Mitschrift : Java Cloud Development 2024-12

Inhalt

[Tag 1 Setup 3](#_Toc185005198)

[Postgres Datenbank zum Verbinden 5](#_Toc185005199)

[Tag 1 5](#_Toc185005200)

[Tag 1 Ansible & Linux 5](#_Toc185005201)

[Tag 1 EC 2 Instanzen 6](#_Toc185005202)

[Ansible Installation um Postgres erweitern 7](#_Toc185005203)

[Postgres Datenbank schema anlegen 7](#_Toc185005204)

[Ansible 8](#_Toc185005205)

[erste Tabelle 8](#_Toc185005206)

[intellij connect db 9](#_Toc185005207)

[TAG 3 11](#_Toc185005208)

[Quarkus projekt anlegen 11](#_Toc185005209)

[Python Virtual Environment 12](#_Toc185005210)

[Quarkus Rest easy für REST services 12](#_Toc185005211)

[Quarkus logging 13](#_Toc185005212)

[Quarkus Server Logging Konfigurieren 13](#_Toc185005213)

[Quarkus DB anbindung 14](#_Toc185005214)

[SKIPT TESTs 14](#_Toc185005215)

[Quarks Datenbank Anbindung Konfigurieren 15](#_Toc185005216)

[TEST Repository Example 16](#_Toc185005217)

[OpenAPI Erweiterung 17](#_Toc185005218)

[Web Rest-Funktion 1.0.0 18](#_Toc185005219)

[Quarkus Test 20](#_Toc185005220)

[Rest test client checks 21](#_Toc185005221)

[Read all entities 21](#_Toc185005222)

[Flyway dependencies configrieren 22](#_Toc185005223)

[Tag 4 23](#_Toc185005224)

[HIbernate Optimistic Locking Konzépt 24](#_Toc185005225)

[Achtung bei JPA detached objekten und TX 25](#_Toc185005226)

[DTO Mapping Framworks 26](#_Toc185005227)

[Disable Junit 5 test 26](#_Toc185005228)

[OpenAPI Grundletende INFOS 27](#_Toc185005229)

[Generatoren 27](#_Toc185005230)

[Monitoring APIS 27](#_Toc185005231)

[Health MP Support 27](#_Toc185005232)

[app initaialize 29](#_Toc185005233)

[Docker Support 30](#_Toc185005234)

[Docker maven run 30](#_Toc185005235)

[Tag 5 : 31](#_Toc185005236)

[AWS Client 31](#_Toc185005237)

[Docker layer 31](#_Toc185005238)

[Download Link neu 31](#_Toc185005239)

[AWS Client Configure 32](#_Toc185005240)

[prüfen welcher user sind wir 32](#_Toc185005241)

[connect Docker mit AWS Login zur Amazon Container Registry 32](#_Toc185005242)

[Docker PUSH to ECR 33](#_Toc185005243)

[Images sind nun in der zentralen Cloud Registry verfügbar 34](#_Toc185005244)

[Container Environement Variablen Environment Variable vom der Task Definiton zu den Tasks = Containern übergeben 35](#_Toc185005245)

[Terraform basis variablen 36](#_Toc185005246)

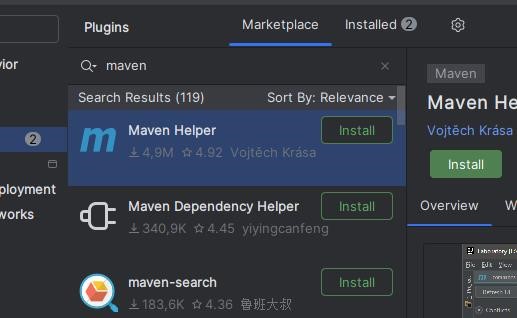
# Tag 1 Setup

Installation:

<https://www.jetbrains.com/idea/download/?section=windows>

Download IntelliJ IDEA – The Leading Java and Kotlin IDE

Download the latest version of IntelliJ IDEA for Windows, macOS or Linux.



Plugins zum Installieren

AWS Toolkit

Maven Helper Plugin

install via Marketplace .. linker plugin tab

downlload putty

<https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>

Download PuTTY: latest release (0.82)

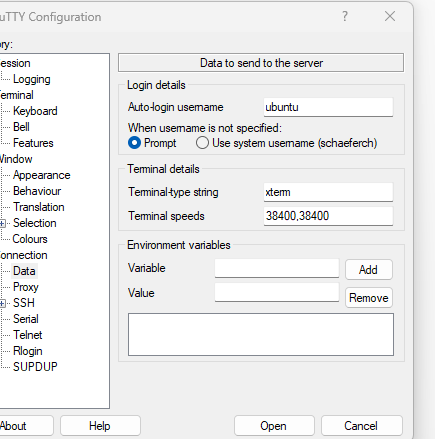
Login in AWS GUI :

<https://cgs-schaefer.signin.aws.amazon.com/console>

--user-name train-004 --password "YourSecurePassword123!"

Linux server :

Login User: ubuntu



sudo apt  install tree

### Postgres Datenbank zum Verbinden

DNS:  
wifi-postgres-rds-db.cts48ysrivsg.eu-central-1.rds.amazonaws.com

User  
postgres

Passwort:  
postgres

-------------------

1. sudo apt update
2. sudo apt install -y libpq-dev
3. cd
4. source myenv/bin/activate
5. cd ansible\_work
6. ansible-playbook list\_ec2\_instances.yml
7. mkdir group\_vars
8. erstellen varablen file
   1. vi group\_vars/all.yml
9. Varablen für die Postgres DB für das all.yml file
10. # PostgreSQL connection details  
    db\_host: "wifi-postgres-rds-db.cts48ysrivsg.eu-central-1.rds.amazonaws.com"         # Remote PostgreSQL server address  
    db\_port: 5432                  # PostgreSQL port  
    db\_user: "postgres"            # Superuser username  
    db\_password: "postgres"   # Superuser password
11. # Trainee-specific variables  
    trainee\_name: "train\_00"       # 001 bis 004 für jeden benutzer eine Datenbank soll erstellt werden  
    sql\_script\_path: "sql/create\_roles\_and\_permissions.sql"

# Tag 1

## Tag 1 Ansible & Linux

Benutzer kann beliebige Pakete nachinstallieren

sudo apt install whois

Prüfe .ssh/ files ob die public keys for das login vorhanden sind

Prüfe .bashrc für eigene bash settings

z.b. echo ..

AWS Tools:

Aws tool prüfung

aws s3 ls

AWS Command Line tool Doku

<https://aws.amazon.com/de/cli/>

Output json vormat

aws s3api list-buckets --output json

## Tag 1 EC 2 Instanzen

Ansbile arbeitet mit python und wir arbeiten mit einer lokalen python umgebung

Zum aktivieren im user home verzeichnis folgendes command ausführen

source myenv/bin/activate

Ansible command to list ec2 instances

ansible-playbook -i inventory list\_ec2\_instance\_simple.yml

------------------

AWS Dokumentation

<https://docs.aws.amazon.com/cli/latest/reference/#available-services>

List instances

aws ec2 describe-instances --query "Reservations[].Instances[].{Instance:InstanceId,KeyName:KeyName,PublicIpAddress:PublicIpAddress,stateName:State.Name}" --output table

Alles von den instanzen

aws ec2 describe-instances --output json

aws ec2 describe-instances --query "Reservations[].Instances[].{Name:Tags[?Key=='Name']|[0].Value,Instance:InstanceId,KeyName:KeyName,PublicIpAddress:PublicIpAddress,StateName:State.Name}" --output table

history | grep aws

Get User information .. if permission is allowed

aws iam get-user

Export profile to be used :

export AWS\_PROFILE=admin

## Ansible Installation um Postgres erweitern

1. Install postgres development tools

sudo apt update

sudo apt install -y libpq-dev

# Postgres Datenbank schema anlegen

Ansible script : create\_database.yml

create\_database.yml

Permissions und Roles

--------------------------------------------------  
-- Roles Creation  
CREATE ROLE "{{ trainee\_name }}\_rw"  
    NOSUPERUSER INHERIT NOCREATEDB NOCREATEROLE NOREPLICATION;

CREATE ROLE "{{ trainee\_name }}\_ro"  
    NOSUPERUSER INHERIT NOCREATEDB NOCREATEROLE NOREPLICATION;

CREATE ROLE "{{ trainee\_name }}\_user" LOGIN  
    NOSUPERUSER INHERIT NOCREATEDB NOCREATEROLE NOREPLICATION;

GRANT "{{ trainee\_name }}\_rw" TO "{{ trainee\_name }}\_user";

-- Set Passwords  
ALTER USER "{{ trainee\_name }}\_user" WITH PASSWORD '{{ trainee\_name }}\_user';

--------------------------------------------------  
-- Permissions  
GRANT CONNECT, TEMPORARY ON DATABASE "{{ trainee\_name }}\_simplechat" TO public;  
GRANT ALL ON DATABASE "{{ trainee\_name }}\_simplechat" TO postgres;  
GRANT CONNECT ON DATABASE "{{ trainee\_name }}\_simplechat" TO "{{ trainee\_name }}\_user";

-- Grant CREATE privilege to the user  
GRANT CREATE ON DATABASE "{{ trainee\_name }}\_simplechat" TO "{{ trainee\_name }}\_user";

## Ansible

call ansible script

ansible-playbook -i inventory create\_database.yml

 pip install psycopg2

pip install psycopg2

## erste Tabelle

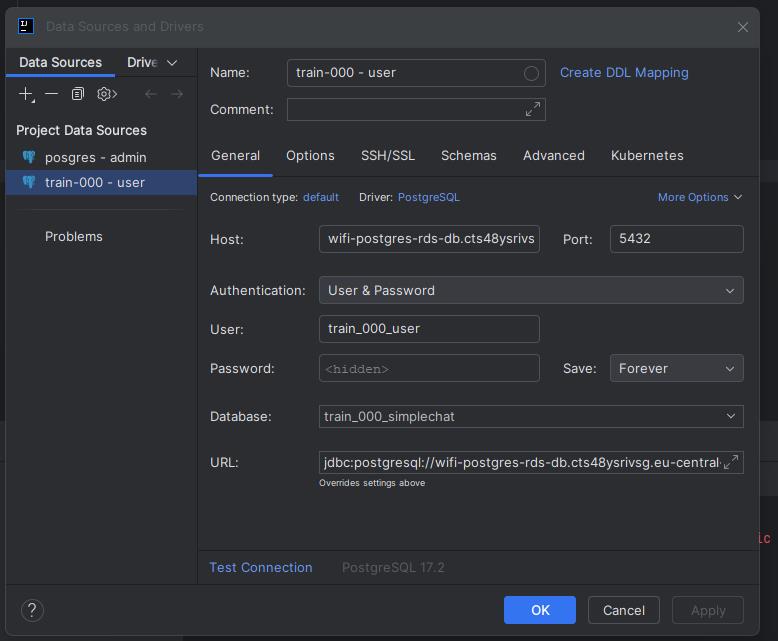
create table test  
(  
   id   serial,  
   name varchar(500)  
);  
  
alter table test  
   owner to train\_000\_user;

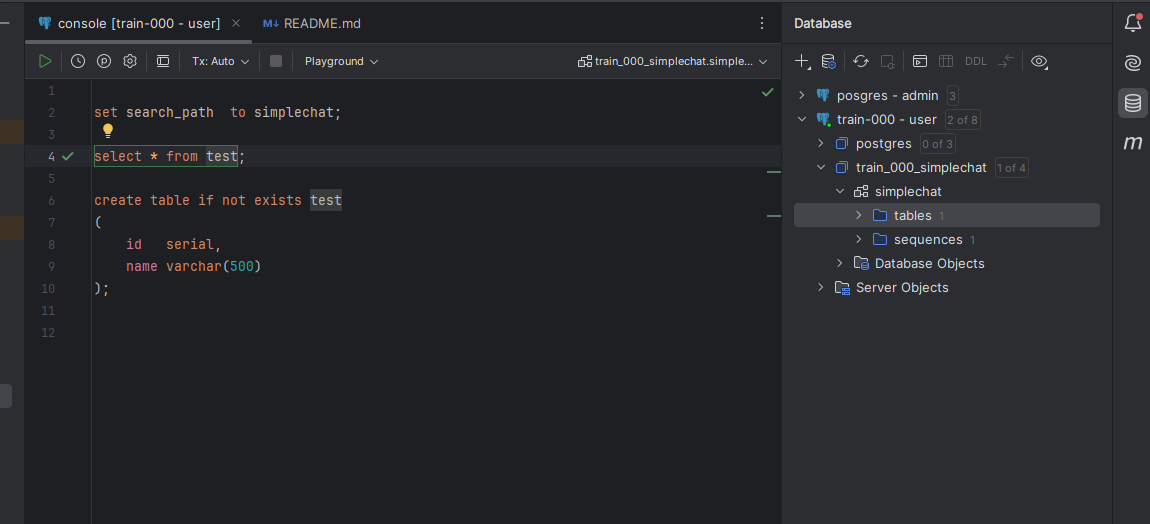
# intellij connect db

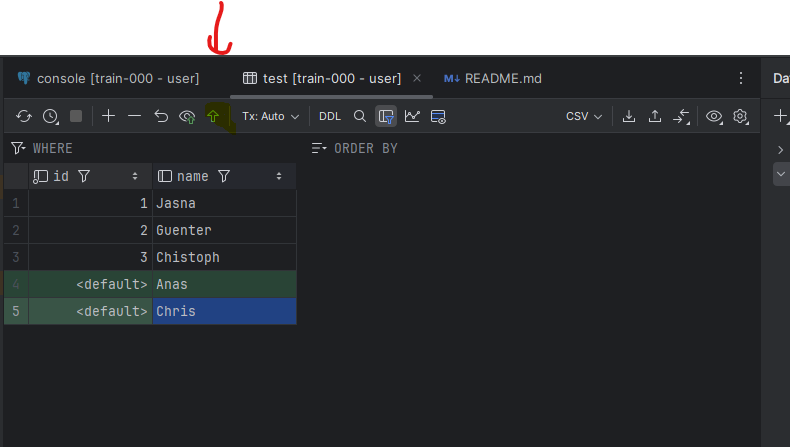
datenbank search path in console

set search\_path  to simplechat;  
  
select \* from test;

connect to new database with new database user







# TAG 3

Tag 3 - Mitschrift

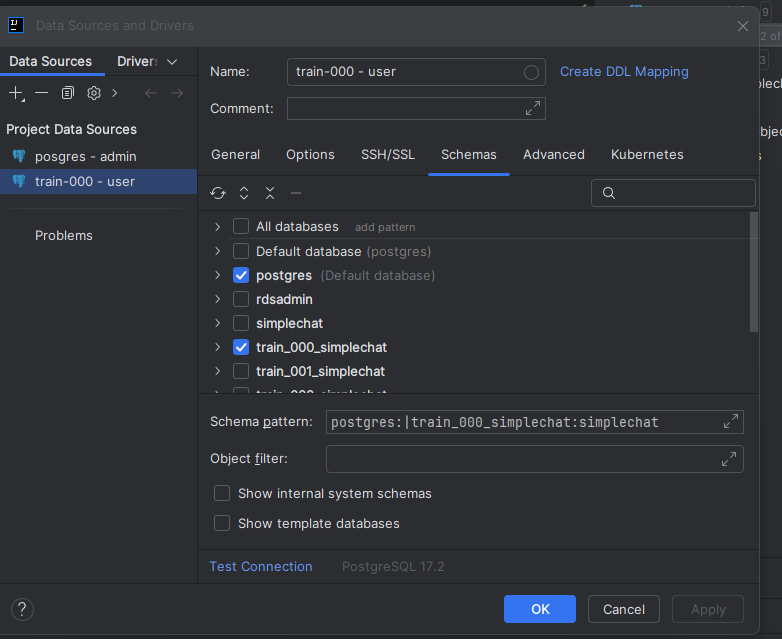
Wiederholung

VPC

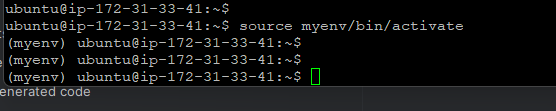
RDS

Datenbank connection

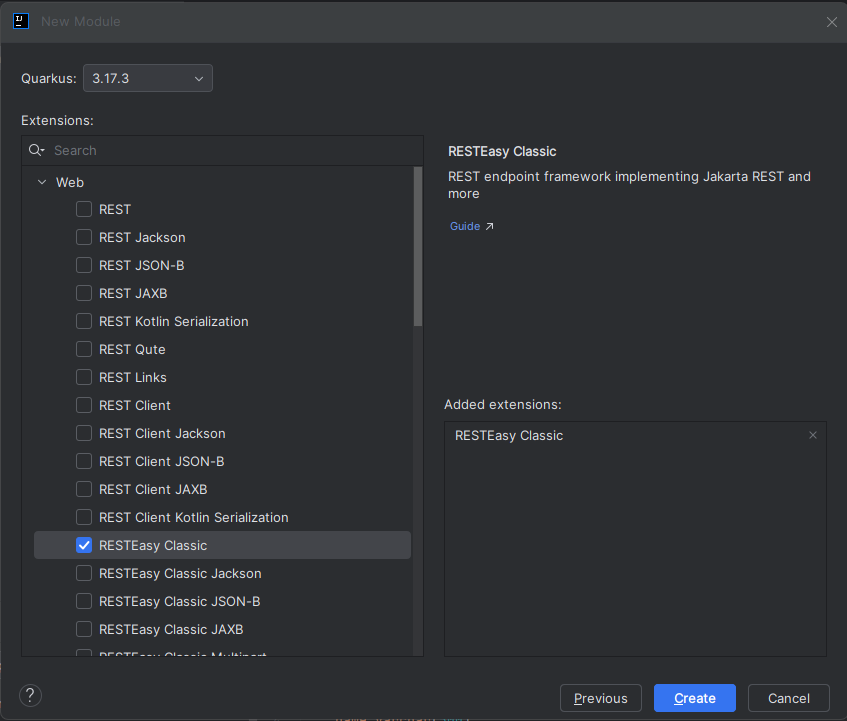
# Quarkus projekt anlegen



## Python Virtual Environment



# Quarkus Rest easy für REST services



# Quarkus logging

Quarkus Logging Dokumentation

* Quarkus Logging

<https://quarkus.io/guides/logging>

* Jboss Logging  
  <https://docs.jboss.org/seam/3/latest/reference/en-US/html/solder-logging.html>

Logging configuration - Quarkus

Quarkus: Supersonic Subatomic Java

### Quarkus Server Logging Konfigurieren

# logging configuration  
quarkus.log.console.format=CGSIT: %d{HH:mm:ss} %-5p [%c{2.}] (%t) %s%e%n   
quarkus.log.console.level=INFO  
quarkus.log.level=DEBUG

quarkus.log.file.enable=true  
quarkus.log.file.path=quarkus-log.log  
quarkus.log.file.level=DEBUG  
quarkus.log.file.format=%d{HH:mm:ss} %-5p [%c{2.}] (%t) %s%e%n

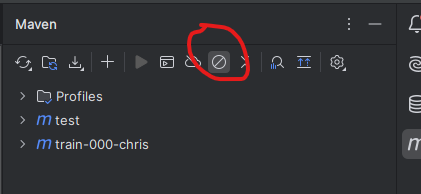
# Quarkus DB anbindung

Datenbank Anbindung maven

        <dependency>  
<groupId>io.quarkus</groupId>  
<artifactId>quarkus-jdbc-postgresql</artifactId>  
</dependency>

        <dependency>  
<groupId>io.quarkus</groupId>  
<artifactId>quarkus-hibernate-orm</artifactId>  
</dependency>

## SKIPT TESTs



# Quarks Datenbank Anbindung Konfigurieren

Datenbank Konfiguration Properties

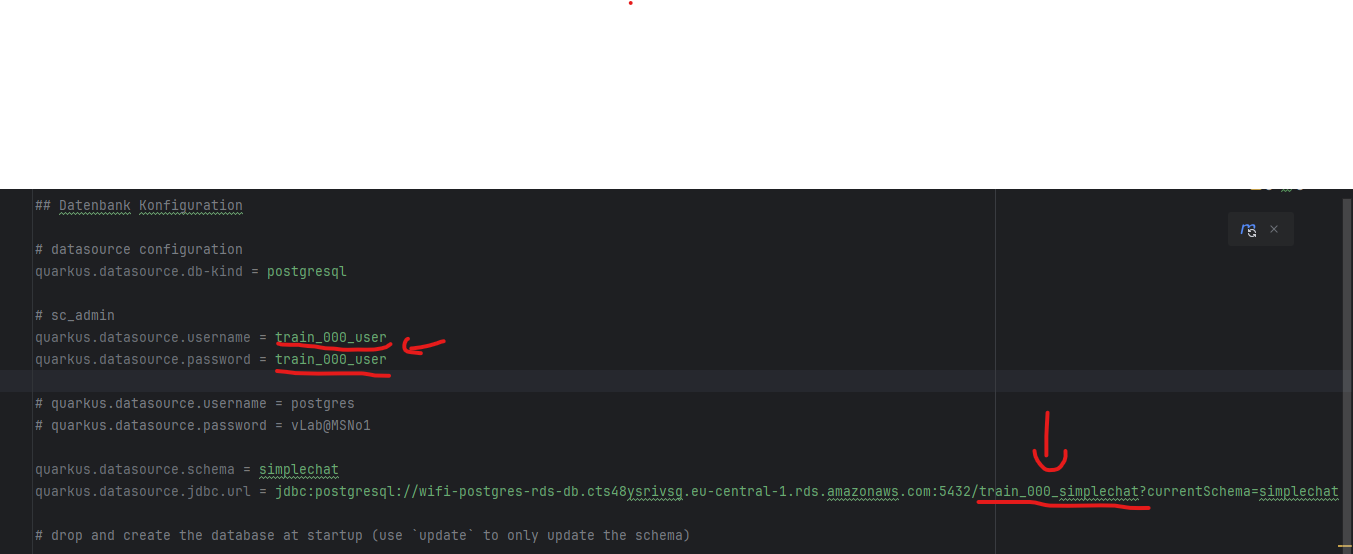
# datasource configuration  
quarkus.datasource.db-kind = postgresql

 # sc\_admin  
quarkus.datasource.username = train\_000\_user  
quarkus.datasource.password = train\_000\_user

 # quarkus.datasource.username = postgres  
# quarkus.datasource.password = vLab@MSNo1

 quarkus.datasource.schema = simplechat  
quarkus.datasource.jdbc.url = jdbc:postgresql://wifi-postgres-rds-db.cts48ysrivsg.eu-central-1.rds.amazonaws.com:5432/train\_000\_simplechat?currentSchema=simplechat

 # drop and create the database at startup (use `update` to only update the schema)



# TEST Repository Example

package at.cgsit.demo.repository;

import at.cgsit.demo.model.TestEntity;

import jakarta.enterprise.context.ApplicationScoped;

import jakarta.inject.Inject;

import jakarta.persistence.EntityManager;

import jakarta.persistence.Query;

import jakarta.transaction.UserTransaction;

import java.util.List;

@ApplicationScoped

public class TestEntityRepository {

@Inject

EntityManager em;

@Inject

UserTransaction utx;

public Long countChatMessags() {

Query query = em.createQuery("select count(e) from TestEntity e");

return (Long) query.getSingleResult();

}

// create insert method

public void insertTestEntity(String name) {

TestEntity testEntity = new TestEntity();

testEntity.setName(name);

em.persist(testEntity);

}

public TestEntity findTestEntity(String name) {

return em.find(TestEntity.class, name);

}

public TestEntity readTestEntityById(Integer id) {

return em.find(TestEntity.class, id);

}

public <T> List<T> findAll(Class<T> entityClass) {

return em.createQuery("SELECT e FROM " + entityClass.getSimpleName() + " e", entityClass)

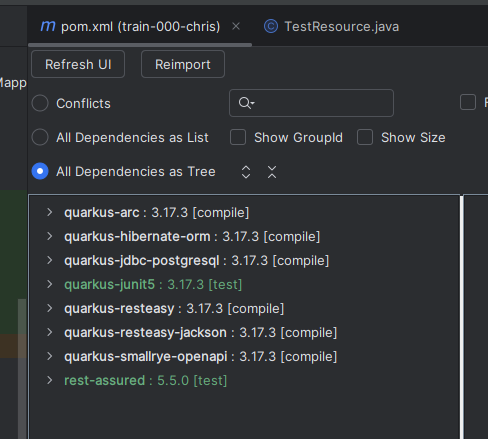
.getResultList();

}

}

# OpenAPI Erweiterung

<dependency>  
 <groupId>io.quarkus</groupId>  
 <artifactId>quarkus-smallrye-openapi</artifactId>  
</dependency>



# Web Rest-Funktion 1.0.0

import at.cgsit.demo.dto.TestDTO;  
import at.cgsit.demo.model.TestEntity;  
import at.cgsit.demo.repository.TestEntityRepository;  
import jakarta.inject.Inject;  
import jakarta.ws.rs.Path;  
import jakarta.ws.rs.GET;  
import jakarta.ws.rs.PathParam;  
import jakarta.ws.rs.Produces;  
import jakarta.ws.rs.core.MediaType;  
import org.eclipse.microprofile.openapi.annotations.Operation;  
import org.eclipse.microprofile.openapi.annotations.parameters.Parameter;  
import org.jboss.logging.Logger;

import java.util.ArrayList;  
import java.util.List;

@Path("test")  
public class TestResource {  
    private static final Logger LOG = Logger.getLogger(TestResource.class);

    @Inject  
    TestEntityRepository testEntityRepository;

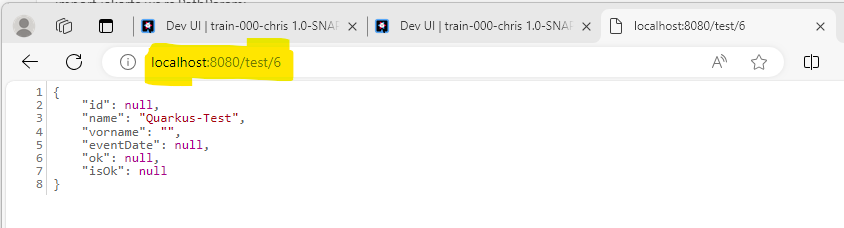
    @Operation( summary = "read a Test DTO Object by ID",  
            description = "read a Test DTO Object by ID and return it",  
            operationId = "readTestDtoById")  
    @GET  
    @Path("/{id}")  
    @Produces(MediaType.APPLICATION\_JSON)  
    public TestDTO readObjectById(  
            @Parameter(name = "input", description = "The TestDTO Input object to store", required = true, allowEmptyValue = false)  
            @PathParam("id") String id  
    ){  
        LOG.infov("input {} , objectOutput {0}",  id, "");

        int pId = 0;  
        try {  
            pId = Integer.parseInt(id);  
        } catch (NumberFormatException e) {  
            LOG.warnv ("invalid input {}", id);  
            throw e;  
        }

        TestEntity result = testEntityRepository.readTestEntityById(pId);

        TestDTO dto = new TestDTO();  
        dto.setName(result.getName());  
        dto.setVorname("");

        return dto;  
    }  
  
}



# Quarkus Test

@QuarkusTest  
class TestResourceTest {  
  
    @Test  
    void testReadObjectById() {  
        given()  
                .when().get("/test/1")  
                .then()  
                .statusCode(200)  
                .body(is(notNullValue()));  
    }

    @Test  
    void readAll() {  
    }  
}

# Rest test client checks

@Test  
    void testReadObjectById() {  
        // .when().get("/test/1") hard coded  
        int id = 1;  
        given()  
                .when().get("/test/{id}", id) // Pass a variable  
                .then()  
                .log().ifValidationFails()  
                .statusCode(200)  
                .body(is(notNullValue()))  
                .body("id", is(id))  
                .body("name", notNullValue())  
                .body("description", notNullValue());  
  
    }

# Read all entities

@GET  
    @Path("")  
    @Produces(MediaType.APPLICATION\_JSON)  
    **public** List<TestDTO> readAll(  
    ){  
  
        List<TestEntity> resultList = testEntityRepository.findAll(TestEntity.**class**);  
  
        List<TestDTO> dtoList = **new** ArrayList<TestDTO>();  
        **for** (TestEntity result : resultList) {  
            TestDTO dto = **new** TestDTO();  
            dto.setId(result.getId().longValue());  
            dto.setName(result.getName());  
            dto.setVorname("");  
            dtoList.add(dto);  
        }  
  
        **return** dtoList;

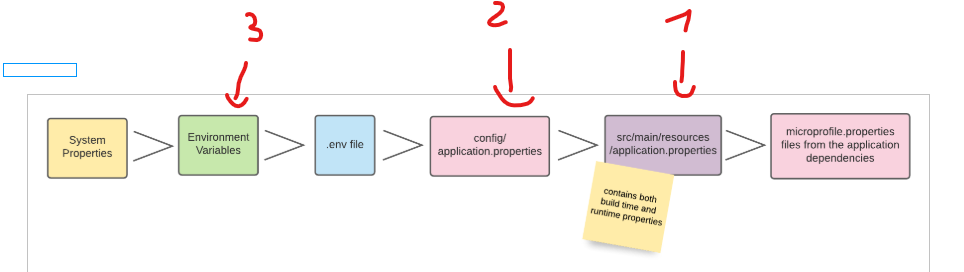
# Flyway dependencies configrieren

# quarkus.hibernate-orm.database.generation=drop-and-create  
# quarkus.hibernate-orm.database.generation=update  
quarkus.hibernate-orm.database.generation=none  
  
# Run Flyway migrations automatically at startup  
quarkus.flyway.migrate-at-start=true  
quarkus.flyway.baseline-on-migrate=true  
quarkus.flyway.locations=classpath:db/migration  
quarkus.flyway.schemas=simplechat

dd

# Tag 4

Config Properties Example  
  
<https://github.com/chris-cgsit/jee_microservices_training_next/blob/main/jee_microservice_restapi/src/main/java/at/cgsit/jeemicro/resource/configproperty/ConfigurationPropertyResource.java>



# HIbernate Optimistic Locking Konzépt

@Version  
    @Column(name = "version\_no", nullable = false)  
    **private** Long versionNo;  
  
     **public** Long getVersionNo() {  
        **return** versionNo;  
    }  
  
        **public** void setVersionNo(**final** Long versionNo) {  
        **this**.versionNo = versionNo;  
    }

neue flyway version file V1.0.2\_create\_version\_number.sql

ALTER TABLE test add column version\_no bigint not null default 0;

@JsonPropertyOrder({ "id", "versionNumber", "name", "vorname" })  
**public** **class** TestDTO {  
  
        @JsonProperty(value = "id", required = true, access = JsonProperty.Access.READ\_WRITE, defaultValue = "-1")  
        **private** Long id;  
  
        @JsonProperty(value = "versionNumber", required = true, access = JsonProperty.Access.READ\_WRITE, defaultValue = "0")  
        **private** Long versionNumber;

# Achtung bei JPA detached objekten und TX

/\*\*  
     \* Updates an existing TestEntity    
     \*  
     \* <p>This method handles both managed and detached entities:  
     \* - If the entity is already managed by the persistence context, it flushes changes to the database.  
     \* - If the entity is detached, it reattaches and updates it using {@code em.merge()}.  
     \*  
     \* <p>The returned entity reflects the current state of the database, including any updated fields such as version numbers.  
     \*  
     \* @param entity The TestEntity to be updated. It may be managed or detached.  
     \* @return The updated TestEntity with the latest database state.  
     \*/  
    @Transactional  
    **public** TestEntity updateTestEntity(TestEntity entity) {  
        **if** (em.contains(entity)) {  
            em.flush(); // flush to DB only .. so work is done for commit  
            **return** entity;  
        } **else** {  
            **return** em.merge(entity); // Reattaches and updates the detached entity  
        }  
    }

# DTO Mapping Framworks

1. <https://github.com/riok/mapperly>
2. <https://mapstruct.org/>
3. https://modelmapper.org/

# Disable Junit 5 test

@QuarkusTest  
@Disabled("Disabled until CustomerService is up!")  
**class** ExampleResourceTest {  
    @Test  
    void testHelloEndpoint() {  
        given()  
                .when().get("/hello")  
                .then()  
                .statusCode(200)  
                .body(is("Hallo Wifi Kurs sdfasdf "));  
    }  
  
}

# OpenAPI Grundletende INFOS

<https://github.com/chris-cgsit/jee_microservices_training_next/blob/main/jee_microservice_chat/src/main/java/at/cgsit/jeemicro/application/ChatApplication.java>

## Generatoren

<https://quarkus.io/extensions/io.quarkiverse.openapi.generator/quarkus-openapi-generator/>

## Monitoring APIS

<dependency>    <groupId>io.quarkus</groupId>    <artifactId>quarkus-smallrye-metrics</artifactId></dependency>

    @Counted(name = "chatMessageFindAllCount", description = "How many primality checks have been performed.")  
    @Timed(name = "chatMessageFindAllTimer", description = "A measure of how long it takes to perform the primality test.", unit = MetricUnits.MILLISECONDS)

## Health MP Support

<dependency>  
            <groupId>io.quarkus</groupId>  
            <artifactId>quarkus-smallrye-health</artifactId>  
        </dependency>

**import** at.cgsit.demo.repository.TestEntityRepository;  
**import** jakarta.inject.Inject;  
**import** org.eclipse.microprofile.health.HealthCheck;  
**import** org.eclipse.microprofile.health.HealthCheckResponse;  
**import** org.eclipse.microprofile.health.Liveness;  
  
**import** jakarta.enterprise.context.ApplicationScoped;  
  
// check if we are ok  
@Liveness  
@ApplicationScoped  
**public** **class** TestHealthCheck **implements** HealthCheck {  
  
    @Inject  
    TestEntityRepository testEntityRepository;  
  
    @Override  
    **public** HealthCheckResponse call() {  
        boolean isHealthy = checkHealth();  
        **return** HealthCheckResponse.named("yes there are test messages. OK ")  
                .status(isHealthy)  
                .build();  
    }  
  
    **private** boolean checkHealth() {  
        // Add custom health logic  
        Long l = testEntityRepository.countChatMessags();  
        **if** (l > 0) {  
            **return** true;  
        }  
        **return** false;  
    }  
}

# app initaialize

**package** at.cgsit.demo.init;  
  
**import** at.cgsit.demo.model.TestEntity;  
**import** at.cgsit.demo.repository.TestEntityRepository;  
**import** io.quarkus.runtime.StartupEvent;  
  
**import** jakarta.enterprise.event.Observes;  
**import** jakarta.inject.Inject;  
**import** jakarta.inject.Singleton;  
**import** jakarta.persistence.EntityManager;  
**import** jakarta.transaction.Transactional;  
**import** jakarta.annotation.Priority;  
**import** org.jboss.logging.Logger;  
  
**import** java.time.LocalDateTime;  
**import** java.util.Random;  
  
@Singleton  
**public** **class** AppInitializer {  
  
    @Inject  
    Logger log;  
  
    @Inject  
    EntityManager em;  
  
    @Inject  
    TestEntityRepository repository;  
  
    @Transactional  
    **public** void loadUsers(@Observes @Priority(2) StartupEvent evt) {  
  
        log.infov("server Startup Event {0}", evt);  
  
        Long l = repository.countChatMessags();  
  
        **if**( l == 0) {  
        **for** (Integer i = 1; i<=5; i++) {  
  
            TestEntity entity = **new** TestEntity();  
            entity.setName("username " + i.toString());  
            repository.insertTestEntity(entity);  
        }  
        }  
  
    }  
}

# Docker Support

<dependency>  
        <groupId>io.quarkus</groupId>  
        <artifactId>quarkus-container-image-docker</artifactId>  
      </dependency>

quarkus.container-image.build=true  
quarkus.container-image.name=trainer-000-app  
quarkus.container-image.group=trainer-000-username  
quarkus.container-image.tag=1.0.0

## Docker maven run

Run maven mit "clean package -Dquarkus.container-image.build=true"

Run maven mit "clean package -Dquarkus.container-image.build=true"

# Tag 5 :

## AWS Client

* also for windows :

<https://docs.aws.amazon.com/cli/latest/userguide/getting-started-quickstart.html>

* Login docker to aws

aws ecr get-login-password --region eu-central-1 | docker login --username AWS --password-stdin 361731702851.dkr.ecr.eu-central-1.amazonaws.com

## Docker layer

ENTRYPOINT [ "/opt/jboss/container/java/run/run-java.sh" ]

## Download Link neu

[Installing or updating to the latest version of the AWS CLI - AWS Command Line Interface](https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html)

Installing or updating to the latest version of the AWS CLI - AWS Command Line Interface

Instructions to install or update the AWS CLI on your system.

# AWS Client Configure

aws configure

* ID und key vom privaten chat
* eu-central-1
* text als default output format

aws s3 ls

## prüfen welcher user sind wir

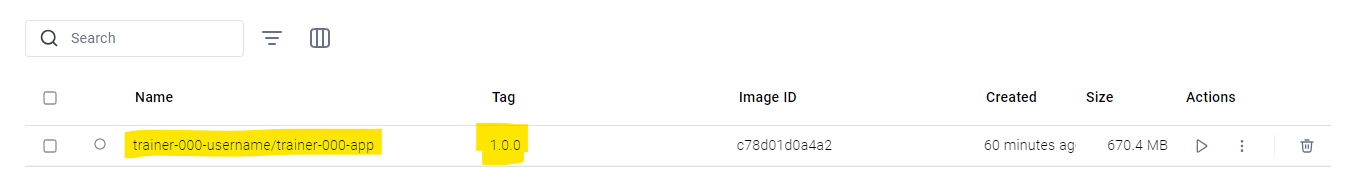
aws sts get-caller-identity

login infos auslesen ... und für docker verwenden

aws ecr get-login-password --region eu-central-1

## connect Docker mit AWS Login zur Amazon Container Registry

aws ecr get-login-password --region eu-central-1 | docker login --username AWS --password-stdin 361731702851.dkr.ecr.eu-central-1.amazonaws.com



docker tag

docker tag trai-xxxxxx-username-old/trainer-XXXXXX-app-old:latest 361731702851.dkr.ecr.eu-central-1.amazonaws.com/cgsit/train202412:XXXXXX-latest

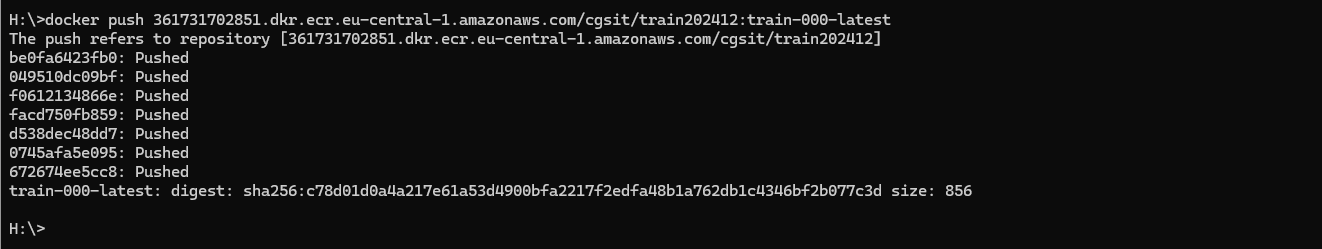
version:  1.0.0

**Account ID ist falsch : korrekt ist 361731702851**

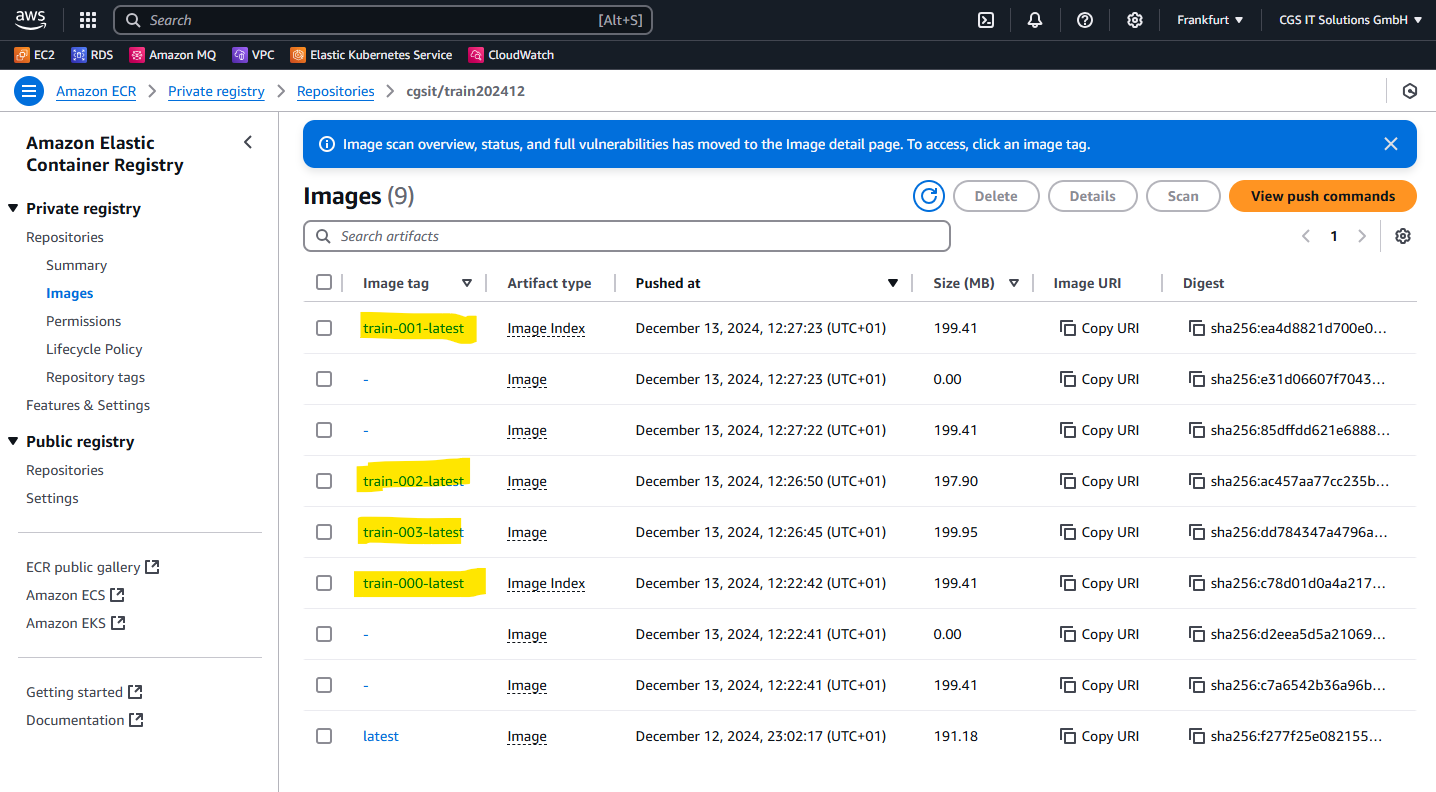
## Docker PUSH to ECR

push tu ECR

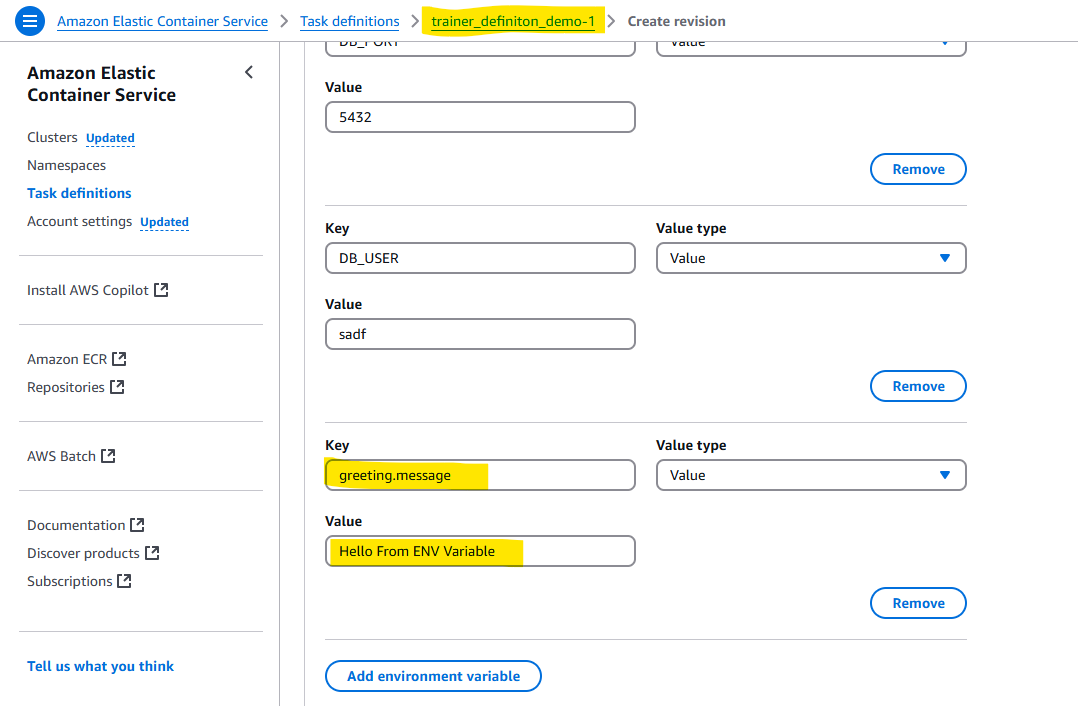
docker push 361731702851.dkr.ecr.eu-central-1.amazonaws.com/cgsit/train202412:train-XXXXXXXXXX-latest



# Images sind nun in der zentralen Cloud Registry verfügbar



# Container Environement Variablen Environment Variable vom der Task Definiton zu den Tasks = Containern übergeben



# Terraform basis variablen

Terraform variables

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = ">= 4.0.0"

}

}

}

provider "aws" {

profile = "default"

region = "eu-central-1"

}

variable "trainee\_name" {

description = "train-000"

type = string

default = "train-000"

}

variable "vpc\_cidr" {

description = "CIDR block for the VPC"

default = "10.0.0.0/16"

}

variable "subnet\_cidrs" {

description = "CIDR blocks for subnets"

type = map(string)

default = {

public = "10.0.1.0/24"

app = "10.0.2.0/24"

db = "10.0.3.0/24"

}

}