Testing a Spring Boot REST API with Testcontainers

- Prerequisites
- Create the Spring Boot project
  - Set up the project
  - Create the JPA entity
  - Create the Spring Data JPA repository
  - Add the schema creation script
  - Create the REST API endpoint
- Write tests with Testcontainers
  - Write the test
- Run tests and next steps
  - Run the tests
  - Summary
  - Further reading

In this guide, you will learn how to:

- Create a Spring Boot application with a REST API endpoint
- Use Spring Data JPA with PostgreSQL to store and retrieve data
- Test the REST API using Testcontainers and REST Assured

Prerequisites

- Java 17+
- Maven or Gradle
- A Docker environment supported by Testcontainers

> [!NOTE] If you're new to Testcontainers, visit the Testcontainers overview to learn more about Testcontainers and the benefits of using it.

Create the Spring Boot project

Set up the project

Create a Spring Boot project from Spring Initializr by selecting the **Spring Web**, **Spring Data JPA**, **PostgreSQL Driver**, and **Testcontainers** starters.

Alternatively, clone the guide repository.

The key dependencies in `pom.xml` are:

```
<properties>
    <java.version>17</java.version>
    <testcontainers.version>2.0.4</testcontainers.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
```

```
    </dependency>
    <dependency>
        <groupId>org.testcontainers</groupId>
        <artifactId>testcontainers-junit-jupiter</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.testcontainers</groupId>
        <artifactId>testcontainers-postgresql</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>io.rest-assured</groupId>
        <artifactId>rest-assured</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Using the Testcontainers BOM (Bill of Materials) is recommended so that you don't have to repeat the version for every Testcontainers module dependency.

Create the JPA entity

Create `Customer.java`:

```
package com.testcontainers.demo;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name = "customers")
class Customer {

  @Id
```

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

@Column(nullable = false)
private String name;

@Column(nullable = false, unique = true)
private String email;

public Customer() {}

public Customer(Long id, String name, String email) {
  this.id = id;
  this.name = name;
  this.email = email;
}

public Long getId() {
  return id;
}

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

public String getName() {
  return name;
}

public void setName(String name) {
  this.name = name;
}

public String getEmail() {
  return email;
}

public void setEmail(String email) {
  this.email = email;
```

```
  }
}
```

Create the Spring Data JPA repository

```
package com.testcontainers.demo;

import org.springframework.data.jpa.repository.JpaRepository;

interface CustomerRepository extends JpaRepository<Customer, Long> {}
```

Add the schema creation script

Create `src/main/resources/schema.sql`:

```
create table if not exists customers (
    id bigserial not null,
    name varchar not null,
    email varchar not null,
    primary key (id),
    UNIQUE (email)
);
```

Enable schema initialization in `src/main/resources/application.properties`:

```
spring.sql.init.mode=always
```

Create the REST API endpoint

Create `CustomerController.java`:

```
package com.testcontainers.demo;

import java.util.List;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController
class CustomerController {

  private final CustomerRepository repo;

  CustomerController(CustomerRepository repo) {
    this.repo = repo;
  }

  @GetMapping("/api/customers")
  List<Customer> getAll() {
    return repo.findAll();
  }
}
```

Write tests with Testcontainers

To test the REST API, you need a running Postgres database and a started Spring context. Testcontainers spins up Postgres in a Docker container and `@DynamicPropertySource` connects it to Spring.

Write the test

Create `CustomerControllerTest.java`:

```
package com.testcontainers.demo;

import static io.restassured.RestAssured.given;
import static org.hamcrest.Matchers.hasSize;

import io.restassured.RestAssured;
import io.restassured.http.ContentType;
import java.util.List;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.web.server.LocalServerPort;
import org.springframework.test.context.DynamicPropertyRegistry;
import org.springframework.test.context.DynamicPropertySource;
import org.testcontainers.postgresql.PostgreSQLContainer;

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
class CustomerControllerTest {

  @LocalServerPort
  private Integer port;

  static PostgreSQLContainer postgres = new PostgreSQLContainer(
    "postgres:16-alpine"
  );

  @BeforeAll
  static void beforeAll() {
    postgres.start();
  }

  @AfterAll
  static void afterAll() {
    postgres.stop();
  }

  @DynamicPropertySource
  static void configureProperties(DynamicPropertyRegistry registry) {
    registry.add("spring.datasource.url", postgres::getJdbcUrl);
    registry.add("spring.datasource.username", postgres::getUsername);
    registry.add("spring.datasource.password", postgres::getPassword);
  }

  @Autowired
  CustomerRepository customerRepository;

  @BeforeEach
  void setUp() {
    RestAssured.baseURI = "http://localhost:" + port;
    customerRepository.deleteAll();
```

```
  }

  @Test
  void shouldGetAllCustomers() {
    List<Customer> customers = List.of(
      new Customer(null, "John", "john@mail.com"),
      new Customer(null, "Dennis", "dennis@mail.com")
    );
    customerRepository.saveAll(customers);

    given()
      .contentType(ContentType.JSON)
      .when()
      .get("/api/customers")
      .then()
      .statusCode(200)
      .body(".", hasSize(2));
  }
}
```

Here's what the test does:

- `@SpringBootTest` starts the full application on a random port.
- A `PostgreSQLContainer` starts in `@BeforeAll` and stops in `@AfterAll`.
- `@DynamicPropertySource` registers the container's JDBC URL, username, and password with Spring so that the datasource connects to the test container.
- `@BeforeEach` deletes all customer rows before each test to prevent test pollution.
- `shouldGetAllCustomers()` inserts two customers, calls `GET /api/customers`, and verifies the response contains 2 records.

Run tests and next steps

Run the tests

```
$ ./mvnw test
```

Or with Gradle:

```
$ ./gradlew test
```

You should see the Postgres Docker container start and all tests pass. After the tests finish, the container stops and is removed automatically.

Summary

The Testcontainers library helps you write integration tests by using the same type of database (Postgres) that you use in production, instead of mocks or in-memory databases. Because you test against real services, you're free to refactor code and still verify that the application works as expected.

To learn more about Testcontainers, visit the Testcontainers overview.

Further reading

- Testcontainers JUnit 5 quickstart
- Testcontainers Postgres module
- Testcontainers JDBC support

□□□ #2
□ □□□ □□□ 2026-10-31 18:48:24 UTC
□ □□□ □□□ 2026-10-31 18:48:25 UTC