… the return of Spring

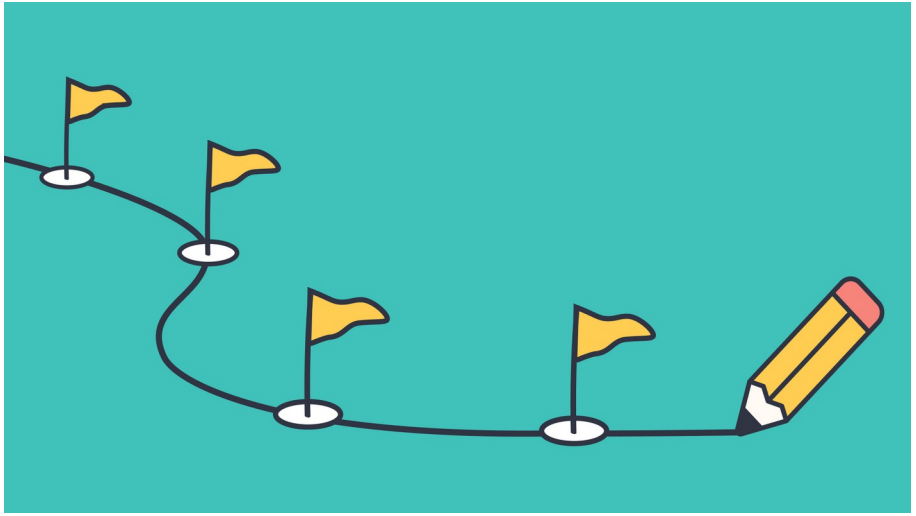
Agenda

- Containers
- Building and running Containers
- From Java SE to Spring Boot
- Introduction to Spring
  - Services and components
  - Controllers
  - Persistance
  - Configuration
- Running Spring in a Container
- Wrap up

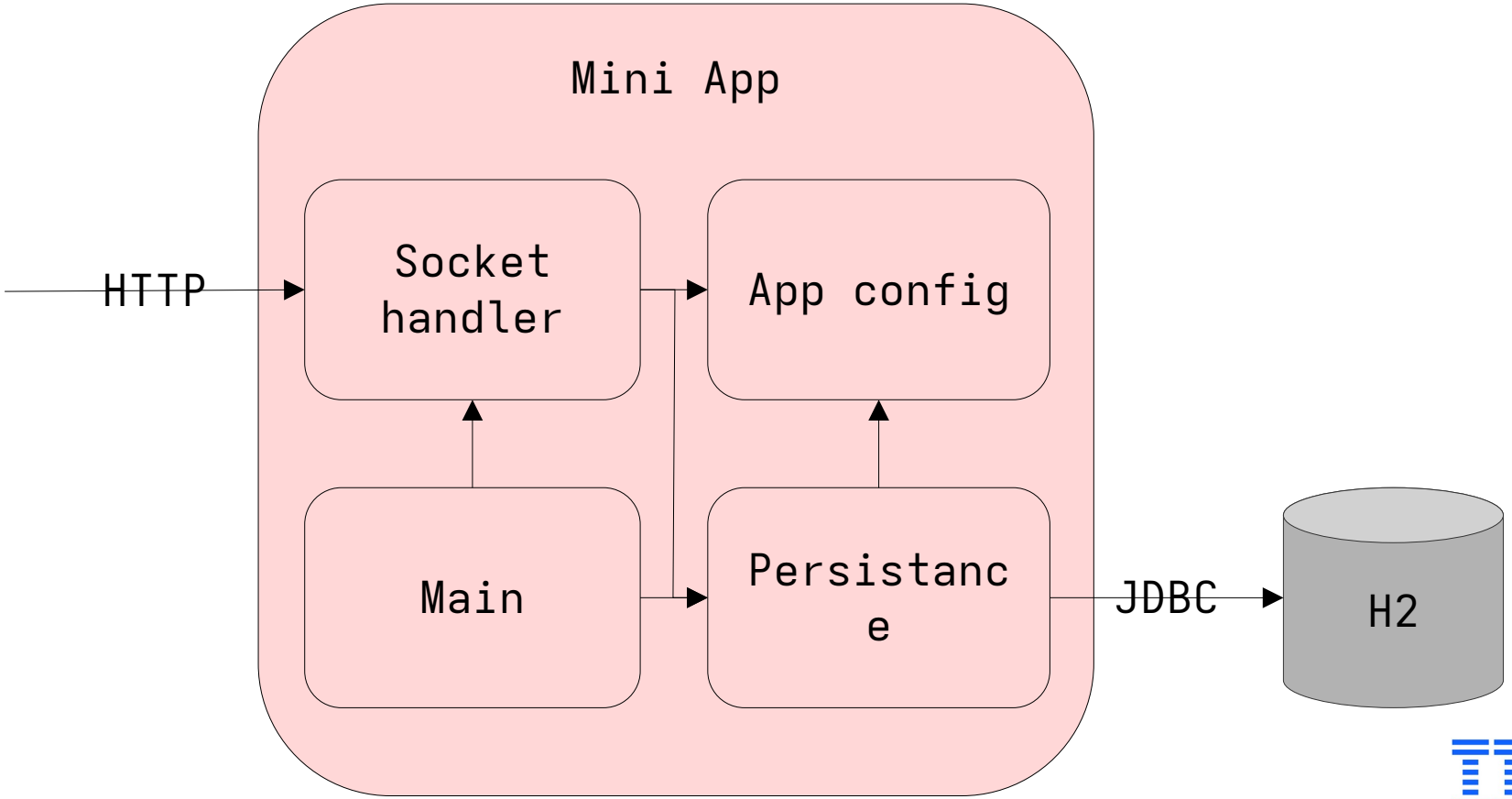
IBM

Learning goals

- Recap simple Java app
- Intro to containers
  - Docker / Podman
  - Practical example
  - Operation tips / tricks
- From Java to Spring
  - Spring Initializr
  - Basics of Spring Framework
  - Implementation and test

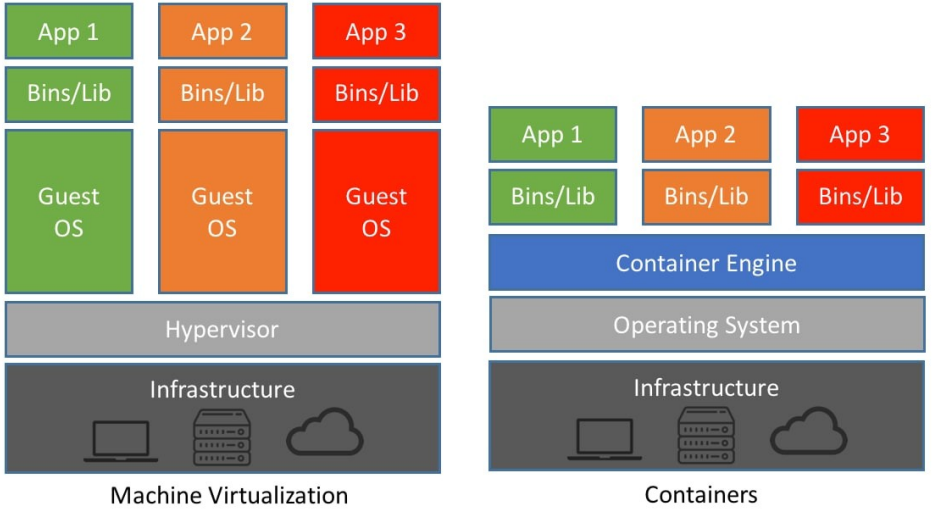


Mini-app design



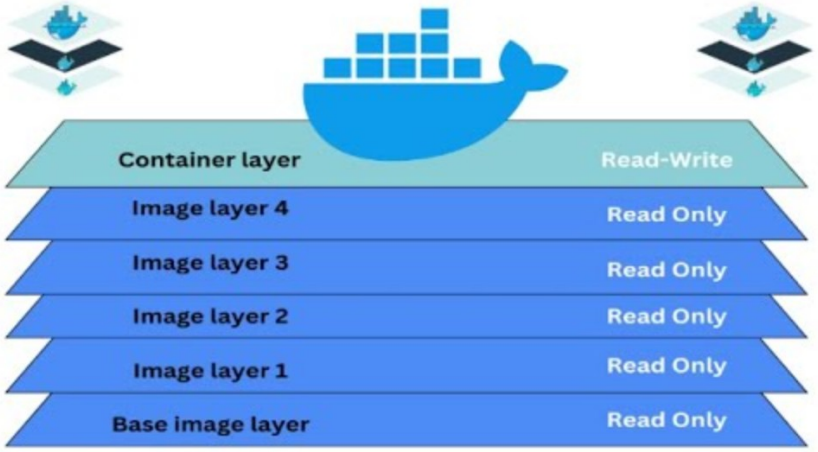
Containers Vs Virtual Machines

- Both used for isolation of apps
- Isolated host OS process behaving as a separate OS
- Different flavors
  - Docker – uses daemon
  - Podman – daemonless and rootless

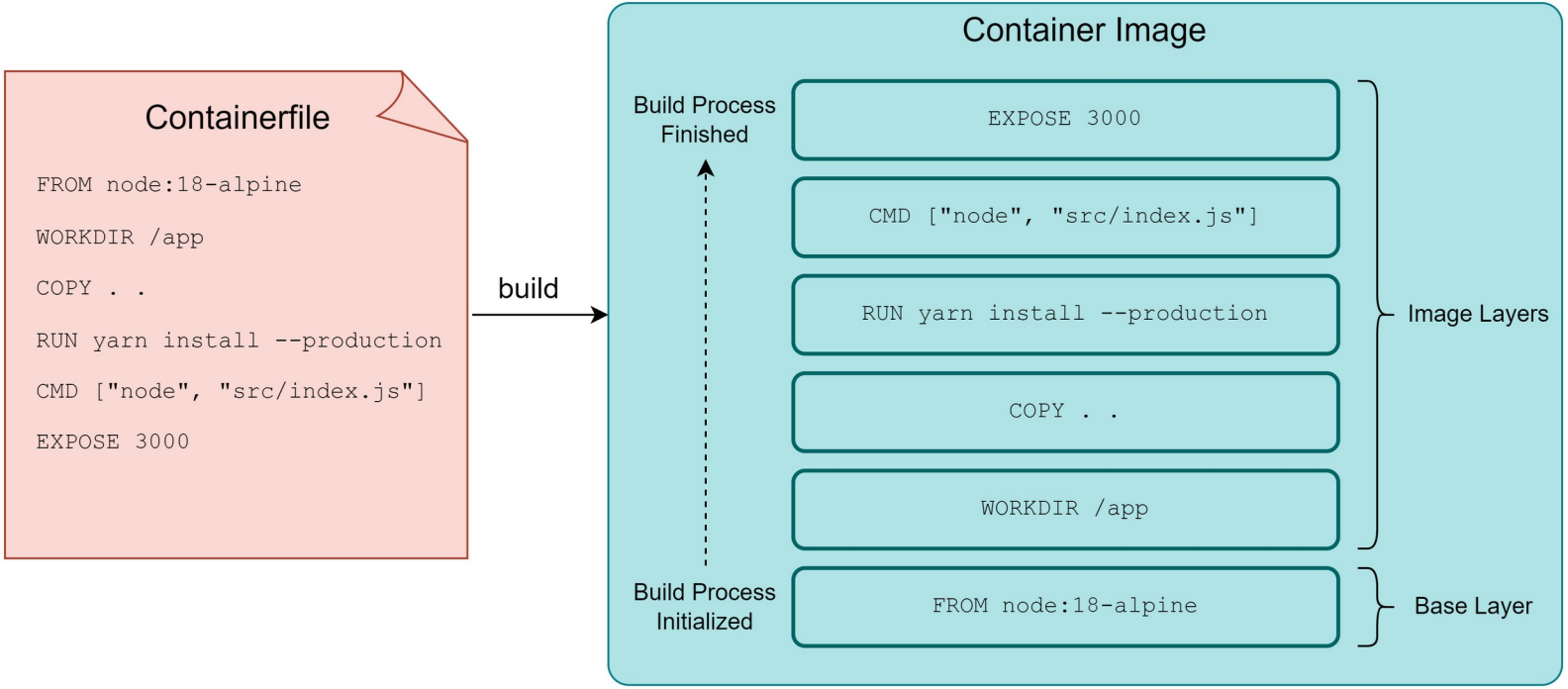


Container images

- Containers made up from layers
  - Read only layers
  - Write only to container laye
- Layers gecached
- Layers shared between container builds
- Same principle to Containerfile / Dockerfile

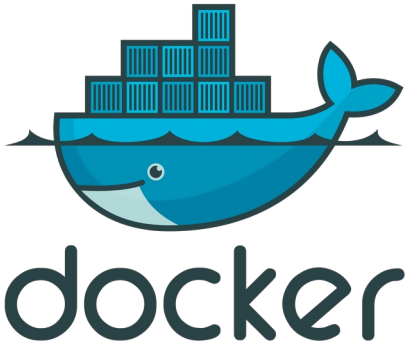


Dockerfiles / Containerfiles



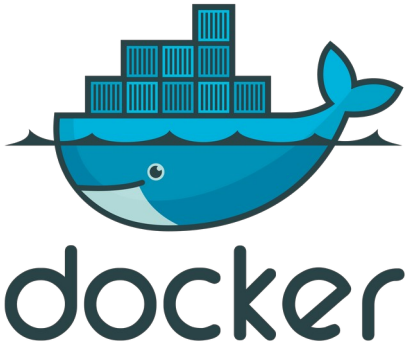
Building and running containers

- Building the container from the containerfile
  - `docker build -t myapp .`
- Run a new container
  - `docker run -p 8080:8080 myapp`
- Check containers running
  - `docker ps`
- Stopping containers
  - `docker stop [id]`



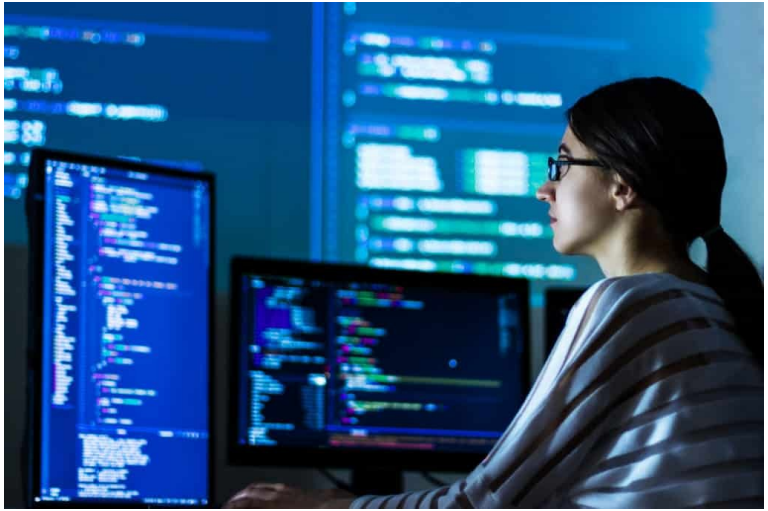
Sharing a Container

- Base images hosted in registries
  - Dockerhub
  - Quay
  - github
  - …
- Image can be exported to a single file
  - `docker save -o myapp.tar myapp`
  - `docker load -i myapp.tar`



DEV: create your Containerfile

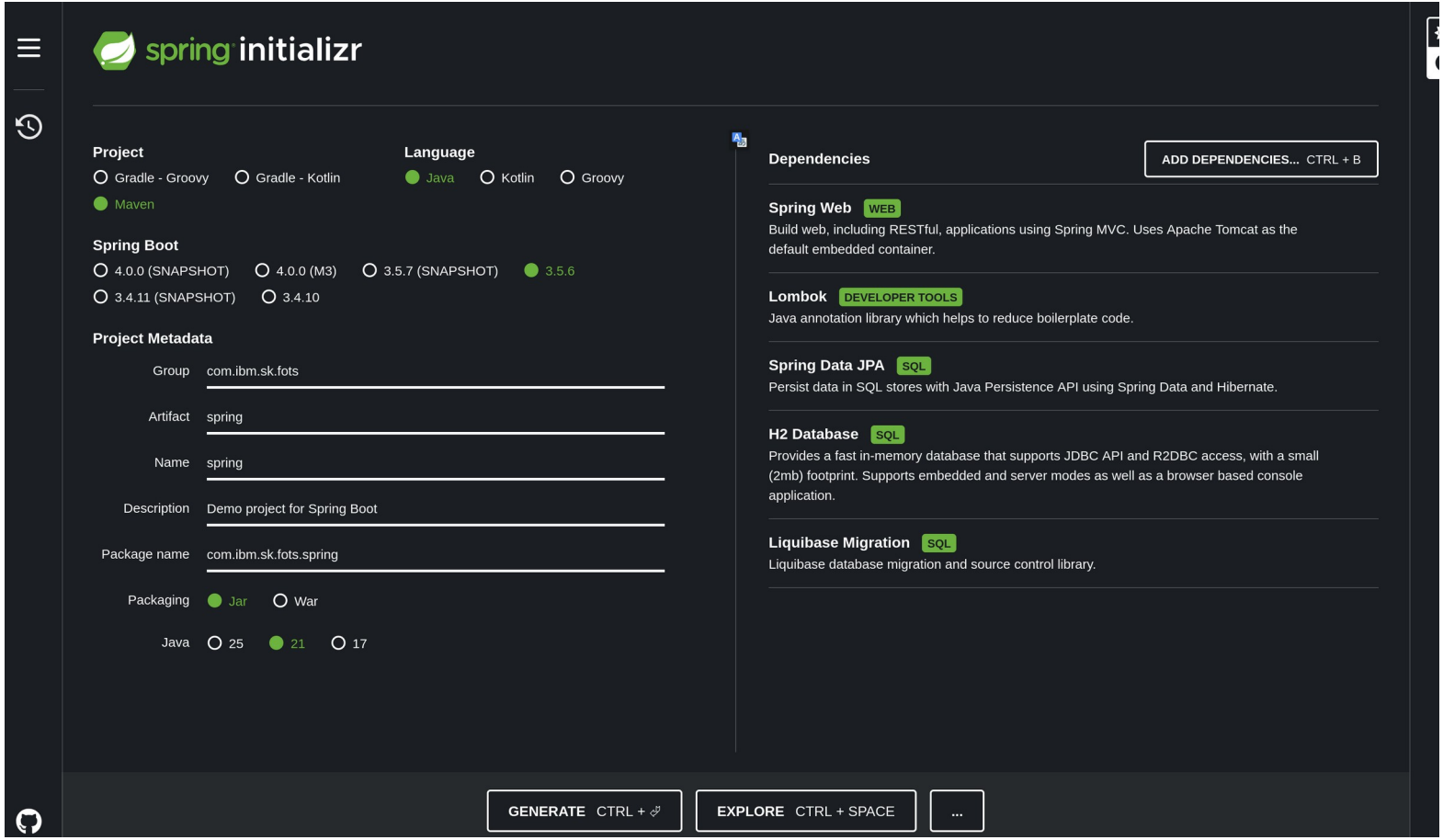
- Package the application from the first session
  - You should have a .jar file in the target folder
  - Copy the h2 library into the target folder as well
- Create a Containerfile
  - Use openjdk:21-jdk-slim base image
  - Copy jars to a target directory
  - Run Java command as entry point



Why Spring

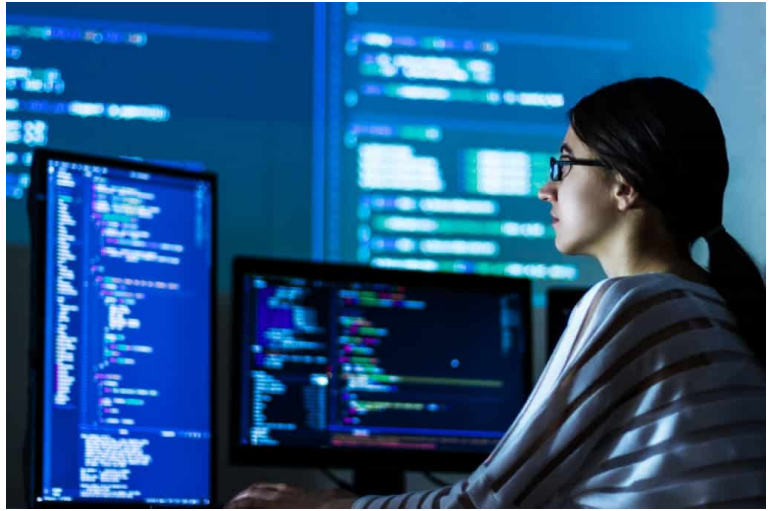
- Plain Java – has its oddities
  - verbose
  - hard wired by default
  - no real guidance – every project is unique
- Spring Framework (Boot)
  - Pre-Configured in most cases
  - Fits the Maven philosophy
    - Convention over configuration
  - Batteries included – embedded web server, configuration handling etc.
  - Easy to start with start.spring.io

Why Spring

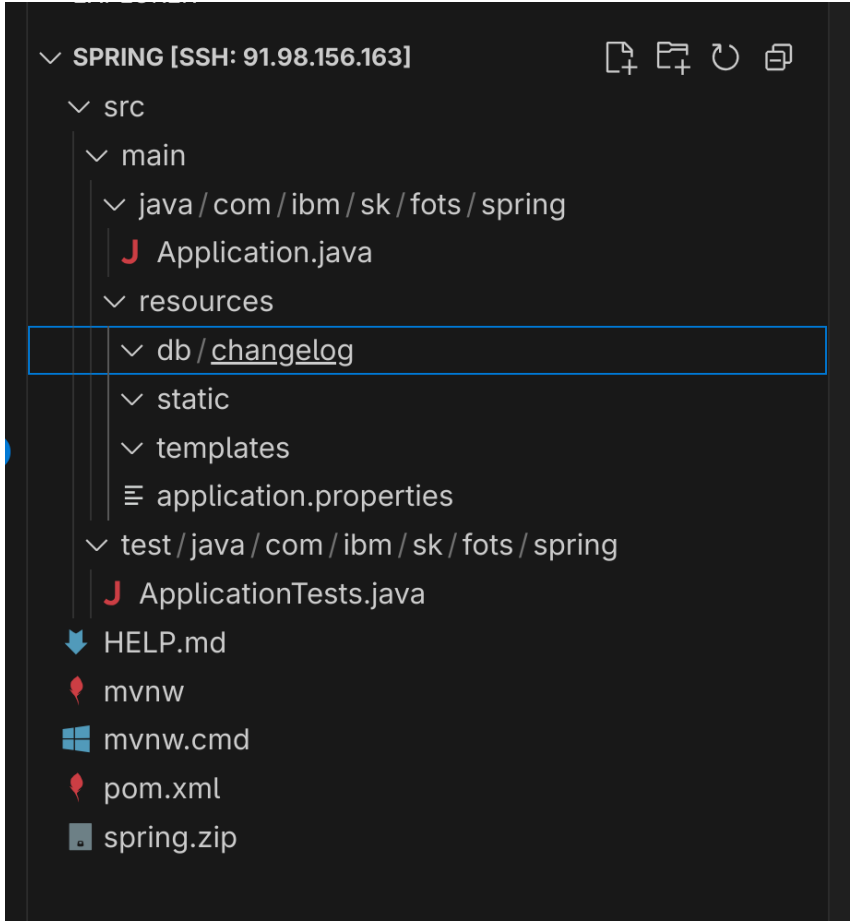

spring initializr
Project
Gradle - Groovy
Gradle - Kotlin
Maven
Language
Java
Kotlin
Groovy
Spring Boot
4.0.0 (SNAPSHOT)
4.0.0 (M3)
3.5.7 (SNAPSHOT)
3.5.6
3.4.11 (SNAPSHOT)
3.4.10
Project Metadata
Group com.ibm.sk.fots
Artifact spring
Name spring
Description Demo project for Spring Boot
Package name com.ibm.sk.fots.spring
Packaging Jar War
Java 25 21 17
Dependencies
ADD DEPENDENCIES... CTRL + B
Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.
Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
H2 Database SQL
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.
Liquibase Migration SQL
Liquibase database migration and source control library.
GENERATE CTRL +
EXPLORE CTRL + SPACE
...

DEV: create your Spring project

- In the IDE
  - Switch to the `~/spring` folder
  - Clone https://github.com/szabogabriel/FotS_2025_spring
  - Create your feature branch
- Create project
  - Start the wizard in the **IDE** by F1 → **Spring Boot Initializr plugin** or **use the web**
  - Enter the parameters
    - Spring Boot Version 3.5.6
    - Project Language: Java
    - Group: com.ibm.sk.fots
    - Artifact: spring
    - Packaging type: Jar
    - Java Version: 21
    - Dependencies: Spring Web, Lombok, Spring Data JPA, H2 Database
    - We add Liquibase Migration later
  - When using the web, copy the created ZIP file to the remote machine:
    - `scp spring.zip student[n]@91.98.156.163:/home/student[n]/spring`
  - Build the project and run the empty application
    - Run the class from IDE
    - From console: `mvn spring-boot:run`



Spring Boot project structure



Spring Boot basics

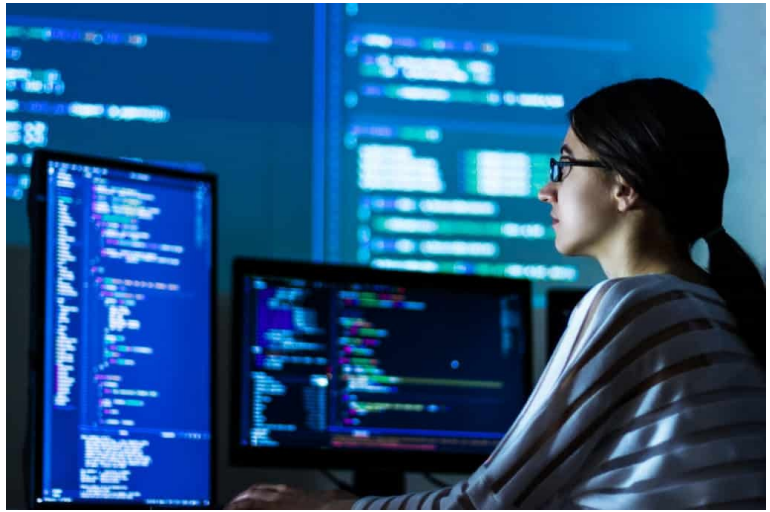
- Entry point: `@SpringBootApplication`
- Auto-config
  - Classpath scan
  - Component scan
- Manual configuration possible
- Projects often define additional conventions
  - package structure
  - naming structure
  - ...
- Packaged into runnable JAR with batteries included

Spring context

- Recognized beans stored in an internal container
- Different ways of defining beans
  - @Component
  - @Service
  - @Repository
- Beans scopes are configured via the @Scope annotation
  - singleton, prototype, request, session, application
- Dependency injection
  - Constructor preferred
  - Can use field level injection (harder to test)

DEV: hello controller

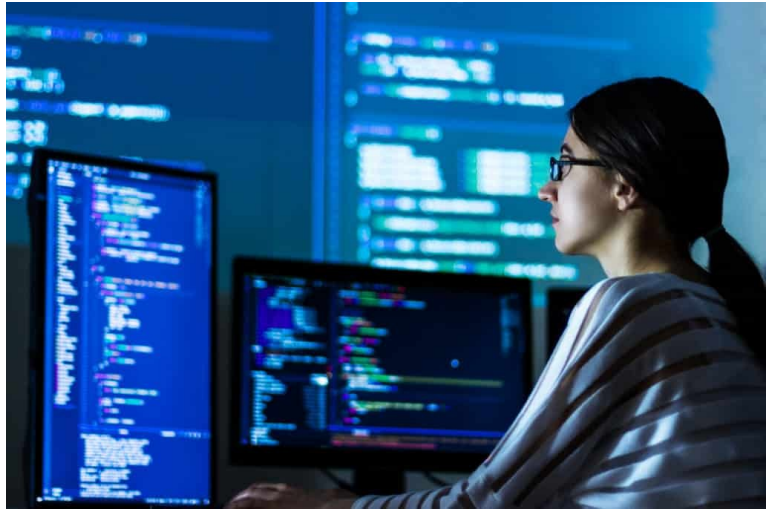
- Create a new package: controller
- Create a new class: `HelloController.java`
- Annotate with `@RestController`
- Create `public String hello()` method
- Return “Hello World” from the method
- Annotate the method with `@GetMapping(“/hello”)`
- Run the app
  - Your app should automatically pick up the pre-set port in the env-variables
  - Check the value via `env` command



DEV: configure a database

- Edit application.properties

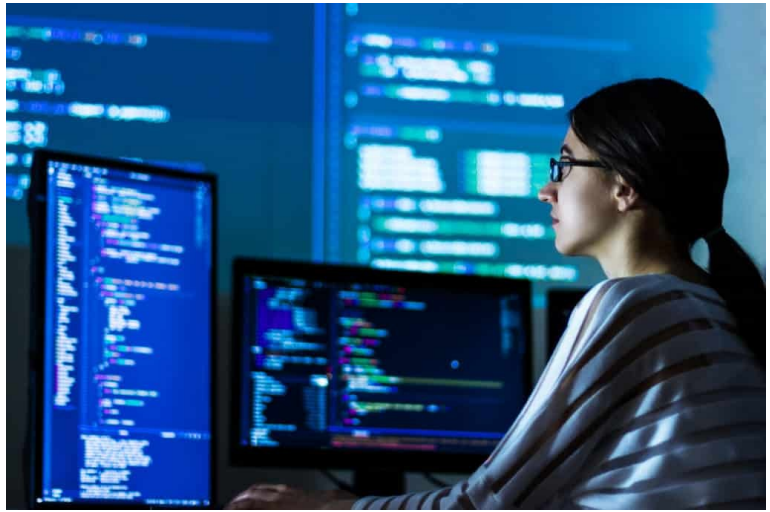
```
spring.datasource.url=jdbc:h2:mem:todo;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EX
IT=FALSE;MODE=PostgreSQL
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.driver-class-name=org.h2.Driver

spring.h2.console.enabled=true
spring.h2.console.settings.web-allow-others: true

spring.jpa.hibernate.ddl-auto=validate
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.show-sql=true
```



- Run the app and try connecting to the H2 console

DEV: creating entity and repository

- Create two packages
  - entity
  - repository
- Create `LogEntriesEntity.java` entity class → see cheat sheet
- Create `LogEntriesRepository.java` interface
  - Extend from `JpaRepository`
  - See cheat sheet

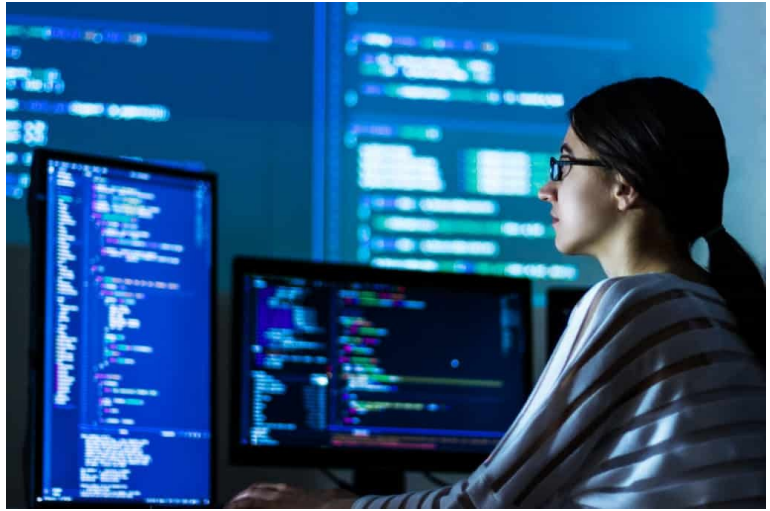


Dependency Injection

- Objects depend on abstract definition, not implementation
- Expect instance of implementation to be provided
- Spring creates and assigns the instances (“injects”) automatically
- Ideally constructor level injection
  - Great for testing
  - Explicit flow

DEV: wire it all together

- Create a LoggingService.java class
  - put it into the `service` package
  - Annotate as `@Service`
  - Add the `LogEntriesRepository` private member
  - Create a constructor with that value
  - Add the `@Autowired` annotation to the constructor
  - Create a method for logging
    - Create LogEntryEntity instance
    - Set message and timestamp
    - Save it via the repository
- Use this service in the controller you created
  - Same way as you used the repo in this service
- Check the log entries in the H2 database via the console /h2-console



Configuration and Profiles

- Basis: application.properties or application.yml
- Can be created for different profiles
  - application-dev.properties
  - application-prod.properties
  - These extend the configuration values from the `application.properties`
  - Activate via `--spring.profiles.active=dev`
- Profiles can be applied also to services
- Order of configuration values handling
  - application.properties, profiled properties, os environment, command line argument, programmatic

Best practices

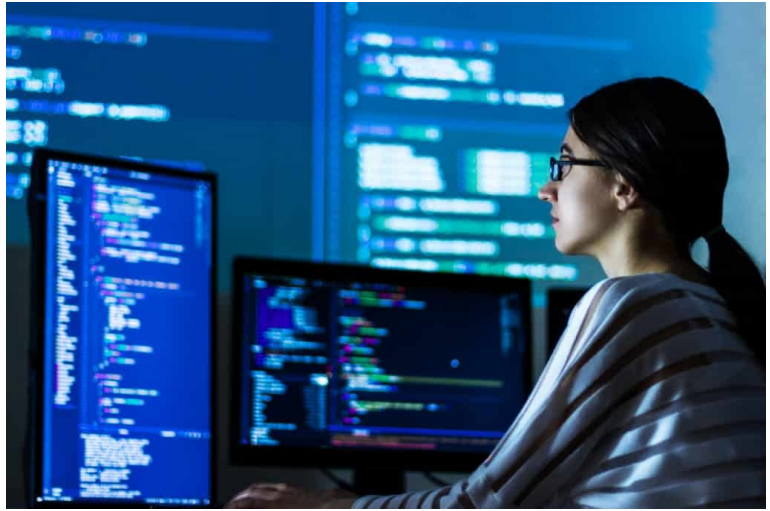
- Controller / service / repository separation
- Avoid logic in controllers
- Externalize configuration
- Use DTOs for REST
- Use single domain and single abstraction services
- Write tests!

DEV: package the app to Docker

- Create a `Containerfile` in the root folder of the project
- Create the content analog to the `Containerfile` from the beginning
- Copy only the single jar created by the build
- Run the app via the -jar switch

  ```
  CMD [“java”, “-jar”, “app.jar”]
  ```

- Build and run the containerized application



End to end test

- Create a few requests
- Show the H2 console
- Change property for the port by increasing it by 50
- Re-run the application

Why Spring

- Convention over configuration
  - Less boilerplate
  - Simple configuration goes a long way
  - Less error prone
- DI / IoC – gives us back some control
- From zero to a running app, fast

Wrap-up, homework and buffer

- Homework (Optional)
  - Try specifying the return type
  - Try adding a Query String argument to the GET request
  - Return the argument in the body
- Discussion

Thank you

Timeplan

- Intro + internet: 10m
- Recap Java App: 20m
- Docker + exercise: 40m
- Spring intro + create app: 30m
- Create Controller: 15m
- Create DB + explain + wiring together: 30m
- Config: 15m
- Package to Docker: 10m
- 170m + 10m backup