Data Monetization

Deployment Instruction

(based on Client Setup)

Version: 1.0

Prepared By: Michal Ostrowski

Contents

[**Overview.** 4](#_Toc148047773)

[1. Code Repository 4](#_Toc148047774)

[1.1 Repo structure 4](#_Toc148047775)

[2. Infrastructure configuration files 5](#_Toc148047776)

[3. Assumptions for Environment installation 7](#_Toc148047777)

[3. Preparation 8](#_Toc148047778)

[3.1. Verify the provided AWS account 8](#_Toc148047779)

[3.3. Use Git client to download the repositories (to local PC/workstation) 9](#_Toc148047780)

[3.4. Install WSL (Windows Subsystem for Linux) (local PC/workstation) 9](#_Toc148047781)

[3.5. Install the software required for installation (local PC/workstation) 9](#_Toc148047782)

[3.5.1 AWSCLI 10](#_Toc148047783)

[3.5.2. EKSCTL 10](#_Toc148047784)

[3.5.3 Kubectl 10](#_Toc148047785)

[3.5.4 Terraform 10](#_Toc148047786)

[3.5.5 Helm 10](#_Toc148047787)

[4. Environment Creation 12](#_Toc148047788)

[4.0. Create the Terraform State Location 12](#_Toc148047789)

[4.1. Create the key pair using ssh-keygen on WSL/Linux: 12](#_Toc148047790)

[4.2. Update Environment Configuration 13](#_Toc148047791)

[4.3. Run the main environment creation routine. 13](#_Toc148047792)

[4.7. Initialize users In RDS database 15](#_Toc148047793)

[4.8. Configure Jenkins Server 16](#_Toc148047794)

[4.8.1 Install Jenkins software on the Jenkins EC2 instance. 16](#_Toc148047795)

[4.8.2 Add EKS access to Jenkins’s server – **This step can be skipped**. 17](#_Toc148047796)

[4.11. Configuring Jenkins Pipelines 17](#_Toc148047797)

[5. Applications Deployment 22](#_Toc148047798)

[5.1. Application Configuration. 22](#_Toc148047799)

[5.1. Kubernetes components. 22](#_Toc148047800)

[5.2. Deploy Network Load Balancer for Kubernetes. 22](#_Toc148047801)

[5.3. Keycloak 24](#_Toc148047802)

[5.4. Zookeeper 25](#_Toc148047803)

[5.5. Build and deploy API Layer 25](#_Toc148047804)

[5.6. Build and deploy Airflow 25](#_Toc148047805)

[5.7. Deploy Nifi 29](#_Toc148047806)

[5.7. Configure Dashboards. 34](#_Toc148047807)

[6. Perform sanity check for environment. 39](#_Toc148047808)

# **Overview.**

This document describes the process of deploying the AMAP IE infrastructure on the Amazon Web Services.

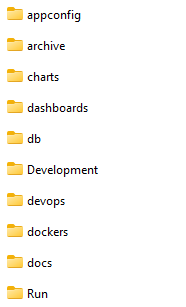
The target for installation is an AWS account with preconfigured IAM containing accounts having roles and assigned policies that allow creation and destruction of all required objects. It’s also assumed that network configuration (VPC,subnets) has been provided.

Document will describe deployment from Jenkins server pipeline installation.

From a high-level implementation perspective, we have the following AWS environments. 

* **DEV**: main development environment, will be used by developers to implement new features and functionalities in the solutions
* **SIT**: System integration test where the integration between AMAP DM and AMAP SA will be tested alongside all integration with external systems
* **UAT**: User Acceptance Test, where final demos will be conducted by the Client stakeholdesr.’
* **PROD**: Main production environment, provides Data Monetization Capabilities

# Code Repository



## Repo structure

Repository with infrastructure and application code.

Repository structure (only relevant folders has been described)

* **appconfig –** a folder where application environment specific configuration is stored.
* **charts –** a folder where open source and custom charts are being stored
* **dashboards –** a copy of CRM analytics dashboards
* **db** – Database configuration scripts. Files that are deployed initially to create users, privileges – one-time installation.
* **javaAPI\DM3-Microservices -** Application files for AMAP DM API
* **dashboards –** Afolder where AMAP IE custom dashboards are stored
* **devops –** Infrastructure deployment files (terraform, Jenkins files, application installation scripts)
* **dockers –** a folder to store custom dockerfiles
* **Run\Airflow\dags –** scripts to build airflow DAGs
* **Run\Nifi\Phase1 –** Nifi flow files and scripts used by Nifi processors
* **Run\Spark\Batch –** ETL scripts used in airflow DAGs
* **Run\Spark\Installation –** SQL scrips used in airflow to create tables in RDS DB
* **Run\Spark**\Setup **–** EMR configuration files used during cluster creation (triggered by airflow DAG) – files are deployed to S3

Jenkins pipelines has been put to dedicated directory: devops/JenkinsPipelines

The infrastructure deployment process uses several Terraform modules that, in general:

1. create the S3 bucket to store the Terraform state file
2. create the main infrastructure resources, including S3 bucket, EKS cluster, EMR configuration, RDS instance, EC2 ,Route53 zone, Jenkins instance.
3. create database users

# Infrastructure configuration files

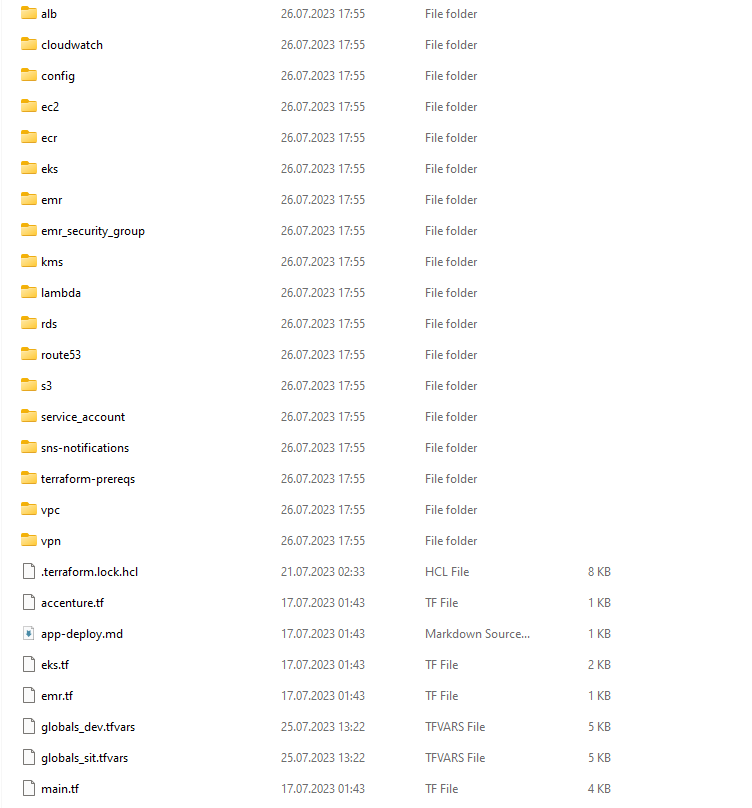
Infrastructure(terraform) configuration files are stored in AMAP Data monetization repository in devops/terraform folder:

* devops/terraform/globals\_<env>.tfvar for the environment specific folders
* terraform/config/<env> for backend configuration.

The table below contains the list and description of the Terraform submodules used:

|  |  |
| --- | --- |
| **Module directory** | **Module description** |
| alb | Creates the Application Load Balancers |
| ebs | UNUSED – creates the Elastic Block Storage volume |
| ec2/common | Configures the base EC2 resources: deployer keypair, instance profile, IAM instance role, EC2 security group |
| ec2/Jenkins | Creats the EC2 Jenkins instance |
| eks | Creates the EKS cluster, including groups, and IAM policy for EKS |
| emr | Creates IAM roles for EMR |
| emr\_security\_group | Crreates the EC2 security group for EMR |
| kms | Configures the KMS key |
| lambda | Creates the lambda functions for environment start/stop and ETL monitoring |
| rds | Creates the RDS instance |
| route53 | creates the Route53 hosted zone |
| S3 | Creates the S3 bucket and associated policy |
| service\_account | Creates the service account for EKS autoscaling including OIDC provider |
| terraform-prereqs | Creates the S3 bucket used for storing the Terraform state file |
| vpc | Creates the VPC configuration. On client accounts it’s not used as usually VPC is provided by ops teams |

Each module is stored in separate directory:



Main module(maint.tf) configures the S3 backend and launches the required submodules.

# Assumptions for Environment installation

1. Access to the relevant code repository and AWS account (via site approved solution e.g Okta or Microsoft AD)
2. Access to internet from Jenkins server (on client Account Jenkins creation is part of infrastructure setup)
3. Access to AWS accounts from Jenkins server

* Small Jenkins server is hosted each AWS accounts and access is granted by roles.

1. VPC and subnets are usually provided by account owners.

# 3. Preparation

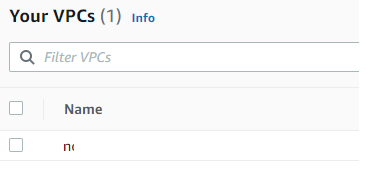
This paragraph describes steps that hast be done by Admin for every new environment.

This are one-time configurations.

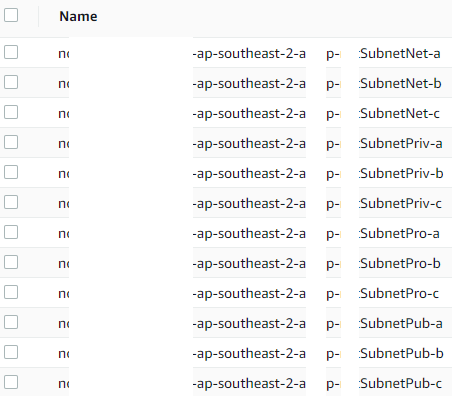
## 3.1. Verify the provided AWS account

Use your credentials to login to the provided AWS account.

Verify that each of the below items is present in the VPC (1)

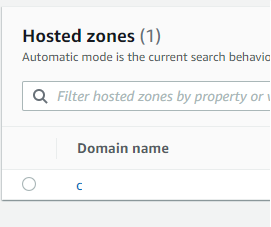


* Subnets in the VPC (12) -



The VPC requires a minimum of 2(usually 3 are created) availability zones. With 1 public and private subnet each per availability zone. The VPC should also have a subnet group for the RDS database. All resources for Data monetization should be built inside this VPC.

* Hosted zone in Route53 if provided by account owner (if not will be created by terraform module)



Also, verify that the SSL Certificate ARN has been provided( this can be verified in AWS console in ACM module)

## 3.3. Use Git client to download the repositories (to local PC/workstation)

Using a Git client, please download bitbucket repository and create proper branch:

* Development – DEV environment
* Environment/SIT – sit environment.
* Environment/UAT – uat environment
* Environment/PROD – PROD environment

## 3.4. Install WSL (Windows Subsystem for Linux) (local PC/workstation)

Please note that this step is to be skipped when performing installation from within the Linux environment.

WSL is a continuously developed software package. The latest official installation instructions can be found at the following URL:

<https://learn.microsoft.com/en-us/windows/wsl/install>

## 3.5. Install the software required for installation (local PC/workstation)

Accessing the AWS account and deploying the required software can be done using the same commands in WSL and in Linux environments.

The software to be installed is as follows:

- AWSCLI

- EKSCTL

- KUBECTL (version 1.21)

- Terraform

- Helm (version 3.8.2)

### 3.5.1 AWSCLI

To install the AWS command line interface, run the following commands:

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

aws --version

### 3.5.2. EKSCTL

To install the EKS command line interface, run the following commands:

curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /tmp

sudo mv /tmp/eksctl /usr/local/bin

eksctl version

### 3.5.3 Kubectl

To install the Kubernetes command line interface, run the following commands:

curl -LO <https://dl.k8s.io/release/v1.21.0/bin/linux/amd64/kubectl>

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

kubectl version --client

### 3.5.4 Terraform

To install the Terraform binary, run the following commands:

curl -LO https://releases.hashicorp.com/terraform/1.2.9/terraform\_1.2.9\_linux\_amd64.zip

unzip terraform\_1.2.9\_linux\_amd64.zip

sudo install -o root -g root -m 0755 terraform /usr/local/bin/terraform

terraform -help

### 3.5.5 Helm

To install the Helm binary, please run the following commands:

curl -LO <https://get.helm.sh/helm-v3.8.2-linux-amd64.tar.gz>

tar xzvf helm-v3.8.2-linux-amd64.tar.gz

cd linux-amd64/

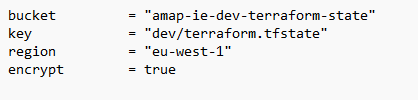
sudo install -o root -g root -m 0755 helm /usr/local/bin/helm

helm help

# 4. Environment Creation

## 4.0. Create the Terraform State Location

In the folder devops/terraform/config/<env> configure the terraform backend bucket name in the file backend.tfvars:



In the command line switch folder to *devops\terraform\terraform-prereqs and* execute commands

* *make init*
* *make validate*
* *make plan ENV=dev* ( replace *dev* with environment acronym)
* review the plan and execute ***make apply ENV=dev*** (replace *dev* with correct environment acronym)

This will create s3 bucket for the terraform state file. You can verify bucket creation from AWS console.

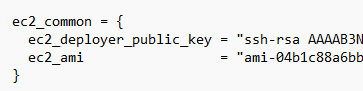
## 4.1. Create the key pair using ssh-keygen on WSL/Linux:

From command line (wsl) execute command:

ssh-keygen -m PEM

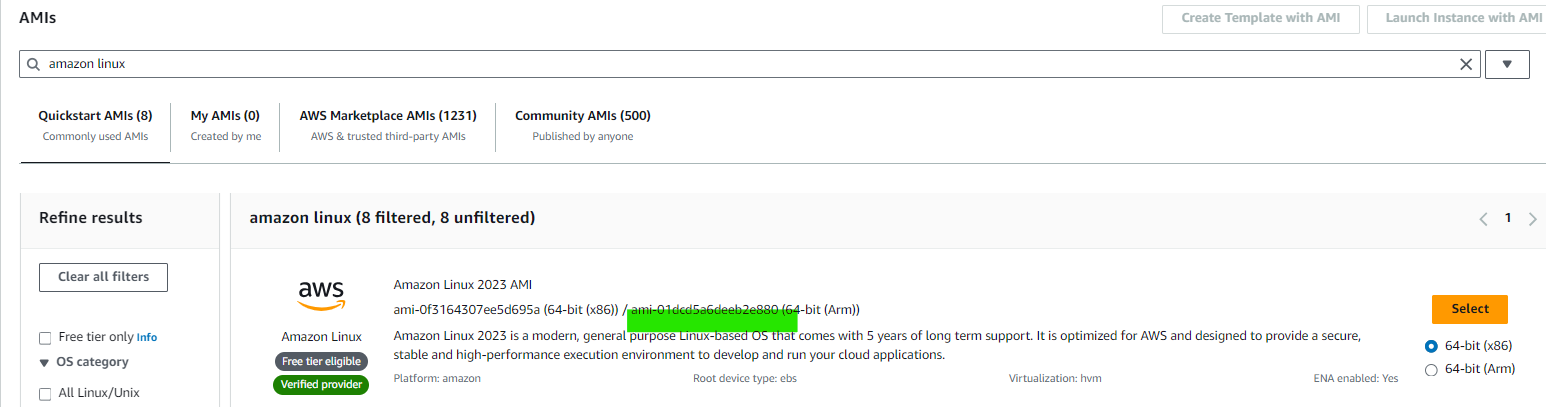
Do not set the passphrase (leave empty)

This will create a public-private key-pair. Public key has to be updated in global\_<env>.tfvars file in the variable *ec2-common. ec2\_deployer\_public\_key*

**

Private key has to be securely stored for the further usage (connect to instances via ssh)

In the ec2\_ami set the latest public image of for the amazon Linux. This can be found in AWS marketplace( on EC2 service page)



## 4.2. Update Environment Configuration

Modify the global global\_<env>.tfvars with the parameters as per the environment configuration and requirements

## 

## 4.3. Run the main environment creation routine.

Run the environment creation routine in the devops/terraform directory by running the commands( replace ***dev*** with environment acronym):

make init ENV=dev  
make plan ENV=dev

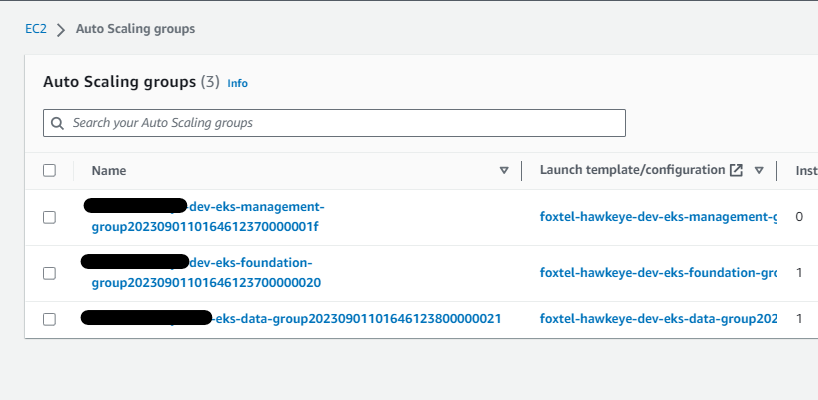
If output looks correct and there are no errors, Pleace review carefully. Some modules may need to be modified dependingon the account configuration/actual needs. After review proceed with:  
make apply ENV=dev  
terraform apply -auto-approve -var-file="./globals\_env.tfvars" -compact-warnings  
(or make apply)

If there were no errors check in AWS if resources (the S3 bucket) have been created as expected.

This will deploy the following:

* IAM roles and instance profile
* Security groups
* S3 bucket for DM data
* EMR cluster configuration (EMR is created itself)
* EKS cluster configuration
* RDS instance and associated configuration
* Jenkins EC2 instance
* Route53 hosted zone
* Internal NLB( for Jenkins)

After installation is successful log to AWS console. Get the IDS of created autoscaling groups autoscaling. If of the groups have to be updated in global\_<env>.tfvars file in variable *eks\_config.auto\_scaling\_group\_ids.* and execute make plan and apply

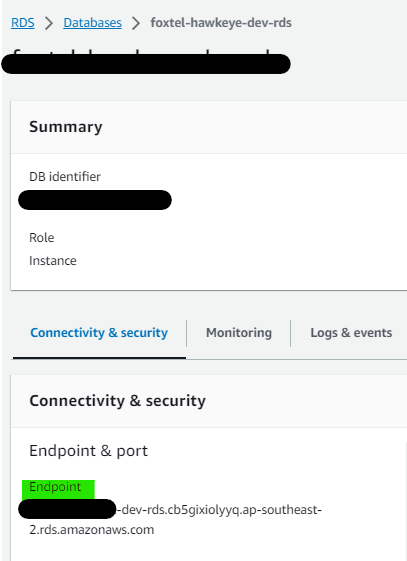


In order to Login RDS and Jenkins access to private subnets must be granted via VPN or other client approved solution:

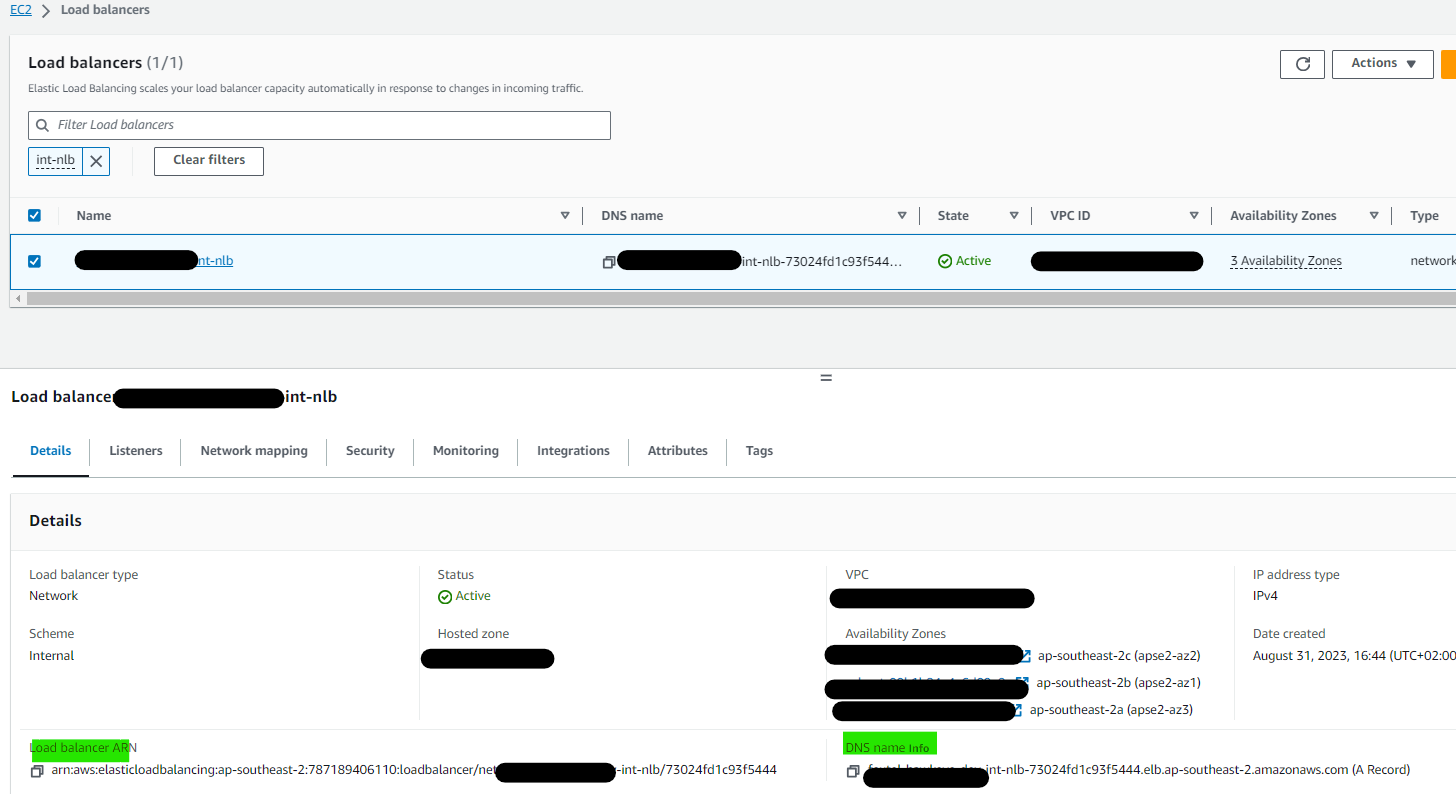
1. Access could be granted to the whole vpc private subnets range
2. Access can be granted only to the specific resources.

In the 2nd option(often applied on client side) an appropriate request has to be done go grant access to NLB DNS and RDS access point. RDS access point and NLB access point can be retrieved from AWS console. Additionally a DNS record has to be created for Jenkins.[domain] to route traffic to NLB

RDS Access Point:



NLB Access point.

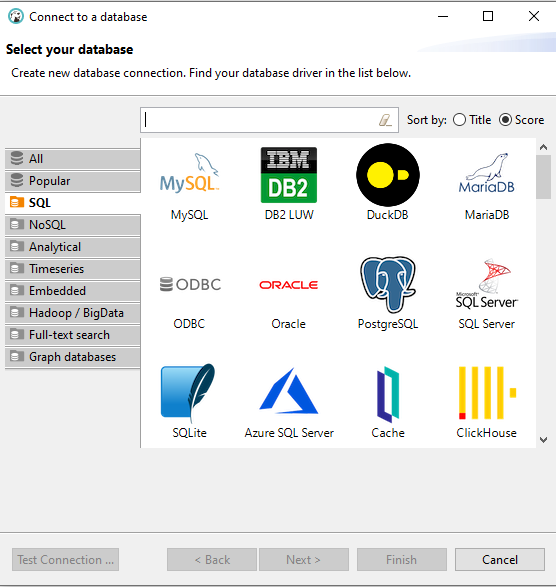


## 4.7. Initialize users In RDS database

Configurate connection to the RDS database – use output from terraform to obtain the hostname or login to AWS Console and check the hostname in RDS.

Download the DBeaver from the following URL: <https://dbeaver.io/files/dbeaver-ce-latest-x86_64-setup.exe>

If installation is successful, open DBeaver and click on the Create New Connection button.



Click on the MySQL logo

Enter the following details:

* host: <rds endpoint>
* port: 3306
* username: dm3admin
* password:

Run SQL commands from file **/db/init.sql** one by one. Remember to generate new secure passwords and save them in a password manager/encrypted file.  
Check if you can login as one of newly created users and if you can see corresponding database.

Log to RDS as **monitoring** user and run SQL commands from **db\data\_monitoring** directory.

## 4.8. Configure Jenkins Server

### 4.8.1 Install Jenkins software on the Jenkins EC2 instance.

Jenkins Server is installed with terraform along with all required application. The installation script

\devops\terraform\ec2\jenkins\startup.sh is attached to terraform and there is no need to run it manually.

### 4.8.2 Add EKS access to Jenkins’s server – **This step can be skipped**.

Adding EKS access to the Jenkins server would allow the user to bypass any Secret ID or Access Key requirements for AWS by utilizing the created IAM role.

1. Get the role arn from the Jenkins instance

* + ssh into the Jenkins instance
  + execute `*aws sts get-caller-identity*`
  + copy the ARN of the role from the output

2. Copy the aws-auth configmap from the created cluster

* + change kubctl context to newly created cluster
    - eksctl utils write-kubeconfig --cluster="<cluste\_name>" --region="<region>
  + execute `*kubectl get configmap -n kube-system aws-auth -o yaml*`
  + this outputs the configmap in yaml format
  + copy and save into a file **aws-auth.yaml**

3. modify the **aws-auth.yaml** file to include the ARN role from the Jenkins instance

Add the following:

```

- "groups":

- "system:masters"

- "eks-console-dashboard-full-access-group"

"rolearn": "arn:aws:iam::AWS\_ACCOUNT\_ID:role/proj-ec2-role-<env>-01" - replace with the role from Jenkins

"username": "proj-ec2-role-<env>-01" - replace with the username

```

4. deploy the ***eks-console-full-access.yaml*** (located in the directory Devops/Terraform/<ENV>/ec2/Jenkins and the edited ***aws-auth.yaml***

eksctl utils write-kubeconfig --cluster="proj-eks-<env>-01" --region="AWS\_REGION"

kubectl apply -f eks-console-full-access.yaml

kubectl apply -f aws-auth.yaml

## 4.11. Configuring Jenkins Pipelines

After installing Jenkins, in web browser in AWS Workspaces, go to https://Jenkins.<host> and complete the initial login process to Jenkins. During this process, accept the installation of recommended plugins. Use dms3admin as the administration user name.

After logging in, perform the following actions:

Go to Manage Jenkins, then to Manage Plugins:

Graphical user interface, text

Description automatically generated

Install the following Jenkins plugins:

* AnsiColor
* Extended Choice Parameter Plug-In
* Generic Webhook Trigger

After installing the plugins, shutdown and the start the EC2 Jenkins instance.

Afterwards,

After logging in to Jenkins again, go to Manage Jenkins, then open the following items:

A picture containing text

Description automatically generated

Crete a new credential that will allow access go GIT repository.

Graphical user interface, text, application, email

Description automatically generated

After adding the GitHub deployment SSH keys, in the Jenkins dashboard, create the following folder structure, using New item->Folder:

* amap-Intelligent-engine
* DEV
  + Admin
  + Applications
    - Airflow
    - API
    - Keycloak
    - Nifi
    - Zookeeper

All the Jenkins pipelines are stored in repository in *devops/JenkinsPipelines* folder

To create a new pipeline in Jenkins, please follow the below process. In this example, the Build Airflow pipeline will be used.

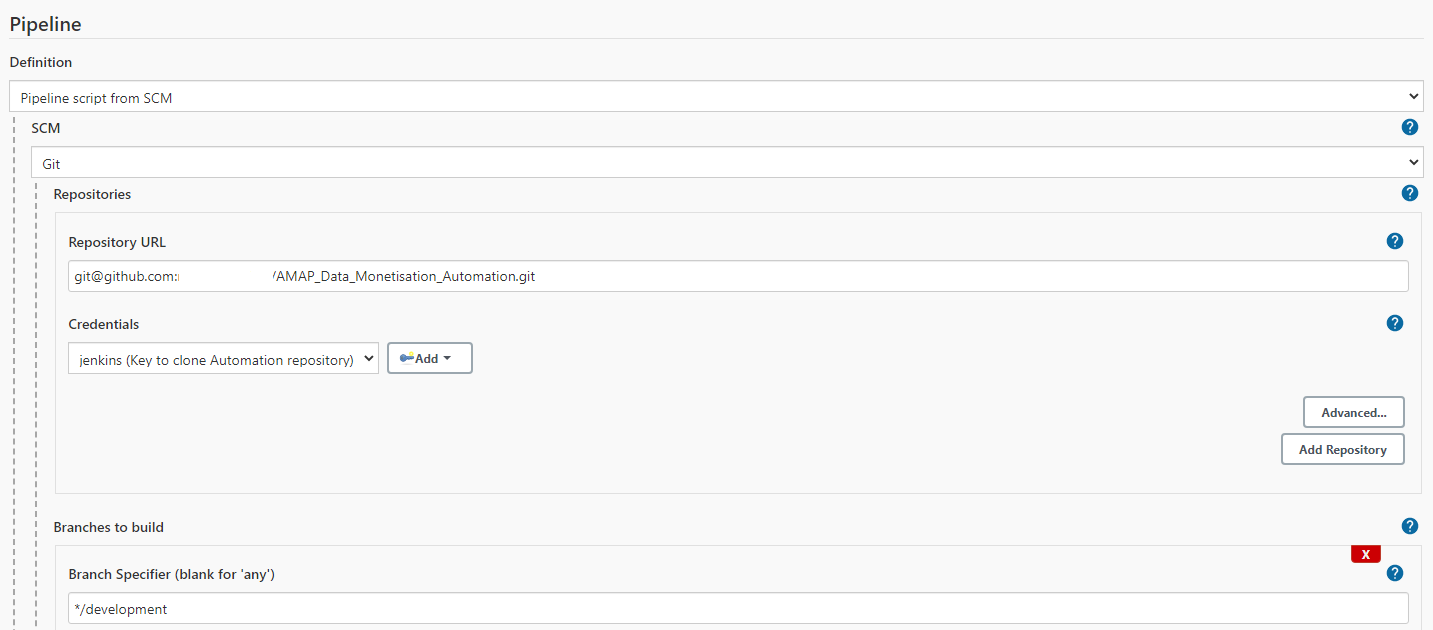
1. In Jenkins dashboard, go to the relevant folder (Airflow), then click on New Item, select Pipeline, enter the pipeline’s name (build\_airflow), then click OK
2. After creating the pipeline, the configuration window will appear.

Near the top, select the following options and enter the desired values:

Graphical user interface, text, application, email

Description automatically generated

Below, in Advanced options, under the Pipeline heading, select the options as below:



For Repository URL, use the GIT repository, and select the credentials associated with it. Specify the branch to checkout from this repository.

Below, select the pipeline script matching the pipeline being added.

Graphical user interface, application

Description automatically generated

After entering the details, click Save.

Here is the complete list of pipelines to be created.

Structure of the pipeline is the same as the structure of the folder *devops/JenkinsPipelines* in git repository.

* amap-intelligent-engine
* DEV
  + Admin
    - deploy\_aws\_loadbalancer\_controler - \devops\jenkinsPipelines\admin\ingress-nginx\deploy-alb-external.gvy
    - [deploy\_cluster\_autoscaler](https://amap.jenkins2.adop.accenture.com/view/all/job/AMAP_SFDC/job/AMAP_DevOps/job/IntelligentEngine/job/DEV/job/Admin/job/deploy_cluster_autoscaler/) - \devops\jenkinsPipelines\admin\cluster\_autoscaler\deploy.gvy
    - [deploy\_external\_dns](https://amap.jenkins2.adop.accenture.com/view/all/job/AMAP_SFDC/job/AMAP_DevOps/job/IntelligentEngine/job/DEV/job/Admin/job/deploy_external_dns/) - \devops\jenkinsPipelines\admin\external\_dns\deploy.gvy
    - [deploy\_ingress\_nginx](https://amap.jenkins2.adop.accenture.com/view/all/job/AMAP_SFDC/job/AMAP_DevOps/job/IntelligentEngine/job/DEV/job/Admin/job/deploy_ingress_nginx/) - amap-intelligent-engine\devops\jenkinsPipelines\admin\ingress-nginx\deploy-ingress-nginx.gvy
    - [deploy\_kube\_dashboard](https://amap.jenkins2.adop.accenture.com/view/all/job/AMAP_SFDC/job/AMAP_DevOps/job/IntelligentEngine/job/DEV/job/Admin/job/deploy_kube_dashboard/) - \devops\jenkinsPipelines\admin\kubernetes\_dashboard\deploy.gvy
    - [Infrastructure](https://amap.jenkins2.adop.accenture.com/view/all/job/AMAP_SFDC/job/AMAP_DevOps/job/IntelligentEngine/job/DEV/job/Admin/job/Infrastructure/)

amap-intelligent-engine\devops\jenkinsPipelines\admin\infrastructure.gvy

* + Applications
    - Airflow
      * [buildAirflow](https://amap.jenkins2.adop.accenture.com/view/all/job/AMAP_SFDC/job/AMAP_DevOps/job/IntelligentEngine/job/DEV/job/Applications/job/Airlfow/job/AirflowBuild/) – build airflow image and push to ECR
      * deployAirflow - apply airflow configuration to EKS along with image tag, synchronize EMR bootstrap scripts to S3
      * buildAndDeployAirflow – executes both build and deploy in one shot.
      * [build\_airflow\_standard\_image](https://amap.jenkins2.adop.accenture.com/view/all/job/AMAP_SFDC/job/AMAP_DevOps/job/IntelligentEngine/job/DEV/job/Applications/job/Airlfow/job/AirflowDeploy/) – executed once to create a default airflow image with all required packaged.
    - API
      * [buildApi](https://amap.jenkins2.adop.accenture.com/view/all/job/AMAP_SFDC/job/AMAP_DevOps/job/IntelligentEngine/job/DEV/job/Applications/job/Airlfow/job/AirflowBuild/) - build Api image and push to ECR
      * deployApi - apply Api configuration to EKS along with image tag
      * buildAndDeployApi - executes both build and deploy in one shot.
    - Keycloak
      * deployKeycloak - apply Keycloak configuration to EKS along with image tag
    - Nifi
      * [build](https://amap.jenkins2.adop.accenture.com/view/all/job/AMAP_SFDC/job/AMAP_DevOps/job/IntelligentEngine/job/DEV/job/Applications/job/Airlfow/job/AirflowBuild/)Nifi – build Nifi image and push to ECR
      * deployNifi - apply Nifi configuration to EKS along with image tag, synchronize Nifi custom scripts to S3
      * buildAndDeployNifi
      * uploadprocessGroup – Creates Nifi Parameter Context (one time activity), uploads Nifi templates to UI
    - Zookeeper
      * [deployZookeper](https://amap.jenkins2.adop.accenture.com/view/all/job/AMAP_SFDC/job/AMAP_DevOps/job/IntelligentEngine/job/DEV/job/Applications/job/Zookeeper/job/ZookeeperDeploy/) – Apply zookeeper configuration to EKS

Configure Credentials for applications:

Login to Jenkins create following credentials (user and password type)

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Username** | **Password** | **Application** |
| api\_user\_password*\_<env>* | api | Password used for api user created in Mysql db | Api |
| airlfow\_db\_user\_password\_<env> | airflow | Password used for airflow user created in Mysql db | Airflow |
| mysql\_password\_for\_etl\_<env> | db\_admin | Password used for db\_admni user created in Mysql db | Airflow |
| keycloak\_db\_password\_<env> | keycloakx | Password used for keycloakx user created in Mysql db | Keycloak |
| hive\_metastore\_password\_<env> | hive\_metastore | Password that will be used to login to hive\_metastore db in MySQL | Airflow and EMR( restart EMR after change) |
| airflow\_admin\_password\_<env> | admin | Password that will be used to login to airflow admin UI | Airflow |
| keycloak\_app\_admin\_user\_password\_<env> | keycloak\_app\_admin\_user\_password | Password that will be used to login to keycloak admin console | Keycloak |
| nifi\_keycloak\_secret\_<env> | nifi\_keycloak\_secret | A secret that will be created after keycloak installation | Nifi |
| amap\_airflow\_fernet\_key\_<env> | amap\_airflow\_fernet\_key | Airflow fernet key | Airflow |
| amap\_airflow\_ssh\_key\_<env> | amap\_airflow\_ssh\_key | A private key generated during environment creation. A key has to be stored as one line in password box(new lines marked with \n) | Airflow |
| amap\_airflow\_webserver\_secret\_key | amap\_airflow\_webserver\_secret\_key | Airflow webserver key | Airflow |
| amap\_airflow\_git\_url\_env | amap\_airflow\_git\_url | Leave empty | Airflow |

# 5. Applications Deployment

## 5.1. Application Configuration.

Before installation application configuration has to be reviewed in folder appconfig/<env> and set as per the environment settings.

## 5.1. Kubernetes components.

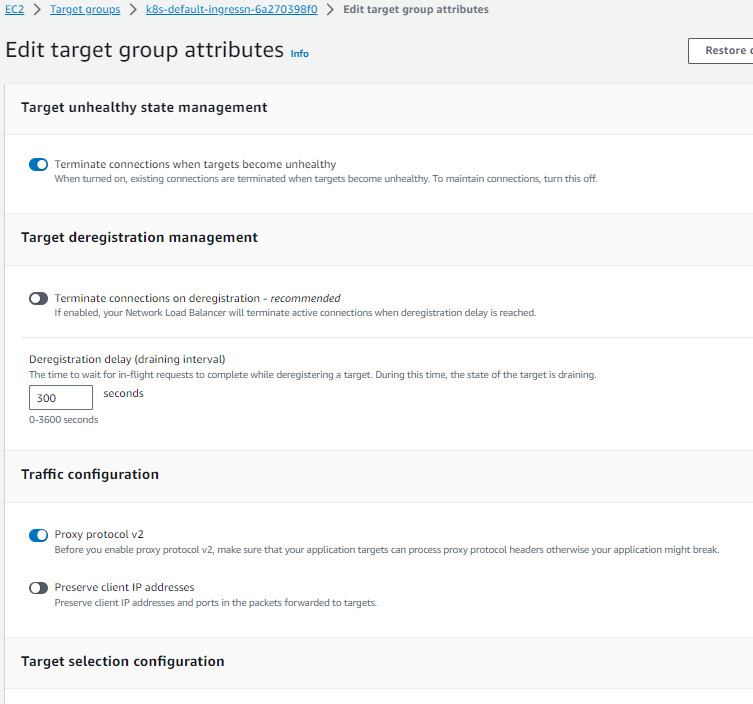
Login to Jenkins application and execute previously configured pipelines.:

* + - deploy\_aws\_loadbalancer\_controler
    - [deploy\_cluster\_autoscaler](https://amap.jenkins2.adop.accenture.com/view/all/job/AMAP_SFDC/job/AMAP_DevOps/job/IntelligentEngine/job/DEV/job/Admin/job/deploy_cluster_autoscaler/)
    - [deploy\_external\_dns](https://amap.jenkins2.adop.accenture.com/view/all/job/AMAP_SFDC/job/AMAP_DevOps/job/IntelligentEngine/job/DEV/job/Admin/job/deploy_external_dns/)
    - [deploy\_kube\_dashboard](https://amap.jenkins2.adop.accenture.com/view/all/job/AMAP_SFDC/job/AMAP_DevOps/job/IntelligentEngine/job/DEV/job/Admin/job/deploy_kube_dashboard/)

## 5.2. Deploy Network Load Balancer for Kubernetes.

Login to Jenkins application and locate **deploy\_ingress\_nginx** pipeline and execute .  
  
  
  
After successful deployment verify creation of new load balancer in AWS console – it will have random string as name. Copy name and go to Route53.

*This step are obsolete. Records in Route 53 should be created by External DNS component.*  
Go into your hosted zone and create a new record – type in your subdomain (most likely priv), pick type A record, turn on Alias slider, pick Alias to Network Load Balancer, pick your region and paste the name of newly created load balancer.

Go back to your Load Balancer, go into Listeners tab and click on Target Group in default action. In Attributes tab click Edit and enable Proxy Protocol v2.  
Open https to https://<host>/dashboard and check if Kubernetes Dashboard is available, where <host> is the priv.\* hostname created in Route53.  


NLB must be accessible via VPN or other client approved solution:

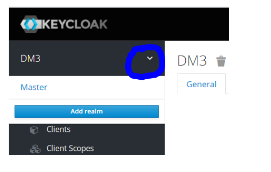
1. Access could be granted to the whole vpc private subnets range
2. Access can be granted only to the specific resources.

In the 2nd option (often applied on client side) an appropriate request has to be done go grant access to NLB. NLB access point can be retrieved from AWS console. Additionally, a DNS record must be created for priv.[domain] to route traffic to NLB.

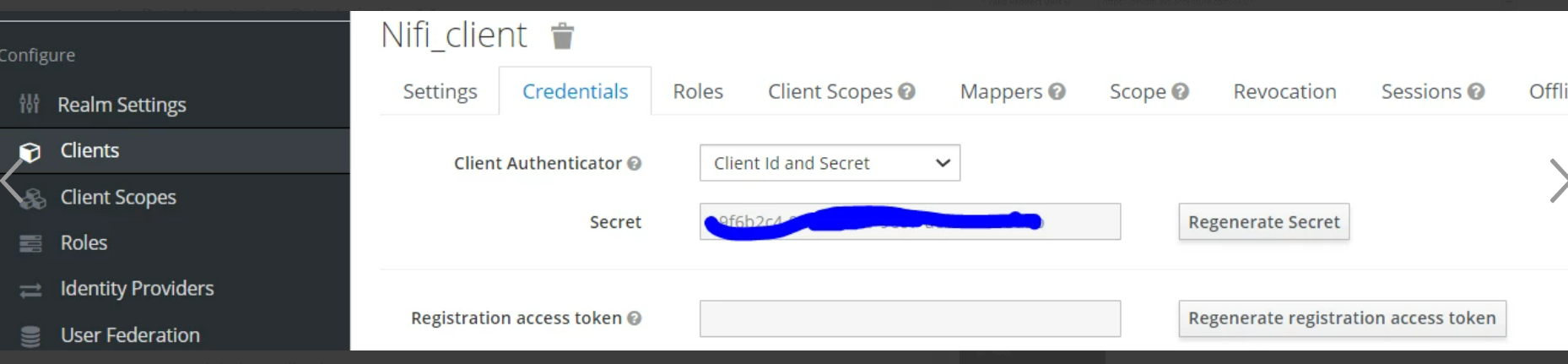
## 5.3. Keycloak

Login to Jenkins application and execute deployKeycloak  
Verify Pipeline variables/parameters and execute the plan.  
After successful deployment go to <host>/keycloak/auth and login into Administration Console.

Hover over realm name (probably Master) under the logo and then click “Add Realm”.

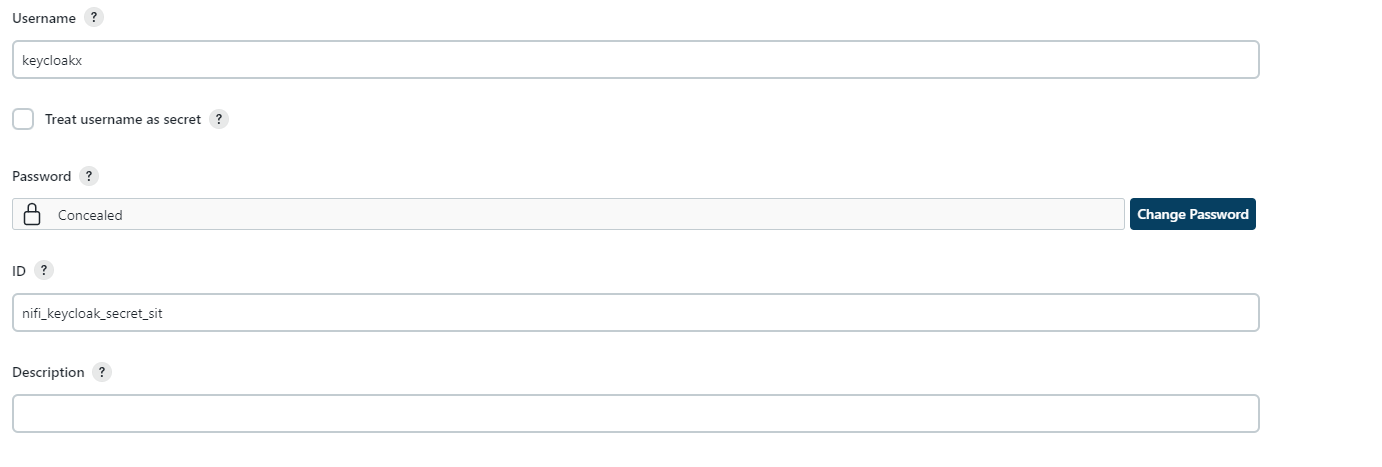
  
Use file stored in repo under location /charts/keycloka/realm-export.json  
Go to Clients -> *nifi-client* and scroll down to *Valid Redirect URIs*. Change environment hosts you used in deployments.

Go to *Credentials* tab of **nifi-client** - generate new secret and save it. This secret will be used to deploy Nifi.

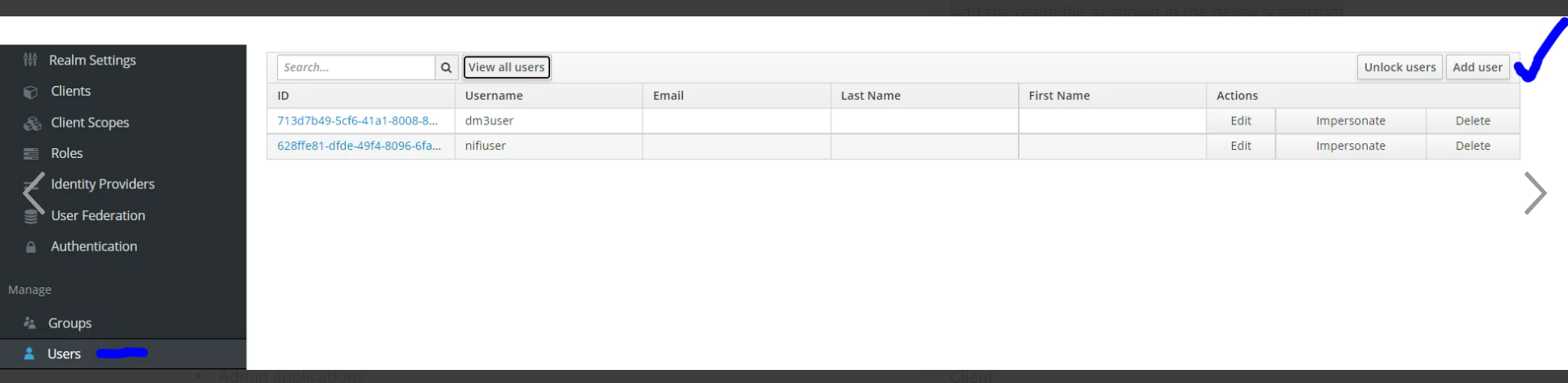


The secret has to be updated in Jenkins credentials (user with password)

*nifi\_keycloak\_secret\_<env>*



Create users for the nif and API layer:  
Go to Users in navigation menu on the left and create new user:   
 - ***nifiuser***  
 - ***dm3\_api\_admin***

  
Create passwords for both users and save them.

## 5.4. Zookeeper

Login to Jenkins application and locate deploy\_zookeeper pipeline  
Verify Pipeline variables/parameters and execute the plan.  
After successful deployment use kubectl to check the status of new pod(zookeeper)

## 5.5. Build and deploy API Layer

Login to Jenkins application and locate **buildAndDeployApi** pipeline  
  
After successful deployment use kubectl to check the status of new pod.  
To look for errors use command:  
  
kubectl logs -n dm <pod-name>  
kubectl logs -n dm <pod-name>

## 5.6. Build and deploy Airflow

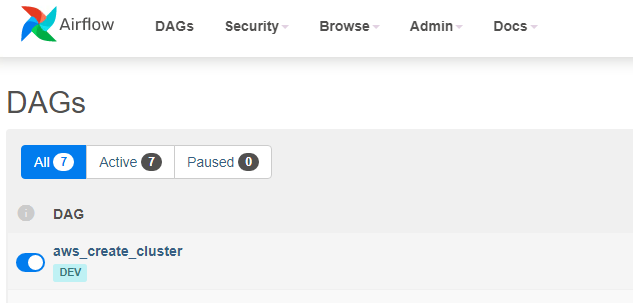
In order speed tup airflow Deployment a

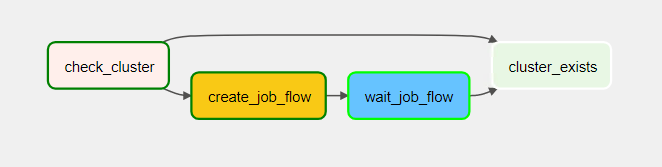
Login to Jenkins application and locate **build\_airflow** pipeline.

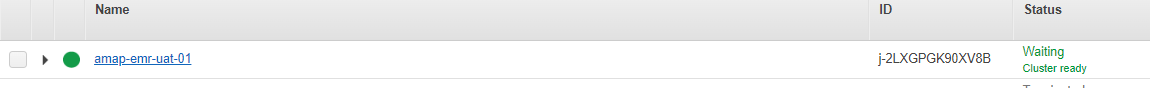
Login to Jenkins application and locate deploy\_airflow pipeline. Verify Pipeline variables/parameters   
and execute pipline.

After successful deployment use kubectl to check the status of new pod.

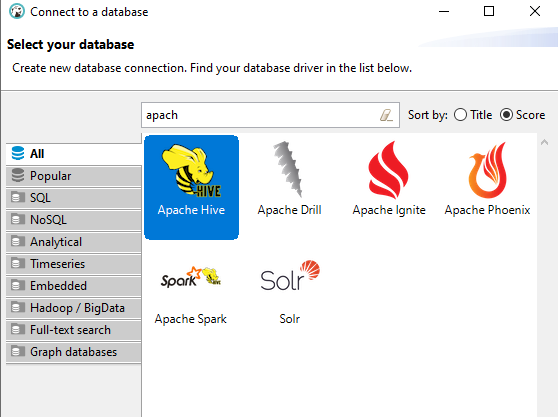
If all pods initiated correctly go to https://<host>/airflow, log in and see if all DAGs are visible. Run **aws\_create\_cluster** DAG to check if cluster creation works as expected.



  
Verify the cluster status in AWS console If EMR will be in status “Waiting Cluster ready”



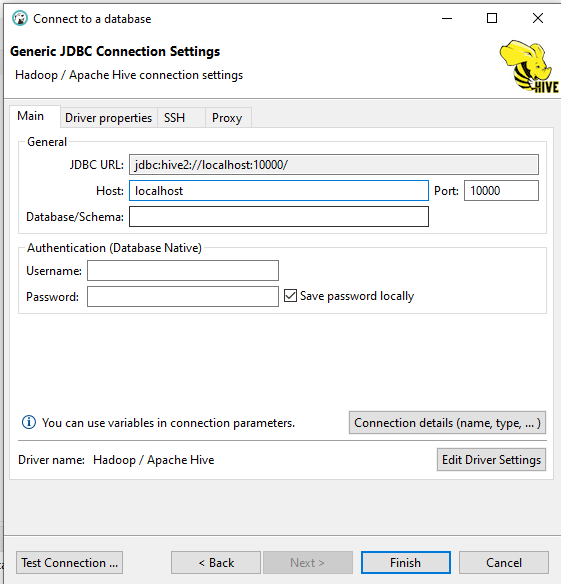
Verify if can connect to Hive DB hosted on EMR  
  
Open DBeaver and click on the Create New Connection button



On the new window, click on ‘All’ on the left side, and type in the search box ‘apache’

Click on the Apache Hive logo.

On the new window:



Enter the following details:

* host: ERM static IP
* port: 10000
* database/schema: default;transportMode=https?hive.txn.manager=org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;
* username: hadoop

.

If can connect to EMR installation was successful.

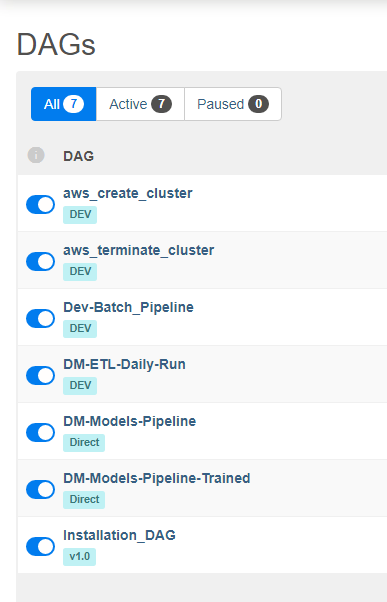
**AMAP DM tables creation**.

After successful installation it’s needed to create tables in data\_activation schema in RDS DB.

Log to airflow application using following url  
https://priv.<host>/airflow entering credentials provided during airflow installation (plan parameters)

In Airflow Activate the Installation\_DAG and execute it.

After successful execution new tables will be created in RDS data\_activation schema.

Activate all remaining DAGs  
  


**Nifi User**

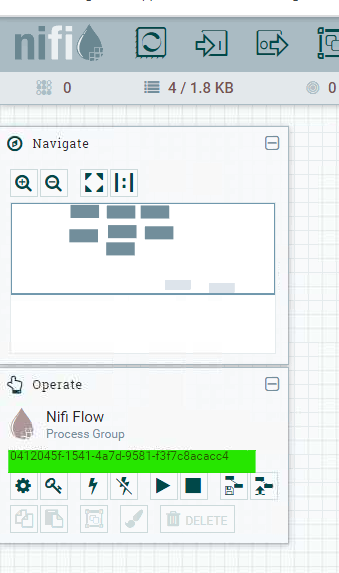
Airflow Daily DAG is triggered from Nifi. A dedicated user has to be created in Airflow. Login to airflow application and create user with id “nifi”. Set password and activate it. Password will have to later set in Nifi context parameter.

## 5.7. Deploy Nifi

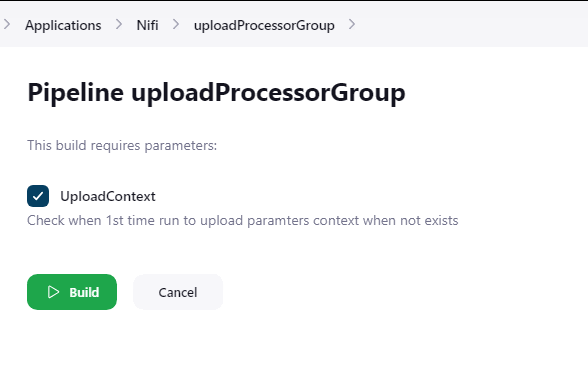
Login to Jenkins application and locate **buildAndDeployNifi**

After successful installation verify you can log(from AWS workspace) to nifi https://priv.<host>/nifi/ using credentials created during Keycloak installation(nifiuser credentials).

In the next step capture the Nifi Flow process group and save it in nifi.yaml file in appconfig folder

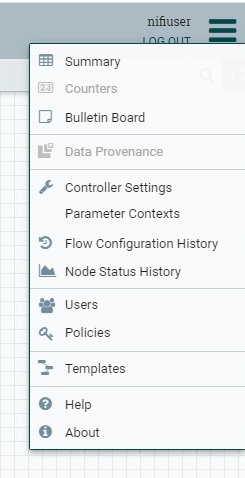


Next deploy parameter context and Templates. Login to Jenkins and execute ***uploadProcesssGroup*** Pipeline. In order to upload context parameter (one time activit) you need to check **UploadContext** checkbox.

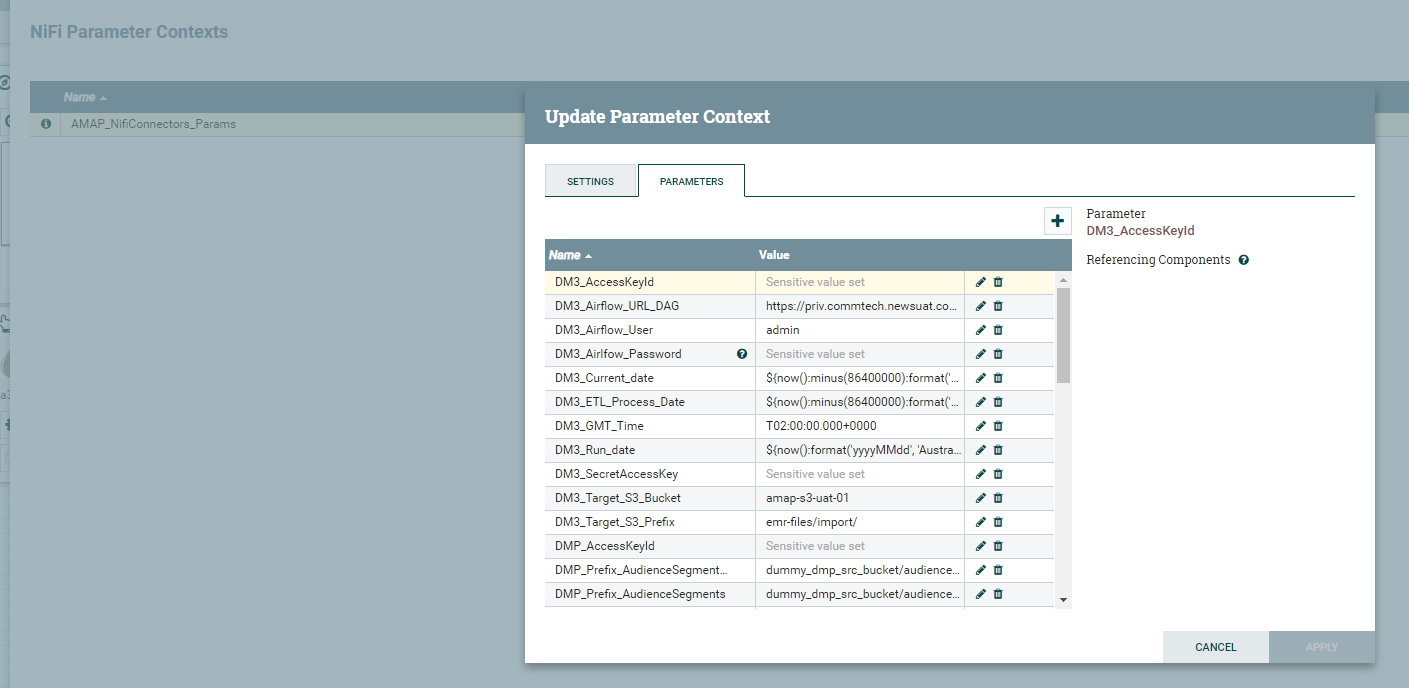


**Parameter Context Configuration:**

Nifi Parameter context stores environment specific variables. It has to be manually created/update every time new environment variable is crated   
Go to https://<host>/nifi/, login into dashboard, click on burger menu icon in upper right corner and select Parameter Contexts.



Edit/Verify/Create context variables by changing values to one used in new environment (AWS components names).



**Nifi Flows configuration.**

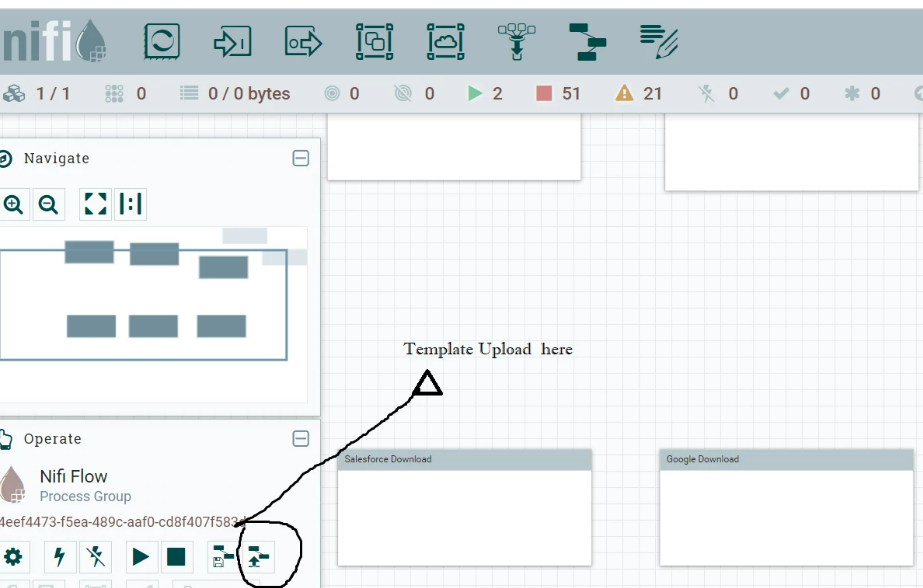
**Nifi templates will be uploade but needs to be manually aplied**

For the Phase1 following templates has to be applied.

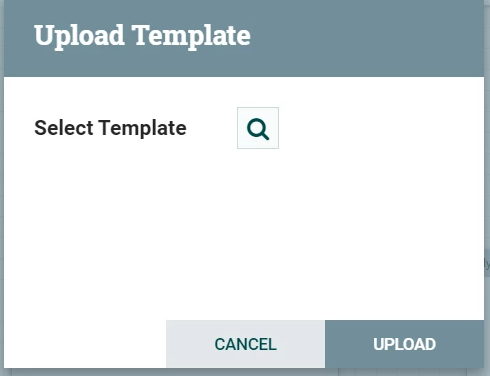
* Execute\_Airflow\_ETL.xml – executed airflow daily dag a scheduled timeframe
* Google\_Download.xml – executes data export from GAM Adserver
* Salesforce\_Download.xml – exports data from Salesforce
* DM\_to\_CRM\_s3.xml – Exports Data to for CRM Analytisc
* DM\_IL.xml – Audience push to Salesforce( trough IL)

**Apply Templates Manually:**

Login to Nifi application and select this option for uploading the template file.



Select the template you want to upload:



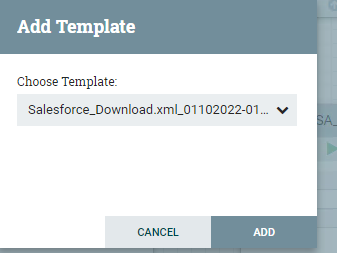
**Configure Templates**

After Templates has been uploaded (via pipeline or manually) they must be configured

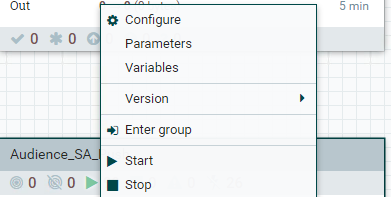
Drag the template on editor and add the template file from here:

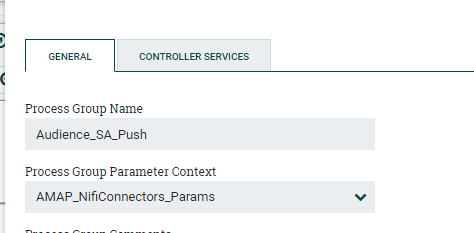


From the list select template to apply:

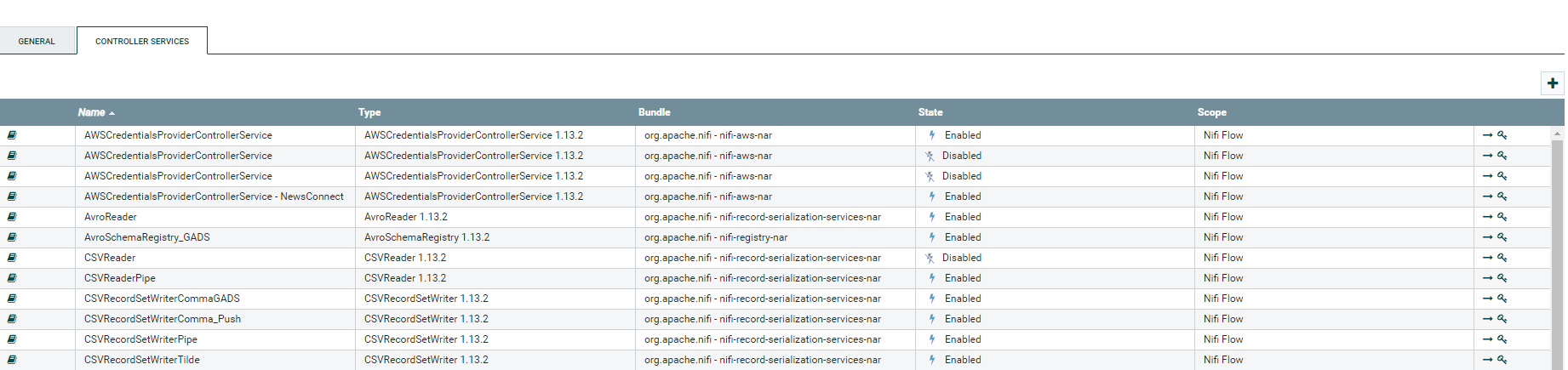


After template upload right click on created process group and select Configure from the list and setup the Process group parameter context





Switch to Controller Services page and verify if all services are enabled:



Review all the errors in the new created process group and solve them. In case templates will be uploaded multiple times a duplicated Controller services will be created. It’s good practice to not enable this controller service but switch to already active one and remove the inactive controller service.

## 5.7. Configure Dashboards.

AMAP IE Dashboards need to be configured in Salesforce Analytics Studio. The Manual configuration requires to complete following Steps:

* S3 connector Definition – configure connection to S3
* Object Configuration
* Receipt configuration
* Dashboards configuration
* Scheduling

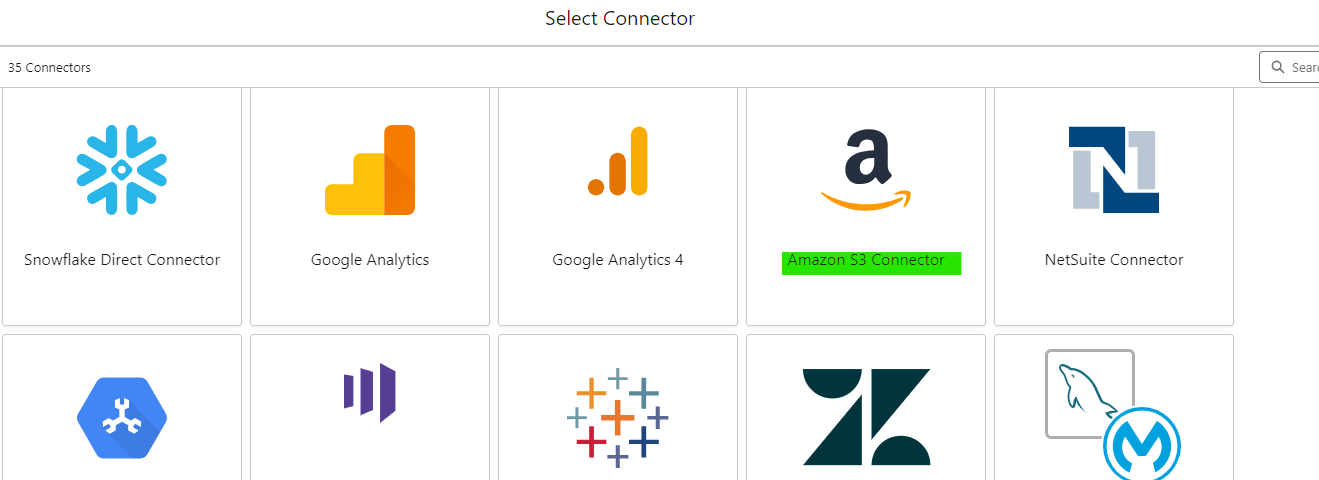
**S3 Connector Definition:**

1. a dedicated user crm\_analytisc is created in AWS account. Its needed to connect to AWS account and generate access keys for that user.

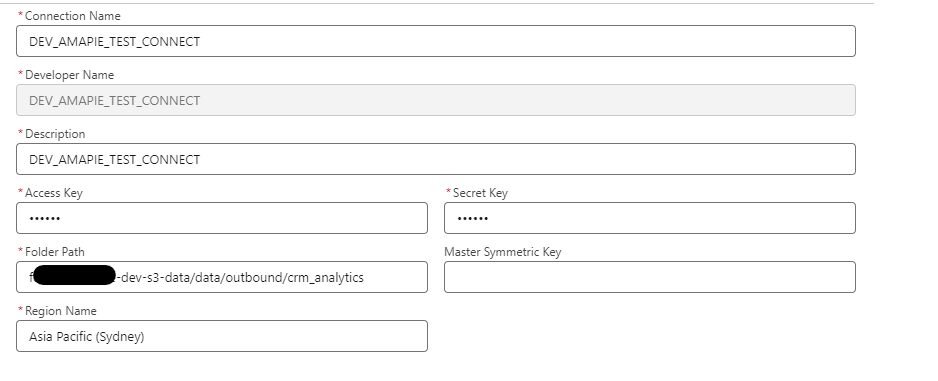
**aws cli command**: *aws iam create-access-key –user-name crm\_analytisc*

This user is having read-only access to data/outbound/crm\_analytics in data backet.

1. Login to Salesforce. Open Data Manager and create Amazon S3 connector.



1. Enter Required Information (with access keys created in step 1). Path to the data hast to be entered together with the bucket name (e.g *amap-data-bucket/outbound/crm\_analytisc*) and save changes

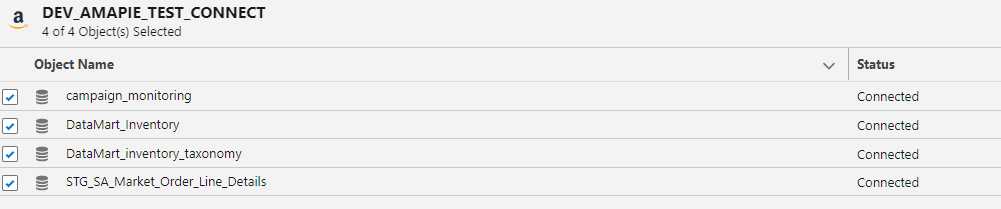


**Object Configuration**

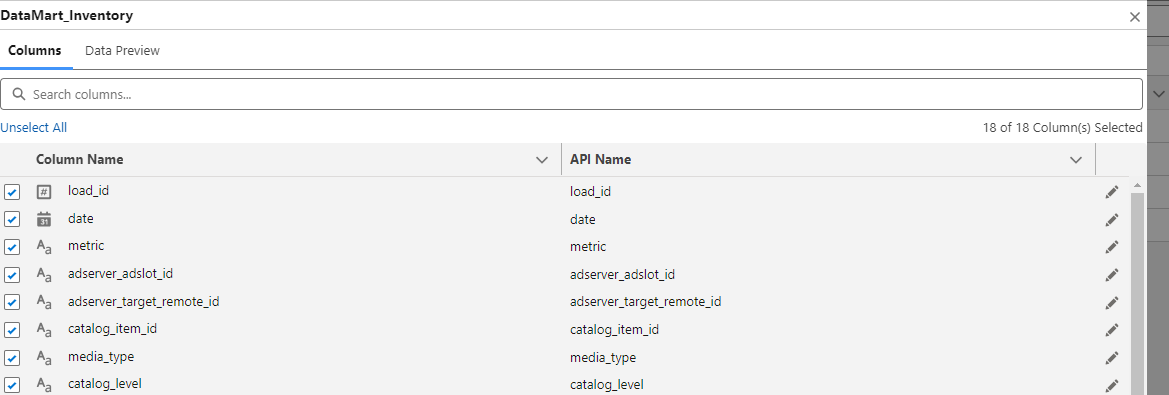
Configure data that will be pulled from AMAP IE S3 bucket. This can be configured only if sample data are available in the bucket.

In the connection created in the previous step select Edit Object from the dropdown menu and mark objects required for the Dashboards.

Below Sample. For the full list of the required object in specific use case contact Data Visualization Team.

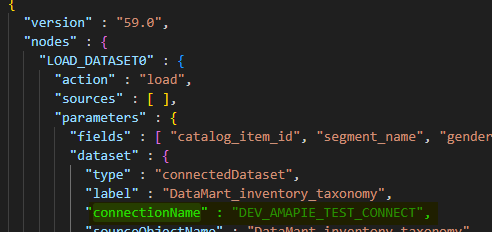


Each object must be edited, and all columns selected.

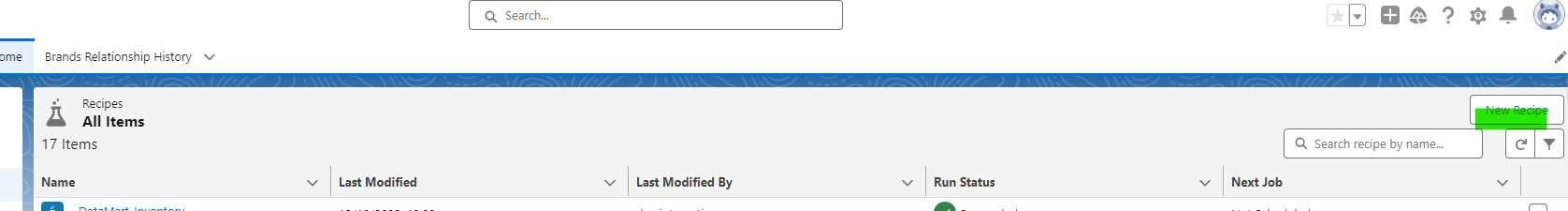


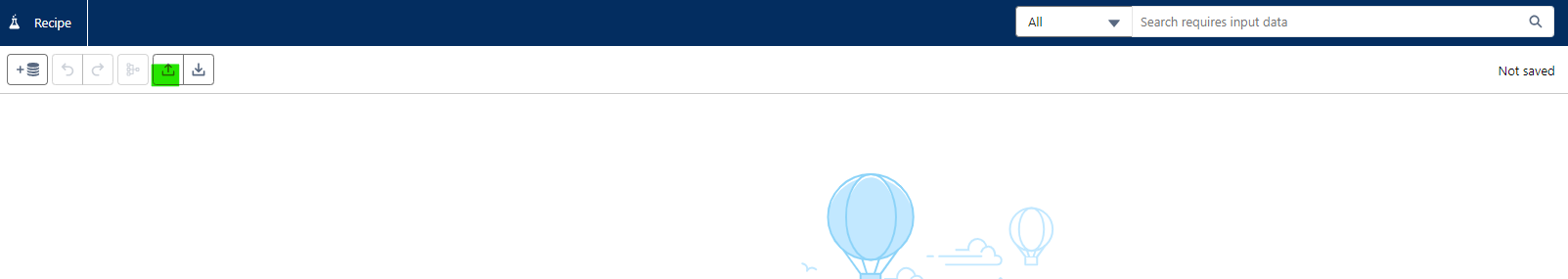
Receipt configuration.

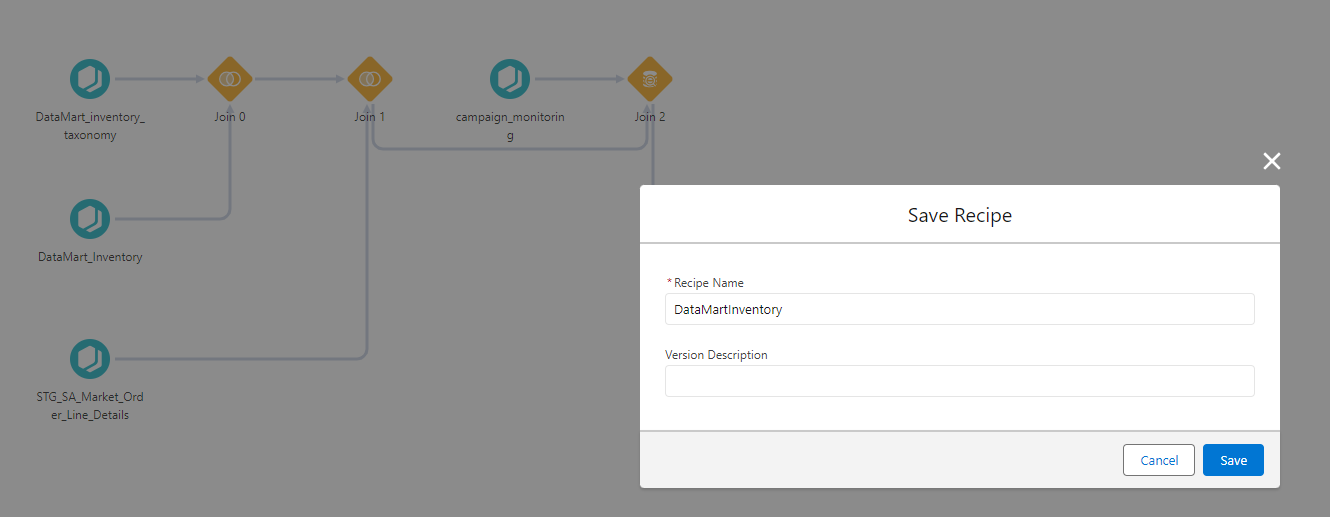
Receipts are stored in data repository in folder dasaboards/RAW/Receipts in json fomat. Before uploading the receipts you need to update directly in the json file the S3 connection name. In json file locate entries with the tag connectionName and update value to creted Amazon S3 connection(use Developer Name)



In Data manager click on new receipt and upload json file from the repository. Save the receipt with the same name as uploaded file.

****

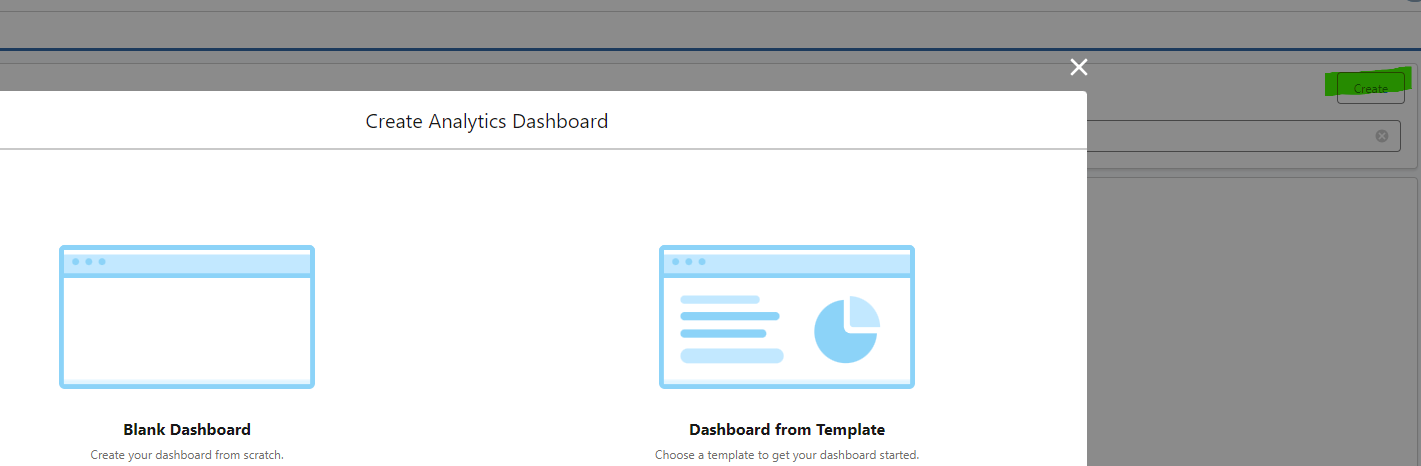
****

****

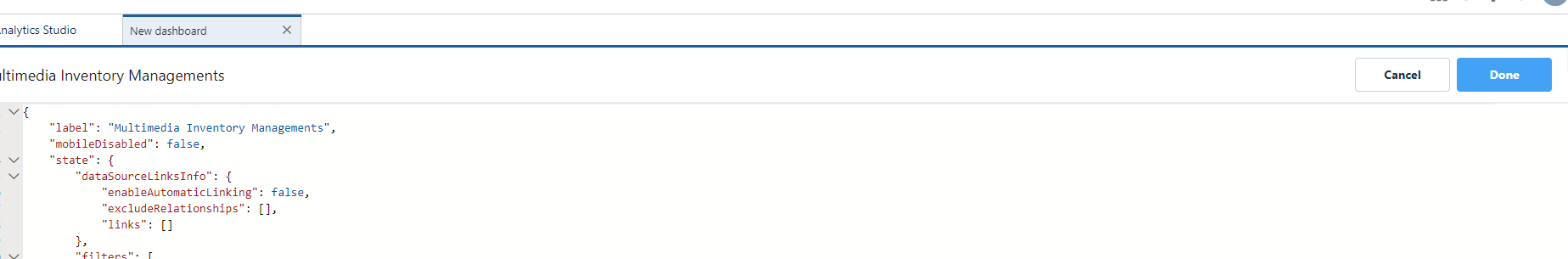
Repeat above Step for All required Receipts.

**Dashboards Configuration**

Dashboards are stored in repository in folders dashboards/RAW/dasboards in json . In order to create Dashboard in Analytics Studio create Blank Dashboards



On keyboard press CTR+E. this will open Code editor. Replace existing code with data from the json files in the repository and press Done button.



# 6. Perform sanity check for environment.

The following check list is to test the readiness of the environment components.   
  
  
  
- AWS COMPONENTS

|  |  |  |  |
| --- | --- | --- | --- |
| FUNCTIONALITY | DESCRIPTION | STATUS | NOTE |
| Elastic Computing (EC2) | Following instances should be visible while environment is running:   * proj-eks-<env>-01-data-group-01-eks\_asg * proj-eks-<env>-01-foundation-group-01 * proj-eks-<env>-01-management-group-01-eks\_asg * proj-emr-<env>-01 (when cluster is running) |  |  |
| Elastic Kubernetes Service (EKS) | proj-eks-<env>-01 cluster should be present and running. |  |  |
| Elastic MapReduce (EMR) | proj-emr-<env>-01 should be in “Waiting Cluster ready” status |  |  |
| Relation Database Service (RDS) | * proj-rds-<env>-01 should be present * Working connection through DBeaver with credentials created during deployment. |  |  |
| Simple Storage Service (S3) | * proj-dm-<env>-terraform-state-01 should be present * proj-s3-<env>-01 should be present * proj-s3-<env>-01 should have the following folders after deployment of all applications:   + cf/   + emr\_logs/   + NiFi/   + Spark/ |  |  |

- APPLICATION COMPONENTS

|  |  |  |  |
| --- | --- | --- | --- |
| FUNCTIONALITY | DESCRIPTION | STATUS | NOTE |
| Nginx | Check each endpoint in browser:   * https://<host>/airflow * https://<host>/nifi * https://<host>/dashboard * https://<host>/keycloak/auth   Check each endpoint with Postman(detailed instruction in Sanity Checks document):   * https://<host>/inventory/inquirity * https://<host>/inventory/check * https://<host>/orders/sync * https://<host>/keycloak/auth/realms/DM3/protocol/openid-connect/token |  |  |
| Apache Airflow | * Should be able to login into dashboard https://<host>/airflow * Should be able see DAGs * Links should return to correct endpoint for the following menu:   + Security   + Browse   + Admin   + Docs |  |  |
| Apache NiFi | * Should be able to login into dashboard: https://<host>/nifi/ * Should be able to create a process * Should be able to generate process groups from processes * Should be able to import/export templates * Should be able to create key:values from Controller Services * Should be able to create key:values from Parameter Contexts |  |  |
| KeyCloak | * Should be able to log into Keycloak dashboard as admin https://<host>/keycloak/auth/admin * Under Manage > Users these accounts should be available   + dm3\_api\_admin - credentials used to test API layers   + nifiuser - credential used to access NiFi dashboard thru Keycloak * Under Configure > Clients "nifi-client" should be present with following configuration:   + URLs should use https://<host> as base domain   + Credential should be created to be used for NiFi installation under Kubernetes cluster |  |  |
| Cluster Autoscaler | * Should be able to log into Kubernetes dashboard: https://<host>/dashboard * Under Cluster > Cluster Roles  “cluster-autoscaler” should be available. |  |  |
| Kubernetes Dashboard | * Should be able to log into Kubernetes dashboard: https://<host>/dashboard * Check all navigations available on the left-side pane, should redirect to correct pages. * Pods of all deployed applications should be visible. |  |  |