Sampler

Table of Contents

1. Technológiai alapok

A technológiák melyeken alapszik:

War

- Java 17
- Maven 3.3.0+
- Jakarta EE 10
- Wildfly 27.0.0.Final

Környezet

- Docker
- Docker compose
- Ant (opcionális)

2. Minta servicek

Minden service azonos API-t használ, azonos porton és környezeten fut.

2.1. REST service

`sample/sample-rest-service`

Ez egy egyszerű alap REST implementáció. Csak feltétlen technológiákat használ. Minden további minta service tulajdonképpen ennek a bővítése.

2.2. JPA service

`sample/sample-jpa-service`

A service célja a JPA használata. Mivel univerzálisra van tervezve ezért H2 adatbázist használ, de minden entitás kezelés olyan mintha rendes DB lenne alatta.

2.3. Redis service

`sample/sample-redis-service`

A service célja a redis kezelés használatának bemutatása egyszerű írás/olvasással. A service a coff:ee redis moduljára épül, erről bővebb információ itt található: https://i-cell-mobilsoft-open-source.github.io/coffee/#common_module_coffee-module-redis

A service használatához először futtatni kell a dockeres redist. pl.: `docker-compose up -f etcd/docker-compose/docker-compose.local.redis.yml up -d`

2.4. Redis pub-sub service

`sample/sample-redispubsub-service`

A service célja a redis Pub/Sub kezelés használatának bemutatása egyszerű publish/subscribe mintával. A service a coff:ee redispubsub moduljára épül, erről bővebb információ itt található: https://i-cell-mobilsoft-open-source.github.io/coffee/#common_module_coffee-module-redispubsub

A service használata

1. futtatni kell a dockeres redist.

   pl.: `docker-compose up -f etcd/docker-compose/docker-compose.local.redis.yml up -d`

2. docker-compose.local.wildfly.yml-t pub-sub ready widlfly modulokkal kell elindítani
    - ezt a compose fájlban a SERVICE_CLI_DIR build arg `wildfly/cli/mp-reactive`-ra állításával tehetjük meg

```
services:
  sample-service:
    container_name: bs-sample-service
    build:
      context: .
      dockerfile: develop.dockerfile
      args:
        WILDFLY_BASE_IMAGE: 'quay.io/wildfly/wildfly:27.0.0.Final-jdk17'
#        custom cli script futtatása, ha a deploymenthez kell
        SERVICE_CLI_DIR: 'wildfly/cli/mp-reactive'
    environment:
      MP_CONFIG_PROFILE: redispubsub
```

    Szükséges a "redispubsub" microprofile-config profile kapcsoló is, mert a konfiguráció összeakadhat a másik mintákkal

2.4.1. Végpontok

GET `/rest/sample`

A válasz `columnA` mezőjében visszaadja `sample-get` redis channelről utoljára olvasott értéket (`GetListener` olvas és `ApplicationContainer`-be ment, innen veszi a REST az értéket).

A channelre beküldeni redis-cli-vel tudunk pl.:

```
docker exec -it bs-sample-redis redis-cli
127.0.0.1:6379> PUBLISH sample-get test-dummy
```

POST `/rest/sample`

1. Üzenetet küld a `sample-post` és `no-sub` redis channelekre.
2. a `PostInListener` olvas a `sample-post` channelről, a kapott üzenetet cacheli `ApplicationContainer` -be és tovább küldi (mégegyszer) a `no-sub` redis channelre.
3. a REST közben vár 1 sec-et, majd kiolvassa az `ApplicationContainer`-ből az üzenetet - vagy BONotFound-ot dob, ha nem találja

Ha szeretnénk a `no-sub` channelre redis-cli-vel fel tudunk iratkozni.

```
docker exec -it bs-sample-redis redis-cli
127.0.0.1:6379> SUBSCRIBE no-sub
```

2.5. gRPC server service

- Main: `sample/sample-grpc-server-service`

A service célja a gRPC szerver használata. CDI alapu gRPC szerver minta implementació, ami a Coffee gRPC alapon működik. Jakarta oldalon folyamatban van a kidolgozása: https://projects.eclipse.org/projects/ee4j.rpc/reviews/creation-review Wildfly feature-pack: https://github.com/wildfly-extras/wildfly-grpc-feature-pack , https://www.youtube.com/watch?v=UYSNM9Dy5M4

2.5.1. GRPC API

gRPC API szinten a következő részekre van bontva:

- api-schema: XSD-ket tartalmazza amiből .proto fileokat tudunk generálni
- api-grpc-xsd2proto: a modul generál .proto file-okat XSD-ből
- api-grpc-service: azokat a .proto file-okat tartalmazza ami definiálja az endpointot, mellette olyan objektumokat tehetünk ide, amihez nem akarunk XSD-t írni
- api-grpc-stub-gen: összefogja az api-grpc-proto-gen és api-service-grpc modulokat, amiből java alapú stub-okat generál. Ezeket a stubokat használja fel a kliens és szerver is egységesen.

proto file doksi: https://developers.google.com/protocol-buffers/docs/proto3

generalt tartalom:

- api-grpc-proto-gen/target/generated-sources/src → generalt *.proto
- api-grpc-proto-gen/target/proto-external → google, coffee *.proto
- api-grpc-stub-gen/target/generated-sources → *.java stub-ok

XSD to proto

common-grpc-api modulba belekerült egy példa XSD-ből protocol-buffer létrehozásra. A schema2proto maven plugin felhasználásával, aminek részletes leírást tartalmazó konfig fájlja az etc/schema2proto/config.yaml helyen található.

Jelenleg a generálásához még szükségesek voltak xsd és proto fájlok dependency-ket másolni, amik a generálás után eltávolításra kerülnek:

- hu.icellmobilsoft.coffeecoffee-dto-xsd (xsd)
- com.google.protobuf:protobuf-java (proto)
- com.google.api.grpc:proto-google-common-protos (proto)

2.5.2. gRPC Observability

Elérhetők a metrikák: http://localhost:9991/metrics/application

- gRPC szerver fogadott request

- gRPC szerver válaszolt response
- gRPC szerver API process duration
- gRPC kliens elküldött request
- gRPC kliens válaszolt response
- gRPC kliens API process duration

A jaeger felé továbbítódnak a span adatok, gRPC + REST + redis stream consumer közös trace flowba kerül.

Két dashboard a /etc/config/grafana/provisioning/dashboards alatt érhető el.

2.5.3. gRPC Health

- The **GrpcHealthCheck** activates the check implemented in **GrpcHealth**.

2.5.4. Service példa

service.proto

```
service DummyService {
    rpc getDummy(DummyRequest) returns (DummyResponse);
    rpc getDummyRequestScope(DummyRequest) returns (DummyResponse);
}
```

service implementáció

```
import hu.icellmobilsoft.sampler.common.sample.grpc.DummyService; ①

@ApplicationScoped ②
public class DummyServiceImpl implements DummyService { ①

    @Inject
    private SampleGrpcAction sampleGrpcAction;

    @Inject
    private SampleGrpcRequestScopeAction sampleGrpcRequestScopeAction; ③

    @Override
    public void getDummy(DummyRequest request, StreamObserver<DummyResponse>
responseObserver) {
        sampleGrpcAction.call(request, responseObserver);
    }

    @Override
    @ActivateRequestContext ③
    public void getDummyRequestScope(DummyRequest request,
StreamObserver<DummyResponse> responseObserver) {
        sampleGrpcRequestScopeAction.call(request, responseObserver);
```

```
    }
}
```

① Generált interface leíró a proto fájlban definiált servicehez

② ApplicationScope szükséges

③ Ha nagyon muszáj lehet Request scope-ú beant is használni, ilyenkor az érintett metódusra ki kell tenni az `@ActivateRequestContext` annotációt.

Kliens

Grpc client kezeléshez Coffee CDI extension alapon működik. Az extension a dependency bekötéssel aktiválódik.

```
<dependency>
    <groupId>hu.icellmobilsoft.coffee</groupId>
    <artifactId>coffee-grpc-client-extension</artifactId>
</dependency>
```

A kliensek használatához konfigurációra van szükség, minta megtalálható a microprofile-config.properties file-ban. Az inject során az itt beállított paraméterekkel azonnal használhatóvá válik hasonlóan a rest kliens-hez.

config DummyServiceGrpc gRPC kliens számára

```
coffee.grpc.client.DummyServiceGrpc.port=8199
coffee.grpc.client.DummyServiceGrpc.host=localhost
```

CDI inject DummyServiceGrpc haszálatához

```
@Inject
@GrpcClient(configKey = "DummyServiceGrpc") ①
private DummyServiceGrpc.DummyServiceBlockingStub dummyGrpcService; ②

...
DummyResponse helloResponse = dummyGrpcService.getDummy(dummyRequest); ③
...
```

① Konfigurációs kulcs a csatlakozási paraméterekről, szerver host és port értéke

② Stub amin definiálva van a service hívás

③ gRPC service hívás

2.5.5. Tesztek

- 3 teszt gRPC kliens haszálat
- egyszerű dummy kérés
- többszálas teszt

- minta hibakezelésre

2.5.6. REST API

Automatikusan lekéréskor generált openapi végpont: http://localhost:8081/openapi (generált API leíró később lessz bekötve).

2.5.7. Konfiguráció

Port beállítás: microprofile-config.properties → coffee.grpc.server.port: 8199

2.6. Mongo service

`sample/sample-mongo-service`

A service célja a mongo adatbáziskezelés használatának bemutatása egyszerű írás/olvasással. A service a coff:ee mongo moduljára épül, erről bővebb információ itt található: https://i-cell-mobilsoft-open-source.github.io/coffee/#common_module_coffee-module-mongodb

A service használatához először futtatni kell a dockeres mongo adatbázist.

2.7. Kafka service

`sample/sample-kafka-service`

Service célja a Kafka kezelés használatának bemutatása egyszerű használati mintákkal.

A service használata

1. futtatni kell a dockeres kafka-t.

   pl.: `docker compose up -f etc/docker-compose/docker-compose.local.kafka.yml up -d`

2. docker-compose.local.wildfly.yml-t microprofile reactive widlfly modulokkal kell elindítani
   - ezt a compose fájlban a SERVICE_CLI_DIR build arg `wildfly/cli/mp-reactive`-ra állításával tehetjük meg

```
services:
  sample-service:
    container_name: bs-sample-service
    build:
      context: .
      dockerfile: develop.dockerfile
      args:
        WILDFLY_BASE_IMAGE: 'quay.io/wildfly/wildfly:27.0.0.Final-jdk17'
#        custom cli script futtatása, ha a deploymenthez kell
        SERVICE_CLI_DIR: 'wildfly/cli/mp-reactive'
    environment:
      MP_CONFIG_PROFILE: kafka
```

Szükséges a "kafka" microprofile-config profile kapcsoló is, mert a konfiguráció összeakadhat a másik mintákkal

2.7.1. Végpontok

GET `/rest/sample`

Nincs egyenlőre Kafka funkcióra kötve.

POST `/rest/sample`

1. Üzenetet küld és olvas `testing` kafka topicra/ból.

2.8. Etcd service

`sample/sample-etcd-service`

A service célja az etcd kezelés használatának bemutatása egyszerű írás/olvasással. A service a coff:ee etcd moduljára épül, erről bővebb információ itt található: https://i-cell-mobilsoft-open-source.github.io/coffee/#common_module_coffee-module-etcd

A service használatához először futtatni kell a dockeres etcd-t. pl.: `docker-compose up -f etc/docker-compose/docker-compose.local.etcd.yml up -d`

2.9. FileWatcher service

`sample/sample-filewatcher-service`

A service célja annak bemutatása, hogy hogyan lehet konfigurációt felolvasni, illetve annak változását is követni, properties fájlból a `java.nio.file.WatchService` segítségével.

A service használata előtt a properties fájl megtalálásához meg kell adni annak a konténeren kívüli mappáját, illetve a properties fájl konténeren belüli elérését, amihez a `docker-compose.local.wildfly.yml` compose fájlt ki kell egészíteni az alábbiak szerint:

- CONFIG_PROPERTIES_FILE_PATH környezeti változóban meg kell adni a konténeren belüli elérési útját a properties fájlnak
  - `CONFIG_PROPERTIES_FILE_PATH: /home/customConfig/config.properties` (ez az alapértelmezett érték)
- a properties fájl tartalmazó mappát fel kell csatolni a konténerhez, mint Docker volume, mivel a fájl változás követés mappán keresztül lehetséges
  - `./customConfig:/home/customConfig'`

2.10. JPA Batch service

`sample/sample-jpa-batch-service`

A service célja, hogy a BatchService-hez példákat és teszteket nyújtson, a lehető legtöbb entitás típust felhasználva.

2.10.1. Service használata

A service használatához szükségünk van egy Oracle / PostgreSQL adatbázisra, majd az adatbázishoz megfelelő Wildfly image-re. A szükséges docker-compose fájlok megtalálhatóak az `etc/docker-compose` alatt.

Oracle adatbázis használata esetén létre kell hoznunk a `bs_sample_service` felhasználót, amelyhez a script az `etc/db/scripts/oracle-bs-sample-service.sql`-ben található meg.

3. Infrastruktúra példák

A példacsomagban különböző thirdparty alkalmazásokhoz is találhatóak példák.

3.1. Redis Exporter

A konfiguráció célja a redis működését metrikákon keresztül lehessen monitorozni akár egy komplexebb alkalmazás keretei között is. Bővebb információ: https://github.com/oliver006/redis_exporter

A csomag használatához futtatni kell a dockeres redist. pl.: `docker-compose up -f etcd/docker-compose/docker-compose.local.redis.yml up -d`

Ezután a redis metrikák lokálisan itt kérdezhetőek le: http://localhost:9121/metrics

Ezen felül a dockeres observibility csomag futtatásával, aminek része a Grafana, illetve a Prometheus is. pl.: `docker-compose up -f etcd/docker-compose/docker-compose.local.observability.yml up -d`

A Redis Exporter segítségével kinyert stream-es metrikákat a Grafanán belül a `redis stream metrics` dashboardon is követhetjük.

3.2. Grafana Alerts

A konfiguráció célja, hogy bemutassa a Grafana Alert-ek működését. A csomag konfiguráció provisioning-hez előkészítve az `etcd/config/grafana/provisioning` könyvtárban.

Az egyik példa egy redis stream hosszának limiten felüli riasztását tartalmazza.

A példához futtatni kell a dockeres redist. pl.: `docker-compose up -f etcd/docker-compose/docker-compose.local.redis.yml up -d`

Illetve futtatni kell a dockeres observibility csomagot, aminek része a Grafana, illetve a Prometheus is. pl.: `docker-compose up -f etcd/docker-compose/docker-compose.local.observability.yml up -d`

A redis metrikákat a Redis Exporter segítségével nyerjük ki, aminek a lekérdezése a Prometheus konfigurációjába be lett kötve. A stream metrikákat a Grafanán belül a `redis stream metrics` dashboardon is követhetjük.

A streamhez hozzáadhatunk értékeket pl az XADD utasítással (pl.: XADD sample 1526919030474-55 cukor borso) `docker exec` parancs használatával, esetleg akár a compose file bővítésével, vagy

grafikus klienssel is (pl.: Medis). Amint 1-nél több elemet rakunk bele, akkor aktiválódik az alertünk (pending, majd firing).

Jelen beállítással emailt küld, amit a MailPit felületén (ami szintén az observability csomag része), lokálisan a http://localhost:8025 címen ellenőrízhetünk is. De van konfiguráció a Google Chat használatához is, itt viszont a saját chatünk webhookját kell majd megadnunk. Tovább részlegesen vannak példák az üzenetek template-jeinke kezeléséhez is.

Ha kitöröljük a plusz elemeket a streamről, akkor pedig a normalizálódott állapotról (resolved) is kapunk értesítést.

Bővebb információkat, paraméterezési lehetőségeket a Grafana dokumentációban lehet olvasni.