Powering a build pipeline with Docker and Jenkins

Benjamin Muschko

VOXXED DAYS BANFF

About the speaker

bmuschko

bmuschko

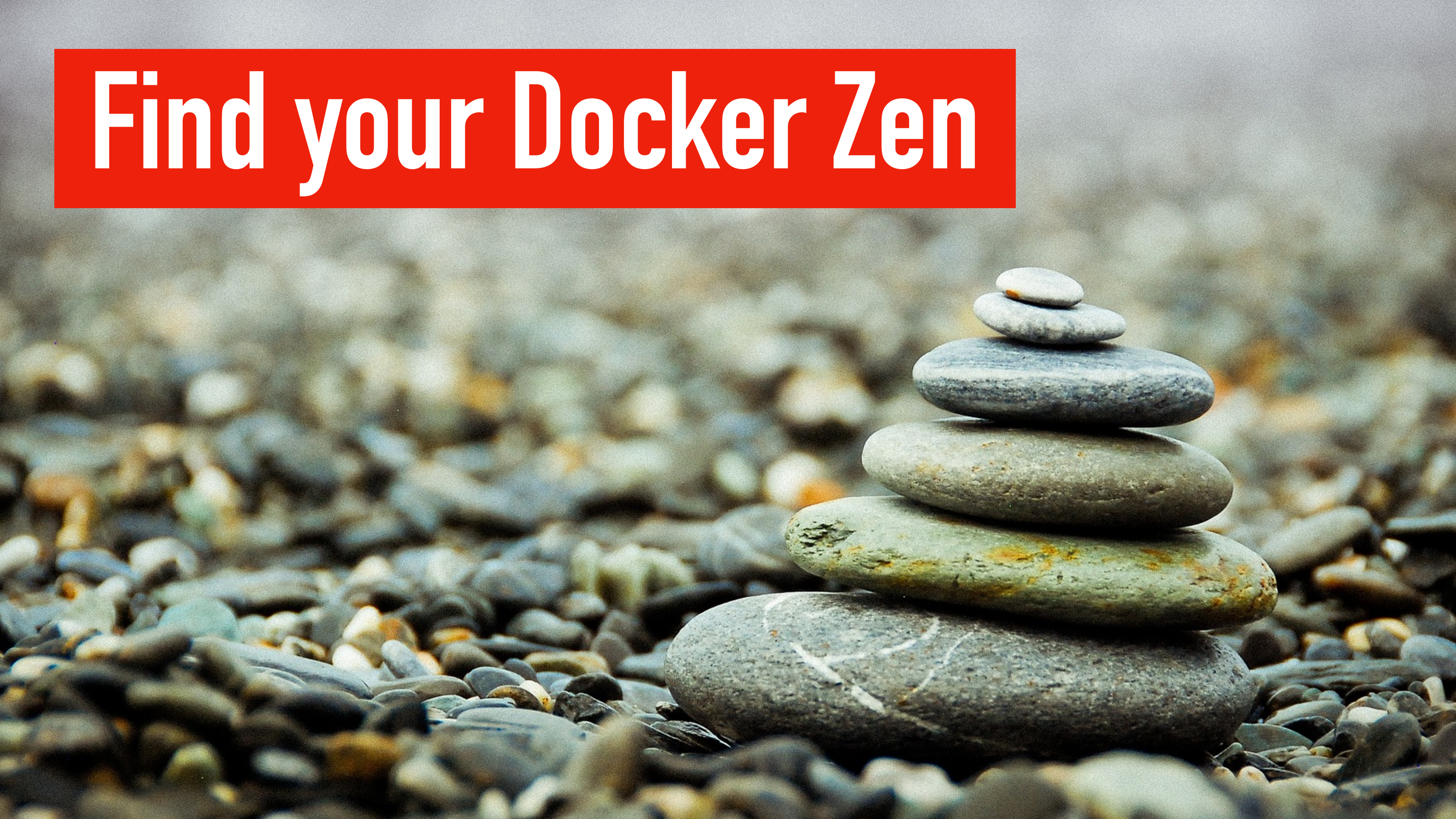
bmuschko.com

AUTOMATED ASCENT

automatedascent.com


As a Java developer,
I want to use Docker but...

Find your Docker Zen



Agenda

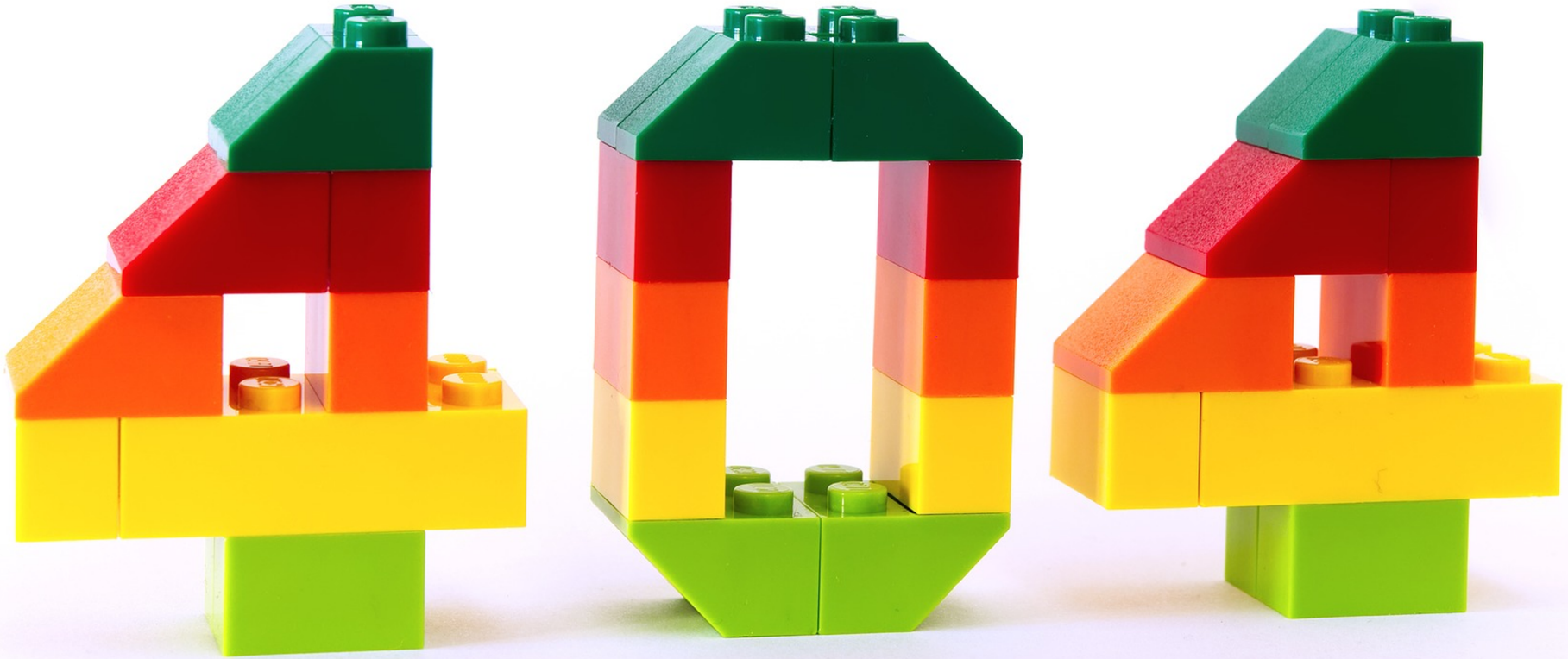
Typical Continuous Delivery uses cases

How can Docker help?

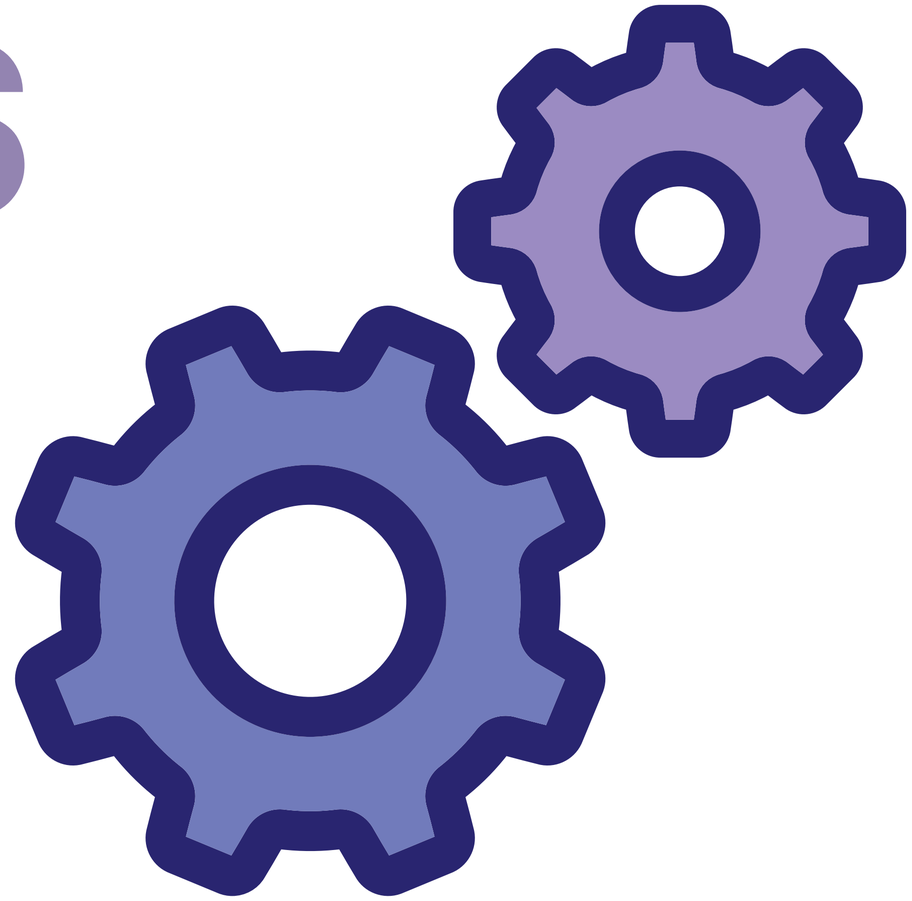
Show me some code!

The glue as code in Jenkins

What this talk is not about



Dev or Ops

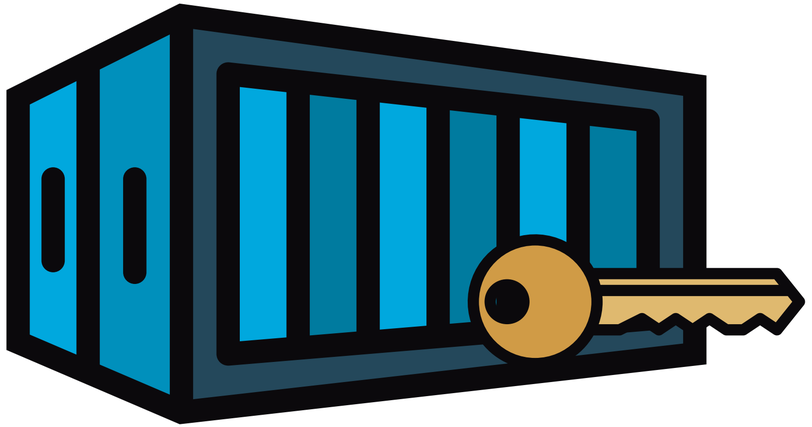


What are we talking about?

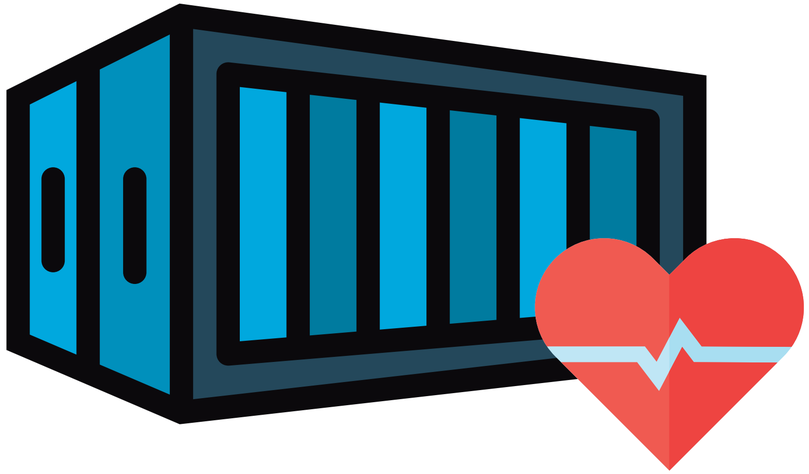
Typical challenges

Supporting different environments

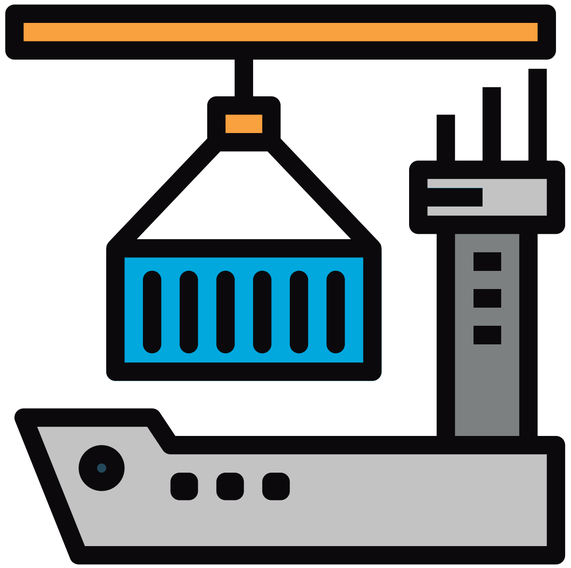
Storing and retrieving credentials



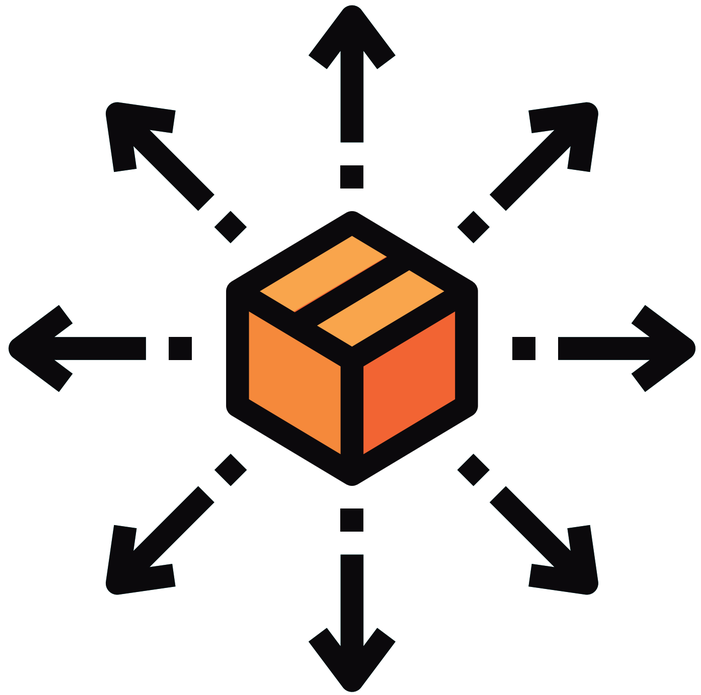
Ensuring healthy state for containers



Services follow individual release cycles



Rolling out services to nodes



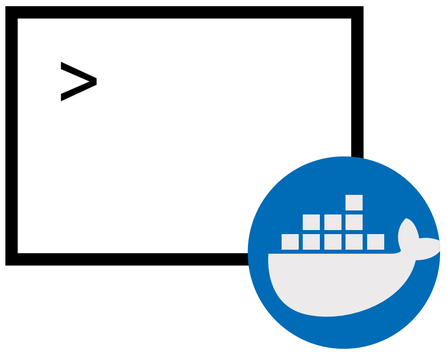
Docker's CLI is great

but do I need to operate it by hand?


Pick the right tool for the job

Other options

Docker CLI



Gradle

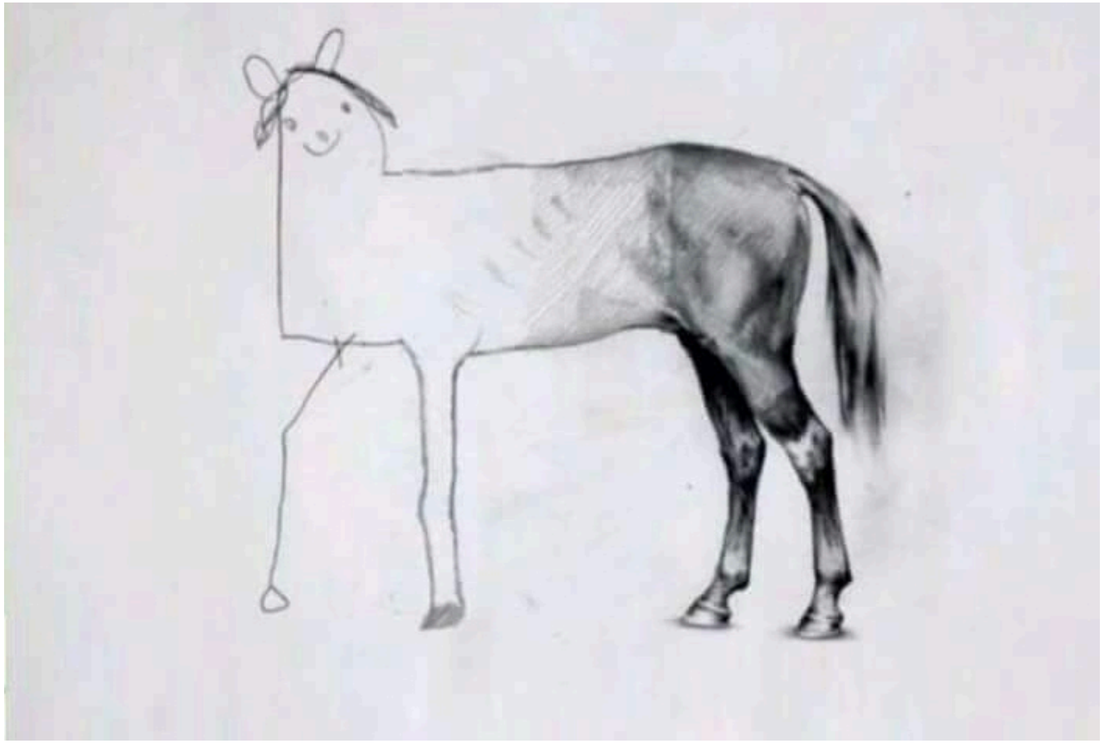
Jib

Containerize your Java application.

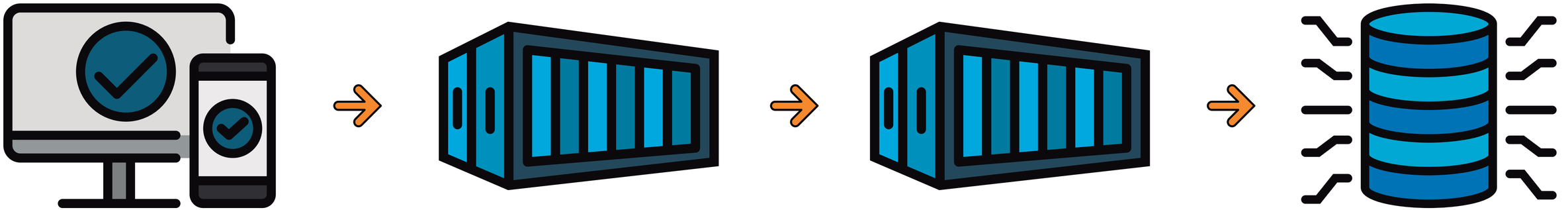
Maven™

TESTCONTAINERS

Show me something beyond Hello World!



Sample architecture whiteboard



Client

Web application

Spring Boot

Web service

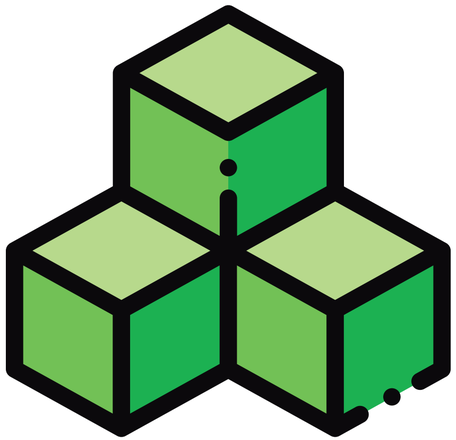
Spring Boot

Database

PostgreSQL

Demo Time

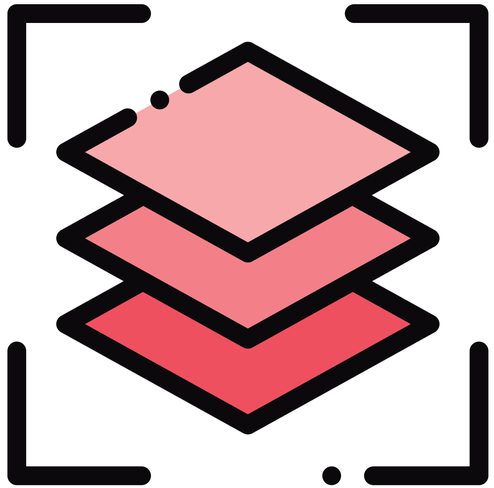
Dockerizing a Java application



Application modules

Bundled as JAR file

Dockerfile



Embedded in Docker image

Dockerfile for Spring Boot app

```
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
```

Building and running image

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

# Run built image in container
docker run -d -p 8080:8080 my-todo-web-service:1.0.0
```

Building the image

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '2.0.6.RELEASE'
    id 'com.bmuschko.docker-java-application' version '4.0.1'
}
```

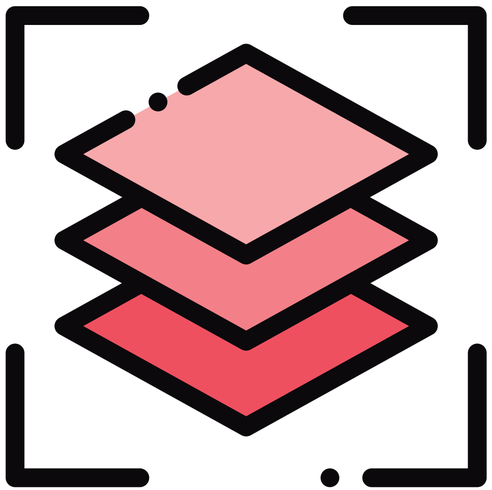
```
$ ./gradlew dockerBuildImage
```

Providing a health check

```
dockerCreateDockerfile {
    instruction 'HEALTHCHECK CMD wget --quiet --tries=1 \
    --spider http://localhost:8080/actuator/health || exit 1'
}
```

Demo Time

Pushing an image to a registry



Tagged Docker image

Docker registry

Pushing image

```
# Log into Docker Hub
docker login --username=bmuschko \
--email=benjamin.muschko@gmail.com

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

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

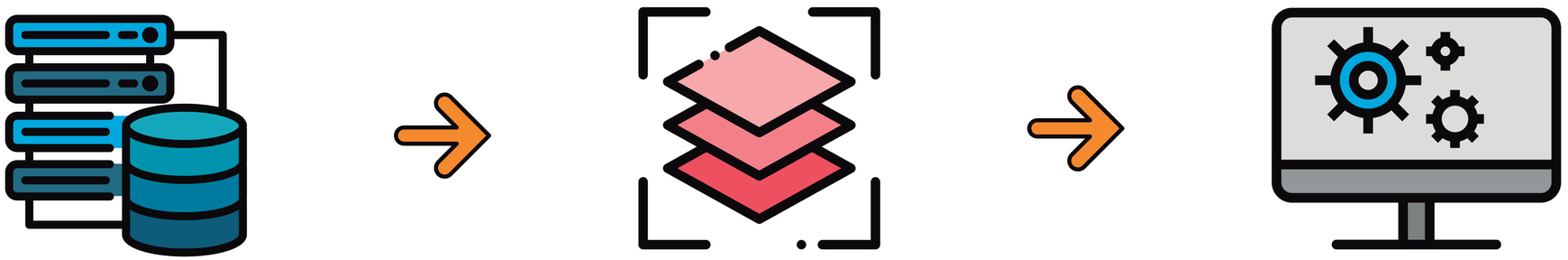
Pushing the image to Docker Hub

```
docker {
    registryCredentials {
        username = 'bmuschko'
        password = System.getenv('DOCKER_PASSWORD')
        email = 'benjamin.muschko@gmail.com'
    }
    springBootApplication {
        tag = 'bmuschko/todo-web-service'
    }
}
```

```
$ ./gradlew dockerPushImage
```

Demo Time

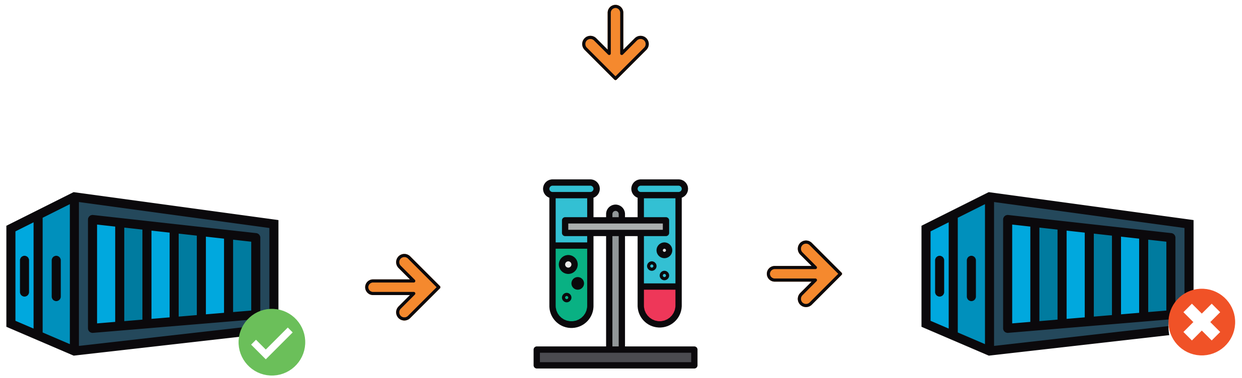
Image as fixture for testing



Docker registry

Docker image

Build machine



Pull image and start/stop container

```
# Pull image from Docker registry
docker pull bmuschko/todo-web-service:latest

# Start container
docker run -d -p 8080:8080 -name todo-web-service \
my-todo-web-service:latest

# Stop container
docker container stop todo-web-service

# Remove container
docker container rm todo-web-service
```

Using container for tests

```
task createContainer(type: DockerCreateContainer) {
    dependsOn dockerBuildImage
    targetImageId dockerBuildImage.imageId
    portBindings = ['8080:8080']
    autoRemove = true
}

task startContainer(type: DockerStartContainer) {
    dependsOn createContainer
    targetContainerId createContainer.containerId
}

task startAndWaitOnHealthyContainer(type: DockerWaitHealthyContainer) {
    dependsOn startContainer
    timeout = 60
    targetContainerId createContainer.containerId
}

task stopContainer(type: DockerStopContainer) {
    targetContainerId createContainer.containerId
}

functionalTest {
    dependsOn startAndWaitOnHealthyContainer
    finalizedBy stopContainer
}
```

```
$ ./gradlew functionalTest
```

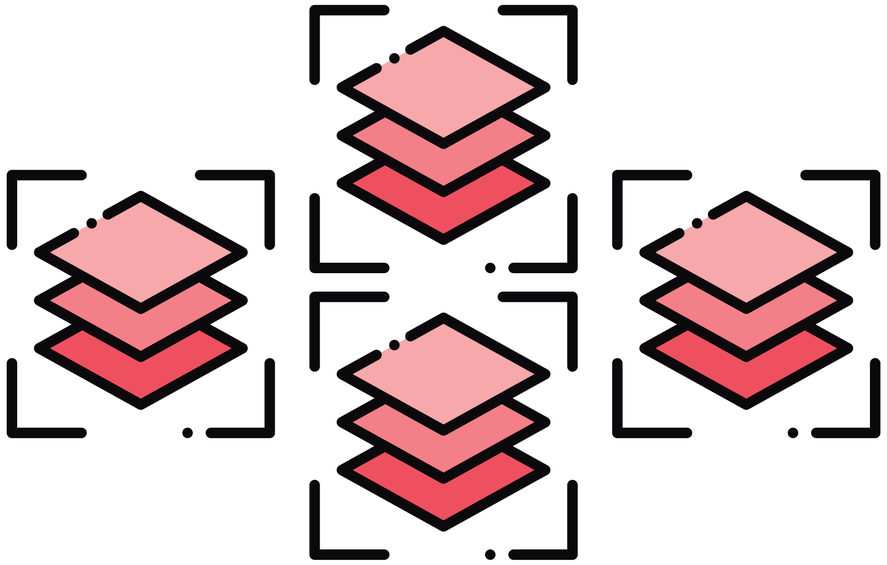
Functional test with JUnit 5

```
public class ToDoWebServiceFunctionalTest {
    @Test
    @DisplayName("can retrieve all items before and after inserting new ones")
    void retrieveAllItems() {
        String allItems = getAllItems();
        assertEquals("[]", allItems);

        ToDoItem toDoItem = new ToDoItem();
        toDoItem.setName("Buy milk");
        toDoItem.setCompleted(false);
        insertItem(toDoItem);

        allItems = getAllItems();
        assertEquals("[{\"id\":1,\"name\":\"Buy milk\",\"completed\":false}]", allItems);
    }
}
```

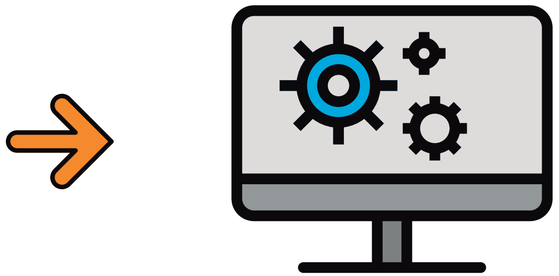
HTTP calls to running container

Demo Time

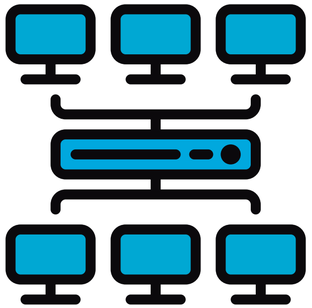
Running application stacks



Multiple Docker images



Build machine



Multi-container application

Docker Compose definition

```yaml
version: "3.7"
services:
  web-service:
    ...
  database:
    ...
networks:
  todo-net:

volumes:
  todo-vol:
```

```yaml
web-service:
    image: "bmuschko/todo-web-service:latest"
    environment:
      - SPRING_PROFILES_ACTIVE=dev
    ports:
      - 8080:8080
    networks:
      - todo-net
    volumes:
      - type: volume
        source: todo-vol
        target: /code
    depends_on:
      - database
    healthcheck:
      test: wget --quiet --tries=1 --spider \
            http://localhost:8080/actuator/health || exit 1
      interval: 10s
      timeout: 5s
      retries: 3
```

Running a multi-container app

```
# Start composed apps
docker-compose up

# Stop composed apps
docker-compose down
```

Using container for tests

```
plugins {
    id 'com.avast.gradle.docker-compose' version '0.8.8'
}

dockerCompose {
    useComposeFiles = ['docker-compose.yml']
    isRequiredBy(project.tasks.integrationTest)
    exposeAsSystemProperties(project.tasks.integrationTest)
}
```

```
$ ./gradlew integrationTest
```

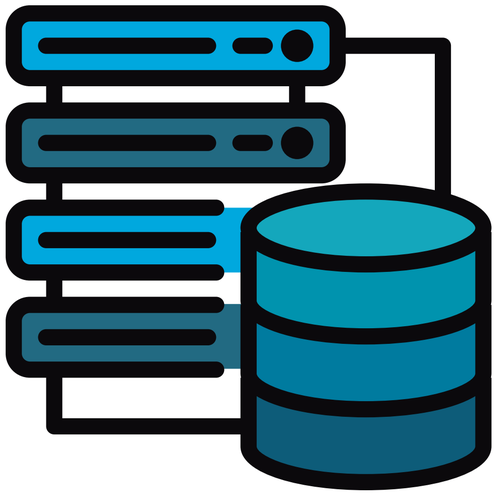
Integration test with JUnit 5

```
@ExtendWith(SpringExtension.class)
@SpringBootTest
public class ToDoServiceImplIntegrationTest {
    @Autowired
     private ToDoService toDoService;

     @Test
     public void canCreateNewItemAndRetrieveIt() {
         ToDoItem newItem = newItem("Buy milk");
         assertNull(newItem.getId());
         toDoService.save(newItem);
         assertNotNull(newItem.getId());
         ToDoItem retrievedItem = toDoService.findOne(newItem.getId());
         assertEquals(newItem, retrievedItem);
     }
}
```

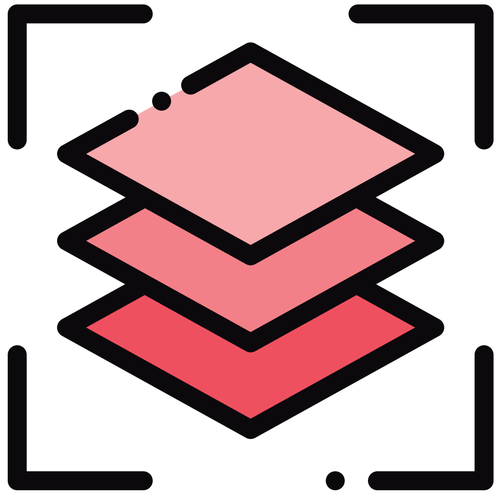
Calls web service endpoint

Demo Time

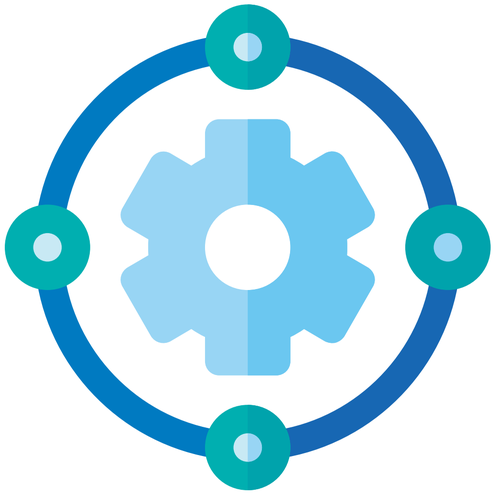
Deploying services to a Caas



Docker registry



Docker image



Docker Swarm

Rolling out services to Swarm

```
# Create a new service
docker service create --name todo-web-service \
--publish 8080:8080 --replicas 5 --secret db-password \
--env SPRING_PROFILES_ACTIVE=prod \
bmuschko/todo-web-service:latest

# Update an existing service from Swarm Leader
docker service update --image \
bmuschko/todo-web-service:latest todo-web-service
```

Creating a Docker secret

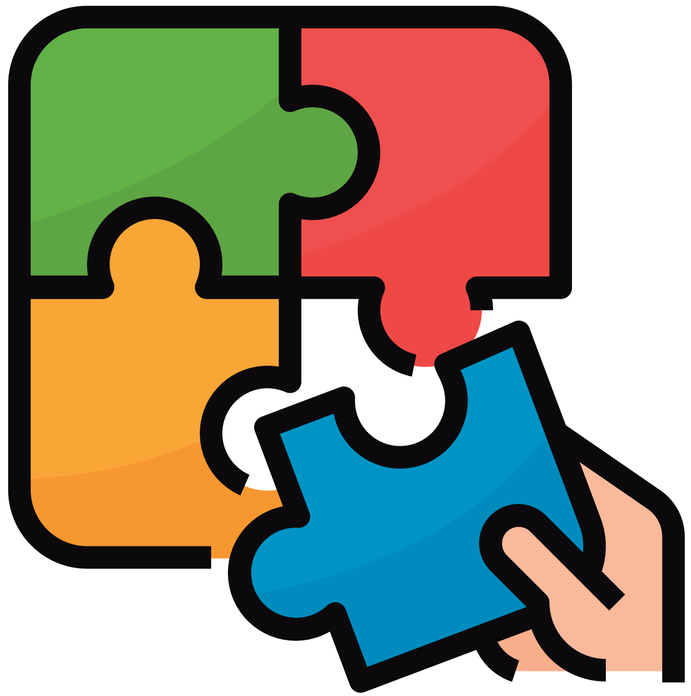
```
# Login into Swarm Leader
ssh swarm1

# Create a password
printf "prodpwd" | docker secret create db-password -
```

Gradle support for service management

Demo Time

Glueing together the pipeline

Jenkins plugins



Pipeline suite

Gradle

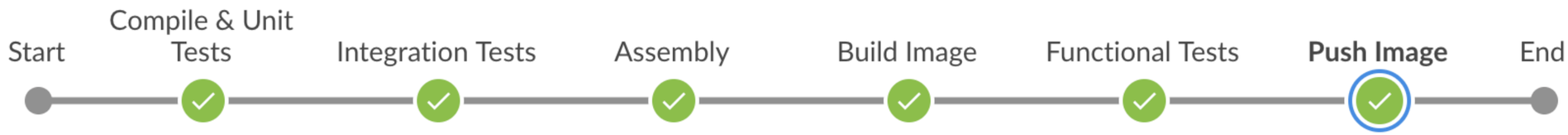
SSH Agent

Optional: Blue Ocean

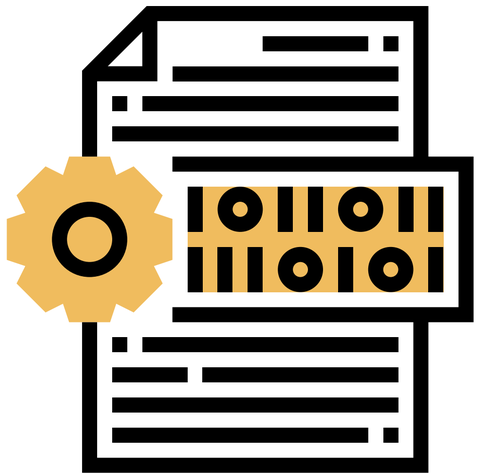
Standard Pipeline

Declarative: Checkout SCM	Compile & Unit Tests	Integration Tests	Assembly	Build Image	Functional Tests	Push Image
854ms	3s	8s	3s	6s	38s	35s
780ms	3s	8s	3s	5s	38s	38s
929ms	3s	8s	3s	6s	38s	32s

Blue Ocean Pipeline



Pipeline as code



Jenkinsfile

Jenkins Pipeline

Calling Gradle from stages

```
pipeline {
    stages {
        stage('Compile & Unit Tests') {
            steps {
                gradlew('clean', 'test')
            }
        }
    }

    ...
}

def gradlew(String... args) {
    sh "./gradlew ${args.join(' ')} -s"
}
```

Credentials

Scope: Global (Jenkins, nodes, items, all child items, etc)

Secret: ••

ID: DOCKER_PASSWORD

Description: DockerHub Password

Environment variables

Global properties

☑ Environment variables

List of variables

Name: DOCKER_EMAIL

Value: benjamin.muschko@gmail.com

Delete

Injecting credentials & env vars

```
stage('Push Image') {
    environment {
        DOCKER_USERNAME = "${env.DOCKER_USERNAME}"
        DOCKER_PASSWORD = credentials('DOCKER_PASSWORD')
        DOCKER_EMAIL = "${env.DOCKER_EMAIL}"
    }
    steps {
        gradlew('dockerPushImage')
    }
}
```

Manual deployment to Swarm

```
stage('Deploy to Production') {
    steps {
        timeout(time: 1, unit: 'DAYS') {
            input 'Deploy to Production?'
        }
        sshagent(credentials: ['ee8346e0-a000-4496-88aa-49977fd97154']) {
            sh "ssh -o StrictHostKeyChecking=no \
            ${env.DOCKER_SWARM_MANAGER_USERNAME}@${env.DOCKER_SWARM_MANAGER_IP} \
            docker service update --image bmuschko/todo-web-service:latest \
            todo-web-service"
        }
    }
}
```

Demo Time

Resources

 github.com/bmuschko/todo-web-service

 github.com/bmuschko/todo-web-app

 github.com/bmuschko/todo-docker-swarm

Thank you!

Please ask questions...