Optimising your inner development loop with Spring Boot 3.1+ and Docker

Brian Matthews

Version 1.0.0, 3 April 2025

About Me

Developing in Java since 1998 and working with the Spring ecosystem since 2006.

@bmatthews68

brian@buralotech.com

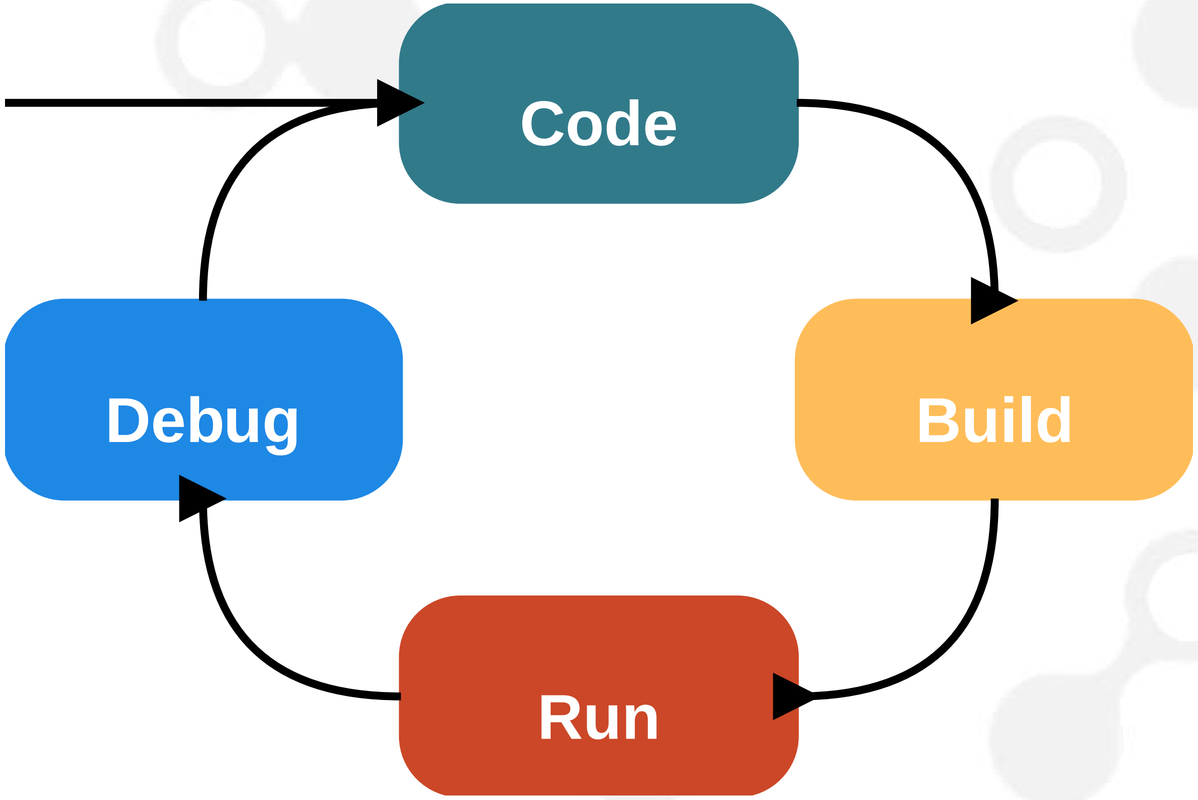
https://www.linkedin.com/in/bmatthews68

Inner Development Loop

Workflow

- **Code**—write or change code
- **Build**—compile the code and execute unit tests
- **Run**—run the compiled code locally
- **Debug**—debug issues while running ad hoc tests

Rinse and repeat



Testing Pyramid

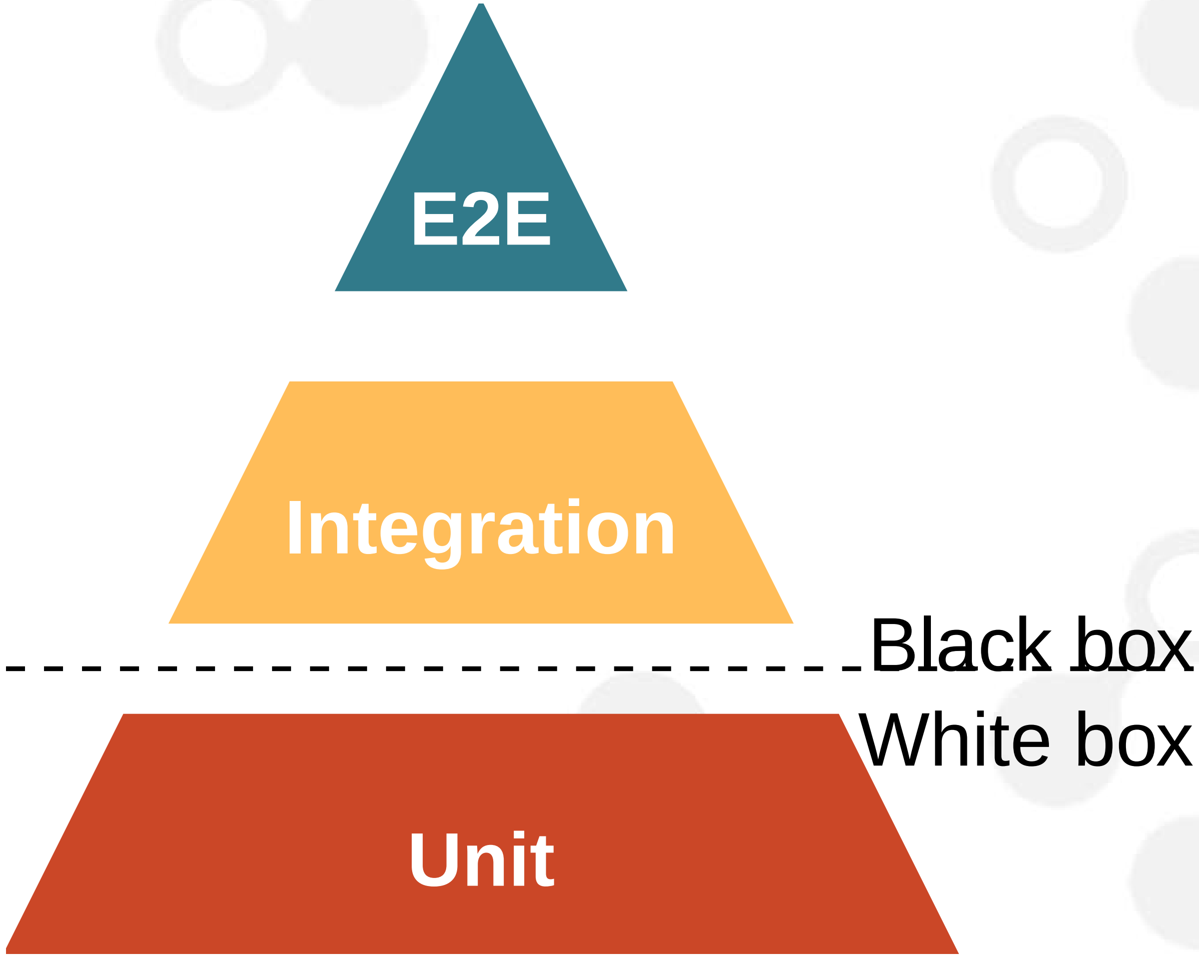
Scope

- **Unit**—Individual classes and methods
- **Integration**—Interactions between components
- **End-to-End (E2E)**—Whole system

Cost to develop increases as you go up the pyramid

Tests lower down the pyramid

- Run faster
- Easier to automate
- Cheaper to write



Integration testing options

Embedded systems (H2, JMemcached, etc)

- Not always an option
- Usually not feature complete
- Dialect and behavioural differences

Shared environments

- Contention with other developers
- Tethered to the network

Docker Compose & Testcontainers

Docker Compose

Pros

No manual installation & configuration

No external dependencies

Consistent experience for everybody

Cons

Hard-coded configuration in test code

Manual container lifecycle management

Testcontainers

Pros

Dynamic port assignment

- No port conflicts
- No hard-coded configuration in test code

Automated container lifecycle management

Runs in the build pipeline

Cons

Extra-code needed to configure application properties

Only suitable for automated unit and integration tests

Spring Boot Development-time Services

Introduced in Spring Boot 3.1

Improves the experience of using Docker Compose and Testcontainers

Manages container lifecycles

Reduces test configuration

Supports many well-known containers out of the box

https://docs.spring.io/spring-boot/reference/features/dev-services.html

ConnectionDetails

Development-time Services required a bit of refactoring

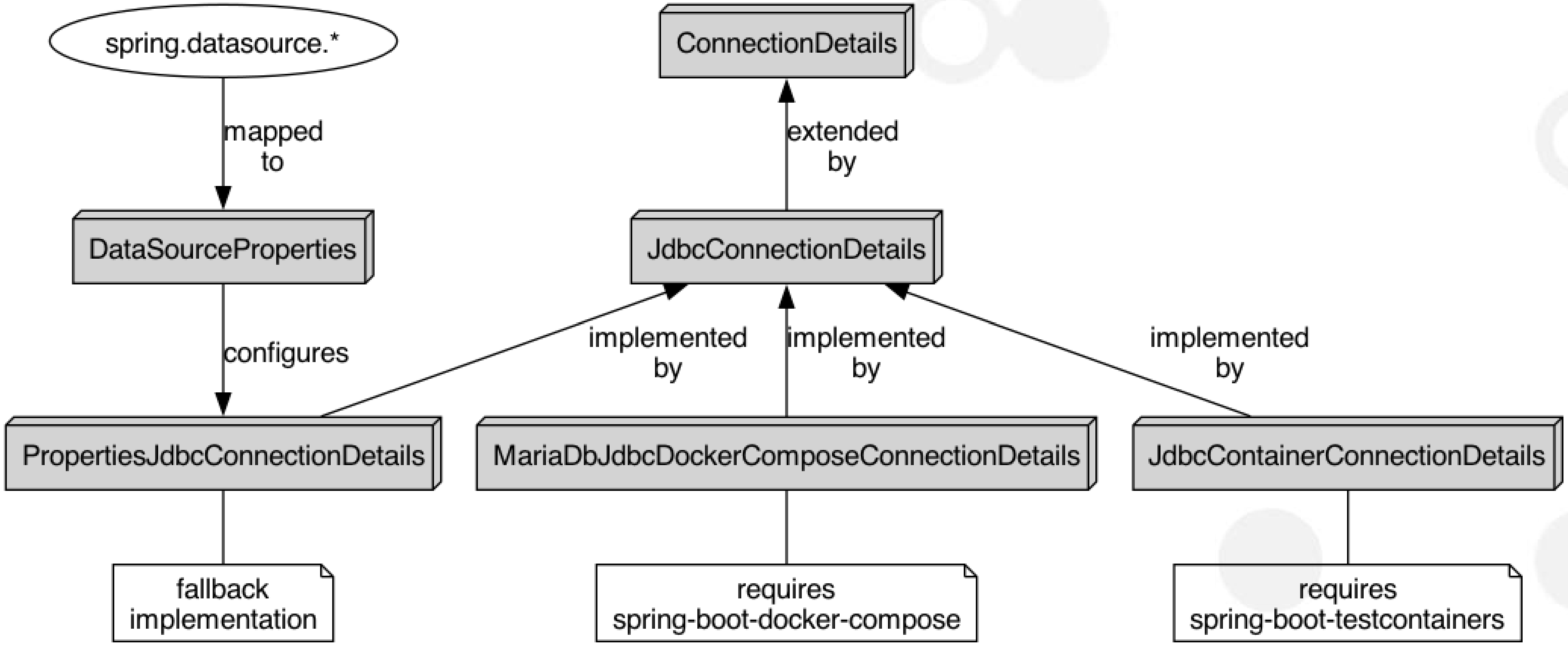
`ConnectionDetials`—new abstraction to encapsulate properties required to connect to an external service

- Autoconfiguration maps properties to simple `ConnectionDetails` implementations
- Factories required to create `ConnectionDetails` for Docker Compose and Testcontainers

Out-of-the box

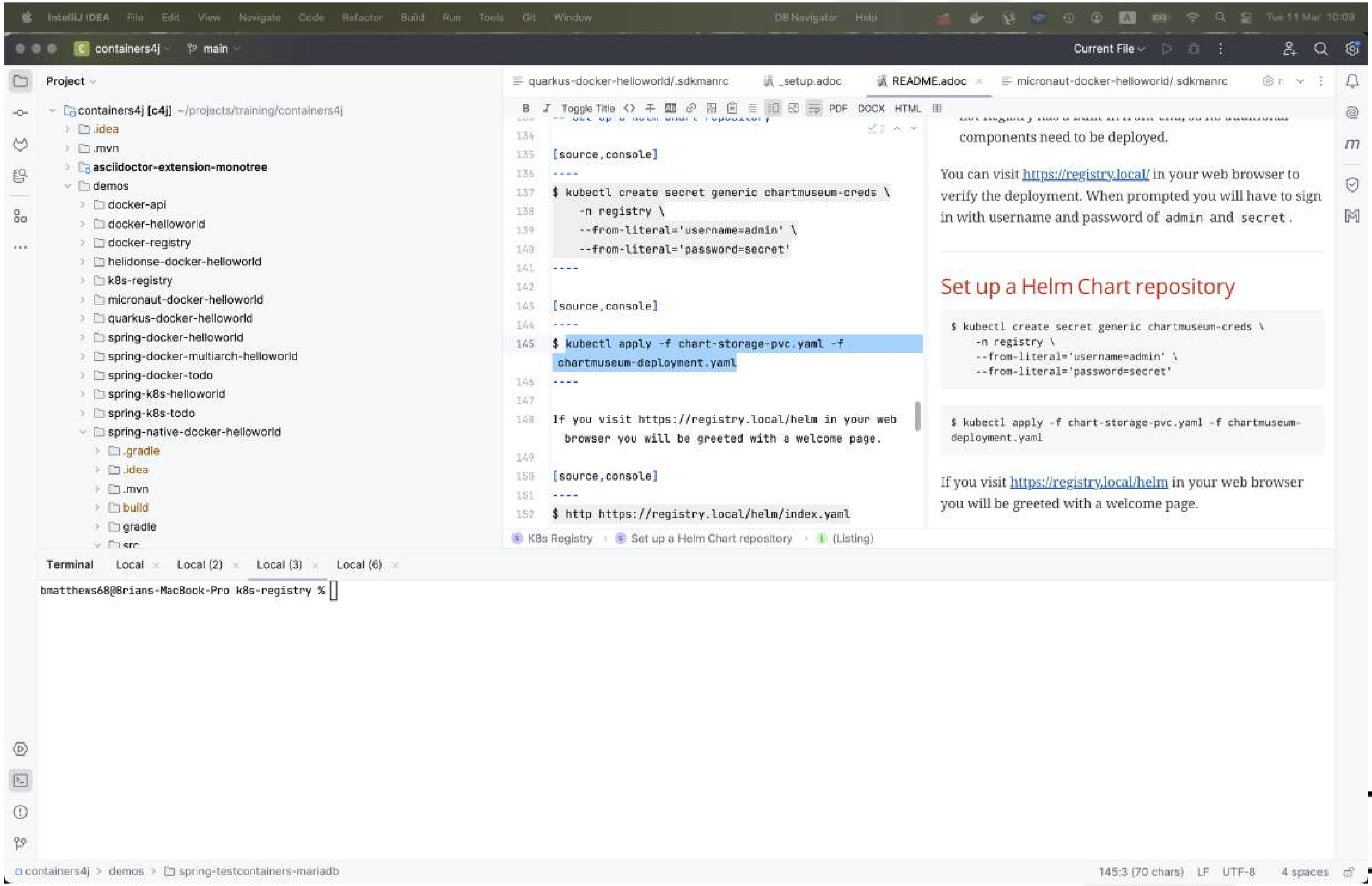
- ActiveMQ
- Artemis
- Cassandra
- Elastic Search
- Hazelcast
- JDBC
- LDAP
- MongoDB
- Neo4J
- OLTP Logging
- OLTP Metrics
- Pulsar
- R2DBC
- Rabbit
- Redis
- Zipkin

JdbcConnectionDetails


spring.datasource.*
mapped to
DataSourceProperties
configures
PropertiesJdbcConnectionDetails
fallback implementation
ConnectionDetails
extended by
JdbcConnectionDetails
implemented by
implemented by
implemented by
MariaDbJdbcDockerComposeConnectionDetails
requires spring-boot-docker-compose
JdbcContainerConnectionDetails
requires spring-boot-testcontainers

Live coding

https://gitlab.com/buralo/talks/spring-boot-containers-support


IntelliJ IDEA File Edit View Navigate Code Refactor Build Run Tools Git Window DB Navigator Help
Tue 11 Mar 10:09
containers4j
main
Current File
Project
containers4j [c4j] ~/projects/training/containers4j
.idea
.mvn
asciidoctor-extension-monotree
demos
docker-api
docker-helloworld
docker-registry
helidonse-docker-helloworld
k8s-registry
micronaut-docker-helloworld
quarkus-docker-helloworld
spring-docker-helloworld
spring-docker-multiarch-helloworld
spring-docker-todo
spring-k8s-helloworld
spring-k8s-todo
spring-native-docker-helloworld
.gradle
.idea
.mvn
build
gradle
src
quarkus-docker-helloworld/.sdkmanrc
_setup.adoc
README.adoc
micronaut-docker-helloworld/.sdkmanrc
B I Toggle Title PDF DOCX HTML
K8s Registry › Set up a Helm Chart repository › (Listing)
Terminal Local Local (2) Local (3) Local (6)
containers4j › demos › spring-testcontainers-mariadb
145:3 (70 chars) LF UTF-8 4 spaces

```
[source,console]
----
$ kubectl create secret generic chartmuseum-creds \
    -n registry \
    --from-literal='username=admin' \
    --from-literal='password=secret'
----

[source,console]
----
$ kubectl apply -f chart-storage-pvc.yaml -f chartmuseum-deployment.yaml
----

If you visit https://registry.local/helm in your web browser you will be greeted with a welcome page.

[source,console]
----
$ http https://registry.local/helm/index.yaml
```

components need to be deployed.

You can visit https://registry.local/ in your web browser to verify the deployment. When prompted you will have to sign in with username and password of `admin` and `secret` .

Set up a Helm Chart repository

```
$ kubectl create secret generic chartmuseum-creds \
    -n registry \
    --from-literal='username=admin' \
    --from-literal='password=secret'
```

```
$ kubectl apply -f chart-storage-pvc.yaml -f chartmuseum-deployment.yaml
```

If you visit https://registry.local/helm in your web browser you will be greeted with a welcome page.

```
bmatthews68@Brians-MacBook-Pro k8s-registry %
```

Conclusions

If this fool can do it—any fool can do it

- Not limited to 3rd party components—use for shared internally developed components
- Go the extra mile and create a Testcontainers module

Alternatives

Spring Boot TestJars is only for Jar files (still experimental)

Skaffold, Tilt or Devspaces

- Requires a Kubernetes cluster
- Better suited for end-to-end testing

Q&A

@bmatthews68 ✉ brian@buralotech.com

https://linkedin.com/in/bmatthews68