INTRODUCTION

GitHub: https://github.com/rwhite35/spring-testcontainer-postgres.git

Greetings API Service is a companion project for Spring Boot Demo App that provides containerized microservices including mock services for test automation. The solution is fully self-contained making local deployment simple and lite-weight. There are required platform and system dependencies, but all third-party frameworks, packages and libraries are included and pulled in automatically from the base project.

PREREQUISITES

Greetings API Service depends on Java OpenJDK v17 or newer for code and compiler. Package management and build functions are provided by Sping Boot 3.3.3 and Gradle v8.9. Web services and a lite-weight web server depends on Apache' Tomcat v10 for localhost service. Spring Boot Testcontainers provides service management inside containers to integrated JUnit hooks and resources for actual or mocked (via Microcks) integrated automated testing.

With the exception of *OpenJDK*, all required packages are configured and included for a "typical" implementation. For service Bean declarations, see:
./test/java/com/demo/greetings/TestcontainersConfiguration.java

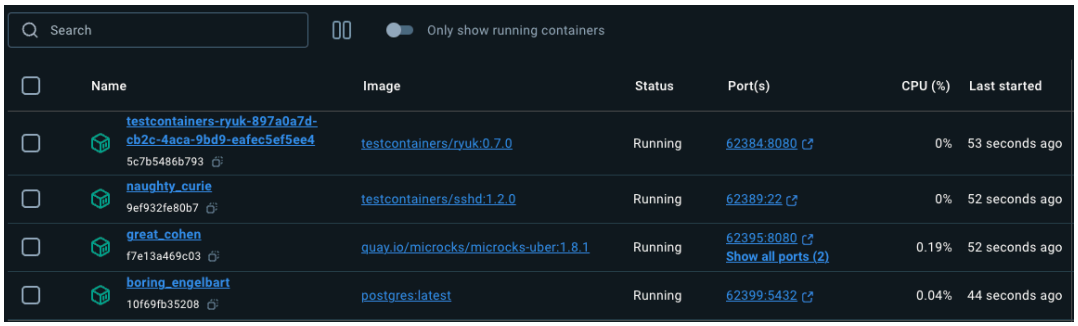
Docker

Either Docker Desktop or Docker CLI is required for daemon service, image building and running a container instance. Docker Desktop provides a GUI point and click interface and is free to install and use when you create a free *Docker Desktop* account using personal email. Both tools works for this project but this doc assumes *Docker Desktop* was installed.

Confirm *docker* command-line is installed and the daemon running before testing the project.
cmd: `docker version` // version: 27.1.1

And while we're at it, pull the latest PostgreSQL(16-alpine) *image* for future reference
cmd: `docker pull postgres:latest`
out: latest: Pulling from library/postgres
f86b8719f2ec: Pull complete...
Digest: sha256:4928..550d
Status: Downloaded newer image for *postgres:latest*

The following screen shows the four created container instances when the projects is started from TestGreetingsApplication.main( ) class.



 ronwhite562@gmail.com

Testcontainer Desktop

Similar to Docker Desktop, Testcontainer Desktop is also free and allows managing Testcontainer containers through a GUI interface. There are other whistles and bell including performance and task monitoring and insights. That's useful when try to debug slow queries or dead links. This project doesn't require Testcontainers Desktop to run locally.

Spring Boot and OpenJDK

The installation and setup for Java OpenJDK is out of scope for this document. In part due to there being so many distros and flavors. However the Spring Boot/OpenJDK System Requirement document does provide additional links and resources for installing OpenJDK for Spring Boot.

PROJECT SETUP

The following step-by-step quick start should allow *Greetings API Service* project to build and run without any additional setup.

Clone Project to Local Workspace

This is personal preference, however, the project will need to create files and directories inside the workstation HOME directory in order to access system resources from a privileges user. For that reason, I generally clone projects under the MacOS /Users/<username>/Sites directory. This location is has a *user:group* ownership of *<username> : wheel* and can generally install any dependencies without `sudo` or `admin` access.

cmd: `git clone https://github.com/rwhite35/spring-testcontainer-postgres.git testcontainers`

TIP: While a directory name argument (*ie testcontainers*) is not required, passing a shorter name does cut down on command-line input.

Create a Work-branch and Open in an IDE

Change directories and list the branch to confirm `origin/master` is the default branch cloned. Assuming that's the case (likely), create a new work-branch for any IDE setting or local change that might be required or desired.

This project was developed on MacOS using VSCode v1.92.2 as the IDE of choice. VSCode includes several productivity extensions like *Prettier* for automatic code formatting. In some cases code may change due to some incompatible IDE preference. Review the IDE's *Problems* tab or console output to confirm no change has broken code.

One Known Issue: v1.0.1 TestcontainersConfiguration.java throws an unassigned closable value warning at line 38. MicrocksContainer( ) is a third-party library for mock data/ services testing. The issue is non-fatal and being ignored for now. There isn't a suitable drop-in replacement for the class being instantiated.

Review Package Dependencies (build.gradle)

As mentioned earlier all dependencies have been defined and are included in the *build.gradle* file. For quick reference, the following are the dependency as of version 1.0.1 release.

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-validation'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.flywaydb:flyway-core'
    implementation 'org.flywaydb:flyway-database-postgresql'
    runtimeOnly 'org.postgresql:postgresql'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'org.springframework.boot:spring-boot-testcontainers'
    testImplementation 'org.testcontainers:postgresql'
    testImplementation 'org.awaitility:awaitility:4.2.2'
    testImplementation 'io.rest-assured:rest-assured:5.5.0'
    testImplementation 'org.testcontainers:junit-jupiter'
    testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
}
```

The following packages are worth noting since they provide core functionality.
Namely Spring's *Testcontainers* implementation is required, as it the correct scopes.

1. **starter-test** is the source for 'test' context and database integration.
   testImplementation 'org.springframework.boot:spring-boot-starter-test'

2. **testcontainers** is for container management including connection and ssh access.
   testImplementation 'org.springframework.boot:spring-boot-testcontainers'

3. **testcontainers:postgresql** is the containerization for PostgresSQL including port mapping.
   testImplementation 'org.testcontainers:postgresql'

NOTE 1: junit-jupiter and junit-platform-launcher need to be last in the list.

Review TestcontainerConfiguration File

Declares and configures request layer containers as service connections wrapped in Spring framework ApplicationContext annotated *Beans*. Beans represent the unit of work thats responsible for some functionality - in a given context. In general, Beans can represent service layer objects, data access objects (DAOs), presentation objects, infrastructure objects such as Hibernate SessionFactories, JMS Queues, and more.

A request to http://localhost:8080 is passed through the Spring DispatchServlet which routes the request to a Tomcat destination running on a Testcontainer. A Bean is bound to that location which in the context of a web request - is configured to handle the request.

TestcontainerConfiguration declares two service connections

```
@TestConfiguration(proxyBeanMethods = false)
public class TestcontainersConfiguration {

    @Bean
    @ServiceConnection
    PostgreSQLContainer<?> postgresContainer() {
        return new PostgreSQLContainer<>(parse("postgres:latest"));
    }
...
}
```

1. *PostgreSQLContainer*<?> postgresContainer( ) {..}
   - return an instance of Docker image *postgres:latest*
   - requires Testcontainer DockerImageName to parse the full image name
   - requires Testcontainer PostgreSQLContainer object for container configuration

   See *Database_Setup.pdf* doc for setup and troubleshooting tips for that resource.

2. *MicrocksContainer* microcksContainer(DynamicPropertyRegistry registry) {..}
   - return an instance of Microcks-Uber mock service for automated testing.
   - uses late static binging to allow the request connection to be passed as a registry argument at runtime. That provides a hot swappable capability.
   - requires Testcontainer MicrocksContainer object for configuration.

   NOTE: Microcks will require a free personal account. The request is actually routed through their cloud infrastructure which has the benefit of mimicking any other cloud provider. Setup the Microcks service before proceeding.

3. LocalstackContainer is a localhost container instanced with AWS IAM credentials implementation. This provides a link between an AWS cloud service and the localhost development environment. It's stubbed in for reference but not implemented in version 1.0.1.

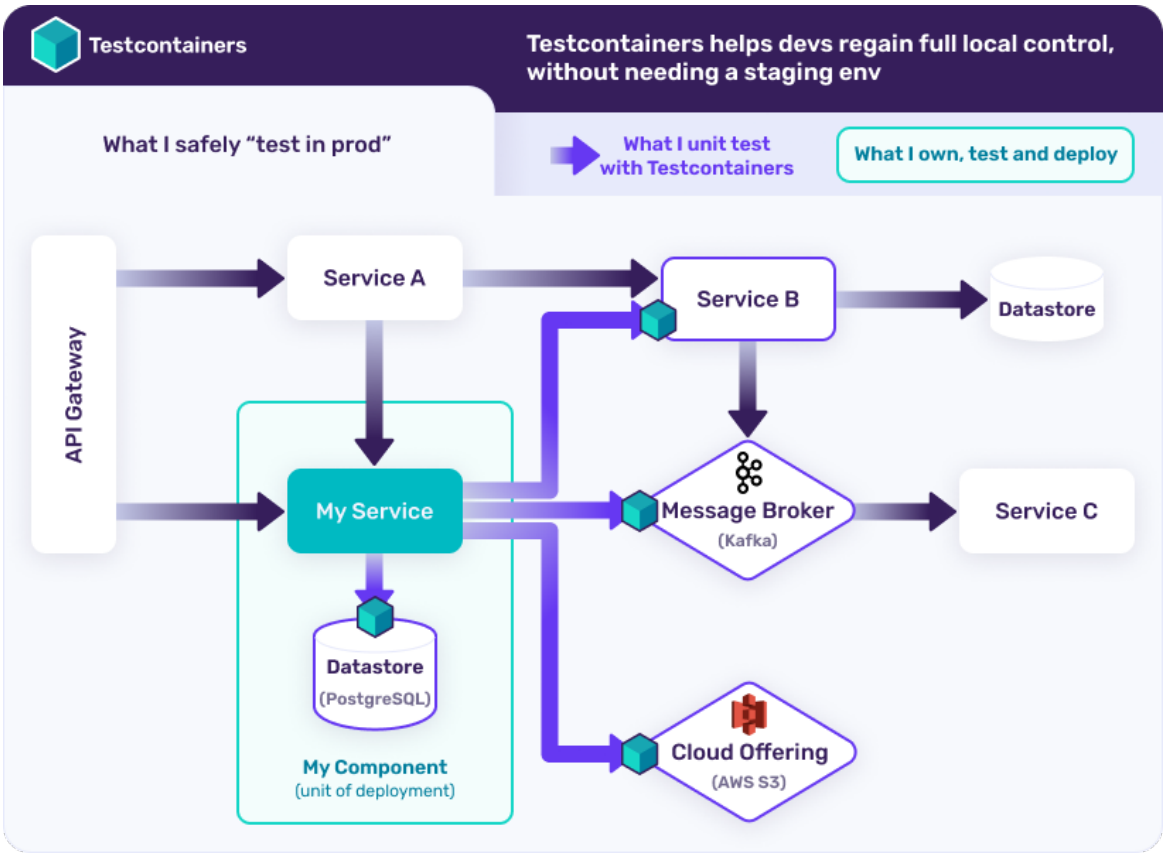
4. KafkaContainer is also stubbed for reference. Kafka can be implemented to handle message brokering and different MIME types including image, video, audio, etc.

Review TestcontainersApplication File

This is the main( ) method for running the project as a Testcontainer local service. Testcontainer is the parent process and launches *Greetings API Service* Application.main( ) as a child process. This has the effect of passing all request through Testcontainer which in-turn routes the request to its service layer containers, be that database or service endpoint.

The following infographic provides the general process flow where the *green cubes* represent injected Testcontainer containers in to a cloud based architecture.

See Testcontainers Getting Started page for more details



TIP: *Greetings API Service* doesn't have to run under the Testcontainer, only when connecting to local containers for testing and development. Testcontainer is a proxy for remote "cloud services".

ronwhite562@gmail.com

Launch Testcontainer Application with Greetings API Service

Testcontainer has to run in Debug mode in order to provision its resources under a context that doesn't interfere with Release or Staging environments. Depending on the IDE, Build and Run in Debug is usually a play button with a bug icon or as an explicitly defined Run Schema.

For VSCode, select the TestcontainersApplication under ./test/java/com/demo/greetings directory and click Debug and Run icon in the left tool bar.

NOTE: this is different than *Task* > *Application* > *Debug and Run*. That will run the Greetings

The Terminal or IDE Console should output the startup process which starts with:

```
[greetings] [main] com.demo.greetings.Application :
    Starting Application using Java 17.0.12 with PID 6943
    (/path/to/project/build/classes/java/main)
                    ... and ends with something like ...
[greetings] [main] o.s.b.w.embedded.tomcat.TomcatWebServer :
    Tomcat started on port 8080 (http) with context path '/'
[greetings] [main] com.demo.greetings.Application :
    Started Application in 26.761 seconds (process running for 27.384)
```

TEST 1: To confirm that the process is running and ready to handling request,
navigate to http://localhost:8080 in a browser window. This will return a *Whitelabel Error Page* with following output expected:

```
This application has no explicit mapping for /error, so you are seeing this as a fallback.
Wed Sep 04 19:52:45 EDT 2024
There was an unexpected error (type=Not Found, status=404).
```

TEST 2: To confirm the Postgres database is up and ready for request,
run the following psql command:

cmd:
```
PGPASSWORD=test psql -h $DB_HOST -U $DB_USER -p $DB_PORT -d $DB_BASE \
-c "select table_schema, table_name from information_schema.tables where \
table_schema in ('public');"
```

output:

```
table_schema |  table_name
-------------+----------------------
public       | flyway_schema_history
public       | greeted                    < this is our table with two columns: id, username.
(2 rows)
```

NOTE: if greeted isn't listed, see Database_Setup.pdf troubleshooting tips.

[Done]