JavaCro'18

Emulate Amazon Web Services infrastructure in single JMV process to reduce development cost and improve productivity.

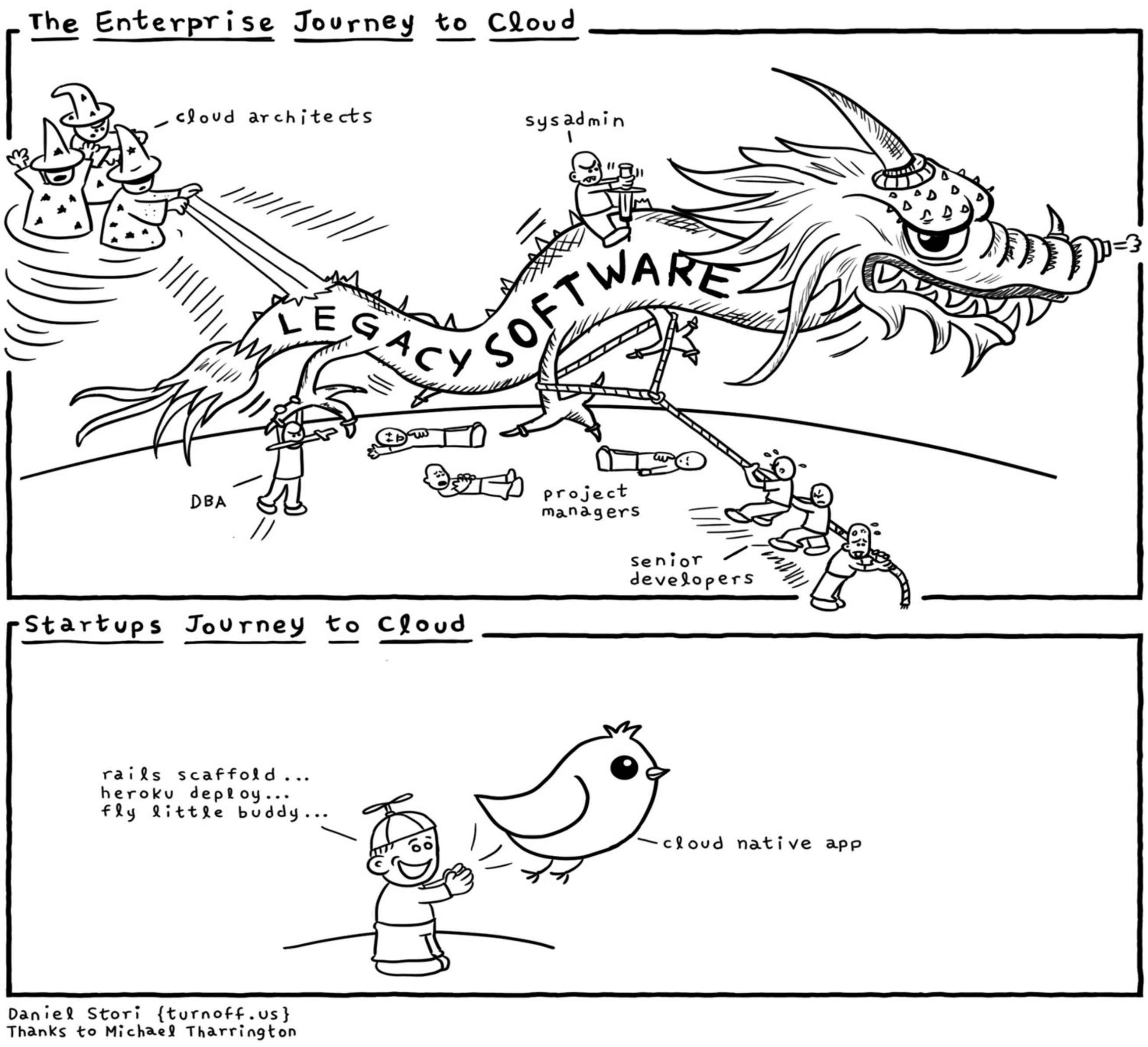
Subjective opinion

Information from this report is my subjective opinion based on my experience, knowledge, mistakes... ;-)

Why you need emulate AWS

- Cost savings
- Testing automation. Isolated environments
- Improve development productivity. Network latency and throughput
- Great Firewall of China / Russia (AWS network blocking)

Marketing buzzwords is your enemy



Marketing buzzwords is your enemy

In economics, vendor lock-in, also known as proprietary lock-in or customer lock-in, makes a customer dependent on a vendor for products and services, unable to use another vendor without substantial switching costs.

https://en.wikipedia.org/wiki/Vendor_lock-in

Marketing buzzwords is your enemy

Amazon MQ is a managed message broker service that provides compatibility with many popular message brokers. **We recommend Amazon MQ for migrating applications from existing message brokers** that rely on compatibility with APIs such as JMS or protocols such as AMQP, MQTT, OpenWire, and STOMP.

Amazon SQS and Amazon SNS are queue and topic services that are highly scalable, simple to use, and don't require you to set up message brokers. **We recommend these services for new applications** that can benefit from nearly unlimited scalability and simple APIs.

https://docs.aws.amazon.com/amazon-mq/latest/developer-guide/welcome.html

Marketing buzzwords is your enemy

Modernizing Your Data Warehouse on AWS

Easy Data Loading

Load virtually any type of data into Amazon Redshift from a range of data sources including Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB, Amazon EMR, and AWS Data Pipeline.

https://aws.amazon.com/big-data/featured-partner-data-warehouse-modernization/

Marketing buzzwords is your enemy

Data Types: SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP, TIMESTAMPTZ

https://docs.aws.amazon.com/redshift/latest/dg/c_Supported_data_types.html

Amazon Redshift is based on PostgreSQL 8.0.2.

https://docs.aws.amazon.com/redshift/latest/dg/c_redshift-and-postgres-sql.html

Dependency inversion principle

You project code should depends on abstract/common API not on concrete cloud provider API.

- High-level modules should not depend on low-level modules. Both should depend on abstractions.
- Abstractions should not depend on details. Details should depend on abstractions.

https://en.wikipedia.org/wiki/Dependency_inversion_principle

Examples: Slf4J, Apache jclouds, micrometer.io, Apache Beam, Contexts and Dependency Injection (CDI)

Emulate deployment infrastructure

- Right choice.

  Examles: Testcontainers + LocalStack

https://github.com/localstack/localstack

https://github.com/testcontainers/testcontainers-java-module-localstack

- Pragmatic approach — in-JVM process libraries.

Emulate Simple Cloud Storage Service (s3)

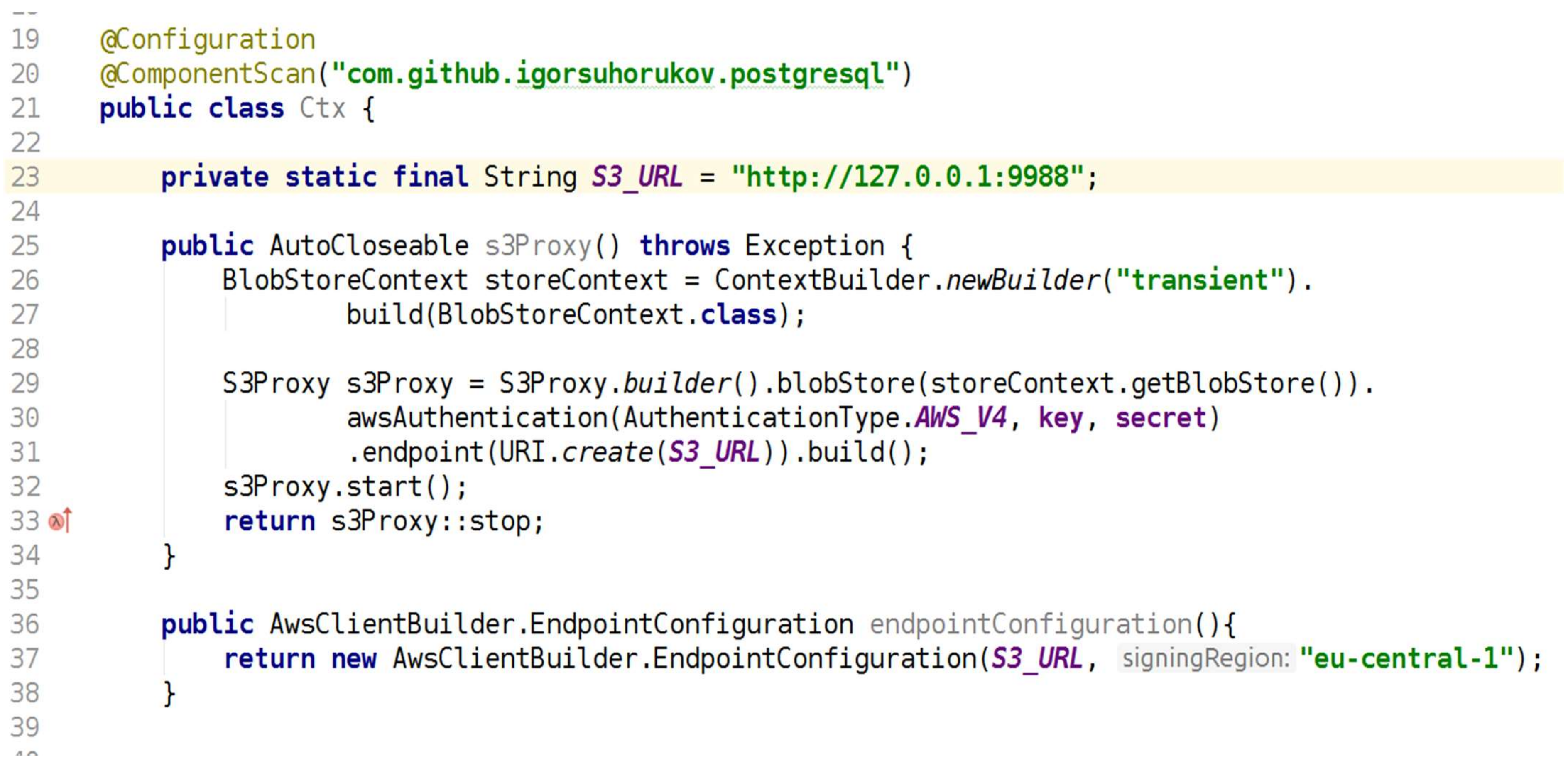
Heavy approach with real distributed object storage:

Ceph Object Gateway S3 API

https://github.com/ceph/s3-tests

Emulate Simple Cloud Storage Service (s3)

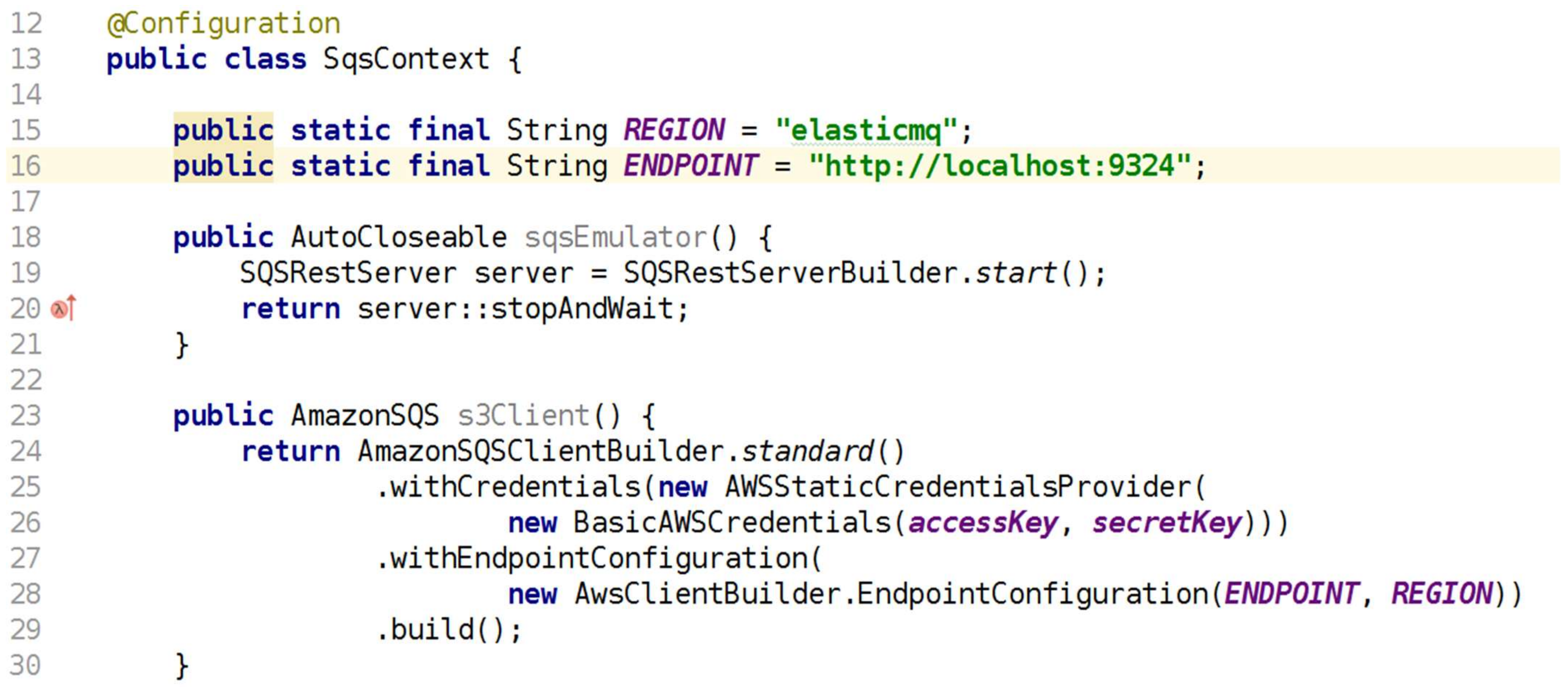
https://github.com/gaul/s3proxy



```java
@Configuration
@ComponentScan("com.github.igorsuhorukov.postgresql")
public class Ctx {

    private static final String S3_URL = "http://127.0.0.1:9988";

    public AutoCloseable s3Proxy() throws Exception {
        BlobStoreContext storeContext = ContextBuilder.newBuilder("transient").
                build(BlobStoreContext.class);

        S3Proxy s3Proxy = S3Proxy.builder().blobStore(storeContext.getBlobStore()).
                awsAuthentication(AuthenticationType.AWS_V4, key, secret)
                .endpoint(URI.create(S3_URL)).build();
        s3Proxy.start();
        return s3Proxy::stop;
    }

    public AwsClientBuilder.EndpointConfiguration endpointConfiguration(){
        return new AwsClientBuilder.EndpointConfiguration(S3_URL, signingRegion: "eu-central-1");
    }

```

Emulate Simple Queue Service (SQS)

https://github.com/adamw/elasticmq

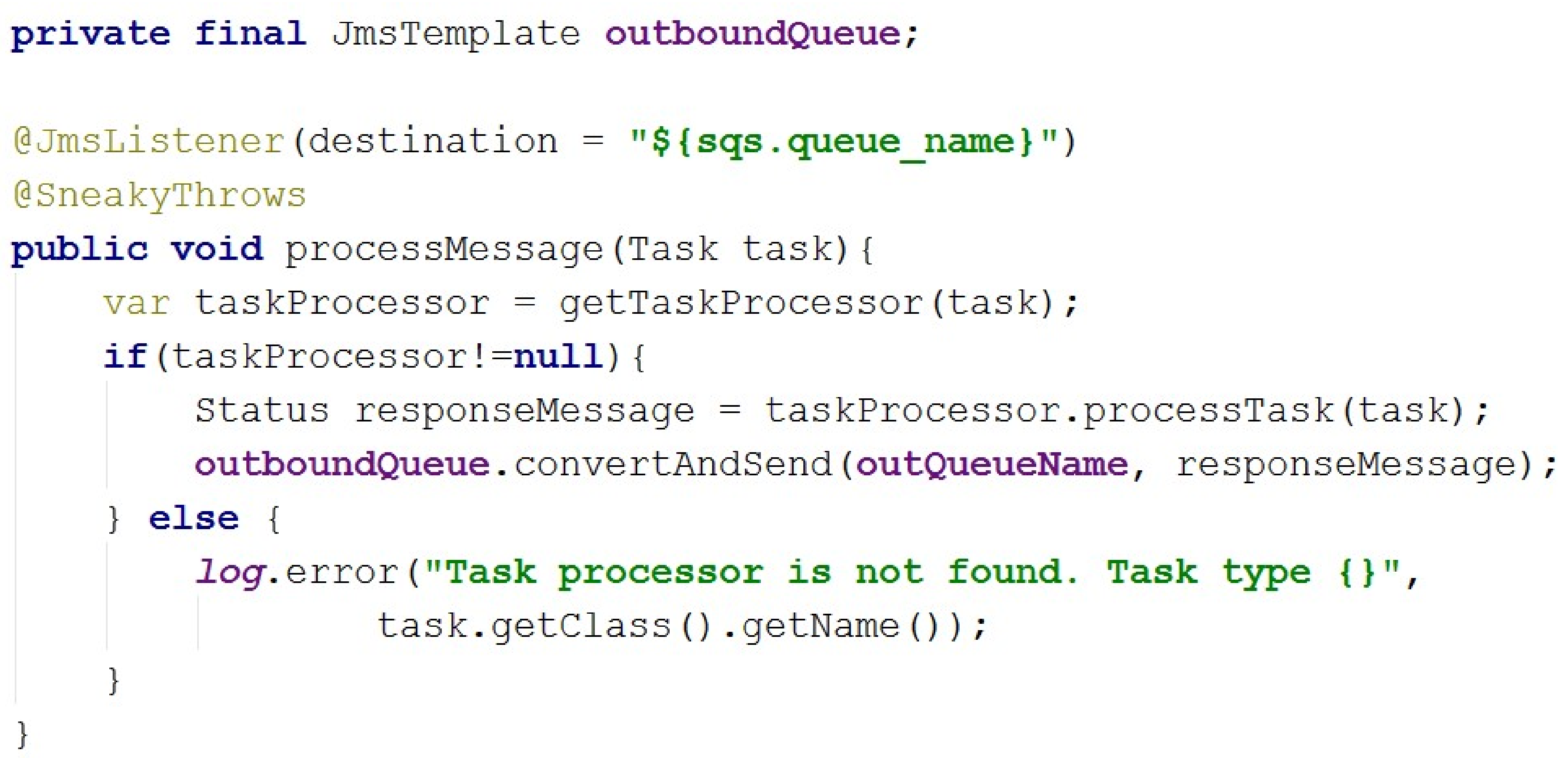


```java
@Configuration
public class SqsContext {

    public static final String REGION = "elasticmq";
    public static final String ENDPOINT = "http://localhost:9324";

    public AutoCloseable sqsEmulator() {
        SQSRestServer server = SQSRestServerBuilder.start();
        return server::stopAndWait;
    }

    public AmazonSQS s3Client() {
        return AmazonSQSClientBuilder.standard()
                .withCredentials(new AWSStaticCredentialsProvider(
                        new BasicAWSCredentials(accessKey, secretKey)))
                .withEndpointConfiguration(
                        new AwsClientBuilder.EndpointConfiguration(ENDPOINT, REGION))
                .build();
    }
```

Emulate Simple Queue Service (SQS): JMS

spring-jms

com.amazonaws:amazon-sqs-java-messaging-lib



```java
private final JmsTemplate outboundQueue;

@JmsListener(destination = "${sqs.queue_name}")
@SneakyThrows
public void processMessage(Task task){
    var taskProcessor = getTaskProcessor(task);
    if(taskProcessor!=null){
        Status responseMessage = taskProcessor.processTask(task);
        outboundQueue.convertAndSend(outQueueName, responseMessage);
    } else {
        log.error("Task processor is not found. Task type {}",
                task.getClass().getName());
    }
}
```

Emulate Simple Queue Service (SQS): JMS

org.springframework.boot:spring-boot-starter-artemis

org.apache.activemq:artemis-jms-server

```
@Import(ArtemisAutoConfiguration.class)
public class TestMessagingContext implements ArtemisConfigurationCustomizer {

    @Autowired
    @Qualifier("jmsConnectionFactory")
    ConnectionFactory testConnectionFactory;

    @Override
    public void customize(Configuration configuration) {
        configuration.setTransactionTimeout(TimeUnit.MINUTES.toMillis( duration: 2));
        Map<String, AddressSettings> addressesSettings =
                configuration.getAddressesSettings();
        AddressSettings addressSetting = new AddressSettings();
        addressSetting.setMaxDeliveryAttempts(0);
        addressesSettings.put("*", addressSetting);
    }

    @Bean
    public ConnectionFactory getConnectionFactory(Credentials credentials, Sqs sqs) {
        return testConnectionFactory;
    }
```

Emulate RDS PostgreSQL

http://www.h2database.com

https://github.com/yandex-qatools/postgresql-embedded

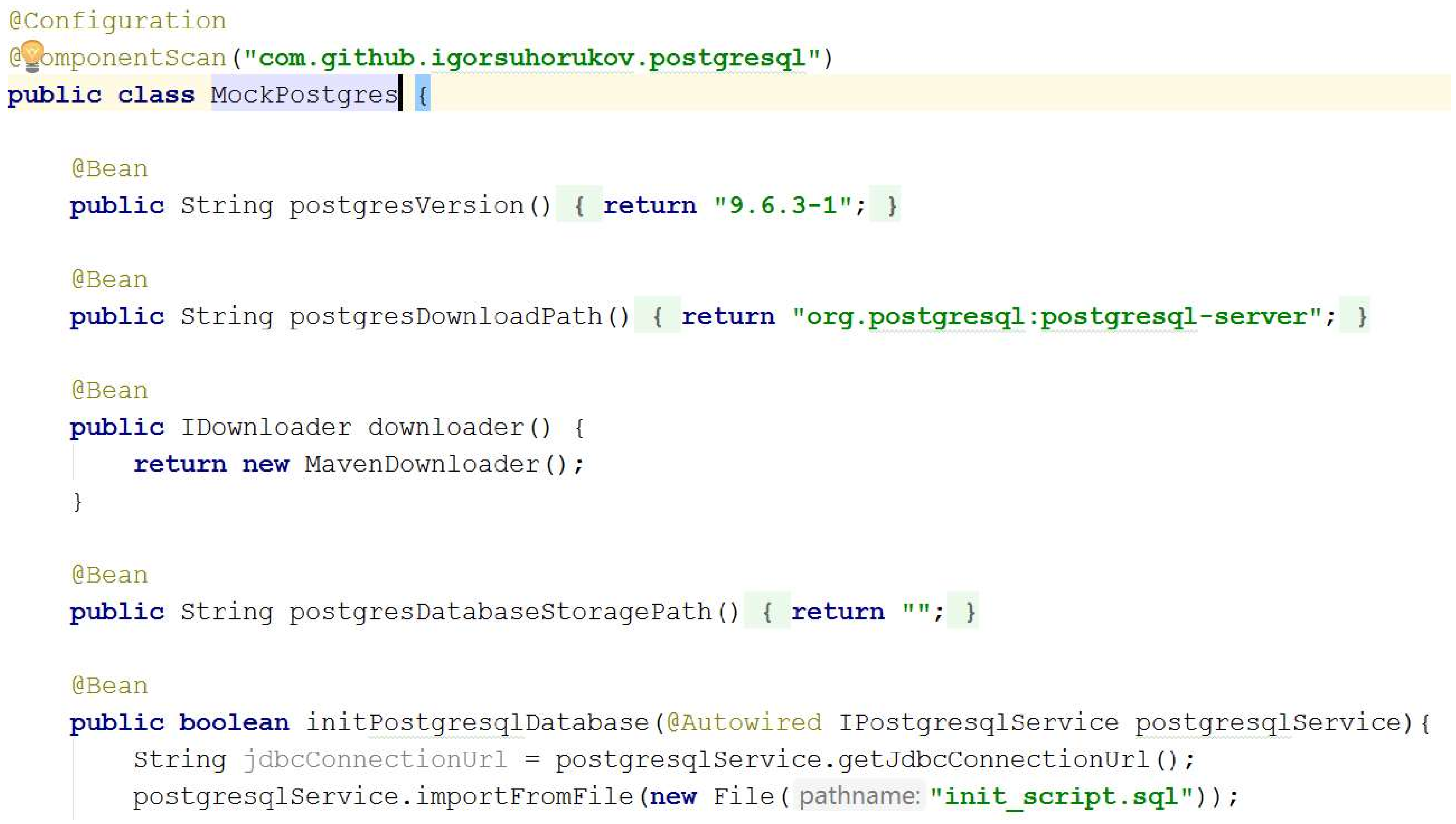
CDI wrapper: https://github.com/igor-suhorukov/postgresql-runner

Expose postgresql-embedded as CDI component

Manage Spring framework PG lifecycle as AutoCloseable…

Maven PostgreSql resolver

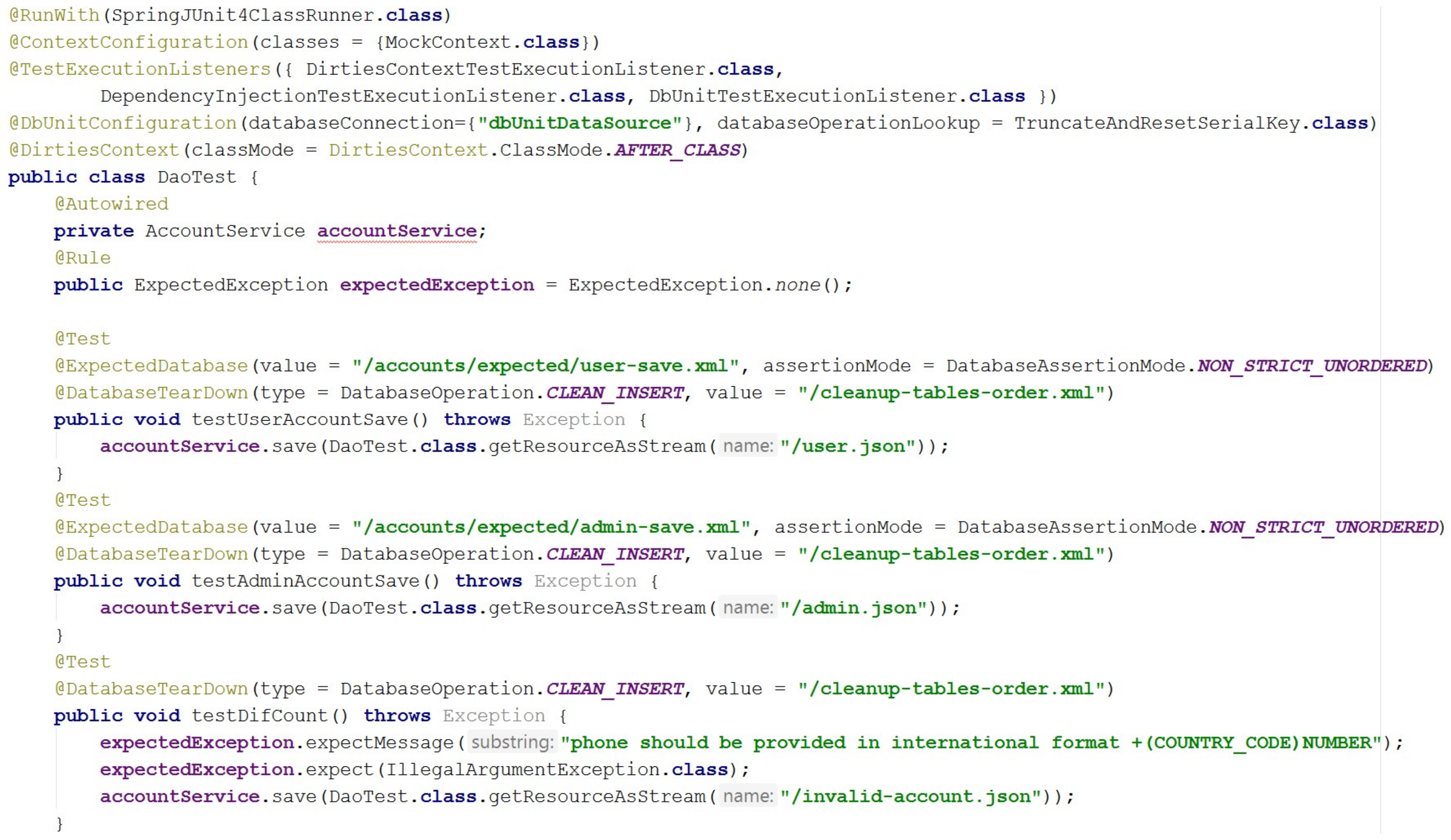
Emulate RDS PostgreSQL



```
@Configuration
@ComponentScan("com.github.igorsuhorukov.postgresql")
public class MockPostgres {

    @Bean
    public String postgresVersion() { return "9.6.3-1"; }

    @Bean
    public String postgresDownloadPath() { return "org.postgresql:postgresql-server"; }

    @Bean
    public IDownloader downloader() {
        return new MavenDownloader();
    }

    @Bean
    public String postgresDatabaseStoragePath() { return ""; }

    @Bean
    public boolean initPostgresqlDatabase(@Autowired IPostgresqlService postgresqlService){
        String jdbcConnectionUrl = postgresqlService.getJdbcConnectionUrl();
        postgresqlService.importFromFile(new File( pathname: "init_script.sql"));
```

Emulate RDS PostgreSQL

com.github.springtestdbunit:spring-test-dbunit



```java
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = {MockContext.class})
@TestExecutionListeners({ DirtiesContextTestExecutionListener.class,
        DependencyInjectionTestExecutionListener.class, DbUnitTestExecutionListener.class })
@DbUnitConfiguration(databaseConnection={"dbUnitDataSource"}, databaseOperationLookup = TruncateAndResetSerialKey.class)
@DirtiesContext(classMode = DirtiesContext.ClassMode.AFTER_CLASS)
public class DaoTest {
    @Autowired
    private AccountService accountService;
    @Rule
    public ExpectedException expectedException = ExpectedException.none();

    @Test
    @ExpectedDatabase(value = "/accounts/expected/user-save.xml", assertionMode = DatabaseAssertionMode.NON_STRICT_UNORDERED)
    @DatabaseTearDown(type = DatabaseOperation.CLEAN_INSERT, value = "/cleanup-tables-order.xml")
    public void testUserAccountSave() throws Exception {
        accountService.save(DaoTest.class.getResourceAsStream( name: "/user.json"));
    }
    @Test
    @ExpectedDatabase(value = "/accounts/expected/admin-save.xml", assertionMode = DatabaseAssertionMode.NON_STRICT_UNORDERED)
    @DatabaseTearDown(type = DatabaseOperation.CLEAN_INSERT, value = "/cleanup-tables-order.xml")
    public void testAdminAccountSave() throws Exception {
        accountService.save(DaoTest.class.getResourceAsStream( name: "/admin.json"));
    }
    @Test
    @DatabaseTearDown(type = DatabaseOperation.CLEAN_INSERT, value = "/cleanup-tables-order.xml")
    public void testDifCount() throws Exception {
        expectedException.expectMessage( substring: "phone should be provided in international format +(COUNTRY_CODE)NUMBER");
        expectedException.expect(IllegalArgumentException.class);
        accountService.save(DaoTest.class.getResourceAsStream( name: "/invalid-account.json"));
    }
```

Emulate Amazon Redshift (COPY)

https://github.com/opt-tech/redshift-fake-driver

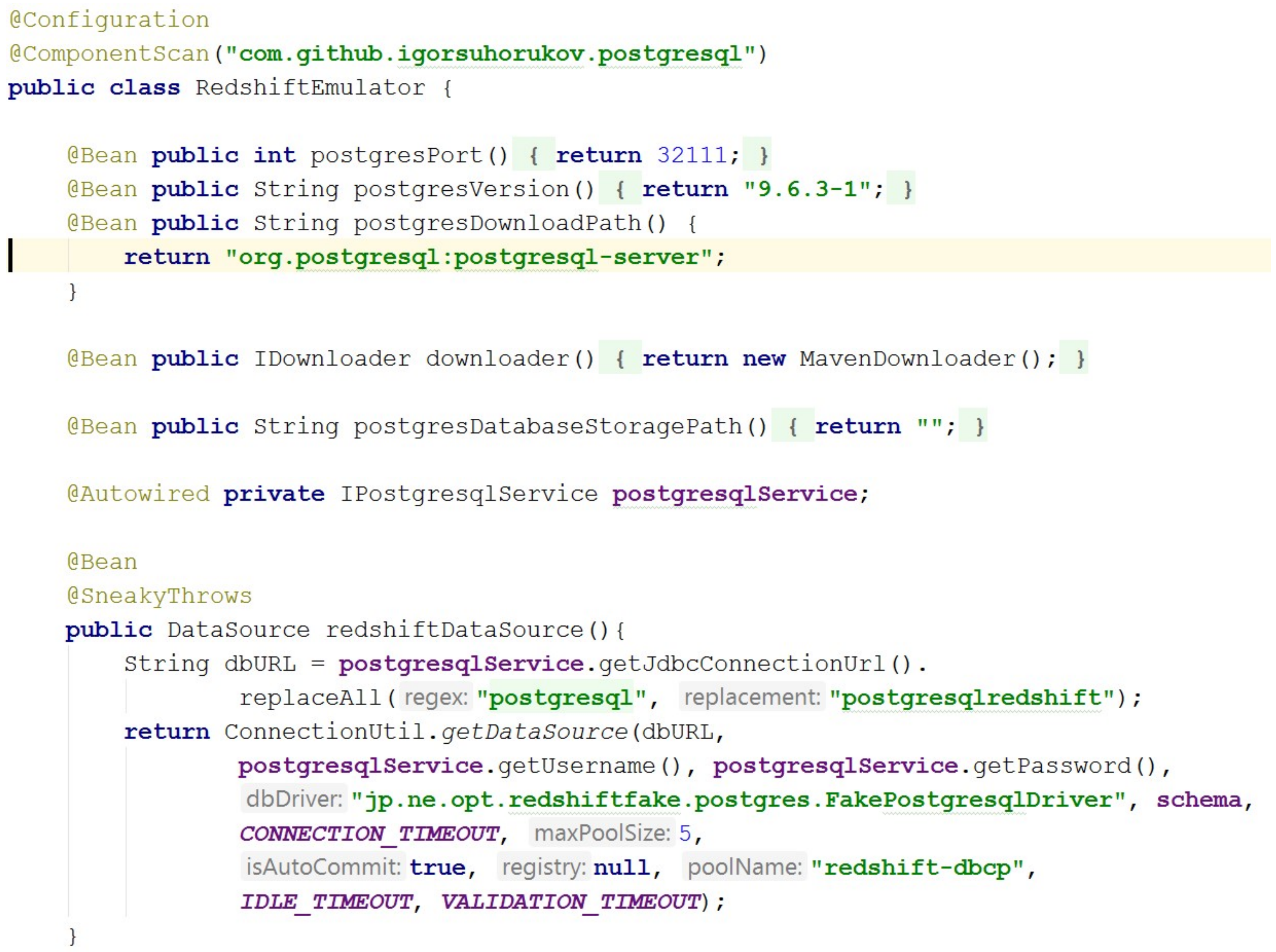
This driver supports:

- json+jsonpath
- Manifest (json file with references to CSV files)

After source code modification driver is almost ready for CSV import

DriverClass: jp.ne.opt.redshiftfake.postgres.FakePostgresqlDriver

Emulate Amazon Redshift (COPY)

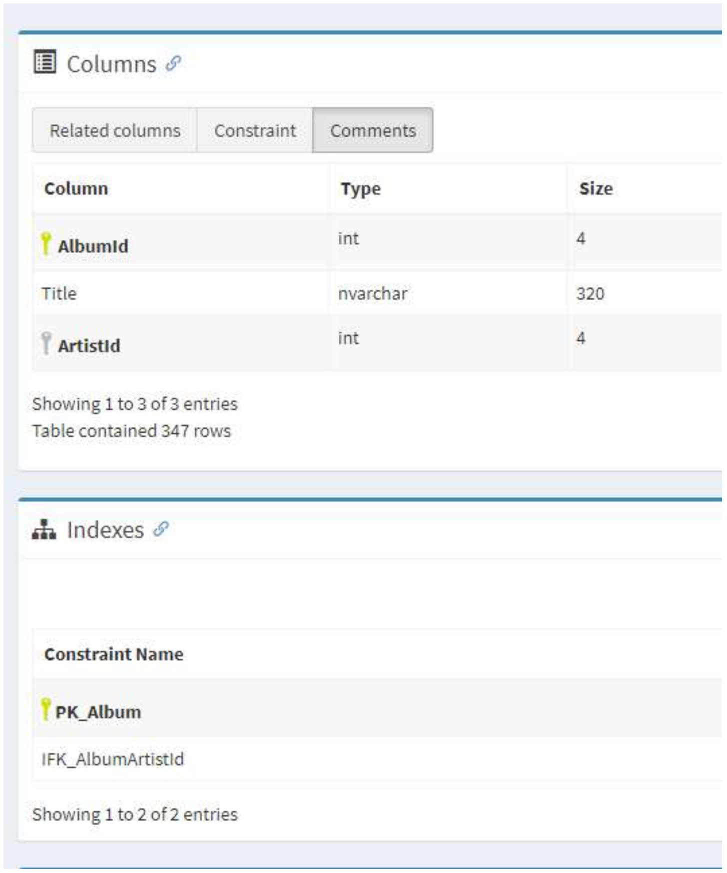


```
@Configuration
@ComponentScan("com.github.igorsuhorukov.postgresql")
public class RedshiftEmulator {

    @Bean public int postgresPort() { return 32111; }
    @Bean public String postgresVersion() { return "9.6.3-1"; }
    @Bean public String postgresDownloadPath() {
        return "org.postgresql:postgresql-server";
    }

    @Bean public IDownloader downloader() { return new MavenDownloader(); }

    @Bean public String postgresDatabaseStoragePath() { return ""; }

    @Autowired private IPostgresqlService postgresqlService;

    @Bean
    @SneakyThrows
    public DataSource redshiftDataSource(){
        String dbURL = postgresqlService.getJdbcConnectionUrl().
                replaceAll( regex: "postgresql",  replacement: "postgresqlredshift");
        return ConnectionUtil.getDataSource(dbURL,
                postgresqlService.getUsername(), postgresqlService.getPassword(),
                 dbDriver: "jp.ne.opt.redshiftfake.postgres.FakePostgresqlDriver", schema,
                CONNECTION_TIMEOUT,  maxPoolSize: 5,
                 isAutoCommit: true,  registry: null,  poolName: "redshift-dbcp",
                IDLE_TIMEOUT, VALIDATION_TIMEOUT);
    }
```

Visualize Redshift table relations

https://www.dbvis.com/

http://schemaspy.org



Columns

Related columns	Constraint	Comments

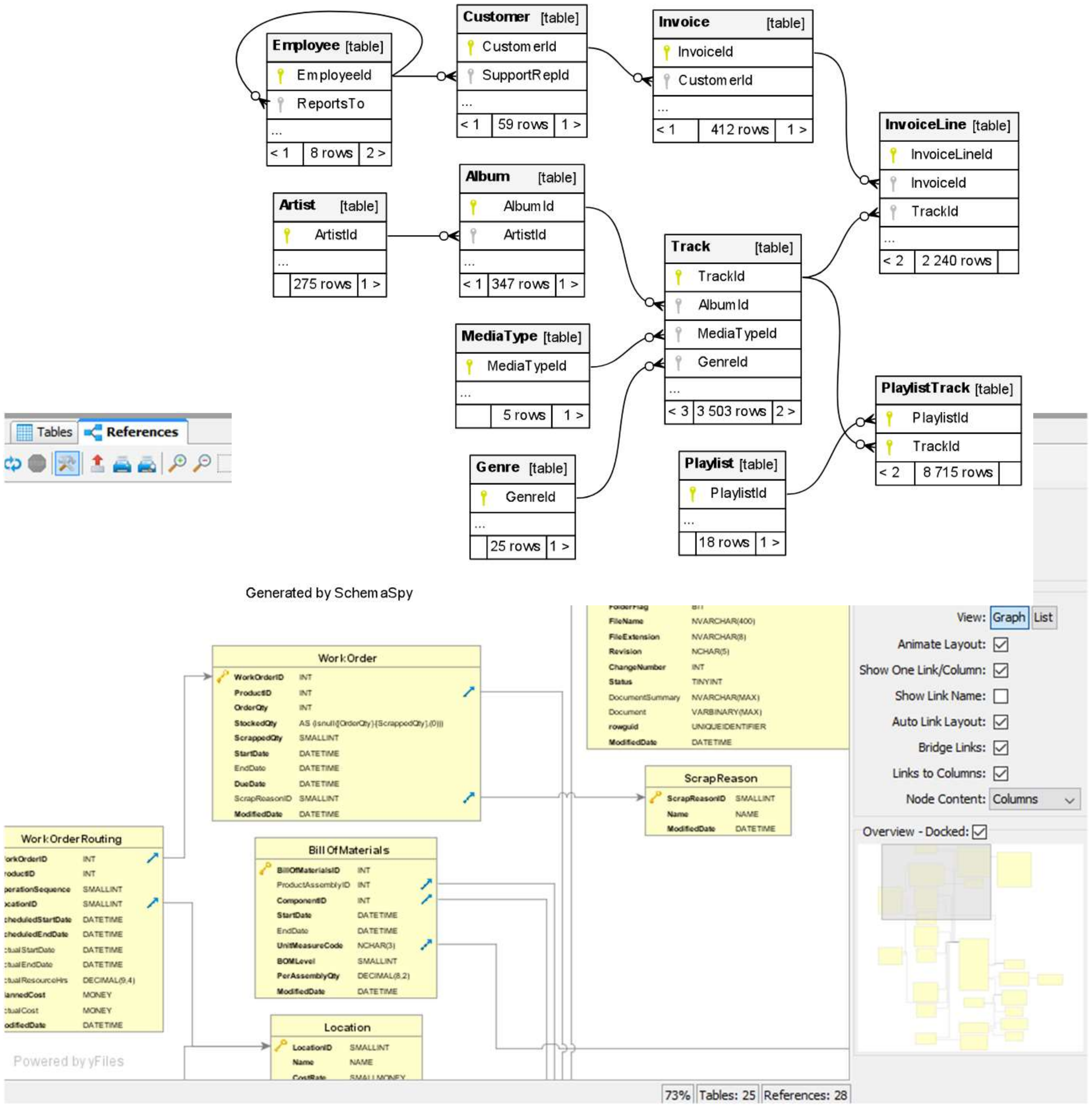
Column	Type	Size
AlbumId	int	4
Title	nvarchar	320
ArtistId	int	4

Showing 1 to 3 of 3 entries
Table contained 347 rows

Indexes

Constraint Name
PK_Album
IFK_AlbumArtistId

Showing 1 to 2 of 2 entries



Redshift JDBC «under the hood»

https://habrahabr.ru/post/345542/

Postgresql wire protocol 8.x

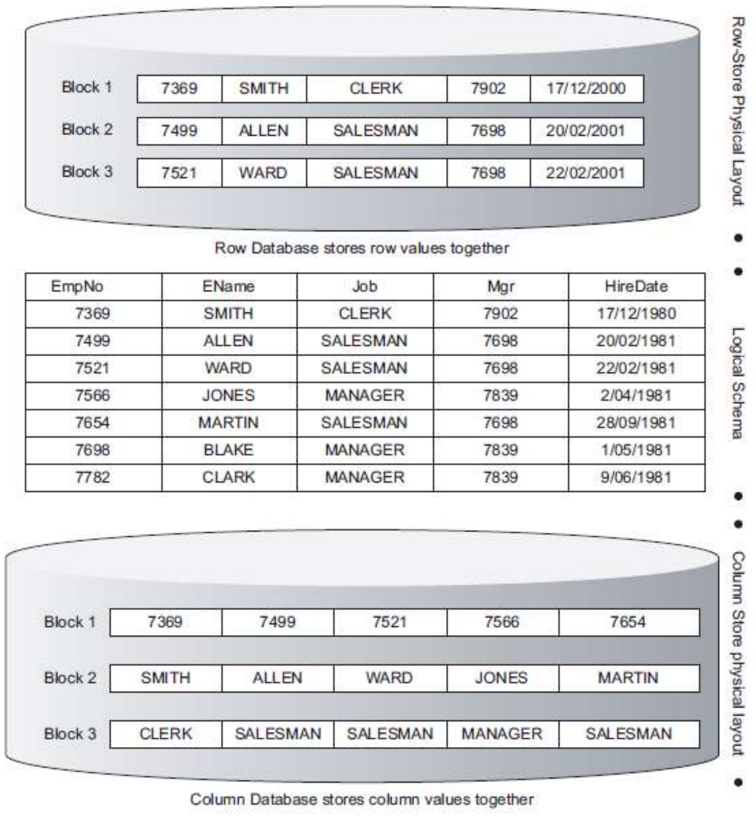
Implements jdbc:redshift and jdbc:postgresql

Fat jar driver (packaged AWS SDK) is not worked with spring boot jar applications

Analytics database

Column-oriented DBMS

Operation	Row-oriented	Column-oriented
Aggregate operations	slow	fast
Insert/Update	fast	slow
Select single record	fast	slow
Select few columns	skip unnecessary data	fast
Compression	high	low

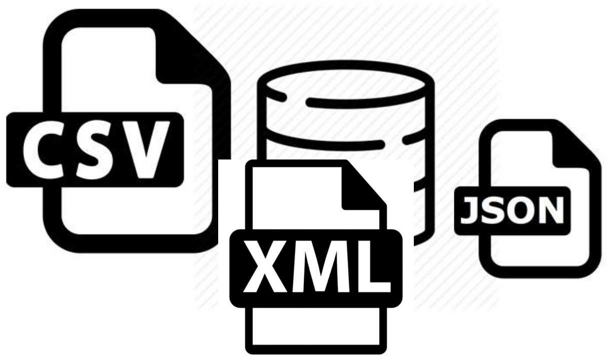

Row-Store Physical Layout
Row Database stores row values together
Logical Schema
Column Store physical layout
Column Database stores column values together

Block 1	7369	SMITH	CLERK	7902	17/12/2000
Block 2	7499	ALLEN	SALESMAN	7698	20/02/2001
Block 3	7521	WARD	SALESMAN	7698	22/02/2001

EmpNo	EName	Job	Mgr	HireDate
7369	SMITH	CLERK	7902	17/12/1980
7499	ALLEN	SALESMAN	7698	20/02/1981
7521	WARD	SALESMAN	7698	22/02/1981
7566	JONES	MANAGER	7839	2/04/1981
7654	MARTIN	SALESMAN	7698	28/09/1981
7698	BLAKE	MANAGER	7839	1/05/1981
7782	CLARK	MANAGER	7839	9/06/1981

Block 1	7369	7499	7521	7566	7654
Block 2	SMITH	ALLEN	WARD	JONES	MARTIN
Block 3	CLERK	SALESMAN	SALESMAN	MANAGER	SALESMAN

Data Lake



Analytics database: Redshift

Based on Postgresql 8.0.2 fork (ParAccel MPP)

v8.0.2 – 2005-04-07

+ AWS service integration, AWS hosted/managed

+ Regular SQL JOINs, support subqueries etc

- constraints, transactions

- Too small function set

- Scaling up downtime

- «slow» inserts/data import

Analytics database: Greenplum

Based on Postgresql 9.0

v9.0 – 2010-09-20

+ Open source PG fork

+ Support complex SQL queries

+ Rich functionality

- Scaling up and maintenance downtime

- Fork with backport of new features

Analytics database: CitusDB

Based on Postgresql 10.0

v10.0 – 2017-10-05

+ Open source PG extension

+ Use latest PG versions and leverage it recent features

+ Distributed transactions

+ Rebalance shards without downtime

- Does not allow subqueries in the WHERE clause

- JOIN a local and a distributed table

Analytics database: Druid

+ Highly optimized for web metrics tasks

+ Very high ingesting rate

- Does not yet have full support for joins

- Limited SQL support

Analytics database: ClickHouse

+ Highly optimized for web metrics tasks

- There is no global query plan for distributed query execution.

- Does not yet have full support for joins

- Limited SQL support

https://github.com/yandex/ClickHouse

https://ruhighload.com/doc/clickhouse/development/architecture.html

Analytics database: CrateDB

Elasticsearch based

+ Full text search, GIS functions

+ Presto SQL parser, PG wire protocol

+ Blob storage

- Constraints, transactions

- Query optimization

- Hash join only for 2 tables

https://habr.com/post/323742/

Analytics database: PrestoDB

+ connectors for external data format

+ dozen of functions: window functions, geo etc

+ transaction support

- primitive table statistics

- query S3 data only with Hive

Analytics database: Apache Drill

+ schema free SQL

+ query S3/HDFS data directly

- ???

Analytics database: Dremio

+ query data from S3, Redshift, Elasticsearch

+ support Apache Arrow

- small OSS community

Analytics database: Apache HAWQ

+ Interactively query Hadoop data, natively via HDFS

+ Cost-Based Optimizer

- Incubating status

Analytics database comparison

Database	Based on	JOIN large tables	Data lake	Full text search, geo data
Redshift	Postgres 8.0.2	Yes	Redshift Spectrum	No
Greenplum	Postgres 9.0	Yes	Postgres FDW	Yes
CitusDB	Postgres extension	Yes	Postgres FDW	Yes
Druid		No		
ClickHouse		No		
CrateDB	Elasticsearch, PrestoDB, Postgres wire protocol	2 tables		Yes
PrestoDB		Yes	Yes	Geo functions only
Apache Drill		Yes	Yes	
Dremio		Yes	Yes	
Apache HAWQ	Greenplum	Yes	Yes	

?
Q
&
A
!

JavaCro'18

Thank you!

igor.suhorukov@gmail.com
github.com/igor-suhorukov