDescription": "Boys's Full Sleeve Shirt with Tie",
6 "canDisplay": "true",
7 "isDeleted": "false",
8 "isAutomatic": "false",
9 "parentCategory": {
10   "id": "12",
11   "categoryName": "Men's Clothing",
12   "categoryDescription": "Men's Branded Designer Clothing"
13 },
14 "category": [
15   "id": "13",
16   "categoryName": "Young Men's Clothing",
17   "categoryDescription": "Young Men's Branded Designer Clothing"
18 ]

```

```

1 {}
2   "message": "Bad Request as Category Can not be invalid",
3   "details": "The Request did not have the correct parameters"
4

```

Let's see what the valid category ids is

The screenshot shows the MySQL Workbench interface with a query window running the SQL command `SELECT * FROM rs_ecommerce.rollingstone_category;`. The result grid displays the following data:

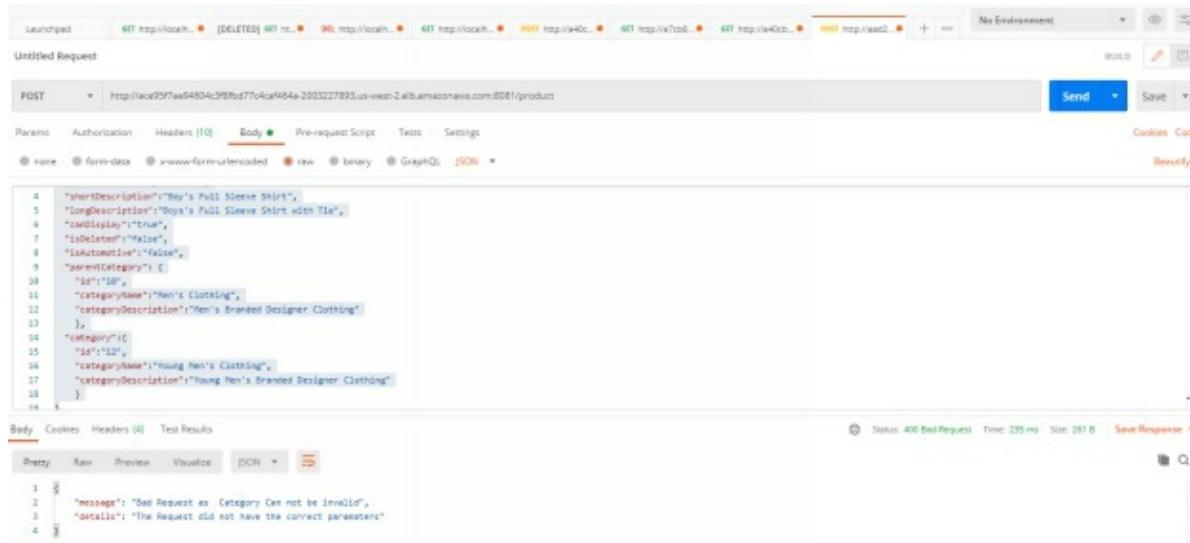
	id	category_description	category_name
▶	5	Food	Food
▶	6	Oranges	Oranges
▶	7	Electronics	Electronics
▶	8	Television	Television
▶	9	Computer	Laptop
▶	10	Clothing	Clothing
▶	11	Mens Shirt	MEns Shirt
▶	12	Young Men's Branded Designer Clothing	Young Men's Clothing
*	NULL	NULL	NULL

Let's change our product body json appropriately to reflect out AWS RDS MySQL Database Category ids.

```
{
  "productCode": "P1249493Z19",
  "productName": "Boy's Shirt",
  "shortDescription": "Boy's Full Sleeve Shirt",
  "longDescription": "Boys's Full Sleeve Shirt with Tie",
  "canDisplay": "true",
  "isDeleted": "false",
  "category": [
    {"id": 12, "name": "Young Men's Clothing", "description": "Young Men's Branded Designer Clothing"}]
```

```
"isAutomotive": "false",
"parentCategory": {
  "id": "10",
  "categoryName": "Men's Clothing",
  "categoryDescription": "Men's Branded Designer Clothing"
},
"category": {
  "id": "12",
  "categoryName": "Young Men's Clothing",
  "categoryDescription": "Young Men's Branded Designer Clothing"
}
}
```

However, the request still is failing



For further debugging, we need to get the logs that the Product Microservice is generating. While for a Production Application, you can expect an ELK cluster receiving your application logs to query, we will get our logs directly from Kubernetes using kubectl.

2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.spring.service.ProductServiceImpl	request port :18000
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.spring.service.ProductServiceImpl	url :http:// http://e-stdfbf212bed44cf59e7ae8d8e0e8d-10550190889.us-west-2.eltb.amazonaws.com:8082/category/
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.spring.service.ProductServiceImpl	request url modified : http://e-stdfbf212bed44cf59e7ae8d8e0e8d-10550190889.us-west-2.eltb.amazonaws.com:8082/category/
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.CustomHttpClientInterceptor	url :http:// http://e-stdfbf212bed44cf59e7ae8d8e0e8d-10550190889.us-west-2.eltb.amazonaws.com:8082/category/
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.CustomHttpClientInterceptor	HTTP Headers :[Accept:"application/json, application/*+json", Content-Length:"0"]
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.CustomHttpClientInterceptor	Product Category is invalid!
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.RestControllerAspect	ATT Method Calls invoke this general aspect method
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.RestControllerAspect	ATT Method Calls invoke this general aspect method
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.RestControllerAspect	ATT Method Calls invoke this general aspect method
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.RestControllerAspect	ATT Method Calls invoke this general aspect method
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.RestControllerAspect	Product Category is not null :com.rollingstone.spring.model.Category@4281d7a
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.RestControllerAspect	Product Category is not null :com.rollingstone.spring.model.Category@44dd7fc8
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.RestControllerAspect	Product Parent Category is not null :null #if #if
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.RestControllerAspect	request port :18000
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.RestControllerAspect	request host : http://e-stdfbf212bed44cf59e7ae8d8e0e8d-10550190889.us-west-2.eltb.amazonaws.com:8082/category/
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.RestControllerAspect	request uri :/category
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.RestControllerAspect	url : http://e-stdfbf212bed44cf59e7ae8d8e0e8d-10550190889.us-west-2.eltb.amazonaws.com:8082/category/
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.RestControllerAspect	request url modified : http://e-stdfbf212bed44cf59e7ae8d8e0e8d-10550190889.us-west-2.eltb.amazonaws.com:8082/category/
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.RestControllerAspect	HTTP Method :GET
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.RestControllerAspect	HTTP Headers :[Accept:"application/json, application/*+json", Content-Length:"0"]
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.RestControllerAspect	Product Category is invalid!
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.RestControllerAspect	ATT Method Calls invoke this general aspect method
2021-07-17 13:40:47.425	INFO	1	[nio-8080-exec-12] c.r.c.RestControllerAspect	Received Bad Request ExceptionRequest to Categories Can not be invalid

Check for the line in the logs as we are logging the full category service URL before making the call.

URI: <http://a7cb07fb723eb443c8f59e7ae580e8e8-1055039089.us-west-2.elb.amazonaws.com:8092/category/12>

The problem here is simple. If our Product Catalog application's property file has an inaccurate category service URL, it would not work. Let us match with your valid Category Service host

With

Kubectl get svc

```
ubuntu@ip-172-31-9-166:~/rollingstone-commerce-product-catalog-k8s-api/src/main/resources$ kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
category-service   LoadBalancer   10.100.217.80   a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com   8092:30994/TCP,8093:30338/TCP   54m
kubernetes       ClusterIP    10.100.0.1    <none>        443/TCP        72m
product-catalog-service   LoadBalancer   ace95f7aa94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com   8081:30233/TCP,8091:32105/TCP   18m
ubuntu@ip-172-31-9-166:~/rollingstone-commerce-product-catalog-k8s-api/src/main/resources$ |
```

Verify if the Category service is working in Postman. Your URL would be different.

<http://a40cb663d90aa45dda46233c2801dbf6-582124535.us-west-2.elb.amazonaws.com:8092/category>

Let's check our property file as we can see the category service URL may be invalid

The container property may be incorrect

Let's make the following change to make the replicas to 0

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-catalog-deployment
spec:
  replicas: 0
  selector:
    matchLabels:
      app: product-catalog-deployment
  template:
    metadata:
      labels:
```

```
app: product-catalog-deployment
spec:
  containers:
    - name: aws-ecr-spring-boot-category
      image: <act-id>.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-product
      ports:
        - containerPort: 8081
        - containerPort: 8091
      env:
        - name: spring.profiles.active
          value: aws
      imagePullPolicy: Always
```

Let's make the pod terminated with the following command

```
kubectl apply -f ./product-kubernetes-deployment.yaml
```

Let us verify that the Product catalog pod was terminated by Kubernetes.

```
ubuntu@ip-172-31-9-166:~/rollingstone-ecommerce-product-catalog-k8s-
api$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
category-deployment-7b675894f5-5v2nc	1/1	Running	0	65m

If the Product Catalog Service application-aws.properties file has the incorrect category service URL,

1. Get the accurate one from `kubectl get svc`,
2. Update the file `application-aws.properties`
3. Rebuild the application
4. Rebuild the Docker Image
5. Tag the Docker Image with AWS ECR URL
6. Login if needed
7. Push the Docker Image to the AWS ECR
8. Now let us deploy the application again

Change the following file with replicas back to 1

```
apiVersion: apps/v1
kind: Deployment
```

```

metadata:
  name: product-catalog-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product-catalog-deployment
  template:
    metadata:
      labels:
        app: product-catalog-deployment
    spec:
      containers:
        - name: aws-ecr-spring-boot-category
          image: <act-id>.dkr.ecr.us-west-2.amazonaws.com/aws-ecr-spring-boot-product
          ports:
            - containerPort: 8081
            - containerPort: 8091
          env:
            - name: spring.profiles.active
              value: aws
      imagePullPolicy: Always

```

With that done, deploy the product application with the following command

```
kubectl apply -f ./product-kubernetes-deployment.yaml
```

Let verify if the new Pod is created or not.

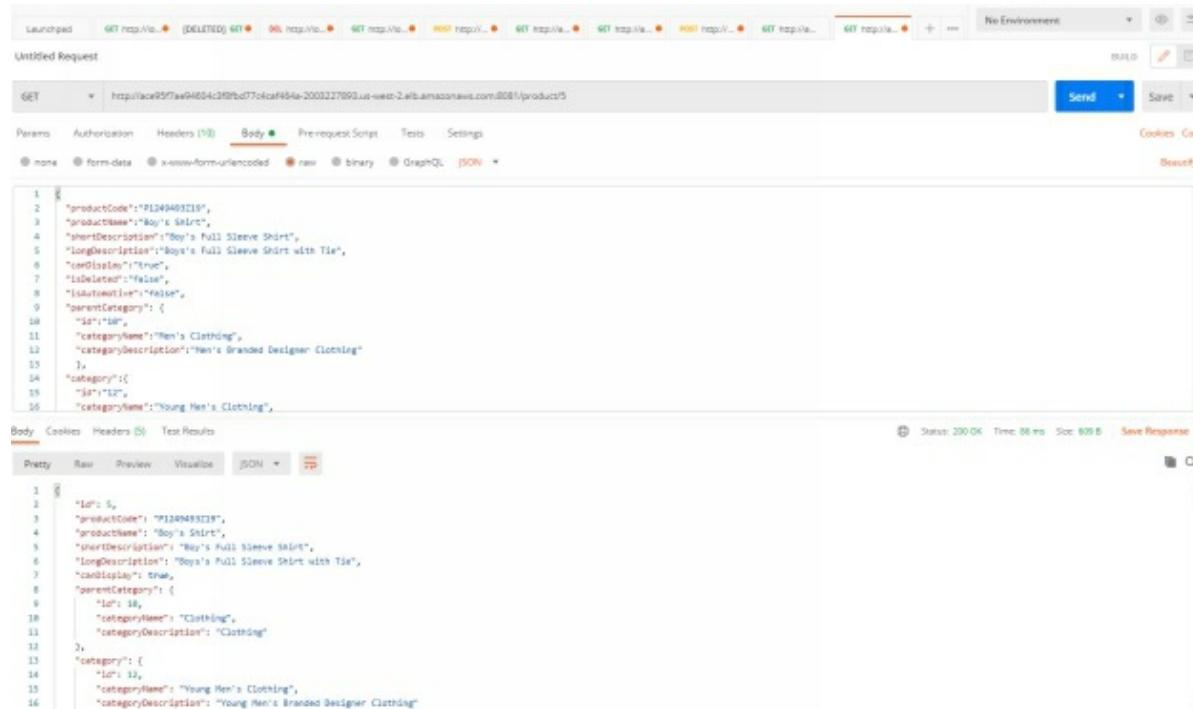
```
kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
category-deployment-7b675894f5-5v2nc	1/1	Running	0	66m
product-catalog-deployment-56bbcb9987-hdmns	1/1	Running	0	20s

Now, lets try the POST request to create a new Product. As shown now it succeeds

7.23 TEST ONE PRODUCT

Let's test one single product retrieval as shown in the following screen.



The screenshot shows a REST client interface with the following details:

- URL:** `http://ace957aa94604c209bd77e4ca454a-2003227892.us-west-2.elb.amazonaws.com:8081/product/5`
- Method:** GET
- Headers:** (None)
- Body:** (JSON)
A JSON object representing a product:

```
1 "productId": "P1149489319",
2 "productName": "Boy's Shirt",
3 "shortDescription": "Boy's Full Sleeve Shirt",
4 "longDescription": "Boy's Full Sleeve Shirt with Tie",
5 "available": "true",
6 "isDeleted": "false",
7 "isInactive": "false",
8 "parentCategory": {
9     "id": 18,
10    "categoryName": "Men's Clothing",
11    "categoryDescription": "Men's Branded Designer Clothing"
12 },
13 "category": {
14     "id": 18,
15     "categoryName": "Young Men's Clothing"
16 }
```
- Response Headers:** Status: 200 OK, Time: 86 ms, Size: 609 B
- Response Body:** (Pretty, Raw, Preview, Visualize, JSON)
The same JSON object as in the body, displayed in pretty-printed format.

7.24 TEST PUT

Let's test the PUT HTTP method to update our product. The full request body is below

```
{  
  "id":5,  
  "productCode":"P1249493Z19",  
  "productName":"Boy's Shirt",  
  "shortDescription":"Boy's Full Sleeve Shirt Updated",  
  "longDescription":"Boys's Full Sleeve Shirt with Tie",  
  "canDisplay":"true",  
  "isDeleted":"false",  
  "isAutomotive":"false",  
  "parentCategory": {  
    "id":"10",  
    "categoryName":"Men's Clothing",  
    "categoryDescription":"Men's Branded Designer Clothing"  
  },  
  "category":{  
    "id":"12",  
    "categoryName":"Young Men's Clothing",  
    "categoryDescription":"Young Men's Branded Designer Clothing"  
  }  
}
```

The screenshot shows the Postman application interface. At the top, there are several tabs for different HTTP methods: GET, HEAD, POST, PUT, PATCH, DELETE, and OPTIONS. Below the tabs, the URL is set to `http://aca95f7ae9-050fc2f9fb77c0caf95fa-2003227893.us-west-2.elb.amazonaws.com:8081/products/5`. The 'Body' tab is selected, showing the JSON payload for the PUT request. The payload is identical to the one provided in the code block above. The 'Send' button is visible at the top right of the main window.

The following image shows the Put was successful .

The screenshot shows the Postman application interface with the following details:

- URL:** `https://ec2-52-73-140-480.compute-1.amazonaws.com:8081/product/5`
- Method:** `PUT`
- Body (JSON):**

```
1  "id": 5,
2  "productCode": "P1234567890",
3  "productName": "Bey's Full Sleeve Shirt",
4  "shortDescription": "Bey's Full Sleeve Shirt to Test PUT Update",
5  "longDescription": "Bey's Full Sleeve Shirt with T1a",
6  "available": "True",
7  "isDeleted": "False",
8  "isInactive": "False",
9  "parentCategory": {
10    "id": 10,
11    "categoryName": "Men's Clothing",
12    "categoryDescription": "Men's Branded Designer Clothing"
13  },
14  "category": {
15    "id": 11,
16    "name": "T-Shirts"
17  }
```
- Status:** 200 OK
- Time:** 196 ms
- Size:** 194 B
- Save Response** button

7.25 VERIFY PUT

We can try the same single GET to check if the Put operation was successful.

The screenshot shows a Postman interface with the following details:

- URL:** `http://aca65f7ae94504c2f9fbcd770caf65fa-2003227893.us-west-2.elb.amazonaws.com:8001/product/5`
- Method:** GET
- Headers (10):** No Environment, BUILD, Send, Save, Cookies, Code, Beautify.
- Body:** Pre-request Script, Tests, Settings, none, form-data, x-www-form-urlencoded, raw, binary, GraphQL, JSON.
- JSON Body:**

```
1 {  
2     "id": 5,  
3     "productCode": "P1240493210",  
4     "productName": "Boy's Shirt",  
5     "shortDescription": "Boy's Full Sleeve Shirt to Test PUT Update",  
6     "longDescription": "Boy's Full Sleeve Shirt with Tie",  
7     "canDisplay": "true",  
8     "isDeleted": "false",  
9     "isAutomatic": "false",  
10    "parentCategory": {  
11        "id": 10,  
12        "categoryName": "Men's Clothing",  
13        "categoryDescription": "Men's Branded Designer Clothing"  
14    },  
15    "category": {  
16        "id": 12,  
17        "categoryName": "Young Men's Clothing",  
18        "categoryDescription": "Young Men's Branded Designer Clothing"  
19    }  
20}
```
- Response Status:** 200 OK, Time: 73 ms, Size: 626 B, Save Response.
- Response Body:**

```
1 {  
2     "id": 5,  
3     "productCode": "P1240493210",  
4     "productName": "Boy's Shirt",  
5     "shortDescription": "Boy's Full Sleeve Shirt to Test PUT Update",  
6     "longDescription": "Boy's Full Sleeve Shirt with Tie",  
7     "canDisplay": true,  
8     "parentCategory": {  
9         "id": 10,  
10        "categoryName": "Clothing",  
11        "categoryDescription": "Clothing"  
12    },  
13    "category": {  
14        "id": 12,  
15        "categoryName": "Young Men's Clothing",  
16        "categoryDescription": "Young Men's Branded Designer Clothing"  
17    }  
18}
```

7.26 TEST DELETE

Finally let us test the Delete HTTP Verb as shown below.

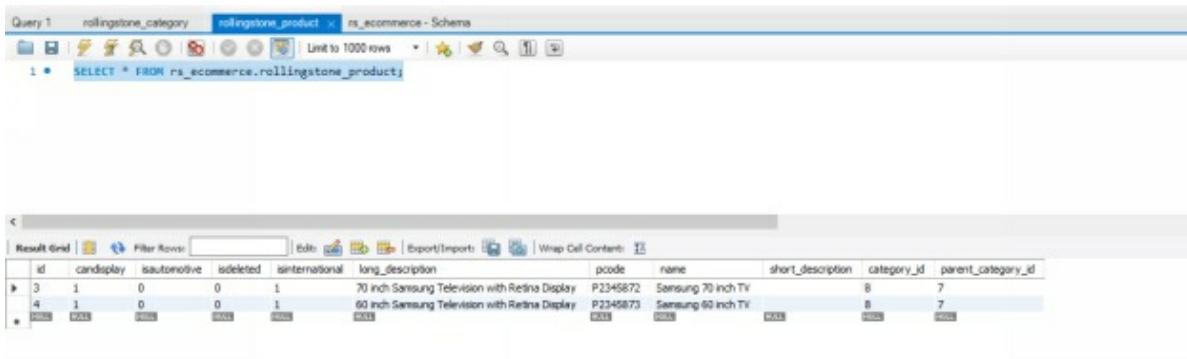
The screenshot shows the Postman application interface. At the top, there are several tabs labeled 'Launched' and various 'GET' and 'DELETE' requests. On the right, it says 'No Environment' and has a 'BUILD' button. Below the tabs, the title 'Untitled Request' is displayed. The main area shows a 'DELETE' request to the URL `http://ace95f9a94b04c3fbfd7704cafa484e-200227092.us-west-2.elb.amazonaws.com:8001/product/5`. The 'Body' tab is selected, containing the following JSON payload:

```
1 "id": 5,
2 "productCode": "P5234R8912IP",
3 "productName": "Boy's SHirt",
4 "shortDescription": "Boy's Full Sleeve Shirt to Test PUT Update",
5 "longDescription": "Boy's Full Sleeve Shirt with Tie",
6 "canDelete": "true",
7 "isDeleted": "false",
8 "isAvailable": "false",
9 "parentCategory": [
10     {
11         "id": "1B",
12         "categoryName": "Men's Clothing",
13         "categoryDescription": "Men's Branded Designer Clothing"
14     }
15 ],
16 "category": {
17     "id": "1B",
18 }
```

Below the body, there are tabs for 'Body', 'Cookies', 'Headers (5)', 'Test Results', and 'Save Response'. The status bar at the bottom indicates a 200 OK response with a time of 114 ms and a size of 194 B.

7.27 VERIFY DB

Verify the Database that the record was indeed deleted.

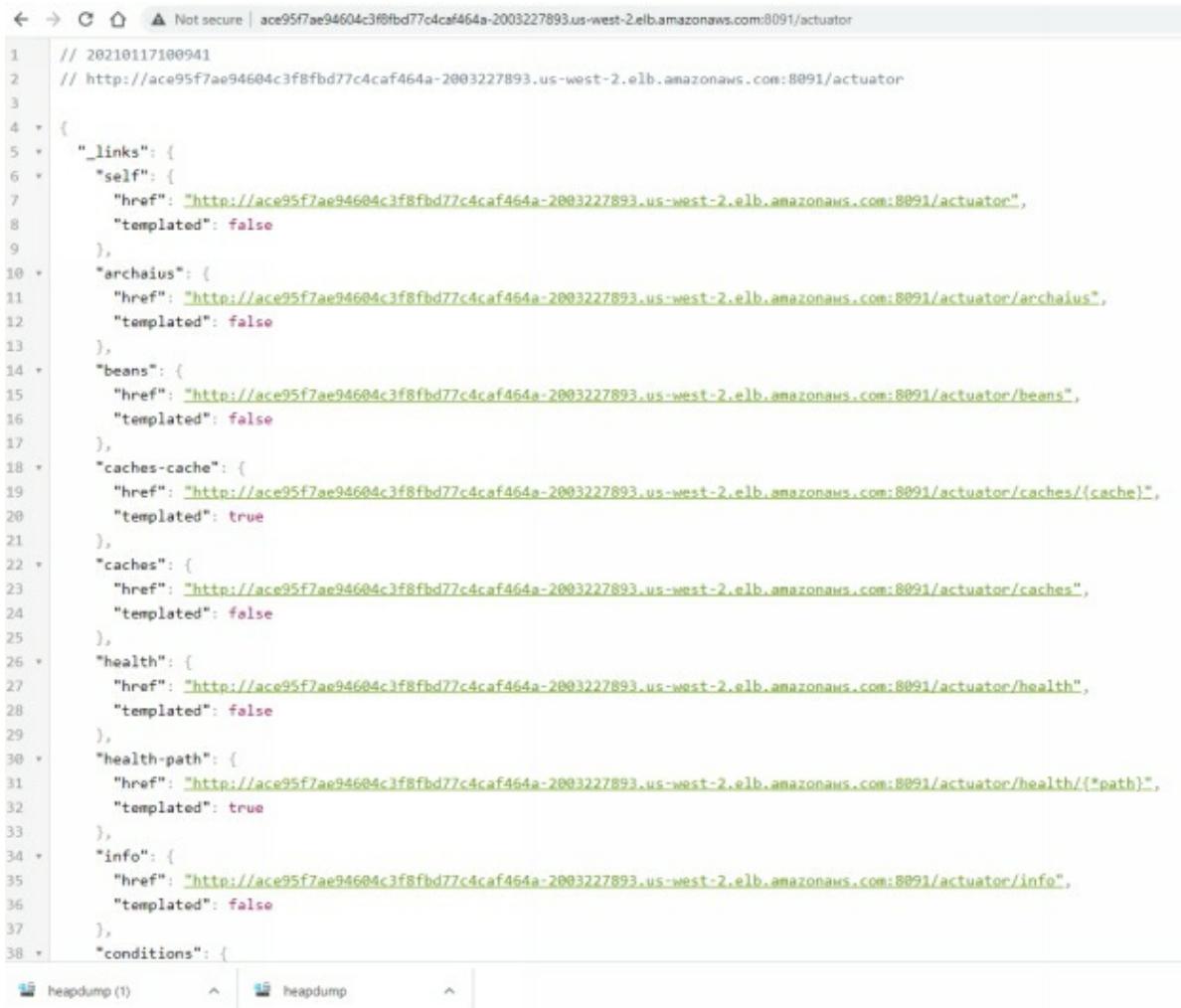


The screenshot shows a MySQL Workbench interface with two tabs: 'rollingstone_category' and 'rollingstone_product'. The 'rollingstone_product' tab is active, displaying a result grid for the query 'SELECT * FROM rs_ecommerce.rollingstone_product;'. The result grid contains the following data:

	id	cardsdisplay	isautomotive	isdeleted	sinternational	long_description	pcode	name	short_description	category_id	parent_category_id
▶	3	1	0	0	1	70 inch Samsung Television with Retina Display	P2345872	Samsung 70 inch TV		8	7
▶	4	1	0	0	1	60 inch Samsung Television with Retina Display	P2345873	Samsung 60 inch TV		8	7

7.28 TEST PRODUCT ACTUATOR

Following image shows how the Product Catalog Actuator exposed on the 8091 port is working.



The screenshot shows a browser window with the URL <http://ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8091/actuator>. The page displays a JSON object representing the actuator endpoints. The JSON structure includes fields like '_links', 'archaius', 'beans', 'caches-cache', 'caches', 'health', 'health-path', 'info', and 'conditions'. Each field contains a href attribute pointing to specific URLs, such as <http://ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8091/actuator>, <http://ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8091/actuator/archaius>, and <http://ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8091/actuator/beans>.

```
// 20210117100941
// http://ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8091/actuator

{
  "_links": {
    "self": {
      "href": "http://ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8091/actuator",
      "templated": false
    },
    "archaius": {
      "href": "http://ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8091/actuator/archaius",
      "templated": false
    },
    "beans": {
      "href": "http://ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8091/actuator/beans",
      "templated": false
    },
    "caches-cache": {
      "href": "http://ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8091/actuator/caches/{cache}",
      "templated": true
    },
    "caches": {
      "href": "http://ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8091/actuator/caches",
      "templated": false
    },
    "health": {
      "href": "http://ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8091/actuator/health",
      "templated": false
    },
    "health-path": {
      "href": "http://ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8091/actuator/health/{path}",
      "templated": true
    },
    "info": {
      "href": "http://ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8091/actuator/info",
      "templated": false
    },
    "conditions": {
      "href": "http://ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8091/actuator/conditions"
    }
  }
}
```

7.29 TEST ACTUATOR HEALTH

If we try the /actuator/health endpoint, Spring Boot would do a lot of work, check connectivity to the Database, if the database driver is in the class path apart from reporting other details. There is a small property, that we need to have enabled to get the detailed status though. Otherwise, it will just show “UP”. Following is that property.

```
endpoint:  
  health:  
    show-details: "always"
```

← → C ⌂ ▲ Not secure | ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8091/actuator/health

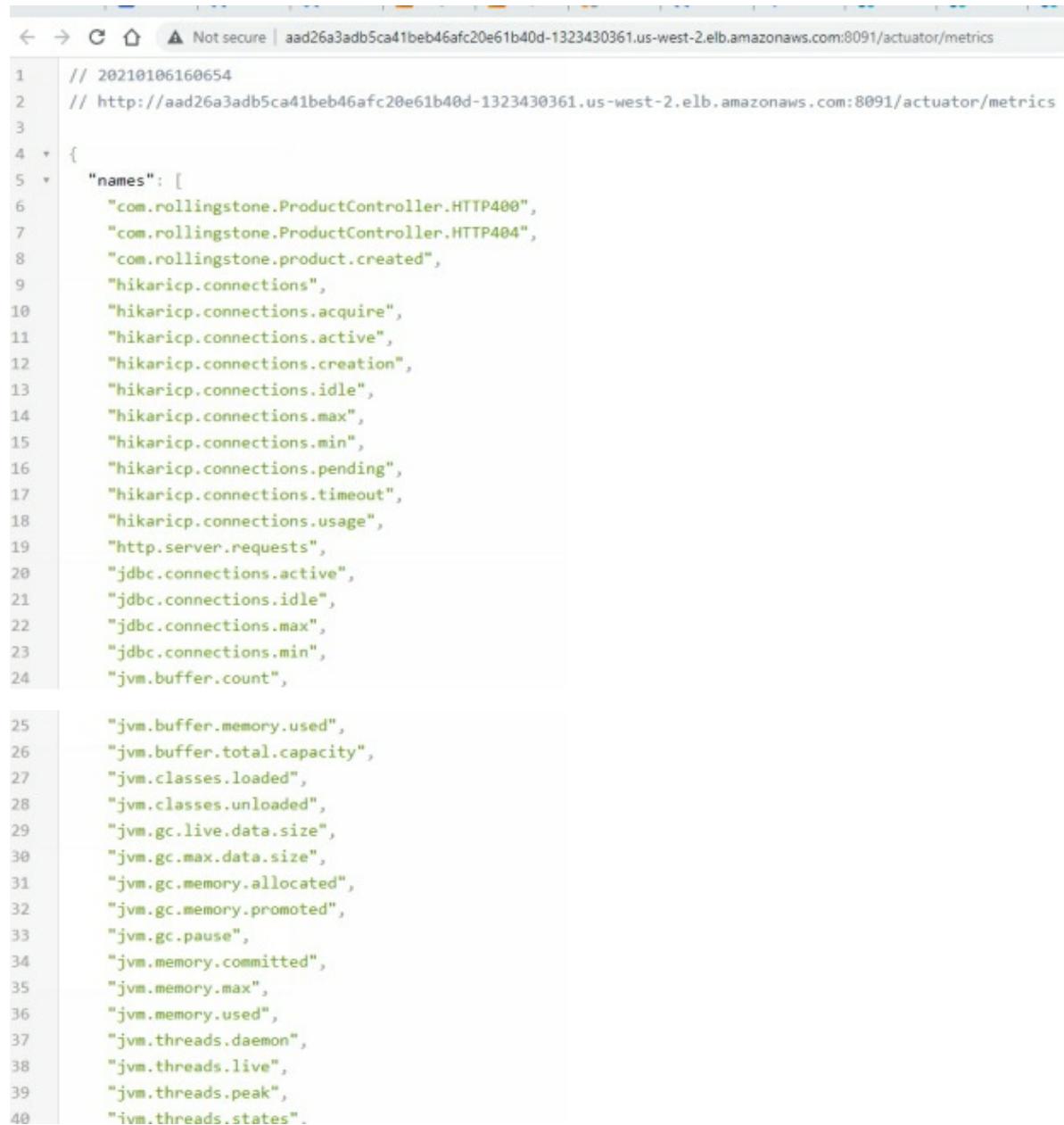
```
1 // 20210117101029
2 // http://ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8091/actuator/health
3
4 +
5     "status": "UP",
6     "components": {
7         "db": {
8             "status": "UP",
9             "details": {
10                 "database": "MySQL",
11                 "validationQuery": "isValid()"
12             }
13         },
14         "discoveryComposite": {
15             "description": "Discovery Client not initialized",
16             "status": "UNKNOWN",
17             "components": {
18                 "discoveryClient": {
19                     "description": "Discovery Client not initialized",
20                     "status": "UNKNOWN"
21                 }
22             }
23         },
24         "diskSpace": {
25             "status": "UP",
26             "details": {
27                 "total": 85886742528,
28                 "free": 83136921600,
29                 "threshold": 10485760,
30                 "exists": true
31             }
32         },
33         "hystrix": {
34             "status": "UP"
35         },
36         "livenessState": {
37             "status": "UP"
38         },
39     }
```

← → C ⌂ ▲ Not secure | ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8091/actuator/metrics/com.rollingstone.product.created

```
1 // 2021011710130
2 // http://ace95f7ae94604c3f8fb77c4caf464a-2003227893.us-west-2.elb.amazonaws.com:8091/actuator/metrics/com.rollingstone.product.created
3
4 +
5     "name": "com.rollingstone.product.created",
6     "description": "Number of Products Created",
7     "baseUnit": null,
8     "measurements": [
9         {
10             "statistic": "COUNT",
11             "value": 1.0
12         }
13     ],
14     "availableTags": [
15         {
16             "tag": "environment",
17             "values": [
18                 "production"
19             ]
20         }
21     ]
22 }
```

7.30 TEST THE PRODUCT SERVICE ACTUATOR METRICS

Actuator generates a lot of general and our custom metrics. We can find that below. The first three are our custom metrics.



The screenshot shows a browser window with the URL `aad26a3adb5ca41beb46afc20e61b40d-1323430361.us-west-2.elb.amazonaws.com:8091/actuator/metrics`. The page displays a JSON response with line numbers on the left. The response lists various metric names, including custom metrics like `com.rollingstone.ProductController.HTTP400`, `com.rollingstone.ProductController.HTTP404`, and `com.rollingstone.product.created`, along with standard JVM and database metrics.

```
1 // 20210106160654
2 // http://aad26a3adb5ca41beb46afc20e61b40d-1323430361.us-west-2.elb.amazonaws.com:8091/actuator/metrics
3
4 {
5   "names": [
6     "com.rollingstone.ProductController.HTTP400",
7     "com.rollingstone.ProductController.HTTP404",
8     "com.rollingstone.product.created",
9     "hikaricp.connections",
10    "hikaricp.connections.acquire",
11    "hikaricp.connections.active",
12    "hikaricp.connections.creation",
13    "hikaricp.connections.idle",
14    "hikaricp.connections.max",
15    "hikaricp.connections.min",
16    "hikaricp.connections.pending",
17    "hikaricp.connections.timeout",
18    "hikaricp.connections.usage",
19    "http.server.requests",
20    "jdbc.connections.active",
21    "jdbc.connections.idle",
22    "jdbc.connections.max",
23    "jdbc.connections.min",
24    "jvm.buffer.count",
25
26    "jvm.buffer.memory.used",
27    "jvm.buffer.total.capacity",
28    "jvm.classes.loaded",
29    "jvm.classes.unloaded",
30    "jvm.gc.live.data.size",
31    "jvm.gc.max.data.size",
32    "jvm.gc.memory.allocated",
33    "jvm.gc.memory.promoted",
34    "jvm.gc.pause",
35    "jvm.memory.committed",
36    "jvm.memory.max",
37    "jvm.memory.used",
38    "jvm.threads.daemon",
39    "jvm.threads.live",
40    "jvm.threads.peak",
41    "jvm.threads.states"
42  ]
43}
```

7.31 TEST ONE CUSTOM METRIC

```
← → C ⌂ Not secure | aad26a3adb5ca41beb46afc20e61b40d-1323430361.us-west-2.elb.amazonaws.com:8091/actuator/metrics/com.rollingstone.product.created
1 // 20210106160742
2 // http://aad26a3adb5ca41beb46afc20e61b40d-1323430361.us-west-2.elb.amazonaws.com:8091/actuator/metrics/com.rollingstone.product.created
3
4 +
5     "name": "com.rollingstone.product.created",
6     "description": "Number of Products Created",
7     "baseUnit": null,
8     "measurements": [
9         {
10             "statistic": "COUNT",
11             "value": 1.0
12         }
13     ],
14     "availableTags": [
15         {
16             "tag": "environment",
17             "values": [
18                 "production"
19             ]
20         }
21     ]
22 }
```

7.32 TERMINATING AWS SERVICES

Understanding how to control AWS Cloud (or Azure / Google/ IBM / Oracle) cost is a high demand skill. All Cloud share the basic structure of their services and differ in quality and detailed features.

Please do the following to delete / terminate AWS services after you are done

1. AWS RDS → Delete the Database using the AWS Management Console
2. AWS EKS → Run the command from your Bastion Host → eksctl delete cluster EKS-Cluster-SpringBoot
3. EC2 Instances → Terminate them or Stop them if you would like to preserve the Bastion Host. We can also make an Amazon Machine Image (AMI) and then delete the EC2 instance
4. Check if any AWS Load Balancers still exists after the completed the three above
5. Check AWS Billing from the AWS Management Console to see the Cost Projection

7.33 WHERE TO GO FROM HERE

Proper learning takes place when we learn something and are absolutely ready to apply that for a paying customer beyond the POCs in our laptops. In this book, we focused on a few Spring Boot specific toolset and features to make ourselves customer ready. Following the same we deployed our services to a real AWS EKS Kubernetes cluster rather than Minikube on our laptops. As I said early in the book, I want to make this book, the next ones in the series to a conversations tool to raise the market value in terms real customer in demand software engineering/architecture skills. While I could cover, a few of them here in this book, there is certainly more to be done. In the series that will follow, I will elaborate on

- How to use Spring Cloud Netflix OSS with Kubernetes and when to use that
 - Circuit Breaker
 - Remote Configuration
 - Service Discovery
 - Client-Side Load Balancing
- High Availability how to productionize the AWS EKS Cluster
- Scalability and Kubernetes Horizontal Pod Scheduler or HPA using
 - CPU Resources and Limits
 - Memory Resources and Limits
- Security
 - OAuth2 for sensitive APIs
 - OAuth2 Implementation using Spring Security
- API Gateways
 - For North South Traffic
- Disaster Recovery
 - Active/Active Provisioning

- Recovery Time Objective (RTO)
- Recovery Point Objective (RPO)
- Logging
 - Using an ELK Cluster
- Monitoring
 - Using Prometheus and Grafana
- AWS CI/CD

Reference

Lockridge, D. (c. 2017) Container ship at the Port of Long Beach. Photo: Jim Park. Trucking History, United States. Retrieved from
<https://www.truckinginfo.com/159847/the-steel-box-that-changed-global-logistics>

Docker, D. (c. 2021) Container Execution. Photo: Docker. Docker Hub, United States. Retrieved from <https://docs.docker.com/get-started/overview/>

Kubernetes. (2021). Kubernetes Documentation. Retrieved from
<https://kubernetes.io/docs/home/>

AWS. (2021). Amazon Web Service. Retrieved from <https://aws.amazon.com>

Oracle Corporation. (2021). JDK Downloads. Retrieved from
<https://www.oracle.com/java/technologies/javase-downloads.html>

JetBrains. (2021). IntelliJ Idea Community Edition Downloads Retrieved from <https://www.jetbrains.com/idea/download/#section=mac>

Eclipse Foundation. (2021). Download Eclipse Technology that is right for you Retrieved from <https://www.eclipse.org/downloads/>