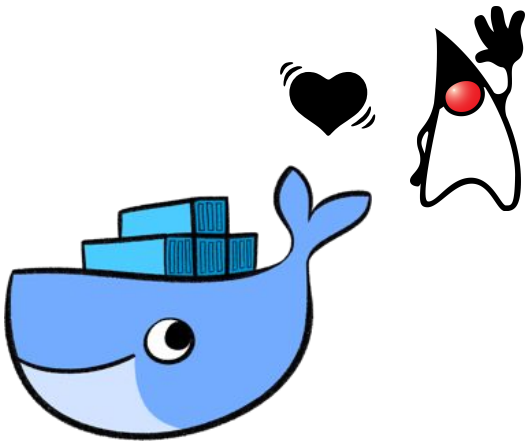


Docker for JVM Projects

Benjamin Muschko

O'REILLY®

About the trainer

bmuschko

bmuschko

bmuschko.com

AUTOMATED ASCENT

automatedascent.com

Training Objectives

Agenda

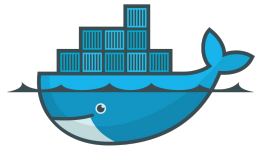
- ➔ Recap: Docker basics
- ➔ Why is Docker important for JVM projects?
- ➔ Containerization and distribution of JVM projects
- ➔ Resilient, reproducible testing with Docker containers
- ➔ Integrating Docker with JVM tooling
- ➔ Conclusion & Wrap-up

Introduction 1

Discussion

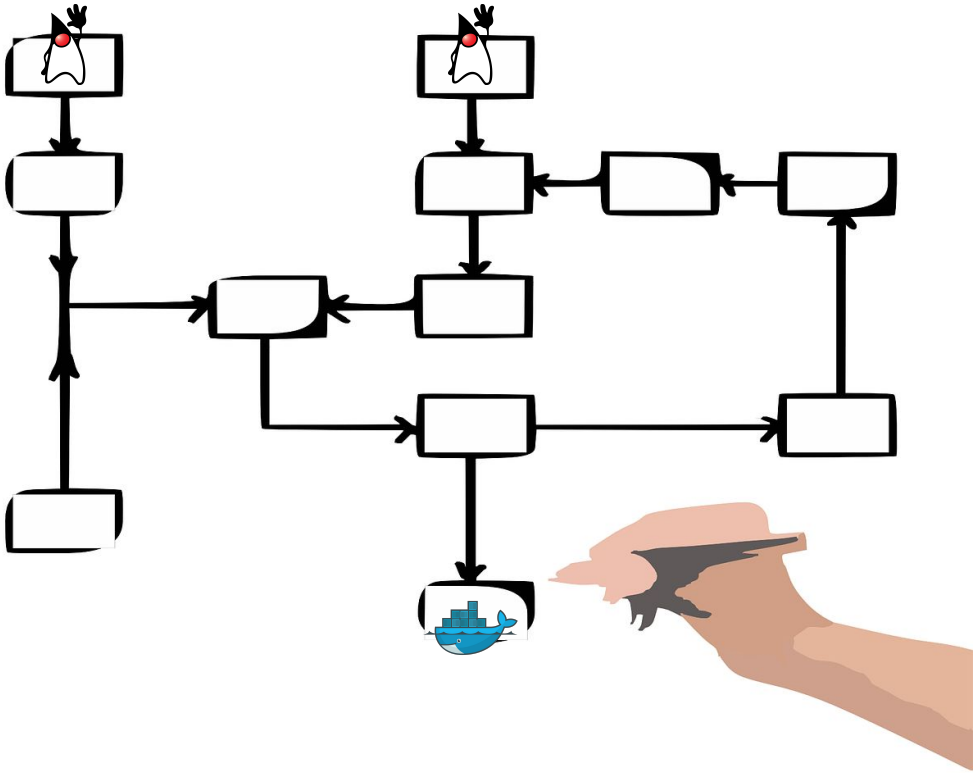
What's your experience with Docker as a JVM developer?

Docker in a nutshell

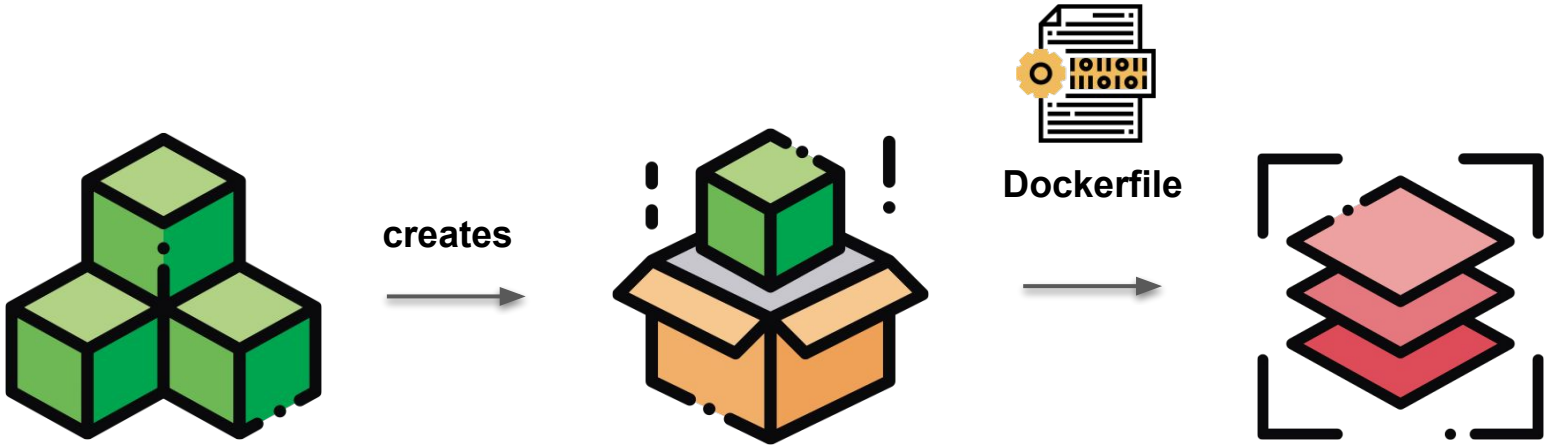


- ➔ Lightweight virtualization technology
- ➔ Creates, manages and orchestrates containers
- ➔ Containers are fast to start and portable
- ➔ Docker Engine is infrastructure plumbing software that runs and orchestrates containers

Typical Workflows



Dockerizing an application



Pushing an image to a registry

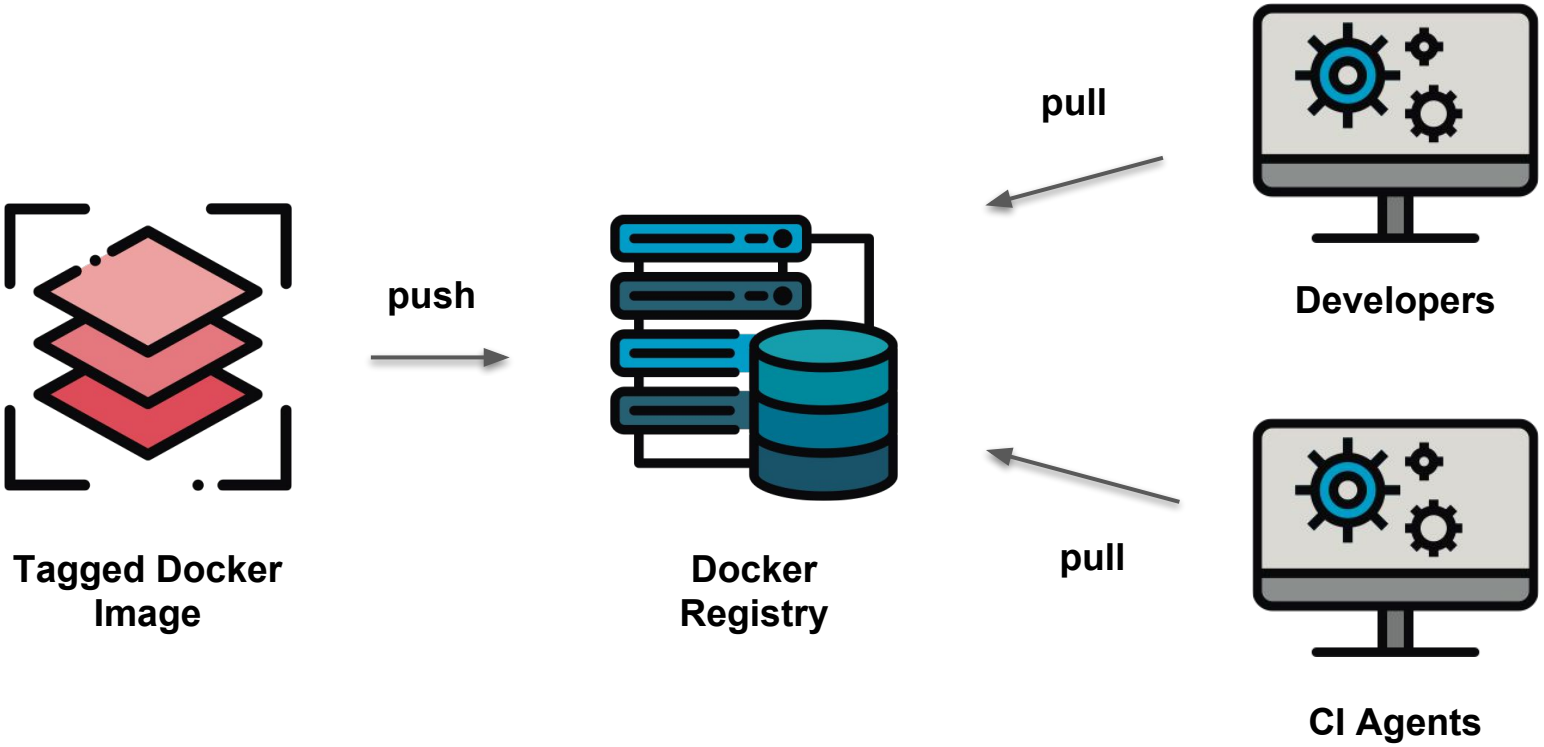
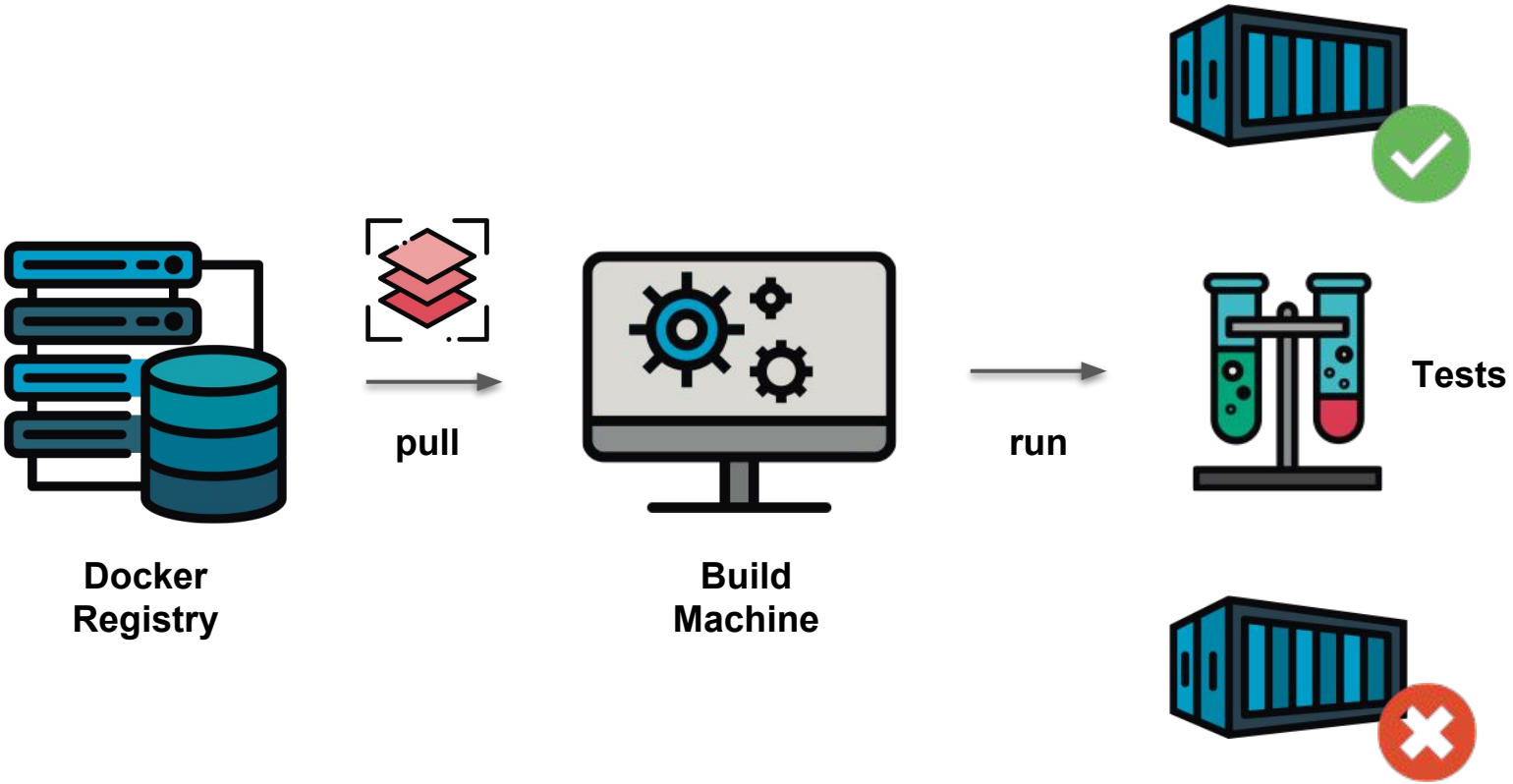
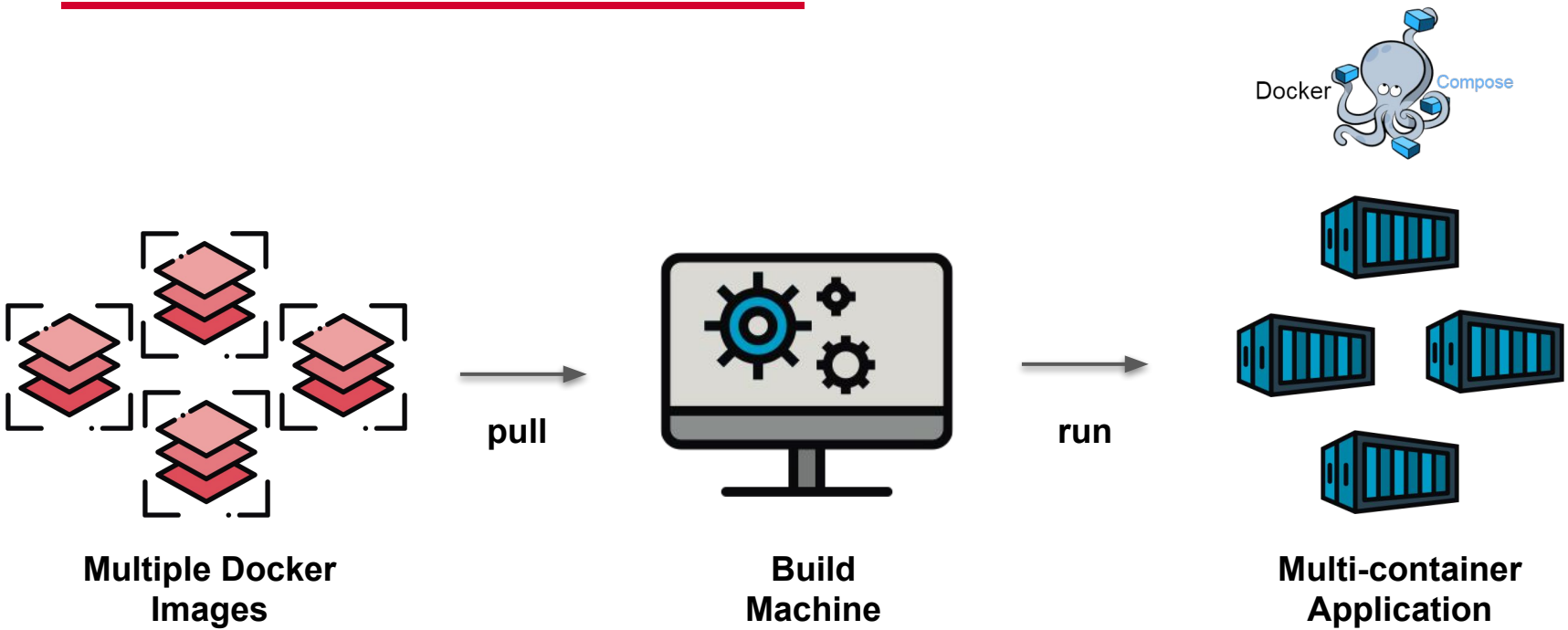


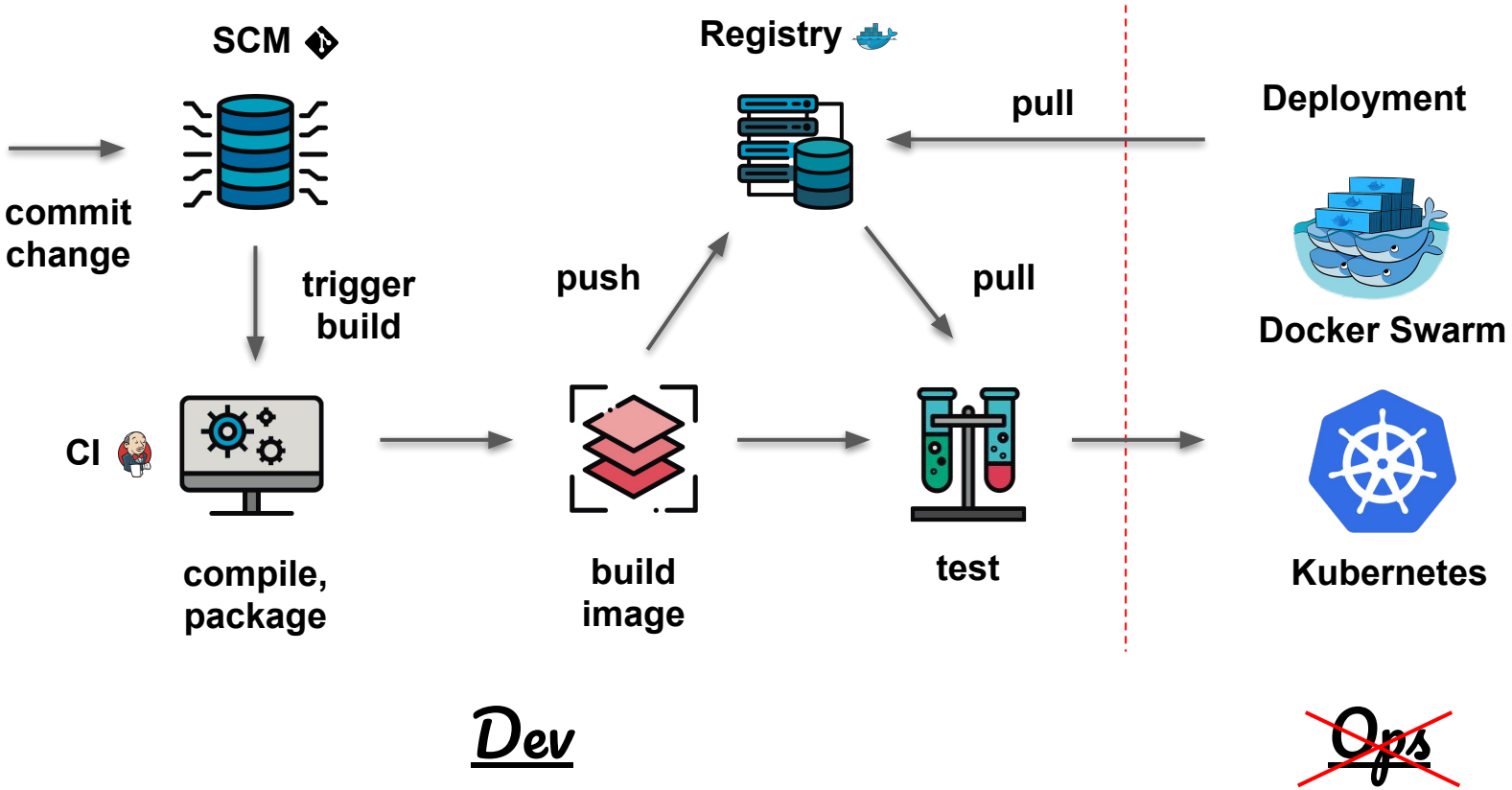
Image as fixture for testing

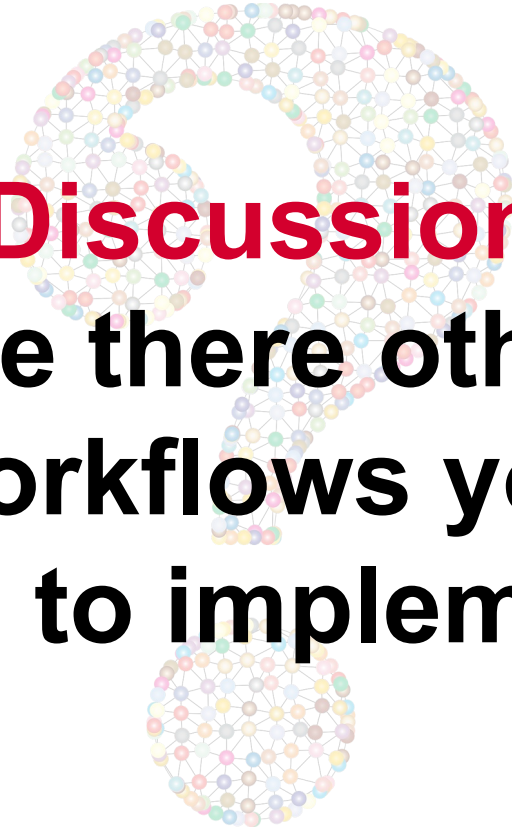


Running application stacks



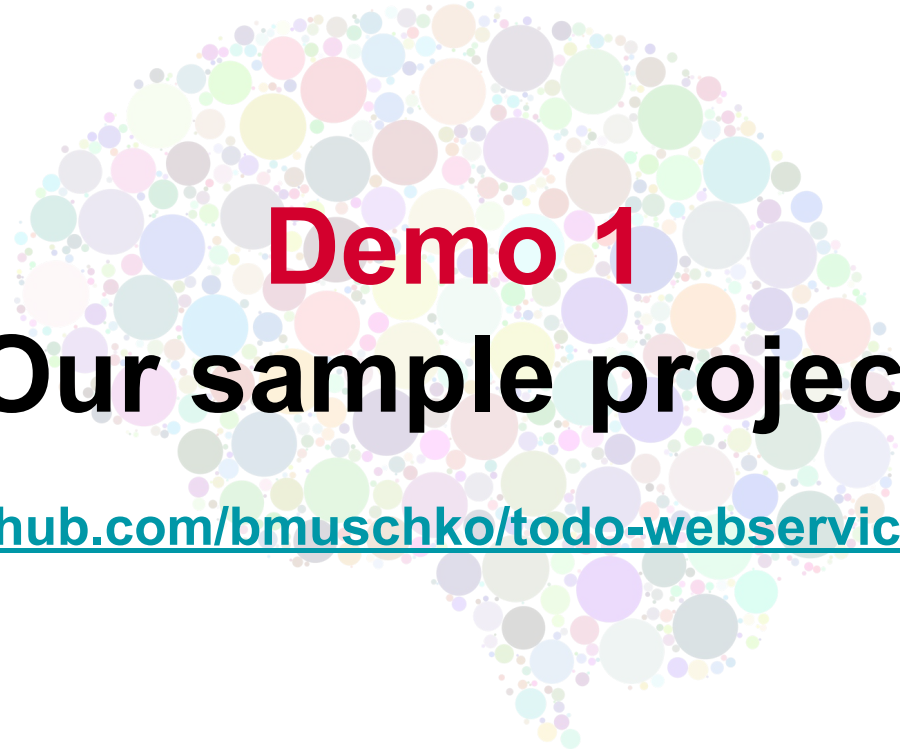
Docker as part of a build pipeline





Discussion

Are there other workflows you have to implement?



Demo 1

Our sample project

https://github.com/bmuschko/todo-webservice-exercise

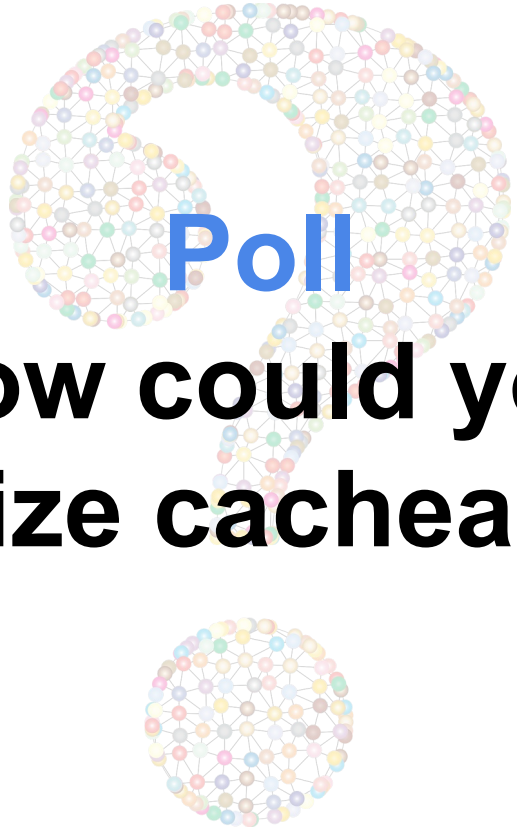
Containerization 2

Writing a Dockerfile for a Java application

- ➔ Pick a base image that’s as small as possible
- ➔ Copy application files as directories or archive?
- ➔ Providing runtime configuration
- ➔ Ensuring application readiness with a health check

Dockerfile for a Spring Boot application

```
FROM openjdk:jre-alpine
COPY todo-webservice-1.0.jar↵
    /app/todo-webservice-1.0.jar
ENTRYPOINT ["java"]
CMD ["-jar", "/app/todo-webservice-1.0.jar"]
HEALTHCHECK CMD wget --quiet --tries=1 --spider↵
    http://localhost:8080/actuator/health || exit 1
EXPOSE 8080
```



Poll

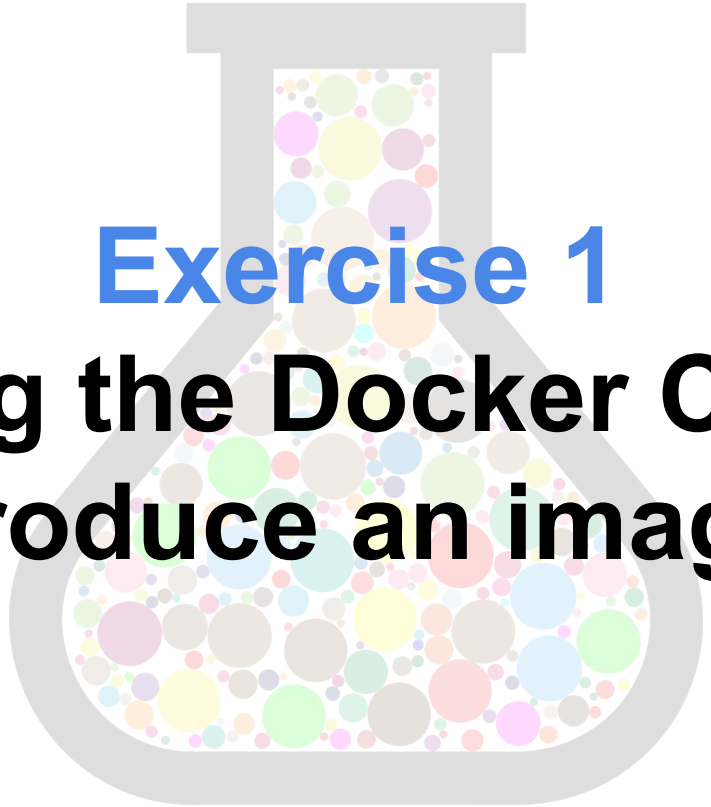
How could you optimize cacheability?

Dockerfile with layers for application files

```
FROM openjdk:jre-alpine
WORKDIR /app
COPY libs libs/
COPY resources resources/
COPY classes classes/
ENTRYPOINT ["java", "-cp",↲
    "/app/resources:/app/classes:/app/libs/*",↲
    "com.bmuschko.MyApplication"]
HEALTHCHECK CMD wget --quiet --tries=1 --spider↲
    http://localhost:8080/actuator/health || exit 1
EXPOSE 8080
```

Building an image

```
# Build image from Dockerfile
$ docker build -t todo-web-service:1.0.0 .
e14f431c33a2

# List the images
$ docker images

REPOSITORY          TAG           IMAGE ID       CREATED         SIZE
todo-web-service    1.0           e14f431c33a2   45 hours ago    99MB
openjdk             jre-alpine    ccfb0c83b2fe   6 months ago    83MB
```

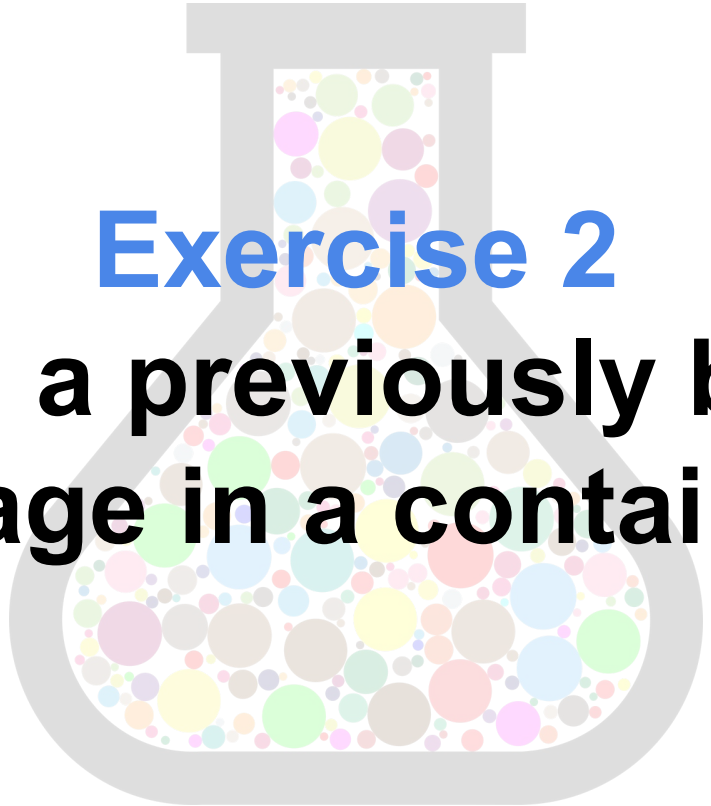


Exercise 1

Using the Docker CLI to produce an image

Running an image

```
# Run built image in container
$ docker run -d -p 8080:8080 todo-web-service:1.0.0
3760ac0bc8c68283473e5dffa123714...

# List the running container
$ docker container ls

CONTAINER ID  IMAGE              COMMAND                  CREATED
3760ac0bc8c6  todo-web-service  "java -cp /app/resou…"  32↵
seconds ago
```



Exercise 2

Run a previously built image in a container

Pushing an image to Docker Hub

```
# Log into Docker Hub
$ docker login --username=bmuschko

# Tag the image
$ docker tag bb38976d03cf↵
    bmuschko/todo-web-service:1.0.0

# Push image to Docker registry
$ docker push bmuschko/todo-web-service:1.0.0
```

Exercise 3

Using the Docker CLI to push image to Docker Hub

Docker’s CLI is great but do I need to operate it by hand?

Automate containerization with Jib

- ➔ Build images without the need for Docker Engine
- ➔ Dockerfile is optional
- ➔ Separates application files for cacheability
- ➔ Runs as Java library or Maven/Gradle plugins

Using the Jib Maven plugin

pom.xml

```
<project>
   ...
   <build>
       <plugins>
           ...
           <plugin>
               <groupId>com.google.cloud.tools</groupId>
               <artifactId>jib-maven-plugin</artifactId>
               <version>1.0.2</version>
               <configuration>
                   <to>
                       <image>myimage</image>
                   </to>
               </configuration>
           </plugin>
           ...
       </plugins>
   </build>
   ...
</project>
```

Building the image

```
$ mvn compile jib:build
```

Using the Jib Gradle plugin

build.gradle

```
plugins {
    id 'java'
    id 'com.google.cloud.tools.jib' version '1.0.2'
}

jib.to.image = 'bmuschko/java-app-jib:1.0'
```

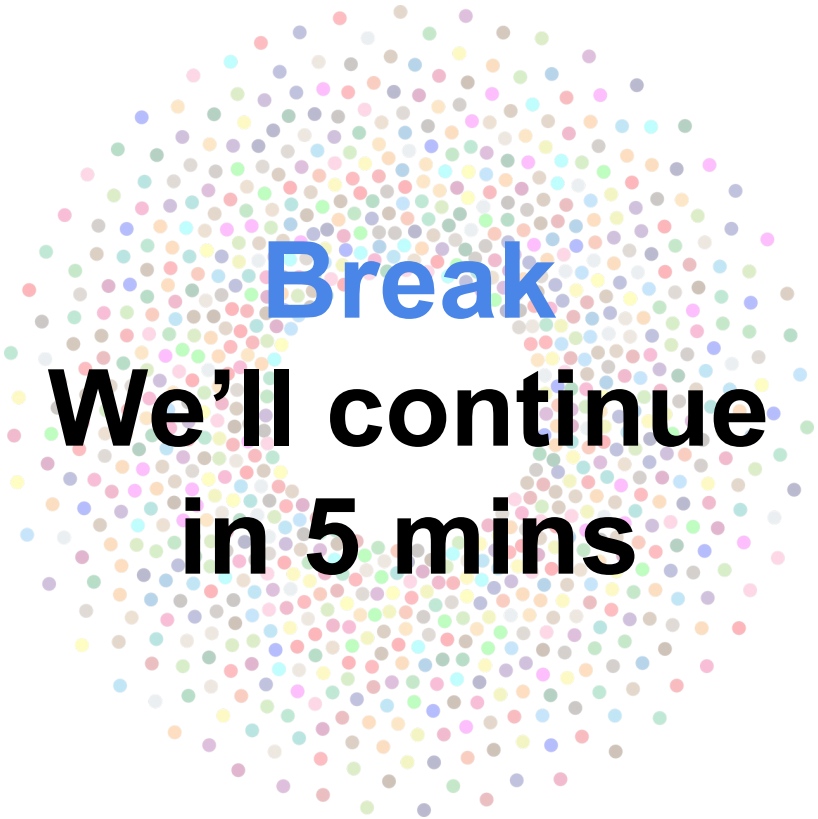
Building the image

```
$ gradle jib
```

Exercise 4

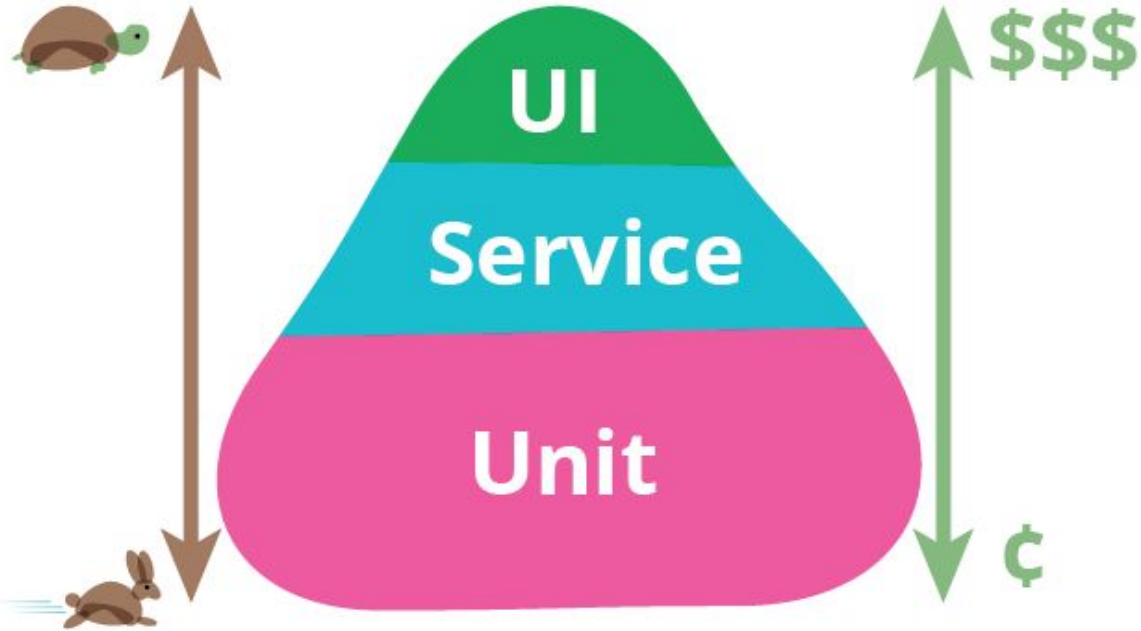
Using Jib to implement the containerization workflow

Please ask questions!


Break
We'll continue
in 5 mins

Testing 3

The testing pyramid

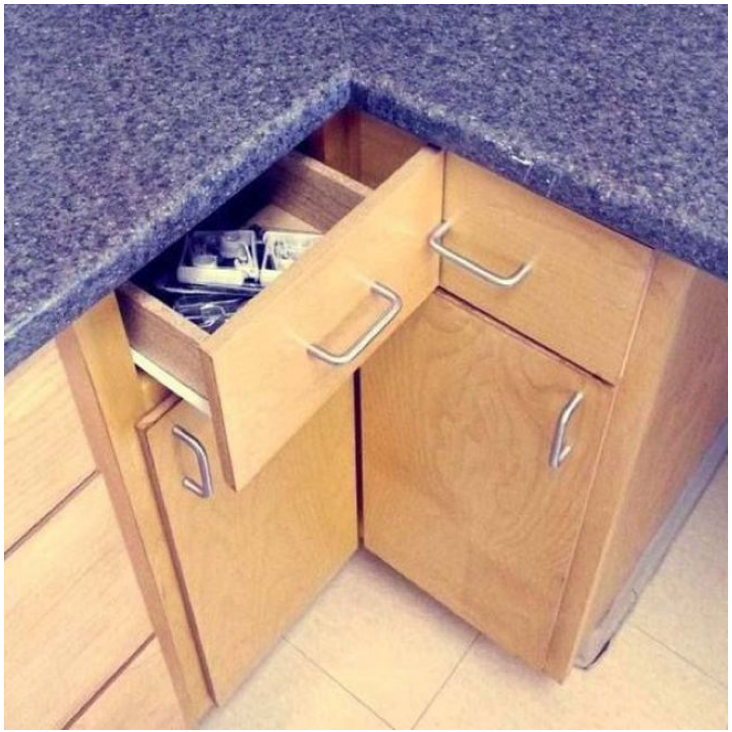


Source: https://martinfowler.com/bliki/TestPyramid.html

Discussion

Who writes integration or functional tests?

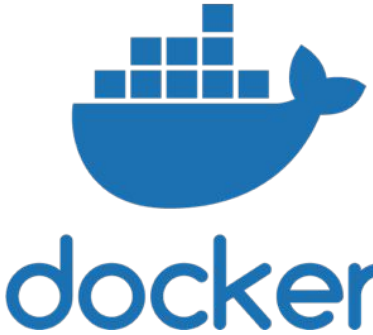
2 unit tests, 0 integration tests



Reproducible environment


Slow startup
times

Isolated & Cross-platform



The obvious choice

Java library for managing
Docker containers in JUnit tests

https://www.testcontainers.org/

Docker Engine Communication

Docker environment discovery

Automatic container cleanup

TestContainers dependencies in Maven

pom.xml

```
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>testcontainers</artifactId>
    <version>1.10.7</version>
    <scope>test</scope>
</dependency>
```

TestContainers libraries available on Maven Central

TestContainers dependencies in Gradle

build.gradle

```
repositories {
    jcenter()
}

dependencies {
    testImplementation 'org.testcontainers:junit-jupiter:1.10.7'
    testImplementation 'org.testcontainers:postgresql:1.10.7'
    testRuntime 'org.postgresql:postgresql'
}
```

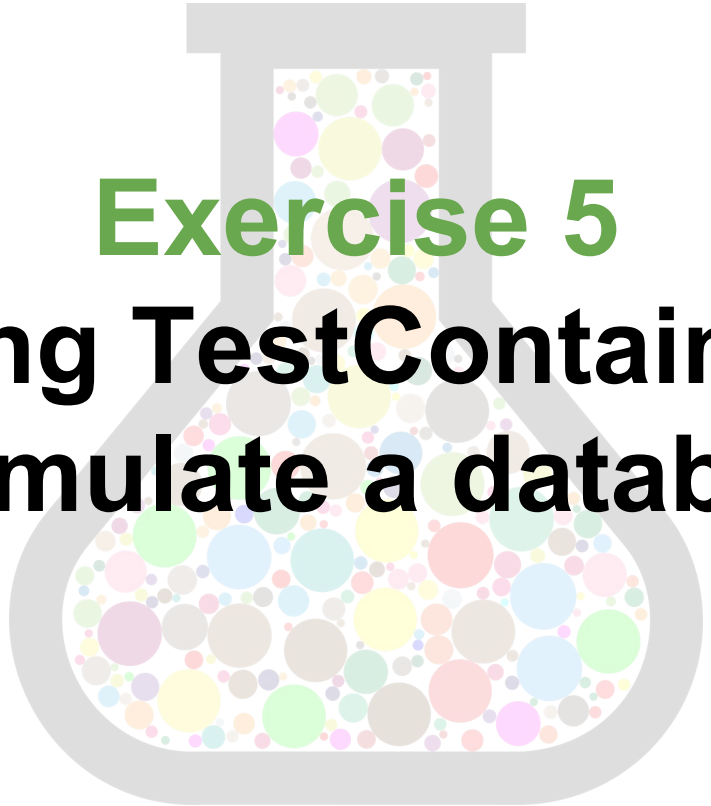
Bintray JCenter as proxy for Maven Central

Creating a database container

```
import org.testcontainers.containers.PostgreSQLContainer;
import org.testcontainers.junit.jupiter. Container;
import org.testcontainers.junit.jupiter. Testcontainers;

@Testcontainers
public class DatabaseIntegrationTest {

    @Container
    public static PostgreSQLContainer container =
        new PostgreSQLContainer("postgres:9.6.10-alpine")
            .withUsername("username")
            .withPassword("pwd")
            .withDatabaseName("todo");
}
```

Accessing runtime container information

```
static class Initializer implements↵
    ApplicationContextInitializer<ConfigurableApplicationContext> {
    @Override
    public void initialize(ConfigurableApplicationContext↵
        configurableApplicationContext) {
        TestPropertyValues.of(
                "spring.datasource.url=" + container.getJdbcUrl(),
                "spring.datasource.username=" + container.getUsername(),
                "spring.datasource.password=" + container.getPassword(),
                "spring.datasource.driver-class-name=org.postgresql.Driver",
                "spring.jpa.generate-ddl=true"
        ).applyTo(configurableApplicationContext.getEnvironment());
    }
}
```



Exercise 5

Using TestContainers to emulate a database

Creating multiple containers

```
import org.testcontainers.containers.DockerComposeContainer;
import org.testcontainers.junit.jupiter.Container;
import org.testcontainers.junit.jupiter.Testcontainers;

import java.io.File;

@Testcontainers
public class DockerComposeIntegrationTest {

   private final static File PROJECT_DIR = new File(System.getProperty("project.dir"));
   private final static String POSTGRES_SERVICE_NAME = "database_1";
   private final static int POSTGRES_SERVICE_PORT = 5432;

   @Container
   public static DockerComposeContainer environment = createComposeContainer();

   private static DockerComposeContainer createComposeContainer() {
       return new DockerComposeContainer(new File(PROJECT_DIR,
               "src/test/resources/compose-test.yml"))
               .withExposedService(POSTGRES_SERVICE_NAME, POSTGRES_SERVICE_PORT);
   }
}
```

Sample Docker Compose file

compose-test.yml

```
database:
  image: "postgres:9.6.10-alpine"
  environment:
    - POSTGRES_USER=postgres
    - POSTGRES_PASSWORD=postgres
    - POSTGRES_DB=todo
elasticsearch:
  image: "elasticsearch"
```

Doesn’t follow the Compose spec 100%

Accessing runtime container information

```
private static String getPostgresServiceUrl() {
    String postgresHost =↲
        environment.getServiceHost(POSTGRES_SERVICE_NAME,↲
                                   POSTGRES_SERVICE_PORT);
    Integer postgresPort =↲
        environment.getServicePort(POSTGRES_SERVICE_NAME,↲
                                   POSTGRES_SERVICE_PORT);
    StringBuilder postgresServiceUrl =  new StringBuilder();
    postgresServiceUrl.append( "jdbc:postgresql://");
    postgresServiceUrl.append(postgresHost);
    postgresServiceUrl.append( ":");
    postgresServiceUrl.append(postgresPort);
    postgresServiceUrl.append( "/todo");
    return postgresServiceUrl.toString();
}
```

Exercise 6

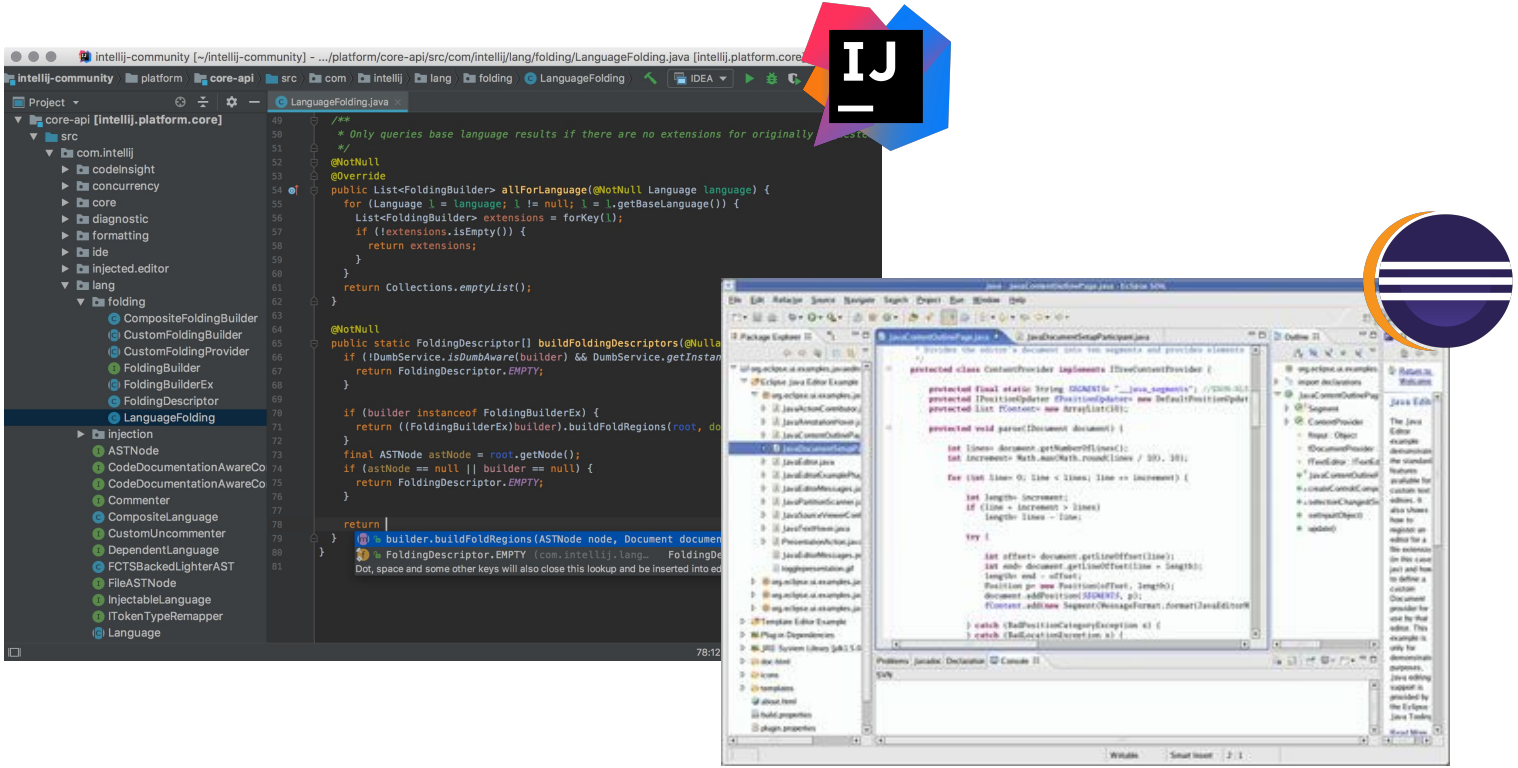
Using Docker Compose with TestContainers

Please ask questions!


Break
We'll continue
in 5 mins

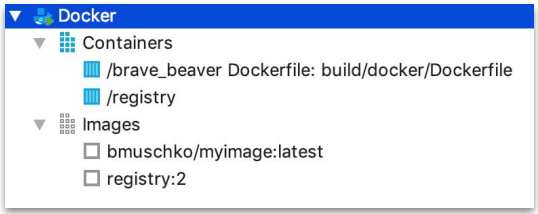
Tooling integration 4

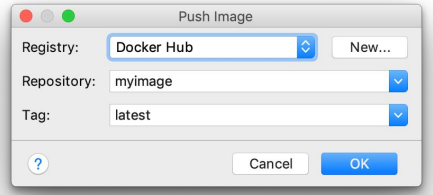
IDE support for Docker



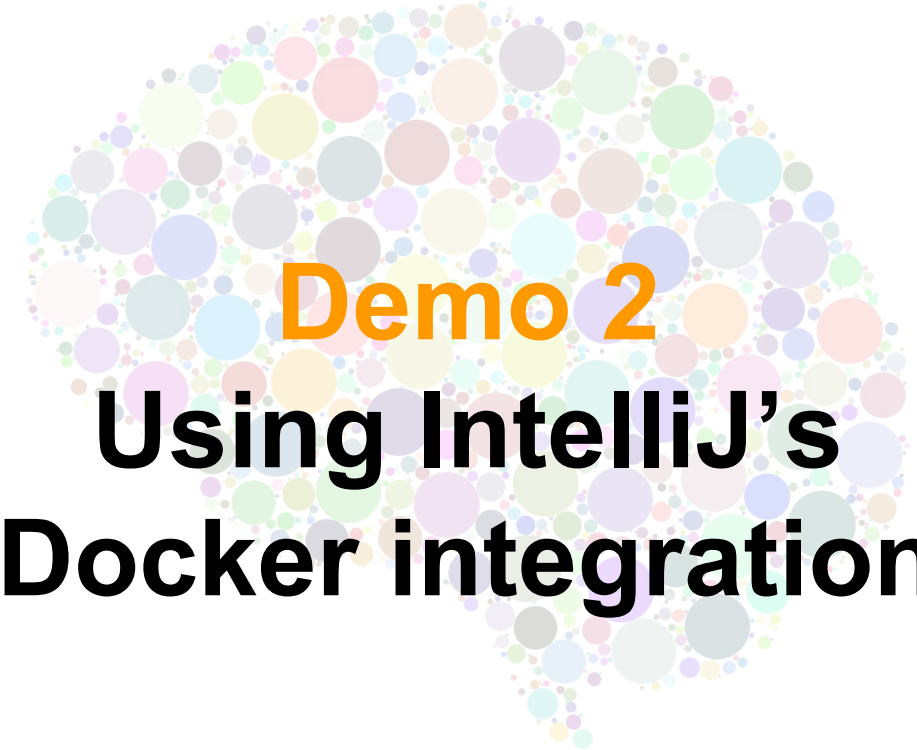
Docker in IntelliJ

- ➔ Browse images and containers
- ➔ Create images from Dockerfile
- ➔ Create a container from an image
- ➔ Push an image to a registry





https://plugins.jetbrains.com/plugin/7724-docker-integration

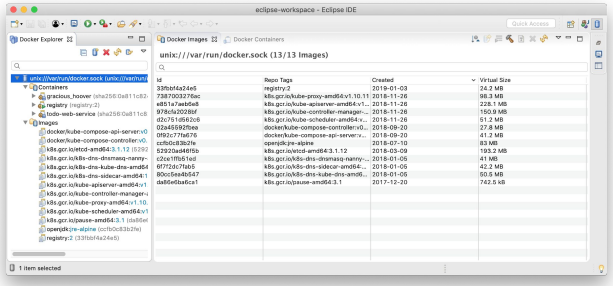


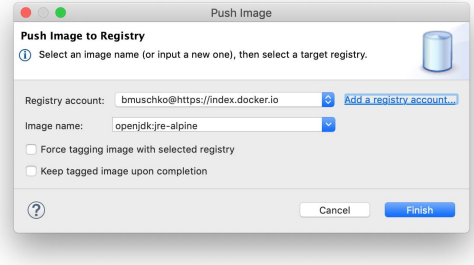
Demo 2

Using IntelliJ's Docker integration

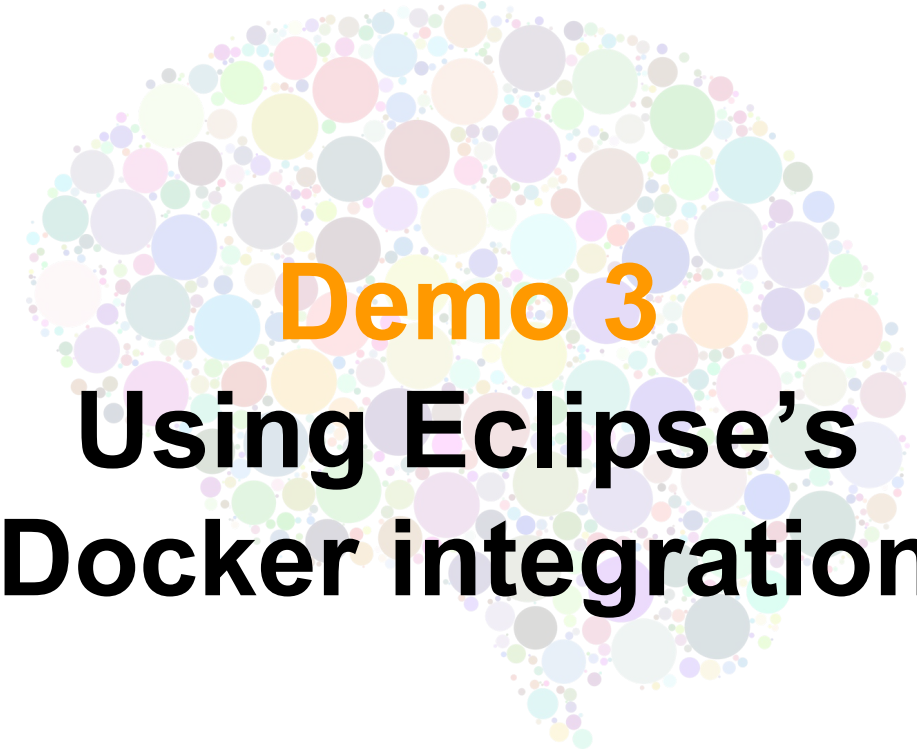
Docker in Eclipse

- ➔ Base capabilities as IntelliJ
- ➔ Loading and saving an image
- ➔ Compatible with Oxygen & Neon
- ➔ Avoid drag & drop installation





https://marketplace.eclipse.org/content/eclipse-docker-tooling



Demo 3

Using Eclipse's Docker integration

Exercise 7

Running a container from IntelliJ IDEA

Fine-grained build tool support for Docker


```
1. bmuschko@ascent: ~/dev/projects/todo-web-service-exercise (zsh)
~/dev/projects/todo-web-service-exercise  master  ./mvnw clean install                1 ↵  9246  14:28:55
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------< com.bmuschko:todo-web-service-exercise >----------------
[INFO] Building todo-web-service-exercise 1.0.0
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ todo-web-service-exercise ---
[INFO] Deleting /Users/bmuschko/dev/projects/todo-web-service-exercise/target
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ todo-web-service-exercise ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ todo-web-service-exercise ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 5 source files to /Users/bmuschko/dev/projects/todo-web-service-exercise/
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:testResources (default-testResources) @ todo-web-s
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /Users/bmuschko/dev/projects/todo-web-service-e
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:testCompile (default-testCompile) @ todo-web-servic
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.22.1:test (default-test) @ todo-web-service-exercise ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:3.1.1:jar (default-jar) @ todo-web-service-exercise ---
[INFO] Building jar: /Users/bmuschko/dev/projects/todo-web-service-exercise/target/todo-we
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ todo-web-service-exercis
Downloading from central: https://repo.maven.apache.org/maven2/commons-codec/commons-codec
Downloaded from central: https://repo.maven.apache.org/maven2/commons-codec/commons-codec/
5 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/mav
s-0.4.pom
```

```
1. bmuschko@ascent: ~/dev/projects/todo-web-service-exercise (zsh)
~/dev/projects/todo-web-service-exercise  master  ./gradlew clean build --console=verbose   ✔  9244  14:27:16
> Task :clean
> Task :compileJava
> Task :processResources
> Task :classes
> Task :bootJar
> Task :jar SKIPPED
> Task :assemble
> Task :compileTestJava NO-SOURCE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test NO-SOURCE
> Task :check UP-TO-DATE
> Task :build

BUILD SUCCESSFUL in 1s
4 actionable tasks: 4 executed
~/dev/projects/todo-web-service-exercise  master                                    ✔  9245  14:27:49
```

Driving Docker from Maven

- ➔ Create Dockerfile, build & push image
- ➔ Starting/stopping containers for testing
- ➔ Support for Docker Compose

https://github.com/fabric8io/docker-maven-plugin

Configuring the plugin

pom.xml

```
<plugin>
    <groupId>io.fabric8</groupId>
    <artifactId>docker-maven-plugin</artifactId>
    <version>0.28.0</version>
    <configuration>
        ...
        <images>
            <image>
                ...
            </image>
            ...
        </images>
    </configuration>
</plugin>
```

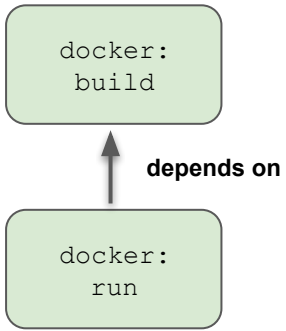
Building an image, running a container

pom.xml

```
<plugin>
    ...
    <configuration>
        <images>
            <image>
                <name>hello/sub/project/java:${project.version}</name>
                <alias>hello-world</alias>
                <build>
                    <from>openjdk:latest</from>
                    <assembly>
                        <descriptorRef>artifact</descriptorRef>
                    </assembly>
                    <cmd>java -jar maven/${project.name}-${project.version}.jar</cmd>
                </build>
                <run>
                    <wait>
                        <log>Hello World!</log>
                    </wait>
                </run>
            </image>
        </images>
    </configuration>
</plugin>
```

Task execution from the command line

```
$ mvn package docker:build docker:run
...
[INFO]
[INFO] --- maven-jar-plugin:3.0.2:jar (default-jar) @ dmp-sample-helloworld ---
[INFO]
[INFO] --- docker-maven-plugin:0.28.0:build (default-cli) @ dmp-sample-helloworld ---
[INFO] Copying files to
/Users/bmuschko/dev/projects/docker-maven-plugin/samples/helloworld/target/docker/hello/sub/p
roject/java/0.28-SNAPSHOT/build/maven
[INFO] Building tar:
/Users/bmuschko/dev/projects/docker-maven-plugin/samples/helloworld/target/docker/hello/sub/p
roject/java/0.28-SNAPSHOT/tmp/docker-build.tar
[INFO] DOCKER> [hello/sub/project/java:0.28-SNAPSHOT] "hello-world": Created docker-build.tar
in 91 milliseconds
[INFO] DOCKER> [hello/sub/project/java:0.28-SNAPSHOT] "hello-world": Built image sha256:611f9
[INFO]
[INFO] --- docker-maven-plugin:0.28.0:run (default-cli) @ dmp-sample-helloworld ---
[INFO] DOCKER> [hello/sub/project/java:0.28-SNAPSHOT] "hello-world": Start container
78f25f507d9a
[INFO] DOCKER> Pattern 'Hello World!' matched for container 78f25f507d9a
hello-world> Hello World!
[INFO] DOCKER> [hello/sub/project/java:0.28-SNAPSHOT] "hello-world": Waited on log out 'Hello
World!' 687 ms
```



Using Docker Compose

pom.xml

```
<image>
    <alias>webapp</alias>
    <name>fabric8/compose-demo:latest</name>

    <external>
        <type>compose</type>
        <basedir>src/main/docker</basedir>
        <composeFile>docker-compose.yml</composeFile>
    </external>

    <build>
        <assembly>....</assembly>
    </build>
    <run>...</run>
    <watch>...</watch>
</image>
```

Driving Docker from Gradle

- ➔ Models most Docker operations as custom tasks
- ➔ Communicates via Docker Engine RESTful API
- ➔ Convention plugins for Java and Spring Boot apps
- ➔ Compatible with Groovy and Kotlin DSL

https://github.com/bmuschko/gradle-docker-plugin

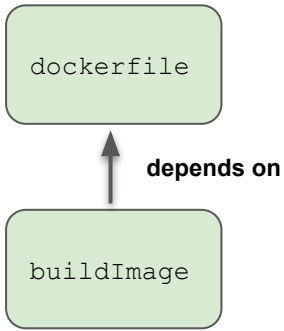
Building a Docker image with custom tasks

build.gradle

```
plugins {
    id 'com.bmuschko.docker-remote-api' version '4.6.2'
}

import com.bmuschko.gradle.docker.tasks.image.Dockerfile
import com.bmuschko.gradle.docker.tasks.image.DockerBuildImage

task dockerfile(type: Dockerfile) {
    from 'ubuntu:12.04'
    label(['maintainer': 'John Doe'])
}

task buildImage(type: DockerBuildImage) {
    dependsOn dockerfile
    tags.add('jdoe/myimage:latest')
}
```

Task execution from the command line

```
$ gradle buildImage --console=verbose
> Task :dockerfile
Step 1/2 : FROM ubuntu:12.04

---> 5b117edd0b76
Step 2/2 : LABEL maintainer="John Doe"

---> Using cache
---> b3edd0b8fac5
Successfully built b3edd0b8fac5
Successfully tagged jdoe/myimage:latest

> Task :buildImage
Building image using context
'/Users/bmuschko/dev/projects/gradle/build/docker'.
Using tags 'jdoe/myimage:latest' for image.
Created image with ID 'b3edd0b8fac5'.

BUILD SUCCESSFUL in 8s
2 actionable tasks: 1 executed, 1 up-to-date
```



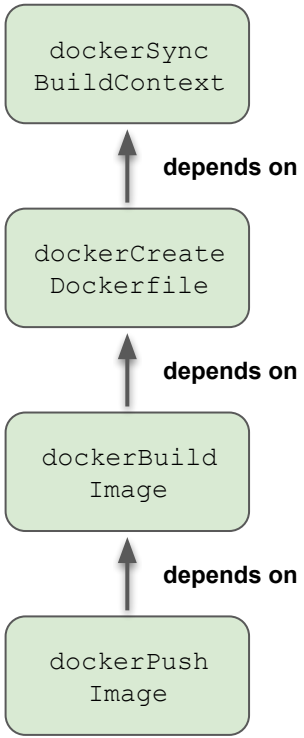
Convention support for Spring Boot apps

build.gradle

```
plugins {
    id 'war'
    id 'org.springframework.boot' version '2.0.3.RELEASE'
    id 'io.spring.dependency-management' version '1.0.5.RELEASE'
    id 'com.bmuschko.docker-spring-boot-application' version '4.6.2'
}

repositories {
    jcenter()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    providedRuntime 'org.springframework.boot:spring-boot-starter-tomcat'
    providedRuntime 'org.apache.tomcat.embed:tomcat-embed-jasper'
}
```

Task execution from the command line

```
$ gradle dockerPushImage --console=verbose
> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes
> Task :dockerSyncBuildContext
> Task :dockerCreateDockerfile
...
Successfully built d677dcfab7c4
Successfully tagged bmuschko/springbootapp:latest

> Task :dockerBuildImage
Created image with ID 'd677dcfab7c4'.

> Task :dockerPushImage
Pushing image with name 'bmuschko/springbootapp'.

BUILD SUCCESSFUL in 26s
6 actionable tasks: 6 executed
```



Using Docker Compose from Gradle

- ➔ There’s no one-stop solution
- ➔ Offers all Docker Compose operations
- ➔ Helpful for local development and integ. testing

https://github.com/avast/gradle-docker-compose-plugin

Using the Avast Docker Compose plugin

build.gradle

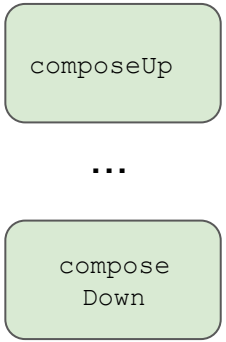
```
plugins {
    id 'com.avast.gradle.docker-compose' version '0.9.1'
}

dockerCompose {
    useComposeFiles = ['docker-compose.yml']
}
```

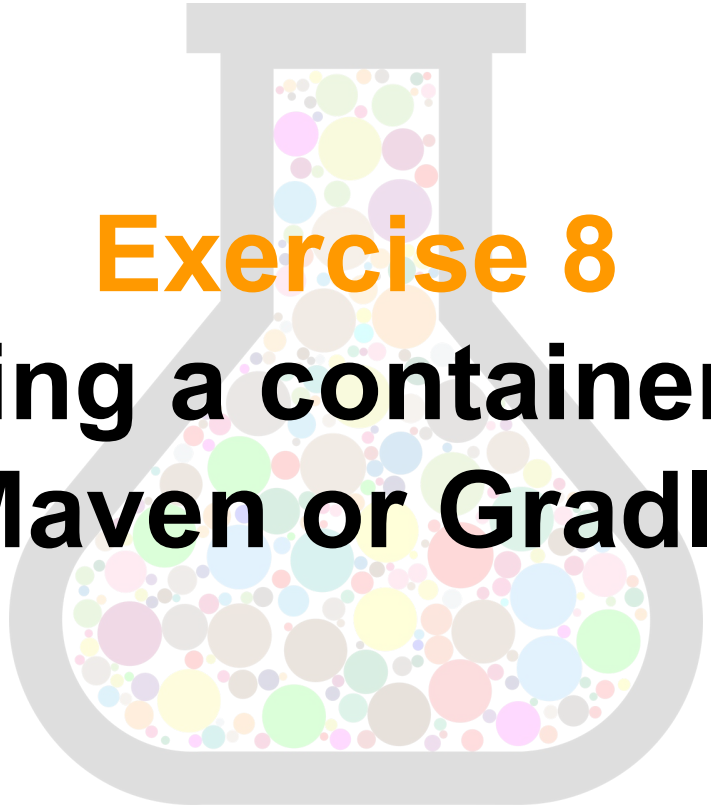
```
plugins {
    id 'java'
}

dockerCompose {
    isRequiredBy(project.tasks.test)
    exposeAsSystemProperties(project.tasks.test)
}
```

Hooking into test execution

Task execution from the command line

```
$ ./gradlew composeUp
> Task :composeUp
redis uses an image, skipping
Building web
Creating network
"dockercomposeintegrationtesting_counter-net" with the
default driver
Creating volume
"dockercomposeintegrationtesting_counter-vol" with default
driver
Creating dockercomposeintegrationtesting_redis_1 ... done
Creating dockercomposeintegrationtesting_web_1 ... done
Will use localhost as host of redis
Will use localhost as host of web
Waiting for redis_1 to become healthy (it's starting)
redis_1 health state reported as 'healthy' - continuing...
Waiting for web_1 to become healthy (it's starting)
web_1 health state reported as 'healthy' - continuing...
Probing TCP socket on localhost:5000 of service 'web_1'
TCP socket on localhost:5000 of service 'web_1' is ready

BUILD SUCCESSFUL in 1m 48s
1 actionable task: 1 executed
```

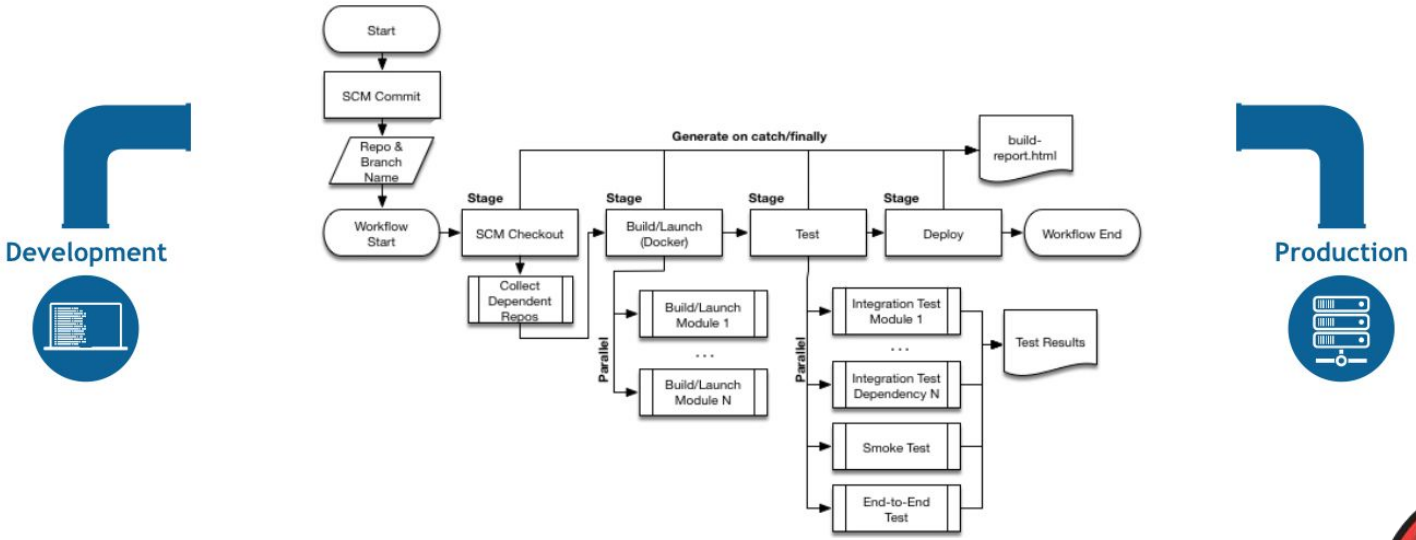




Exercise 8

Running a container from Maven or Gradle

CI/CD pipelines involving Docker



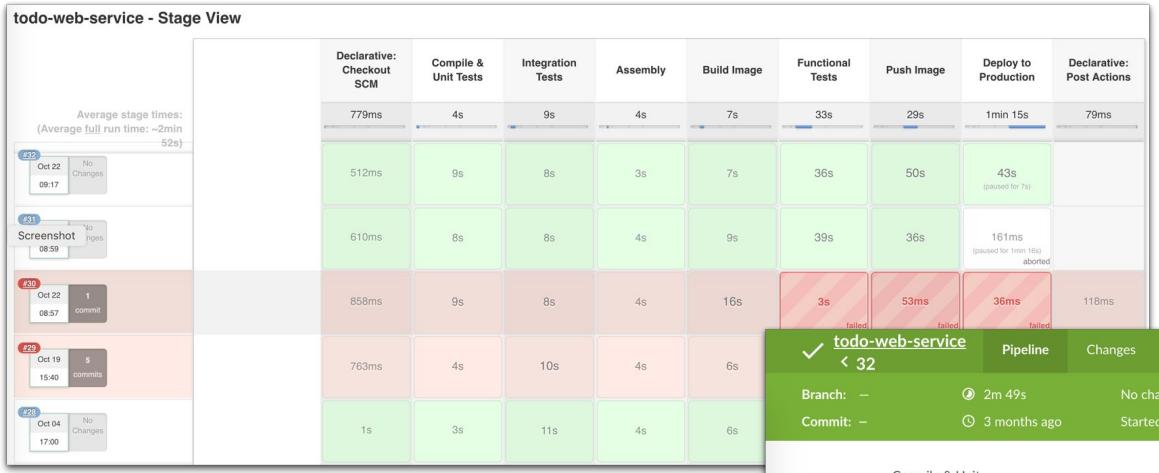
Source: https://jenkins.io/doc/book/pipeline/

Issues with manual job creation

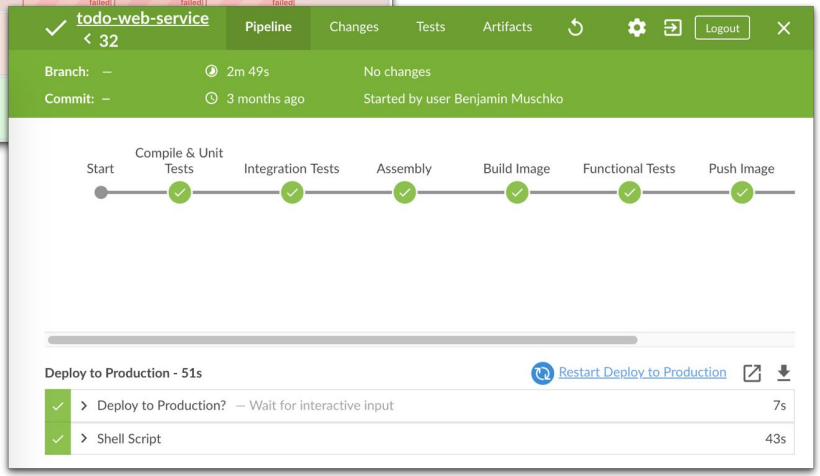
Jenkins 2 pipelines

- ➔ Configuration as code
- ➔ Groovy as DSL in Jenkinsfile
- ➔ Declarative pipeline definition
- ➔ Syntax documentation

Pipeline visualization



Standard

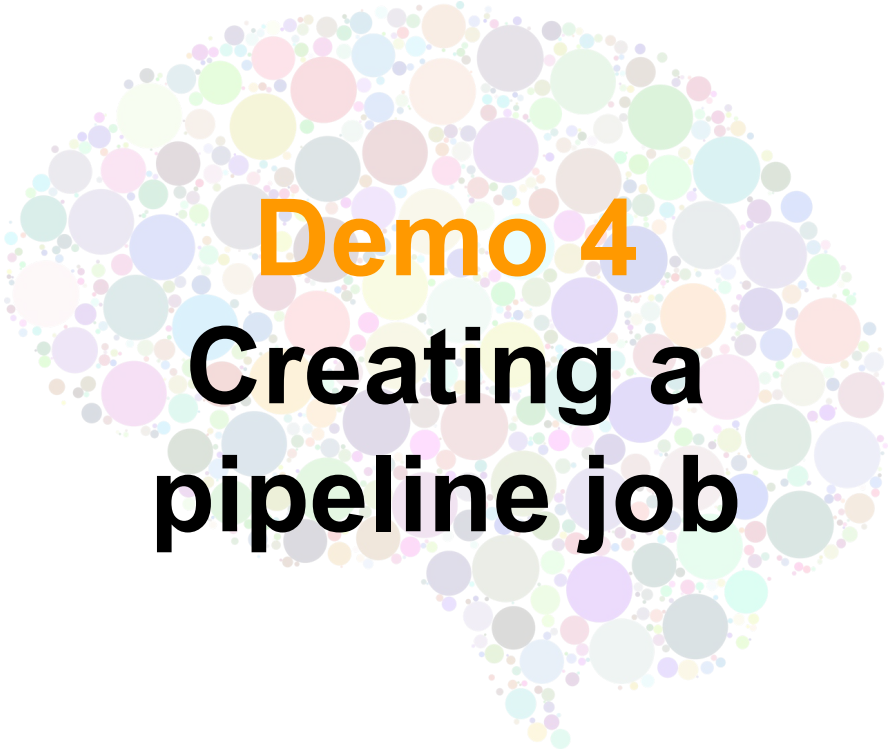


Blue Ocean

Required Jenkins plugins

➔ Suite of pipeline plugin (standard)

➔ Blue Ocean plugin (optional)



Demo 4

Creating a pipeline job

Syntax element: Pipeline

```
pipelines {
    ...
}
```

Models CI pipeline including stages for building, testing, releasing application

Syntax element: Agent

```
pipelines {
    agent any
}
```

```
pipelines {
    agent {
        docker {
            image 'maven3-alpine'
            label
'my-defined-label'
            arg '-v /tmp:tmp'
        }
    }
}
```

“where does the pipeline execute”

Syntax element: Stage

```
pipelines {
    agent any {
        stages {
            stage('Build') {
                steps {
                    ...
                }
            }
        }
    }
}
```

Subset of tasks, visualized in UI

Syntax element: Steps

```
pipelines {
    agent any {
        stages {
            stage('Build') {
                steps {
                    ...
                }
            }
        }
    }
}
```

Single task, “what to execute”

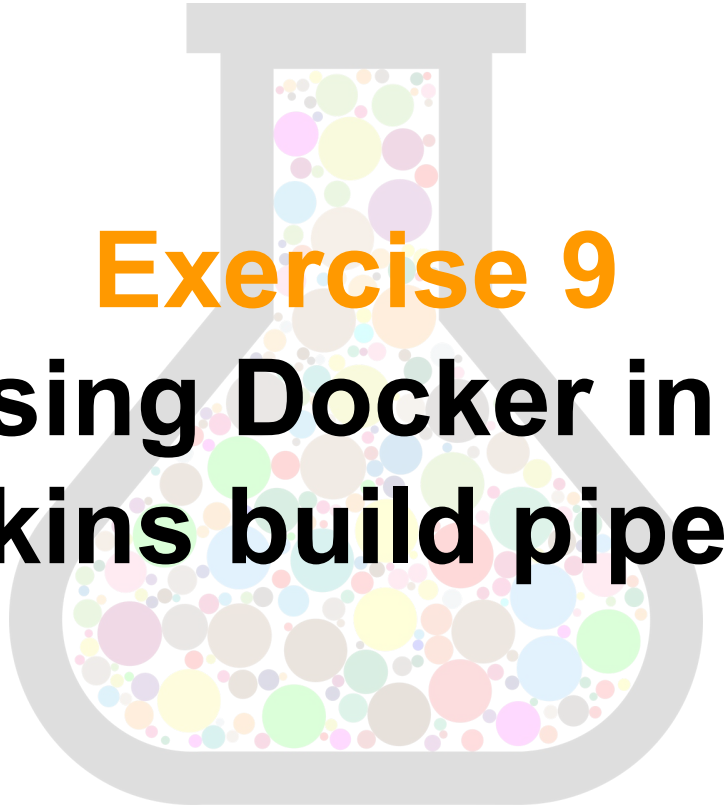
There’s more...

- ➔ Using pre-defined tools
- ➔ Using environment variables and credentials
- ➔ Post-build notifications
- ➔ Requesting user input
- ➔ Conditional execution of stages

Example step for compiling code

```
pipelines {
    agent any {
        stages {
            stage('Compile') {
                steps {
                    sh './mvnw compile'
                }
            }
            stage('...') {
                ...
            }
        }
    }
}
```

Shell Command



Exercise 9

Using Docker in a Jenkins build pipeline

Q & A

Please ask questions!

Conclusion

Let’s summarize what we’ve learned