1. Java Graal

import org.graalvm.compiler.api.directives.GraalDirectives;

import org.graalvm.compiler.api.replacements.Fold;

public class MyClass {

@Fold

@CompileStatic

public static int add(int a, int b) {

return a + b;

}

public static void main(String[] args) {

int result = add(1, 2);

GraalDirectives.instrumentationBegin();

System.out.println(result);

GraalDirectives.instrumentationEnd();

}

}

This code defines a simple Java class with a main method that calls the add method and prints the result. The add method is annotated with the @CompileStatic annotation, which tells the GraalVM compiler to statically compile the method. The @Fold annotation tells the compiler to perform constant folding, which can further improve the performance of the method.

To compile this code with GraalVM, you can use the native-image command as shown above. This will create a native executable that contains the compiled code for the add method.

1. Testcontainers

public class MyIntegrationTest {

@Rule

public PostgreSQLContainer postgres = new PostgreSQLContainer();

@Test

public void testWithDatabase() {

// Connect to the database using the host, port, and credentials

// provided by the PostgreSQLContainer

Connection conn = DriverManager.getConnection(

postgres.getJdbcUrl(),

postgres.getUsername(),

postgres.getPassword());

// Run your tests against the database

// ...

}

}

This code creates a PostgreSQLContainer object, which is annotated with the @Rule annotation. This tells JUnit to create and start the container before running the test method, and to stop and remove the container after the test method has completed. The getJdbcUrl, getUsername, and getPassword methods can be used to obtain the connection details for the PostgreSQL instance.he add method.

3.Spring boot application

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication

@RestController

public class HelloWorldApplication {

public static void main(String[] args) {

SpringApplication.run(HelloWorldApplication.class, args);

}

@GetMapping("/hello")

public String sayHello() {

return "Hello, World!";

}

}

Output -

HTTP/1.1 200 OK

Content-Type: text/plain;charset=UTF-8

Content-Length: 12

Hello, World!

This code defines a simple Spring Boot application with a single @RestController that has a single @GetMapping for the /hello endpoint. When the application is run and you send a GET request to /hello, the controller will return the string "Hello, World!".

To run this application, you will need to have the Spring Boot framework on your classpath. You can then use the SpringApplication.run method to start the application.

4.Connecting springboot application to the database

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

import org.testcontainers.containers.PostgreSQLContainer;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

@SpringBootApplication

public class HelloWorldApplication {

@Autowired

private PostgreSQLContainer postgres;

public static void main(String[] args) {

SpringApplication.run(HelloWorldApplication.class, args);

}

@Bean

public CommandLineRunner run() throws SQLException {

return args -> {

// Connect to the database using the host, port, and credentials

// provided by the PostgreSQLContainer

Connection conn = DriverManager.getConnection(

postgres.getJdbcUrl(),

postgres.getUsername(),

postgres.getPassword());

// Use the connection to execute a query

// ...

};

}

}

This code defines a Spring Boot application with a PostgreSQLContainer bean that is started as part of the application's context. The container is then injected into the application using the @Autowired annotation.

The code also defines a CommandLineRunner bean that is run when the application starts. This bean uses the PostgreSQLContainer to obtain a JDBC connection to the database and can use the connection to execute queries or perform other database operations.

1. Terraform script to deploy spring boot application

1.Package the application as a standalone JAR file.

2.Create an AWS Lambda function using the aws\_lambda\_function resource.

3.Set the function's runtime to java8.

4.Set the function's handler to the fully-qualified name of the class that contains your application's main method.

5.Set the function's code to the location of the JAR file that you created in step 1.

provider "aws" {

region = "us-east-1"

}

resource "aws\_lambda\_function" "lambda" {

function\_name = "hello-world"

runtime = "java8"

handler = "com.example.HelloWorldApplication::main"

code = "./hello-world.jar"

}

This script creates an AWS Lambda function with the name "hello-world" and sets the runtime to java8. The function's handler is set to the fully-qualified name of the HelloWorldApplication class and its code is set to the location of the hello-world.jar file.