



# Tools for Zero Downtime Migrations

For relational databases

Daniane P. Gomes



## Agenda

- Zero downtime deployment techniques.
- Migrations for relational databases.
- Tools to handle and automatize migrations.
- Little demo.
- Use case for continuous delivery pipeline with GitLab and Amazon Web Services.



# Zero Downtime Deployment

1

## Blue green deployments.

Old is called blue, new version is called version green.

Transfer traffic to green and if something goes wrong, transfer it back to blue.

Active user sessions won't be impacted.

[What is blue green deployment \(Red Hat\)](#)

2

## Feature toggle.

A piece of code to turn on/off functionalities during runtime.

A new release can be rolled out to only a fraction of the users.



# What about the database?

Nice theory, but what about the database?

Once a table or a column is dropped, there's no turning back...

How to handle that?





# What is a migration?

- All database changes are migrations.
- DDL or DML commands.

```
1 INSERT INTO person(name, powers) VALUES('Daniane P. Gomes', 'Knows what a proparoxitona is');
2 INSERT INTO person(name, powers) VALUES('Franckson Sirena', 'Cooks delightfully');
3 INSERT INTO person(name, powers) VALUES('Hermione Granger', 'Psionic Blast');
4 INSERT INTO person(name, powers) VALUES('Thomas Shelby', 'Waterbreathing');
5 INSERT INTO person(name, powers) VALUES('Daario Naharis', 'Superhuman Invention Skill');
6 INSERT INTO person(name, powers) VALUES('June Osborne', 'Danger Warning');
7 INSERT INTO person(name, powers) VALUES('Amy Farrah Fowler', 'Super Mentality');
8 INSERT INTO person(name, powers) VALUES('Yusuke Urameshi', 'Telepathy');
9 INSERT INTO person(name, powers) VALUES('Lagertha', 'Phasing');
10 INSERT INTO person(name, powers) VALUES('Eleanor Shellstrop', 'Super Learning');
```

```
1 CREATE TABLE `person` (
2
3   `id` int NOT NULL AUTO_INCREMENT PRIMARY KEY,
4   `name` varchar(20),
5   `powers` varchar(100)
6
7 )ENGINE=InnoDB DEFAULT CHARSET=UTF8;
```



# Good practices for database migrations

1

Keep backward and forward compatibility.

3

Keep scripts version and code together.  
Submit them to code review.

2

Have your own local database instance.

4

Write small changes and integrate often  
with the main database.



**Backward compatibility:**  
**Newer code can read data that was written by older code.**

**Forward compatibility:**  
**Older code can read data that was written by newer code.\***

\* older code to ignore additions made by a newer version.

[Martin Kleppmann - Designing Data-Intensive Applications](#)



# Backward and forward compatibility

Do NOT perform destructive changes!

Do NOT drop tables or columns at once!

Do NOT alter columns at once!





# Backward and forward compatibility

It takes more effort but it's safe!

How rename a column without downtime?



1. Add a new column
2. Code reads from the old column and writes on both
3. Shard data while copying from old column to new (break large chunks into small ones)
4. Code reads from the new column and writes on both
5. Code reads and writes from the new column
6. Delete the column (future)

Edson Yanaga



# Backward and forward compatibility

Example: renaming column 'name' to 'fullname'.

[GitHub repo.](#)

```
8 /**
9  * SAMPLE CODE!!! This service is an SAMPLE example on how to handle a column
10 * rename migration. It's not suitable for production environments.
11 *
12 * @author Daniane
13 *
14 */
15 @Service
16 public class PersonCompatibilityService_State2 {
17
18     private PersonRepository repository;
19
20     public PersonCompatibilityService_State2(PersonRepository repository) {
21         this.repository = repository;
22     }
23
24     // State 2 for application and V1_4_0 for database: Code reads from the old
25     // column and writes on both
26     // V1_4_1 is also executed and data is copied in small shards
27     public void save(String name) {
28         Person p = new Person();
29         p.setName(name);
30         p.setFullname(name);
31         repository.save(p);
32     }
33
34     public String findNameById(Long id) {
35         return repository.findById(id)
36             .orElseThrow()
37             .getName();
38     }
39 }
40
```

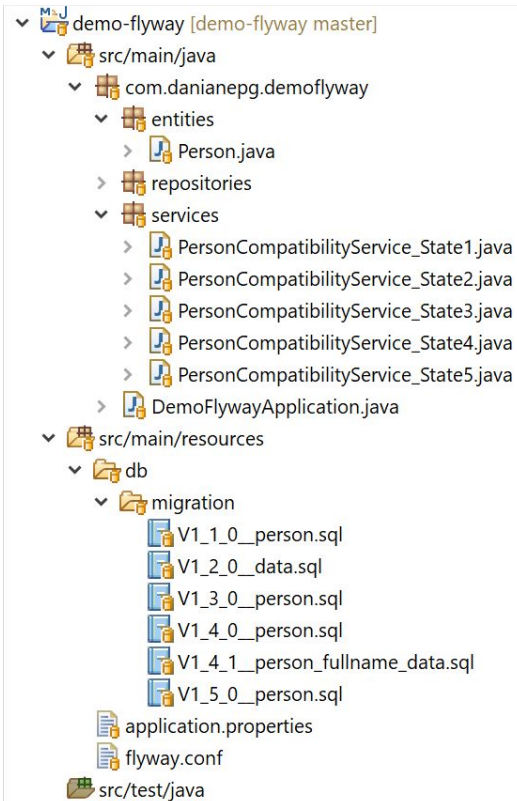


# Local database instance

- With container images (e.g. Docker) it's easy to have a local instance of the database.
- Easily re-create the container using the base image.
- Developers can perform changes and tests without impact peers.
- Integrate continuously to avoid conflicts.

# Keep migrations and code together

- Everything you need to run the application is grouped.
- Use version control.






## Keep it small

- Split multiple scripts in several migrations.
- Shard updates to avoid long locks.

```
1 /* Shard data */  
2 UPDATE person SET fullname = name WHERE id > 0 and id <=5;  
3 UPDATE person SET fullname = name WHERE id > 5 and id <=10;
```



# Automatize your migrations



Liquibase.org



Flyway





# Liquibase

- Handles multiple file formats: xml, yaml, sql, json.
- Can perform rollbacks (paid version).
- Generates scripts automatically.
- <https://www.liquibase.org/>





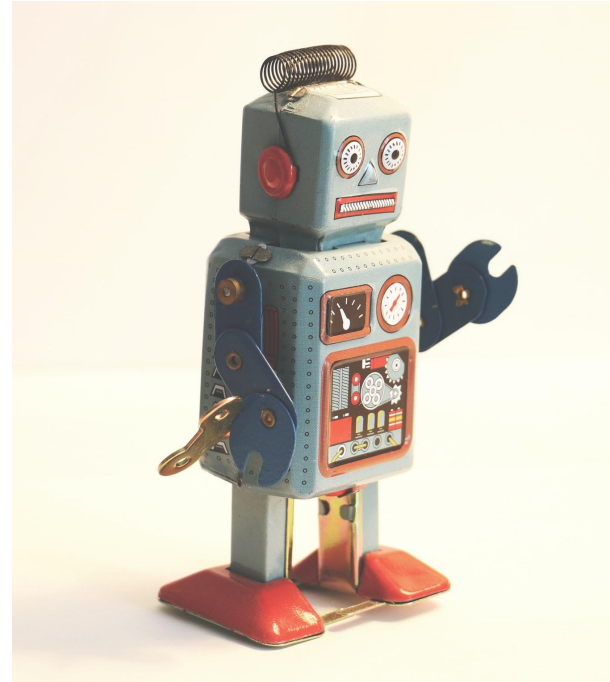
# Flyway

- Simple.
- Handles sql files.
- Naming convention to define scripts order.
- <https://flywaydb.org/>



## Little Demo :)

- Can be triggered on application startup.
- Can be trigger by Maven.
- Can be trigger by command line tool.





# Use case

- Java 11 & Spring Boot
- MySQL
- GitLab
- AWS services:
  - ECR to manage container images.
  - ECS to run images.
  - Lambda to run code without manage servers.
  - CloudWatch Alarms for logs and alarms.
- Docker
- Flyway



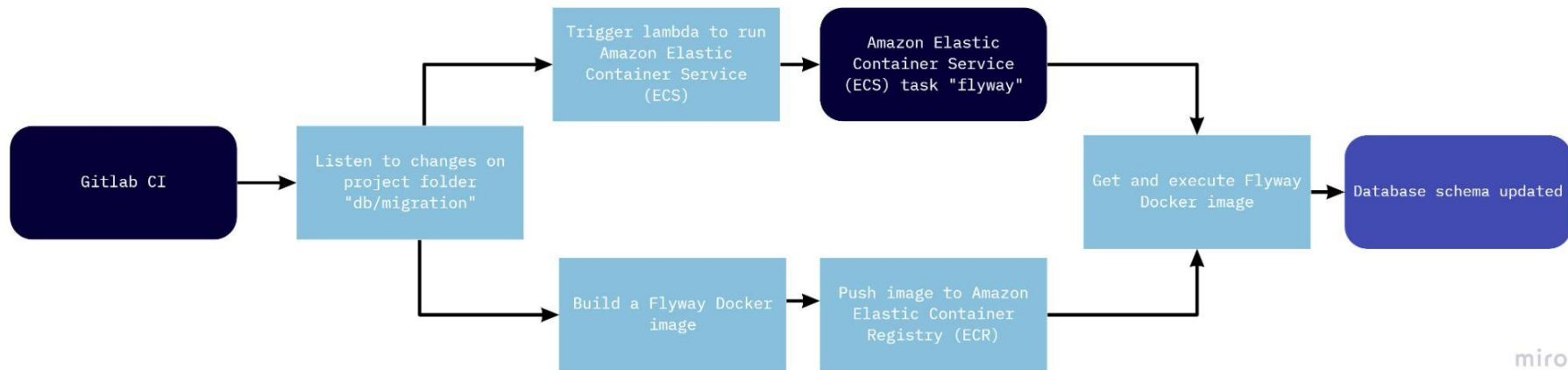


## Use case

- [Amazon Elastic Container Registry \(ECR\)](#): to push/pull and store my Docker image.
- [Amazon Elastic Container Service \(ECS\)](#): to get and run the my Docker image.
- [Amazon Lambda](#): to trigger my ECS task (container).



# Use case





## Use case: build the Docker image

- GitLab CI is listening to changes pushed into a repository folder.
- A Docker image is created containing Flyway image.
- SQL scripts are copied to the image.
- Image is pushed to [Amazon Elastic Container Registry \(ECR\)](#) (to manage container images).

```
1 # Get image "flyway" from Flyway's repository
2 FROM flyway/flyway
3
4 WORKDIR /flyway
5
6 # Database credentials
7 COPY db/flyway.conf /flyway/conf
8
9 # Add the scripts I've pushed to my project folder to the Docker image
10 ADD db/migration /flyway/sql
11
12 # Execute the command migrate
13 CMD [ "migrate" ]
```

Dockerfile.sh hosted with ❤ by GitHub

[view raw](#)



## Use case: execute the Docker image

- An [Amazon Lambda](#) (service to run serverless code without provisioning servers) is triggered at the same time.
- Lambda runs [Amazon Elastic Container Service \(ECS\)](#) (to run containers) task.
- Docker image is executed and command 'migrate' is triggered.
- Database schema is updated.

# Use case: the AWS Lambda

- [Gist](#).

```
myLambdaToTriggerFlyway.js
1  /**
2   * myLambdaToTriggerFlyway
3   *
4   * This lambda relies on 3 environment variables: ENV_CLUSTER, ENV_SUBNET, ENV_SECURITY_GROUP.
5   *
6   */
7
8  const aws = require('aws-sdk');
9  const ecs = new aws.ECS();
10
11  exports.handler = async (event, context) => {
12
13      let taskDefinition = null;
14
15      const CLUSTER = process.env.ENV_CLUSTER;
16      const SUBNET = process.env.ENV_SUBNET.split(",");
17      const SECURITY_GROUP = process.env.ENV_SECURITY_GROUP;
18      const LAUNCH_TYPE = "FARGATE";
19      const FAMILY_PREFIX = "flyway";
20      const CONTAINER_NAME = "flyway";
21
22      const taskParams = {
23          familyPrefix: FAMILY_PREFIX
24      };
25
26      let listTaskDefinitionsResult;
27      try {
28          listTaskDefinitionsResult = await ecs.listTaskDefinitions(taskParams).promise();
29      } catch(e) {
30          console.error("Error! failed to load task definition", e);
31      }
32
33      if (listTaskDefinitionsResult) {
34          taskDefinition = listTaskDefinitionsResult.taskDefinitionArns[listTaskDefinitionsResult.taskDefinitionArns.length-1];
35          taskDefinition = taskDefinition.split("/")[1];
36      }
37
38      const params = {
39          cluster: CLUSTER,
40          count: 1,
41          launchType: LAUNCH_TYPE,
42          networkConfiguration: {
43              "awsVpcConfiguration": {
44                  "subnets": SUBNET,
45                  "securityGroups": [SECURITY_GROUP]
46              }
47          },
48          taskDefinition: taskDefinition,
49      };
50
51      const runTaskResult = await ecs.runTask(params).promise();
52
53      if (runTaskResult.failures && runTaskResult.failures.length > 0) {
54          console.log("Error!");
55          console.log(runTaskResult.failures);
56      }
57
58      return runTaskResult;
59  };
60  }
```





## Use case: what if something goes wrong

- Cloudwatch alarms will send an email to the team.
- After fix scripts, there are commands, such as
  - 'validate' - validate if the migrations applied match the scripts.
  - 'repair' - to repair the history table.
- Commands will be executed manually.



## Use case: why this structure

- To keep it decoupled.
- When a script is changed it's not necessary to update the whole application.
- Couldn't I apply the migrations on application startup?
  - Kind of... but it's not suitable for multiple application instances.
- Lambdas make the process more flexible: can send emails, get the execution results, statistics, etc.



# References

- [Evolutionary Database Design - Pramod Sadalage and Martin Fowler](#)
- [Talk: Migrating to Microservice Databases - Edson Yanaga](#)
- [Book: Microservice database migration guide - Edson Yanaga](#)



# Appendix

- <https://github.com/danianepg/demo-flyway>
- <https://github.com/danianepg/demo-liquibase>
- [Building a continuous delivery pipeline for database migrations with GitLab and AWS](#)
- [Database Migrations with Liquibase and Flyway](#)
- [Collaborative book in progress \(pt-BR\): Deploy em produção para desenvolvedores - Capítulo Migrações em Banco de Dados Relacionais](#)





# Thank you! Enjoy TDC!

Daniane P. Gomes

E-mail | [danianepg@gmail.com](mailto:danianepg@gmail.com)

Twitter | [danianepg](https://twitter.com/danianepg)

Medium | [danianepg.medium.com](https://medium.com/@danianepg)

LinkedIn | [linkedin.com/in/danianepg](https://linkedin.com/in/danianepg)