

# Development of infrastructure

For the test task EastRockValley - steps of the development.

The task description

- Accepted assumptions:
  - Realization:
  - S3
  - Airflow
  - MySQL
  - Metabase (Reporting)
- 

## Accepted assumptions:

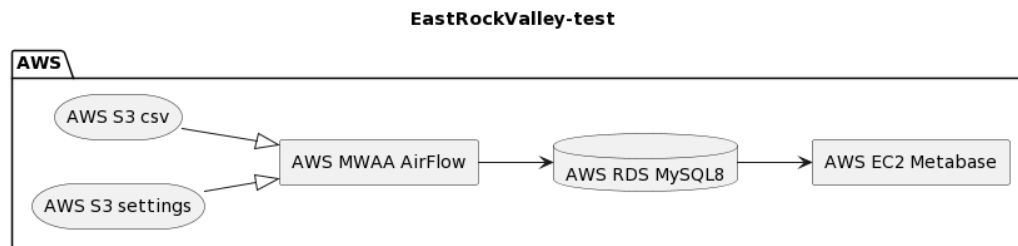
Based on the task, we operate with incoming data.

1. Apparently, this is not the original data on transactions and actions of player users. These are the amounts of time that players spend in a particular game, already aggregated for a date (day). We do not know anything about bets or gameplay results. In another file, we see player profiles. In the third file, we see financial transactions. Since Transaction ID is present, it is no longer possible to call this data aggregated to the date (day). So, we assume that information about the exact time and order of transactions is lost or not important. Let's assume the UserID in files 'games' and 'payments' contains the UserID which was imported before or exists in the current file 'users'.
2. Let us assume that these three files are all that we know about the data to be processed.
3. Let's assume that this data appears in our online storage in the form of CSV files.

## Realization:

Chosen platform and software: AWS S3, MySQL 8 RDS, AirFlow (MWAA and Docker), Metabase (EC2)

Software: IE (Chrome), Github, DataGrip, PyCharm, VSCode

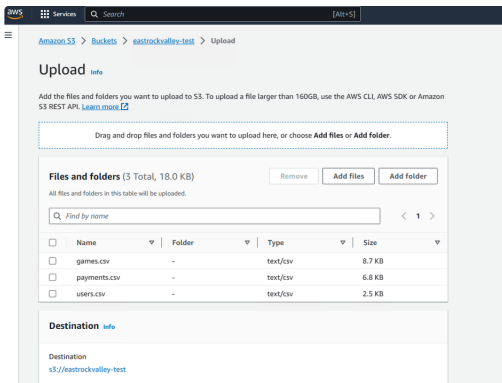


common UML schema

---

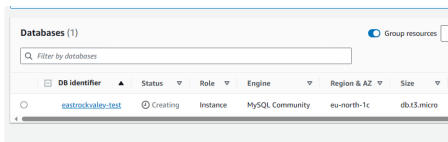
## S3

Create the S3 storage for the incoming files:



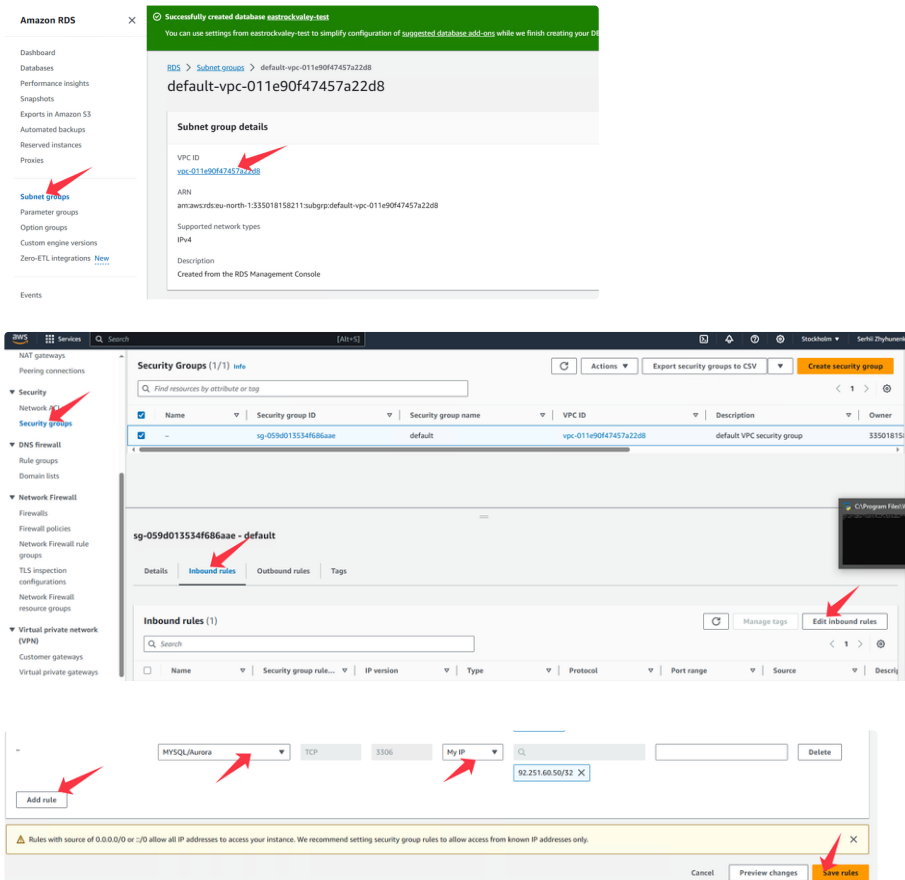
3 files uploaded

Create the RDS database in AWS. For this example, I created the MySQL database

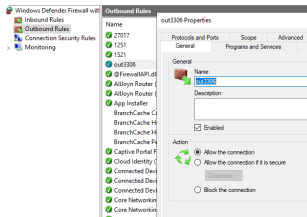
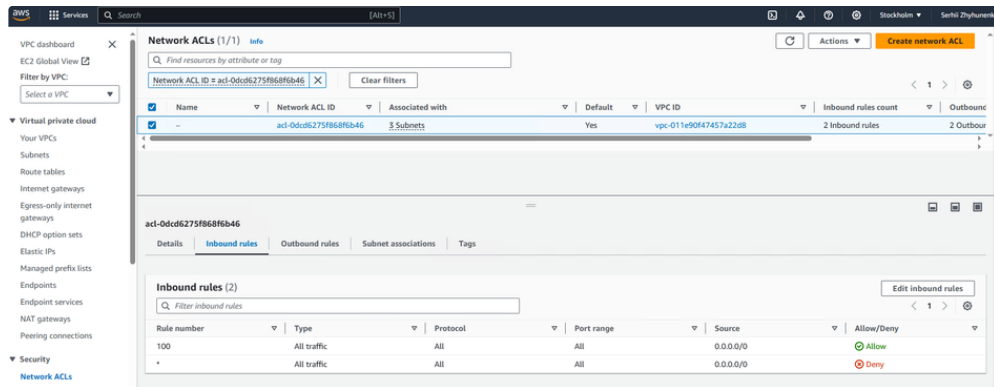


MySQL 8, inside AWS. No special settings

For the new database, need to open inbound rules to operate it remotely:

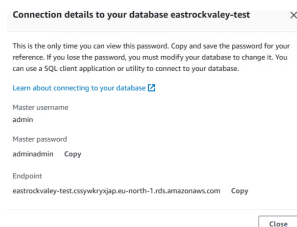


Should be set for whitelisted IP of the company's office for sure



well I have to open the MySQL  
port in my firewall as well

The database was created and I can note its creds:



simple password was taken  
for example

Before doing any code, let's create the project in GitHub for this:

### Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?  
[Import a repository.](#)

Required fields are marked with an asterisk (\*).

#### Repository template

No template

Start your repository with a template repository's contents.

Owner \*

Repository name \*

eastrockvalley-test is available.

Great repository names are short and memorable. Need inspiration? How about [jubilant-pancake](#)?

Description (optional)

the test task for EastRock

Public

Anyone on the internet can see this repository. You choose who can commit.

Private

You choose who can see and commit to this repository.

Initialize this repository with:

☐ Add a README file  
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

gitignore template: None

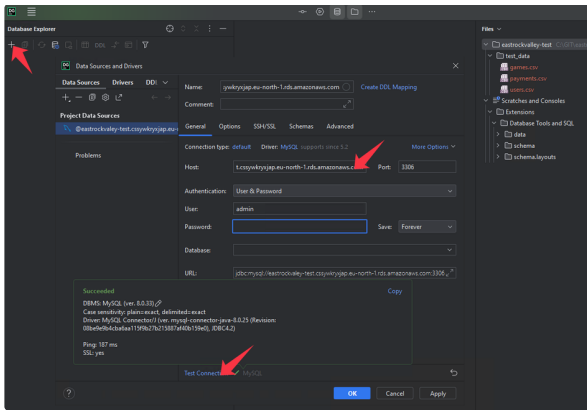
Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None

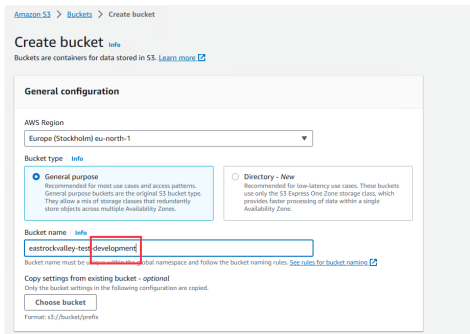
A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

Now everything is ready to create the project and connect my DataGrip to the new database and Github:

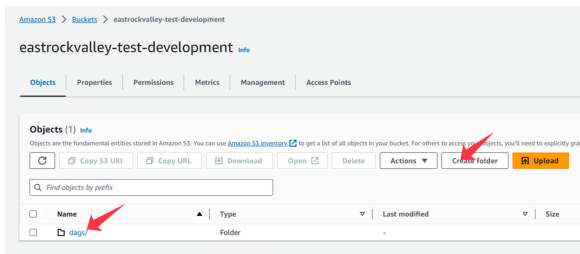


Before MySQL code, will put here all the other files of the project

Before creating AirFlow instance in AWS we need to create another bucket for the code of DAGs:



I did it again in S3



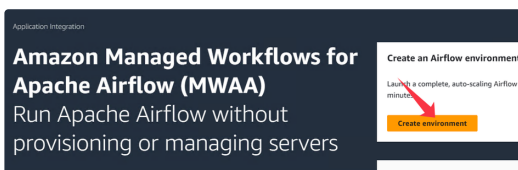
and folders in it (dags, plugins)

## Airflow

Create file `requirements.txt` with two additional Python packages:

download from the site [pandas](#) last wheel, zip it into `pligins.zip`, upload it in the development bucket, and select for airflow (not on images)

Later I decided to use `airflow.providers.amazon.aws.transfers.s3_to_sql` instead



**Environment details** [info](#)

Name


Use only letters, numbers, dashes, or underscores. Max 80 characters.

Airflow version  
2.7.2 (latest)

Weekly maintenance window start (UTC)  
Friday 02:30

**DAG code in Amazon S3** [info](#)

Amazon MWAA uses your Amazon S3 bucket to load your DAGs and supporting files. Specify your S3 bucket, and the paths of your DAG folder, plugins.zip, and requirements.txt.



Create or specify an S3 bucket to store your DAG code. The bucket must have versioning enabled. You can create a new bucket in the [Amazon S3 console](#).

S3 Bucket  
The S3 bucket where your source code is stored. Enter an S3 URI or browse and select a bucket.  
 View [Browse S3](#)

Format: s3://mybucketname

Cancel **Next**

Browse the created before DAG bucket and requirements.txt

**Environment details** [info](#)

Airflow version  
2.7.2 (latest)

Weekly maintenance window start (UTC)  
Friday 02:30

**DAG code in Amazon S3** [info](#)

S3 Bucket  
The S3 bucket where your source code is stored. Enter an S3 URI or browse and select a bucket.  
 View [Browse S3](#)

Format: s3://mybucketname

DAGs folder  
The S3 bucket folder that contains your DAG code. Enter an S3 URI or browse and select a folder.  
 View [Browse S3](#)

Format: s3://mybucketname/mydagfolder

Plugins file - optional  
The S3 bucket URI file that contains your DAG plugins. Enter an S3 URI or browse and select a file object and version.  
 Choose a version View [Browse S3](#)

Format: s3://mybucketname/myplugins.zip

Requirements file - optional  
The S3 bucket URI file that contains your DAG requirements.txt. Enter an S3 URI or browse and select a file object and version.  
 Choose a version View [Browse S3](#)

Format: s3://mybucketname/myrequirements.txt

Startup script file - optional  
The shell script will be executed for customized image. Enter an S3 URI or browse and select a file object and version.  
 Choose a version View [Browse S3](#)

Format: s3://mybucketname/startup.sh

Cancel **Next**

Amazon MWAA > Environments > Create environment

Step 1  
[Specify details](#)

Step 2  
**Configure advanced settings**

Step 3  
Review and create

**Configure advanced settings**

**Networking**

Virtual private cloud (VPC) [info](#)  
Subnet(s) representing subnetwork setup of your Airflow environment. An environment needs 2 private subnets in diff zones. To create a new VPC with private subnets, choose Create MWAA VPC. [Learn more](#)

Create MWAA VPC [info](#)

Subnet 1  
Private subnet for the first availability zone. Each environment occupies 2 availability zones.

[info](#)

Subnet 2  
Private subnet for the second availability zone. Each environment occupies 2 availability zones.

[info](#)

[VPC and subnet selections can't be changed after an environment is created.](#)

The creation of a cheap airflow instance can take a lot of time on AWS side. Have to wait before it's available.

While it is going, we can prepare the DAG.

Prepare the Python on a local computer:

(full code in GIT env.txt)

Install or update the latest version of the AWS CLI - AWS Command Line Interface (amazon.com)

Create and retrieve and set AWS access keys on the local computer:

**Access key created**  
This is the only time that the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key.

IAM > [Security credentials](#) > Create access key

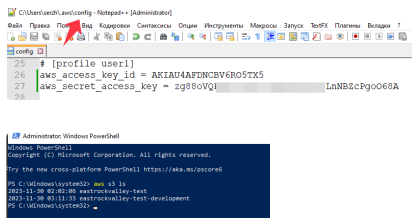
Step 1  
[Alternatives to root user access keys](#)

Step 2  
**Retrieve access key**

**Retrieve access key** [info](#)

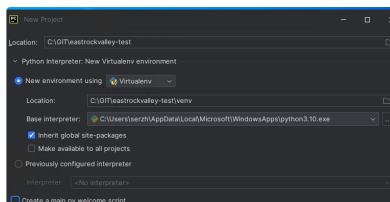
Access key  
If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make it your default.

Access key  Secret access key  [Show](#)

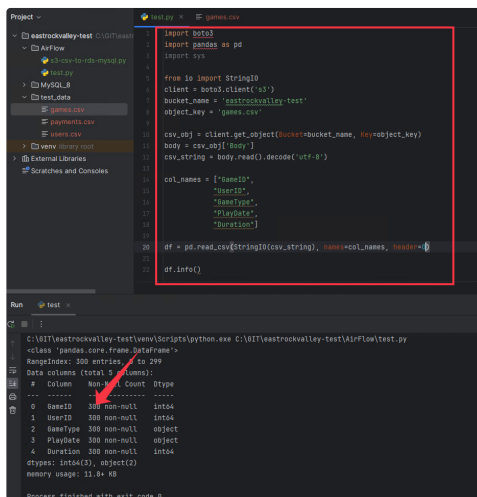


check the s3 is available  
from aws cli

Open PyCharm and create a new project in the same folder

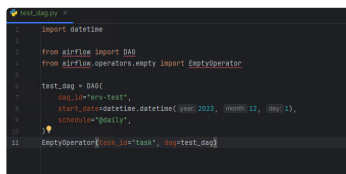


We can create a small test app to check the data availability:



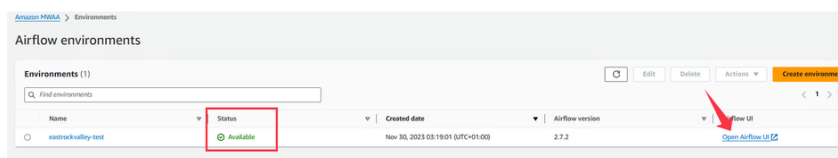
300 is a number of rows of data are available

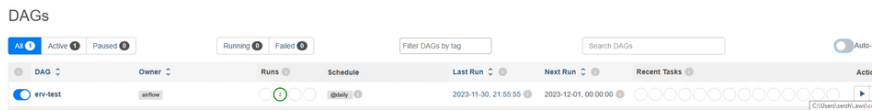
Also, we can create some test DAG .py file



and put it into Dags folder (bucket for development)

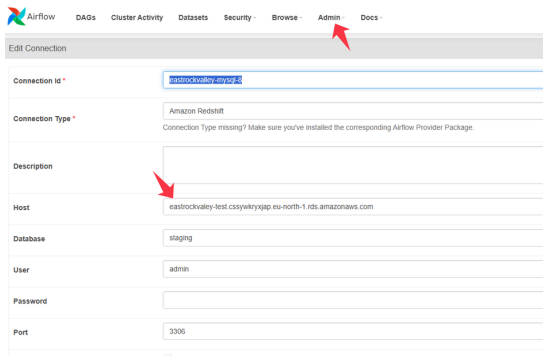
Once Airflow is ready, we can open its user interface



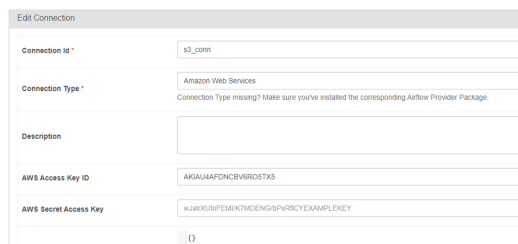


and we can see the test DAG can be triggered

In AirFlow UI Go to Admin - connections add next two:



connection to MySQL



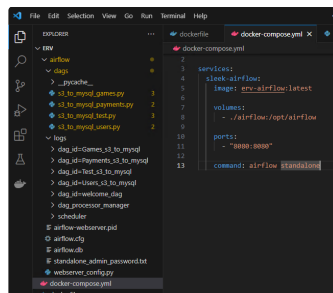
S3 connection

Time to write our S3-to-MySQL DAG code:

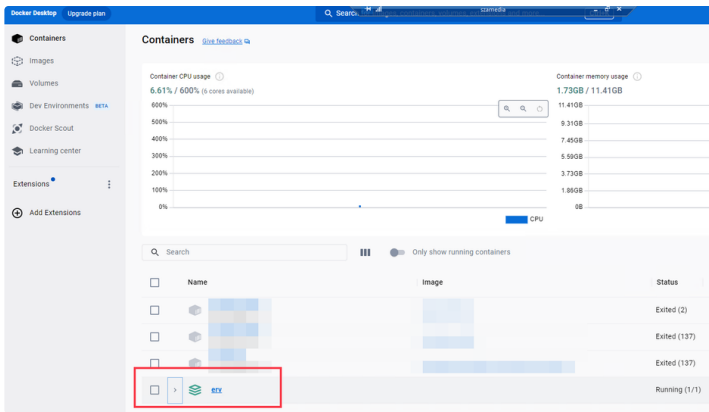
(presented in GitHub and attached).

I found I did not want to use pandas and took `S3ToSqlOperator` from Amazon provider.

Also, I found out that Amazon started billing me real money for Airflow usage online, So I decided to delete this online instance and rebuild it locally on Docker:

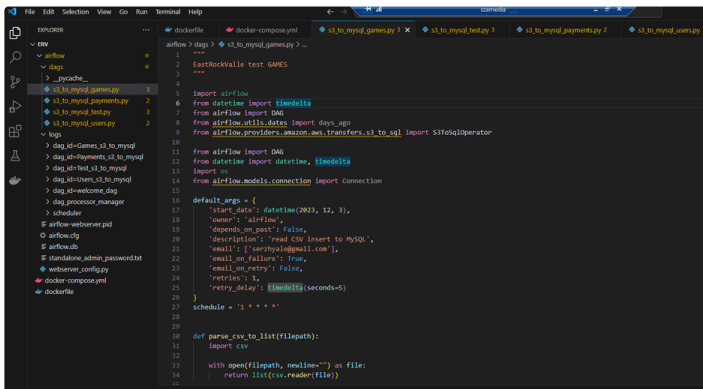


New project in VSCode with  
docker compose (windows)

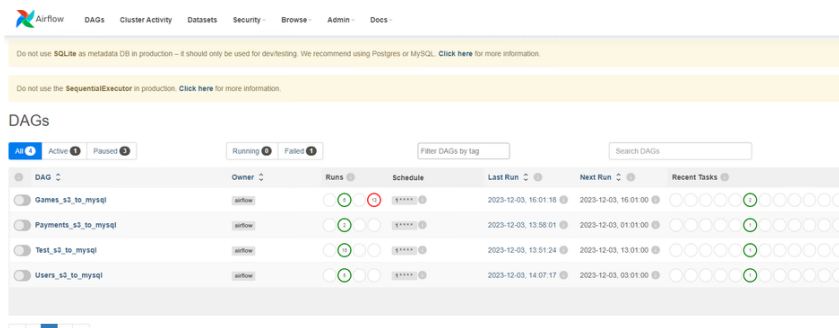


docker Airflow container

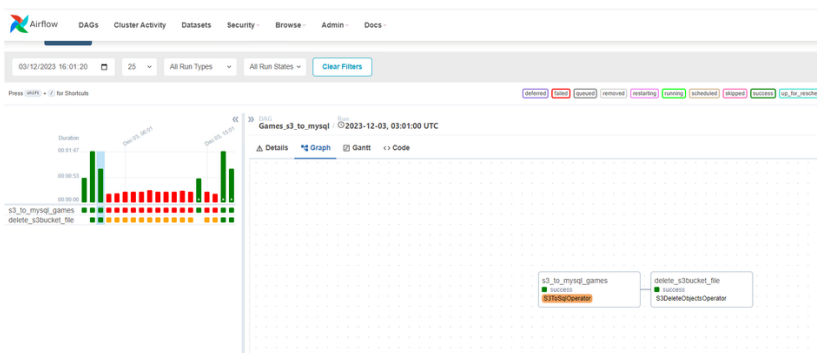
I repeat the connections in this local Airflow instances (as shown above for online)



put all dags in the Airflow docker-linked folder



dags ready to be triggered



triggering DAGs to move data to MySQL database and delete the source file

In my scenario, new files will appear in the same place with the same name.



```

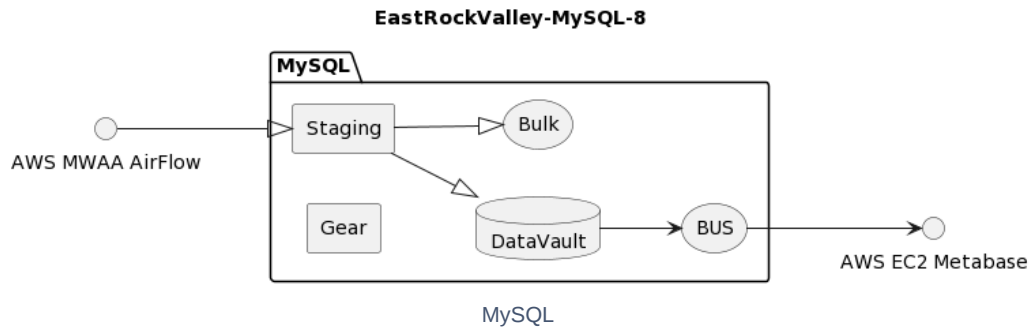
select * from staging-games g;
select * from staging-payments p;
select * from staging-users u;

```

userID	username	signupdate	country	imports
1	User1	20/10/2023	ES	2023-12-01
2	User2	02/11/2023	IT	2023-12-01
3	User3	10/10/2023	DE	2023-12-01
4	User4	10/10/2023	USA	2023-12-01

In MySQL we can see data is coming

## MySQL



MySQL code. Return to DataGrip

(Code is presented in Github or attached)

schema Staging - a place for landing all incoming data 1:1

schema Bulk – to archive already processed data

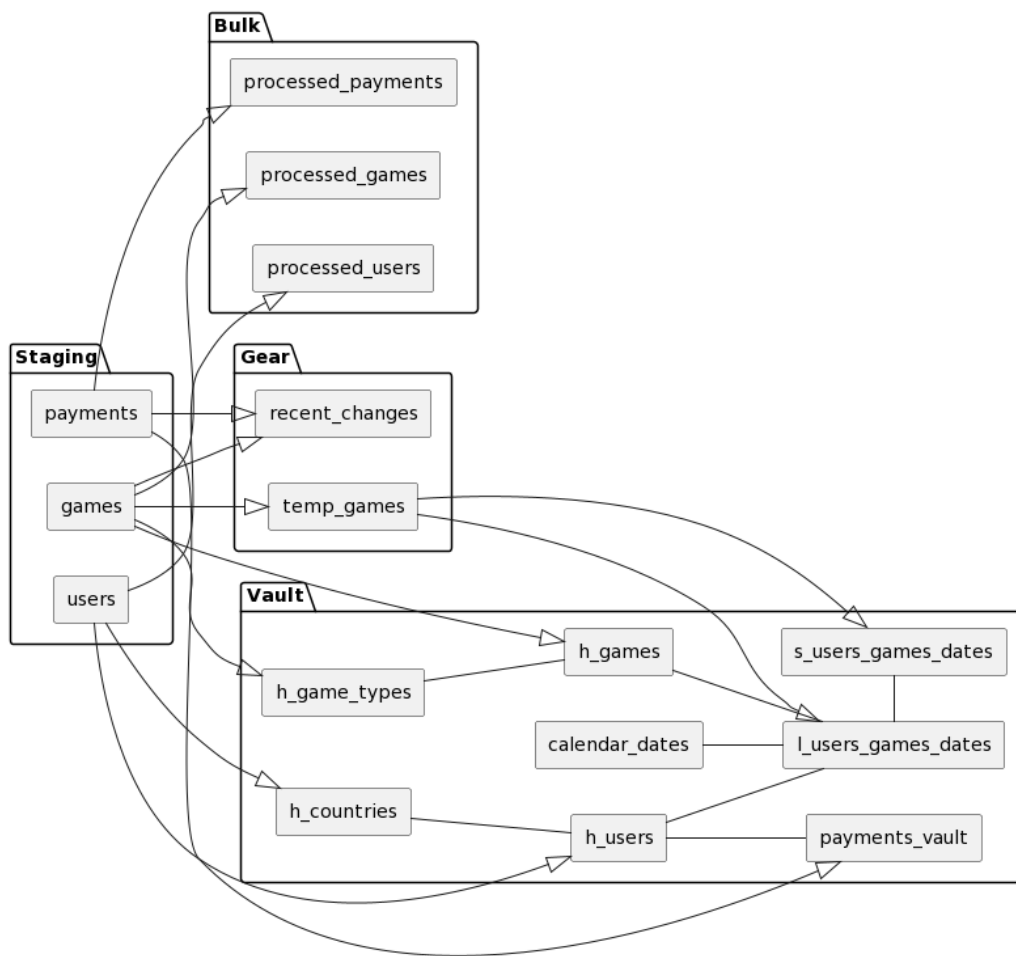
schema Gear - all processing stored procedures (routines) and temporary tables

schema Vault - data storage with datavault policy

schema BUS – for report layer aggregated tables and views. Source for Reporting System

calendar\_dates - a table of dates generated additionally (taken from my other project)

## EastRockValley MySQL IMPORT



uml schema presented in txt file as well

The landing is in an “every 10 minutes” event:

```

--
DROP PROCEDURE IF EXISTS gear_exec_process_staging;
CREATE PROCEDURE gear_exec_process_staging()
BEGIN
    -- report
    CALL gear_exec_process_users();
    CALL gear_exec_process_games();
    CALL gear_exec_link_users_and_games();
    CALL gear_exec_process_payments();
END;

DROP EVENT IF EXISTS gear_ten_minutesRoutine;
CREATE EVENT gear_ten_minutesRoutine
ON SCHEDULE EVERY 10 MINUTE
STARTS '2023-12-01 00:00:01' ON COMPLETION PRESERVE ENABLE DO
CALL gear_exec_process_staging();

```

The next step is aggregation. This step is independent of the landing one.

By the code, it is waiting for the changes in table “recent\_changes” and recalculates only aggregations for the updated (added) period and for updated (added) group elements. Everything is realized as a series of stored procedures.

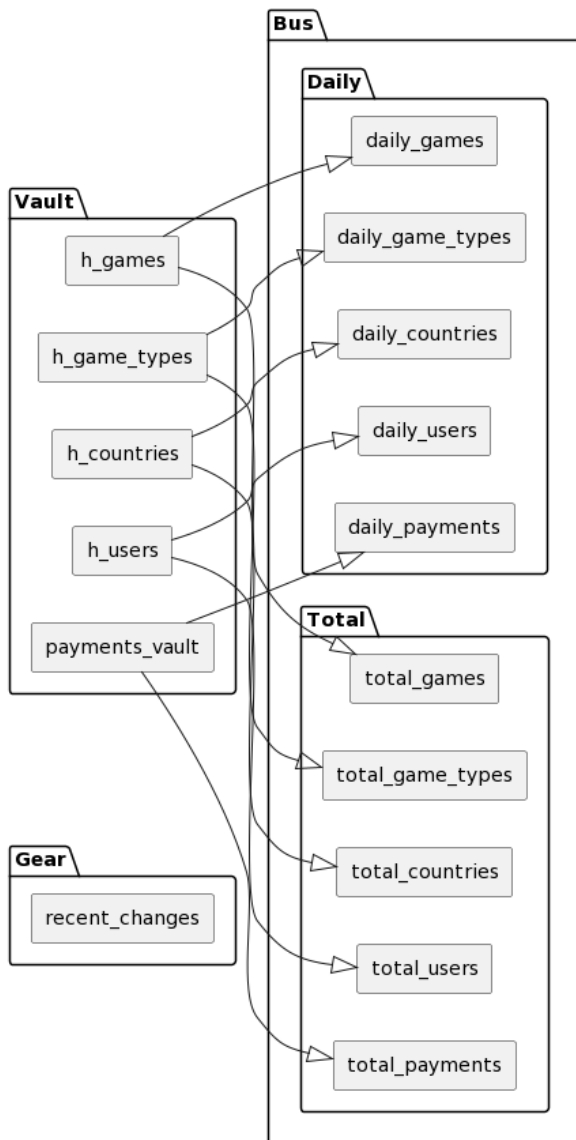
```

--
DROP EVENT IF EXISTS gear_scheduleRoutine_aggregations;
CREATE EVENT gear_scheduleRoutine_aggregations
ON SCHEDULE EVERY 10 MINUTE
STARTS '2023-12-01 00:00:01' ON COMPLETION PRESERVE ENABLE DO
CALL gear_calc_aggregations();

DROP EVENT IF EXISTS gear_scheduleRoutine_aggregations;
CREATE EVENT gear_scheduleRoutine_aggregations
ON SCHEDULE EVERY 24 HOUR
STARTS '2023-12-01 00:00:01' ON COMPLETION PRESERVE ENABLE DO
CALL gear_aggregate_total_payments();

```

## EastRockValley MySQL PRESENTATION



BUS layer is going to be shown in Metabase

## Metabase (Reporting)

Take JAR file of the free version of Metabase from <https://www.metabase.com/start/oss/jar>

Create EC2 instance with Ubuntu in AWS

**Summary**

Number of instances:  [info](#)

Software image (AMI)  
Canonical, Ubuntu, 22.04 LTS, \_amd64\_ [read more](#)  
ami-0f8b0c4f5d15c754

Virtual server type (instance type)  
t3.micro

Firewall (security group)  
default

Storage (volumes)  
1 volume(s) - 10 GiB

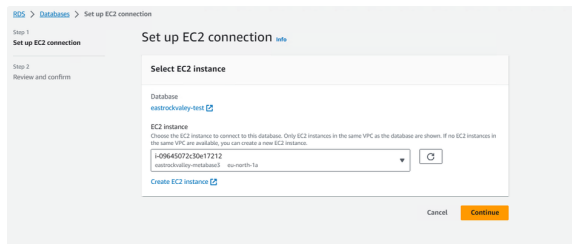
**Free tier** to your first year includes:  
750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage, 2 million IOPS, 1 GiB of snapshots, and 100 GiB of bandwidth to the Internet.

[Cancel](#) [Launch instance](#) [Review commands](#)

create pem, copy jar, yum install java-devel, java -jar metabase.jar

add inbound rule for port 3000 (metabase)

for mysql instance open connection to metabase:

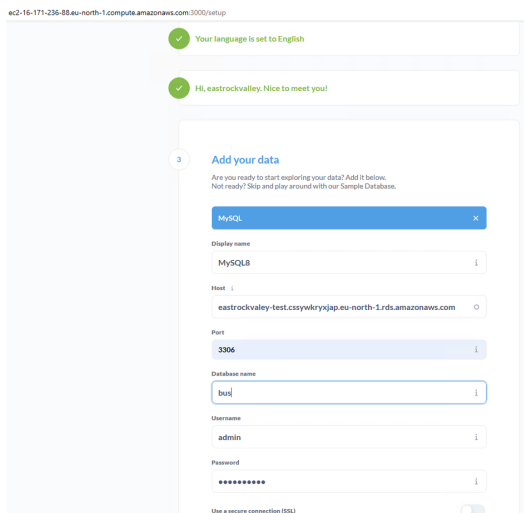


Metabase (ec2-16-171-236-88.eu-north-1.compute.amazonaws.com)

Metabase

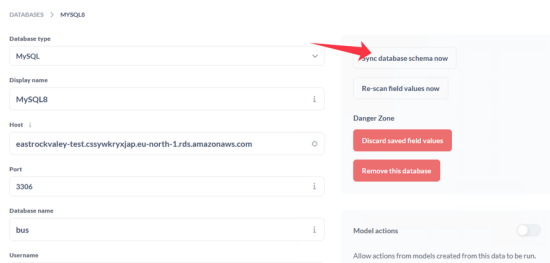
login serzhayle@gmail.com

password: eastrockvalley1



connect metabase on bus schema

Metabase (ec2-16-171-236-88.eu-north-1.compute.amazonaws.com)



refresh schema

Not secure | es2-13-51-137-193.eu-north-1.compute.amazonaws.com:3000/collection/2-users

Search...

Home

COLLECTIONS

Our analytics

Your personal collection

GAMES

PAYMENTS

USERS

DATA

Browse data

USERS

Reports about Users

Useful data

Start new explorations about USERS here

Total Users

A model

Daily Countries

A model

Total Countries

A model

Daily Users

A model

<input type="checkbox"/>	Type	Name ^	Last edited by	Last edited at
<input type="checkbox"/>	III	Most Depositors	eastrockvalley test	December 04, 2023
<input type="checkbox"/>	uI	Payments for Users (chart)	eastrockvalley test	December 04, 2023
<input type="checkbox"/>	IF	Sign-Ups and Transactions by Countries (chart)	eastrockvalley test	December 04, 2023
<input type="checkbox"/>	III	Total Users	eastrockvalley test	December 04, 2023
<input type="checkbox"/>	✓	Transactions for countries (char)	eastrockvalley test	December 04, 2023
<input type="checkbox"/>	III	Users Activity	eastrockvalley test	December 04, 2023

In Metabase all Reports placed in blocks