TESTIRANJE SOFTVERA

Testiranje integracije u distribuiranim sistemima i arhitekturi mikroservisa

Unapređeni kurs testiranja softvera

Java • Maven • Javalin • Pact • Swagger

PLAN PREDAVANJA

Šta ćemo danas pokriti?

PROBLEM

Ograničenja tradicionalnog integracionog testiranja.

REŠENJE

Ugovorom vođeno testiranje (Contract-Based Testing).

FOKUS

Testiranje vođeno potrošačem (Consumer-Driven Contract Testing).

PRAKSA I DEMO

SemVer, API Breaking Changes, Pact, Java, Javalin i Swagger.

Cilj: proveriti usaglašenost servisa bez podizanja celog ekosistema.

Evolucija arhitekture i testiranja

MONOLITNI SISTEMI

- Sve komponente dele isti memorijski prostor ili proces.

DISTRIBUIRANI SISTEMI

- Komponente komuniciraju preko mreže.
- Primeri protokola: HTTP, gRPC i poruke.

Promena u QA: fokus se pomera sa testiranja unutrašnjih metoda na mrežnu komunikaciju (API).

Kako smo ranije pristupali integraciji?

BIG BANG

- Sve komponente se spajaju odjednom.
- Testiranje se obavlja nakon spajanja.

TOP-DOWN

- Testiranje kreće od najviših kontrolnih struktura naniže.
- Koriste se stubs.

BOTTOM-UP

- Testiranje kreće od osnovnih modula naviše.
- Koriste se drivers.

Ove strategije su nastale u kontekstu integracije komponenti monolitnog sistema.

Zašto stare metode nisu dovoljne za mikroservise?

PRETPOSTAVKA MONOLITA

- Tradicionalne metode pretpostavljaju da su sve komponente dostupne u istom okruženju.

E2E ZAMKA

- Podizanje celog sistema je skupo i sporo.
- Potrebni su mikroservisi, baze i redovi poruka.

FLAKY TESTOVI

- Mrežni zastoji i nestabilna testna okruženja čine E2E testove nepouzdanim.

Potrebna je provera interfejsa između servisa, bez zavisnosti od celog sistema.

Contract-Based Testing: konceptualno rešenje

CONSUMER

Komponenta koja inicira API poziv i koristi odgovor.

UGOVOR

JSON/YAML datoteka opisuje tačne zahteve i očekivane odgovore API-ja.

PROVIDER

Server ili backend komponenta koja pruža podatke.

Prednost: testira se usaglašenost sa ugovorom izolovano, a ne celokupan živi sistem.

Consumer i Provider: ko je ko?

CONSUMER • POTROŠAČ

- Klijentska komponenta koja inicira API poziv.
- Konzumira podatke koje dobija od servisa.
- Primeri: frontend aplikacija ili Mikroservis A.

PROVIDER • DOBAVLJAČ

- Server ili backend komponenta koja pruža podatke.
- Odgovara na zahteve potrošača.
- Primer: Mikroservis B.

Dva smera definisanja ugovora

PROVIDER-DRIVEN

- Server diktira pravila i objavljuje specifikaciju.
- Primer: Swagger / OpenAPI.
- Klijenti moraju da se prilagode.

CONSUMER-DRIVEN

- Klijent definiše tačno šta mu je potrebno od servera.
- Server garantuje da postojeći ugovor neće biti narušen.

Zašto je fokus na potrošaču važan?

EFIKASNOST

Backend ne razvija polja koja niko ne koristi.

ZAŠTITA PRODUKCIJE

Ako backend menja polje, ugovor signalizira koji klijent može prestati da radi.

BRZ FEEDBACK

Testovi se izvršavaju tokom build faze (CI/CD), bez podizanja celog ekosistema.

Naš primer: Javalin mikroservisi



Protokol razmene: REST preko HTTP-a, format podataka: JSON.

Maven postavka: Javalin i JSON

Osnovne zavisnosti u pom.xml za User-Service:

XML • POM.XML

```xml
<dependencies>
  <dependency>
    <groupId>io.javalin</groupId>
    <artifactId>javalin</artifactId>
    <version>6.1.3</version>
  </dependency>
  <dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.17.0</version>
  </dependency>
</dependencies>
```

ZAVISNOSTI

- Javalin — web framework.
- Jackson Databind — JSON serijalizacija.

User-Service: Javalin endpoint

JAVA • USERSERVICEAPP

```
Javalin app = Javalin.create(config→{...}).start(8081);

app.get("/users/{id}", UserServiceApp::getUser);

public static void getUser(Context ctx) {
    String id = ctx.pathParam("id");
    userRepository.findById(id)
        .ifPresentOrElse(
            user -> ctx.status(200).json(user),
            () -> ctx.status(404).result("User not found")
        );
}
```

ID = 123

Odgovor: 200 i JSON profil korisnika.

DRUGI ID

Odgovor: 404 — User not found.

Swagger / OpenAPI i Pact imaju različite uloge

SWAGGER / OPENAPI

- Definiše strukturu API-ja i dozvoljene rute.
- Obezbeđuje javnu dokumentaciju i ljudima čitljiv format šeme.
- Dokumentacija može odstupati od stvarnog ponašanja ako se ne validira automatski.

PACT

- Implementira Consumer-Driven Contract testiranje.
- Dinamički proverava ponašanje stvarnog koda prema ugovoru.
- Povezuje očekivanja potrošača sa implementacijom provajdera.

Javalin OpenAPI plugin generiše Swagger UI dokumentaciju pomoću Java anotacija na handlerima.

Javalin OpenAPI: zavisnosti, anotacije i UI

MAVEN • POM.XML

```
<dependency>
    <groupId>io.javalin</groupId>
    <artifactId>javalin</artifactId>
    <version>6.1.3</version>
</dependency>
<dependency>
    <groupId>io.javalin.community.openapi</groupId>
    <artifactId>javalin-openapi-plugin</artifactId>
    <version>6.1.3</version>
</dependency>
```

Aktivacija plugin-a:

U konfigurisanju Javalin:

```
config.registerPlugin(new OpenApiPlugin());
config.registerPlugin(new SwaggerPlugin());
```

Pozivanje aplikacije (aktiviranje end pointa):

```
mvn compile exec:java
-Dexec.mainClass="primer_pact_swagger.UserServiceApp"
```

U UserServiceApp.main imamo test kod:

```
app.get("/users/{id}", UserServiceApp::getUser);
```

Aktiviranje Swagger UI:
http://localhost:8081/swagger

JAVA • OPENAPI ANOTACIJA

```
@OpenApi(
    path = "/users/{id}",
    methods = {HttpMethod.GET},
    summary = "Preuzmi korisnički profil",
    pathParams = @OpenApiParam(name = "id", description = "Jedinstveni ID
korisnika", required = true),
    responses = {
        @OpenApiResponse(status = "200", content = @OpenApiContent(from =
User.class), description = "Uspešno pronađen korisnik"),
        @OpenApiResponse(status = "404", description = "Korisnik ne postoji")
    }
)
public static void getUser(Context ctx) {
    String id = ctx.pathParam("id");
    userRepository.findById(id)
        .ifPresentOrElse(
            user -> ctx.status(200).json(user),
            () -> ctx.status(404).result("User not found")
        );
}
```

Kako funkcioniše Consumer-Driven Contract Testing?



Pact je open-source alat: Consumer generiše ugovor, Provider ga verifikuje protiv stvarne implementacije.

Maven zavisnost i izolacija kroz Mock Server

MAVEN • POM.XML

```
<dependency>
  <groupId>au.com.dius.pact.consumer</groupId>
  <artifactId>junit5</artifactId>
  <version>4.6.5</version>
  <scope>test</scope>
</dependency>
```

MOCK I STUB U PACT-U

- Pact podiže lokalni mock server tokom testiranja Consumer-a.
- Mock server imitira pravi backend prema pravilima iz testa.
- Uspešan test generiše JSON ugovor.

Definisanje ugovora u Consumer testu

JAVA • ORDERSERVICEPACTTEST

```java
@ExtendWith(PactConsumerTestExt.class)
@PactTestFor(providerName = "User-Service", port = "8081")
public class OrderServicePactTest {

    @Pact(consumer = "Order-Service")
    public V4Pact createUserProfilePact(PactDslWithProvider builder) {
        return builder
            .given("User 123 exists")
            .uponReceiving("A request for user 123")
                .path("/users/123")
                .method("GET")
            .willRespondWith()
                .status(200)
                .headers(Map.of("Content-Type", "application/json"))
                .body("{\"id\": \"123\", \"name\": \"Petar Petrovic\", \"email\": \"petar@email.com\"}")
            .toPact(V4Pact.class);
    }
```

Izvršavanje zahteva i provera klijentskog koda

JAVA • CONSUMER TEST

```
    @Test
    @PactTestFor(pactMethod = "createUserProfilePact")
    void testGetUserProfile(MockServer mockServer) throws Exception {
        HttpClient client = HttpClient.newHttpClient();
        HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(mockServer.getUrl() + "/users/123"))
            .GET().build();

        HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());

        assertEquals(200, response.statusCode());
        assertTrue(response.body().contains("Petar Petrovic"));
    }
}
```

Generisani ugovor: Pact JSON fajl

JSON • TARGET/PACTS/ORDER-SERVICE-USER-SERVICE.JSON


```
{
  "consumer": {
    "name": "Order-Service"
  },
  "interactions": [
    {
      "description": "A request for user 123",
      "key": "8f331a7d",
      "pending": false,
      "providerStates": [
        {
          "name": "User 123 exists"
        }
      ],
      "request": {
        "method": "GET",
        "path": "/users/123"
      },
      "response": {
        "body": {
          "content": {
            "email": "petar@email.com",
            "id": "123",
            "name": "Petar Petrovic"
          },
          "contentType": "application/json",
          "encoded": false
        },
        "headers": {
          "Content-Type": [
            "application/json"
          ]
        },
        "status": 200
      },
      "transport": "https",
      "type": "Synchronous/HTTP"
    }
  ],
  "metadata": {
    "pact-jvm": {
      "version": "4.6.5"
    },
    "pactSpecification": {
      "version": "4.0"
    }
  },
  "provider": {
    "name": "User-Service"
  }
}
```

Verifikacija ugovora na strani Provider-a

PROCES VERIFIKACIJE

- Provider preuzima JSON ugovor i pokreće testove nad svojim kontrolerima (Javalin aplikacija).
- Ne mock-uje se kod: pokreće se stvarni HTTP server u testnom režimu.
- Pact šalje zahteve iz ugovora i proverava statuse i JSON strukturu.

MAVEN • POM.XML

```
<dependency>
  <groupId>au.com.dius.pact.provider</groupId>
  <artifactId>junit5</artifactId>
  <version>4.6.5</version>
  <scope>test</scope>
</dependency>
```

Pact ima ulogu klijenta i proverava stvarno ponašanje provajdera.

Pact Provider test: provera Javalin koda

JAVA • VERIFICATIONTEST

```
@Provider("User-Service")
@PactFolder("src/test/resources/pacts")
public class VerificationTest {
  @BeforeEach
    void setup(PactVerificationContext context) {
        if (app == null) {
            app = Javalin.create().start(8081);
            app.get("/users/{id}", UserServiceApp::getUser);
        }
        context.setTarget(new HttpTestTarget("localhost", 8081));
    }

  @TemplateVerificationOption
  @TestTemplate
  void pactVerificationTestTemplate(PactVerificationContext context) {
    context.verifyInteraction();
  }
}
```

Zašto su potrebna stanja na Provider strani?

PROBLEM

- Pact šalje stvarni zahtev, npr. GET /users/123.
- Ako korisnik 123 ne postoji u bazi, server vraća 404 Not Found.
- Ugovor koji očekuje 200 OK tada pada.

REŠENJE • given()

- Consumer u ugovoru definiše stanje: given("User 123 exists").
- Provider priprema sistem za dato stanje pre slanja HTTP zahteva.
- Pact traži odgovarajuću @State metodu.

Implementacija @State i dobre prakse

JAVA • PROVIDER STATE HANDLERS

```
@Provider("User-Service")
public class VerificationTest {
  private static UserRepository userRepository =
      UserRepository.getInstance();

  @State("User 123 exists")
  public void setupUserExistsState() {
    User testUser = new User("123", "Petar Petrovic", "petar@email.com");
    userRepository.save(testUser);
  }

}
```

AT OMIČNOST

Svaka @State metoda čisti bazu i unosi samo potrebne podatke.

BEZ HARDCODING-A

Ne menjati logiku Javalin handlera zbog testova; priprema se radi u test objektima/bazi.

IMENOVANJE

Tekst u given() mora se tačno poklapati sa vrednošću u @State().

API Breaking Changes: šta lomi postojeće klijente?

Izmene API-ja koje narušavaju kompatibilnost unazad, zbog čega postojeći klijenti prestaju da rade.

UKLANJANJE POLJA

JSON polje email se uklanja iz odgovora.

PROMENA TIPA

id se menja iz String u Long.

PROMENA RUTE

/users/{id} se menja u /api/v1/users/{id}.

Kako Pact sprečava Breaking Changes?

1 • IZMENA

Programer u User-Service uklanja polje email.

2 • mvn test

Pact verification proverava Provider prema ugovoru.

3 • BUILD PADA

Ugovor Order-Service i dalje zahteva polje email.

Neispravan kod se zaustavlja pre nego što stigne na produkciju.

SemVer: Major.Minor.Patch

Primer formata verzije: 2.1.4

MAJOR

Izmene koje nisu kompatibilne unazad (Breaking Changes).

MINOR

Dodavanje funkcionalnosti koje su kompatibilne unazad.

PATCH

Ispravke bagova koje su kompatibilne unazad.

Kako verzionisanje utiče na regresiono testiranje?

MINOR / PATCH

Zahtevaju standardne regresione testove (garancija kompatibilnosti unazad).

MAJOR

Zahteva novu verziju ugovora i koordinaciju timova.

Pact Broker koristi SemVer oznake i matricu kompatibilnosti da proceni da li je deployment bezbedan (can-i-deploy).

Centralno skladište i sinhronizacija ugovora

CONSUMER

Automatski objavljuje (push) generisani ugovor na Broker.

PACT BROKER

Centralni aplikativni server
Skladište i tačka sinhronizacije

PROVIDER

Automatski preuzima (pull) ugovore sa Brokera tokom build procesa radi verifikacije.

Ručno prenošenje JSON fajlova nije održivo u CI/CD arhitekturi.

can-i-deploy: sprečavanje nebezbednog deployment-a

PROVERA KOMPATIBILNOSTI

- CLI alat unutar Pact ekosistema.
- Proverava matricu kompatibilnosti u Pact Broker-u.
- Pita da li je trenutni Consumer verifikovan od strane Provider verzije koja je na produkciji.

ISHOD U PIPELINE-U

- Ako je odgovor NE, deployment se blokira.
- Provera se obavlja pre puštanja Order-Service na produkciju.
- Primeri pipeline alata: Jenkins / GitHub Actions.

Maven konfiguracija za objavljivanje ugovora

XML • POM.XML

```xml
<plugin>
  <groupId>au.com.dius.pact.provider</groupId>
  <artifactId>maven</artifactId>
  <version>4.6.5</version>
  <configuration>
    <pactBrokerUrl>http://localhost:9292</pactBrokerUrl>
    <pactBrokerToken>${env.PACT_BROKER_TOKEN}</pactBrokerToken>
  </configuration>
</plugin>
```

TOKEN

PACT_BROKER_TOKEN se čita iz environment promenljive.

POKRETANJE

```
mvn pact:publish
```

Komanda se pokreće nakon uspešno izvršenih testova.

Zašto je idempotentnost važna u distribuiranim sistemima?

DEFINICIJA

Svojstvo API zahteva da višestruko izvršavanje istog zahteva ima isti efekat na stanje sistema kao jedno izvršavanje.

ZAŠTO JE VAŽNA ZA QA?

- Mrežne greške mogu navesti klijenta da ponovi zahtev.
- Retry mehanizmi su česti u distribuiranim sistemima.
- Testovi proveravaju ponašanje pri ponovljenim zahtevima.

Idempotentnost po HTTP metodama

HTTP metoda	Idempotentna?	Opis
GET	**Da**	Samo čita podatke; ponovljeni pozivi ne menjaju stanje.
PUT	**Da**	Potpuno zamenjuje resurs; dva ista poziva daju isti krajnji rezultat.
DELETE	**Da**	Prvi poziv briše, ostali mogu vratiti 404, ali stanje ostaje isto.
POST	**Ne**	Kreira novi resurs; dva poziva mogu kreirati dva različita objekta.

Testiranje idempotentnosti u praksi

KORACI TESTA

- Prvi poziv: PUT /users/123 sa novim imenom; proveriti 200 OK.
- Ponovljeni poziv: poslati potpuno isti zahtev.
- Validacija: baza ostaje konzistentna, nema duplih zapisa, odgovor je ispravan.

JAVA • JAVALIN ENDPOINT

```java
app.put("/users/{id}", UserServiceApp::updateUser);

public static void updateUser(Context ctx) {
    String id = ctx.pathParam("id");
    User body = ctx.bodyAsClass(User.class);

    // save() upisuje po ključu (id) i sprečava dupliranje podataka
    User updatedUser = new User(id, body.name(), body.email());
    userRepository.save(updatedUser);

    ctx.status(200).json(updatedUser);
}
```

Ključne poruke i pitanja za diskusiju

REZIME PREDAVANJA

- E2E testovi su spori i skupi; Contract testiranje pruža brzu alternativu izolovanim testovima.
- Consumer-Driven pristup omogućava klijentima da vode evoluciju API-ja bez straha od breaking promena.
- Pact automatizuje kreiranje i verifikaciju ugovora kroz Java ekosistem.
- SemVer i idempotentnost su važni za projektovanje i testiranje robusnih API-ja.

PITANJA I ODGOVORI

Hvala na pažnji!

Kako biste testirali asinhronu integraciju (npr. RabbitMQ/Kafka poruke) koristeći Pact?

Gde saznati više?

PACT	pact.io
JAVALIN	javalin.io
SEMANTIC VERSIONING	semver.org
MARTIN FOWLER BLOG	Consumer-Driven Contracts: A Service Evolution Pattern