**Matillion** – LLD

Version: 2.0

[1. Document Demographics 4](#_Toc145414361)

[2. Introduction 5](#_Toc145414362)

[2.1 Purpose 5](#_Toc145414363)

[2.2 Scope of this document 5](#_Toc145414364)

[3. Matillion Environment Setup 6](#_Toc145414365)

[4. Register the newly Created Matillion instance to Matillion HUB 10](#_Toc145414366)

[5. Matillion Active Directory Setup 11](#_Toc145414367)

[6. Matillion User Access 12](#_Toc145414368)

[7. Matillion User Accounts For ETL Project setup 12](#_Toc145414369)

[8. Matillion ETL Project Setup 12](#_Toc145414370)

[9. Import ETL Projects 14](#_Toc145414371)

[10. Export ETL jobs 15](#_Toc145414372)

[11. Key Vault Integration 15](#_Toc145414373)

[12. Blob Storage Integration 18](#_Toc145414374)

[13. Matillion DR 19](#_Toc145414375)

[13.1 Architecture Diagram of Azure to Azure DR Setup, Active Passive mode 19](#_Toc145414376)

[13.2 Steps for DR Setup 19](#_Toc145414377)

[13.3 Firewall Rules 21](#_Toc145414378)

[13.4 DNS Entry and Load Balancer 21](#_Toc145414379)

[13.5 DR Testing 21](#_Toc145414380)

[14. Technical Steps for Password Rotation 22](#_Toc145414381)

[14.1 Architecture Diagram 22](#_Toc145414382)

[15. Matillion DevOps 23](#_Toc145414383)

[15.1 Introduction 23](#_Toc145414384)

[15.1.1 Purpose of the document 23](#_Toc145414385)

[15.1.2 Scope 23](#_Toc145414386)

[15.1.3 Tools/Services Used 23](#_Toc145414387)

[15.1.3.1.1 Azure SaaS 23](#_Toc145414388)

[15.1.3.1.2 Azure PaaS 23](#_Toc145414389)

[15.1.3.1.3 Other Software 23](#_Toc145414390)

[15.1.4 Intended Audience 24](#_Toc145414391)

[15.1.5 References 24](#_Toc145414392)

[16. DevOps Solution Architecture for Matillion 25](#_Toc145414393)

[16.1 Solution Architecture 25](#_Toc145414394)

[16.1.1 Architecture Components 25](#_Toc145414395)

[16.1.1.1 Azure DevOps 25](#_Toc145414396)

[16.1.1.1.1 Azure Repos 25](#_Toc145414397)

[16.1.1.1.2 Azure Artifacts 25](#_Toc145414398)

[16.1.1.1.3 Azure Pipeline 25](#_Toc145414399)

[16.1.1.2 CI – Continuous Integration 25](#_Toc145414400)

[16.1.1.2.1 Repo Details 25](#_Toc145414401)

[16.1.1.2.2 Environment and Matillion server details 26](#_Toc145414402)

[16.1.1.2.3 Branching Strategy 26](#_Toc145414403)

[16.1.1.2.4 Branch Policies 27](#_Toc145414404)

[16.1.1.3 Matillion GIT Integration Set Up 28](#_Toc145414405)

[16.1.1.4 Steps for Developer 33](#_Toc145414406)

[16.1.1.5 Continuous Delivery 37](#_Toc145414407)

[16.1.1.5.1 Multi Environment Deployment Diagram 37](#_Toc145414408)

[17. Appendix 44](#_Toc145414409)

# Document Demographics

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Print Name** | **E-mail** | **Date** |
| **Author** |  |  |  |
| **Reviewed by** |  |  |  |
| **Approved by** |  |  |  |

# Introduction

CLIENT X onboarded Matillion to create Data Hub and a modern data platform. This document covers the topics that includes modern application architecture framework as applicable to a data platform.

## Purpose

The document provides complete low-level design and configuration details of Matillion modern data hub platform.

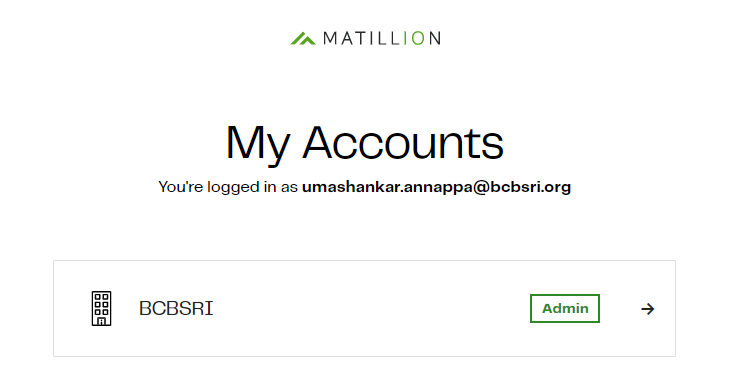
## Scope of this document

The scope of this document is to provide the complete configuration details of Matillion ETL tool:

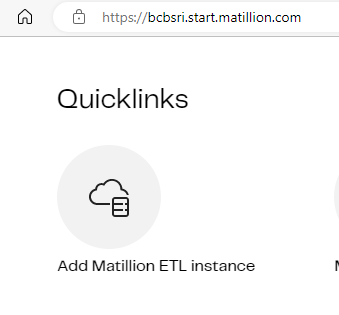
1. Matillion Environment Setup
2. Register the newly Created Matillion instance to Matillion HUB.
3. Matillion Active Directory Setup
4. Matillion User Access
5. Matillion User Accounts For ETL Project setup.
6. Matillion ETL Project Setup.
7. Import ETL Projects
8. Export ETL Projects
9. Integrate Key Vaults
10. Integrate Blob Storage
11. Disaster Recovery
12. Password Rotation
13. DevOps

# Matillion Environment Setup

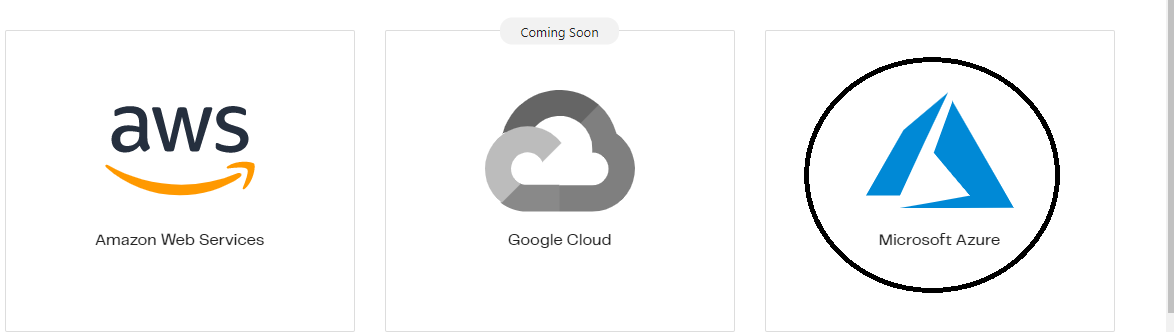
1. Matillion ETL instances are provisioned via hub.matillion.com
2. Login to Matillion <https://hub.matillion.com> with a user who has as Admin Role:
3. Click on Admin option.



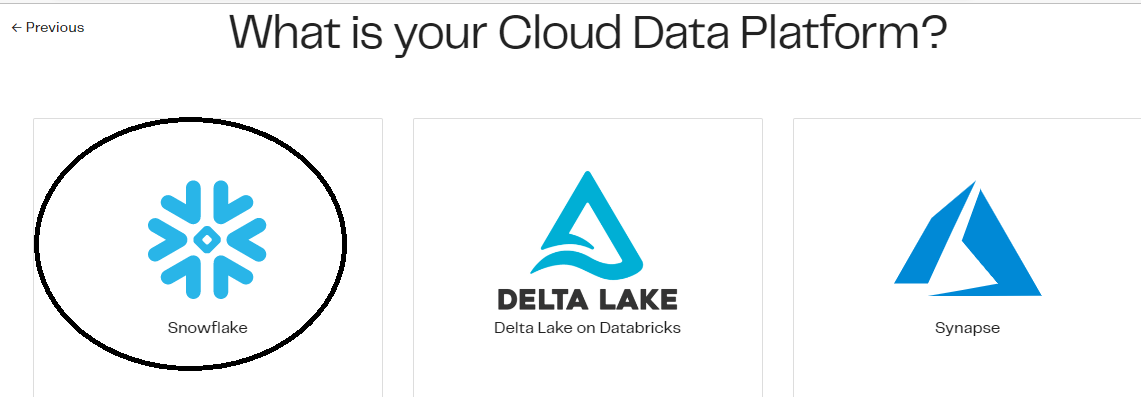
1. Click on Add Matillion ETL instance as below



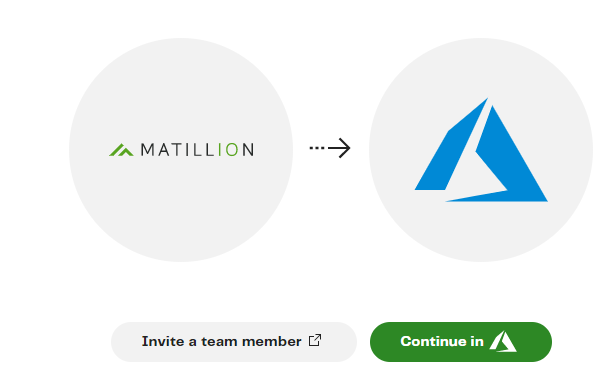
1. Choose the Cloud provider as Microsoft Azure



1. Choose cloud data Platform as Snowflake



1. Matillion will redirect the URL to Azure and please invite HCL Infra Team (Linux Team) to deploy Matillion Instance and click on continue in



1. Deploy Matillion Instance as below, click on Create



1. Create Matillion Instance with the following Details

Subscription = CLIENT X-DATA

Resource Group=rg-eus2-data-Matillion-001

Region=East-US 2

Matillion Product = Matillion ETL for Snowflake

Virtual Machine Name= Provide VM name as per BCBS naming standard

Username and Password

A screenshot of a computer

Description automatically generated

1. Choose Size of the VM, vnet and subnet as below

Graphical user interface, text, application

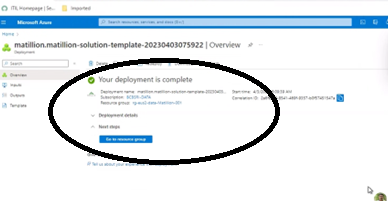
Description automatically generated

1. Review the Configuration and then click on Create

Graphical user interface, application

Description automatically generated

1. Matillion Instance deployment complete



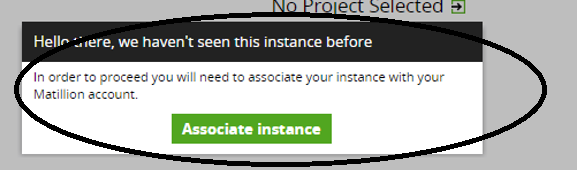
1. Login to Azure portal and make sure the Newly Deployed VM is in the Running Status as below



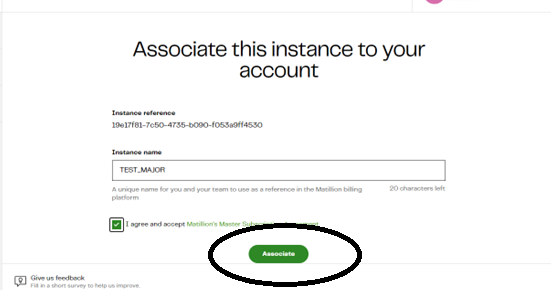
# Register the newly Created Matillion instance to Matillion HUB

1. Login to newly created Matillion Instance via web URL – <https://10.31.8.9/> and click on Associate instance

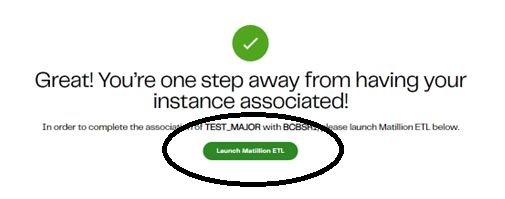
The default username is azure-user and the default password is azure-user.

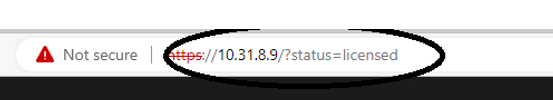


1. Click on associate instance will redirect to Matillion Hub Portal, Click Admin and associate the instance with Admin Account



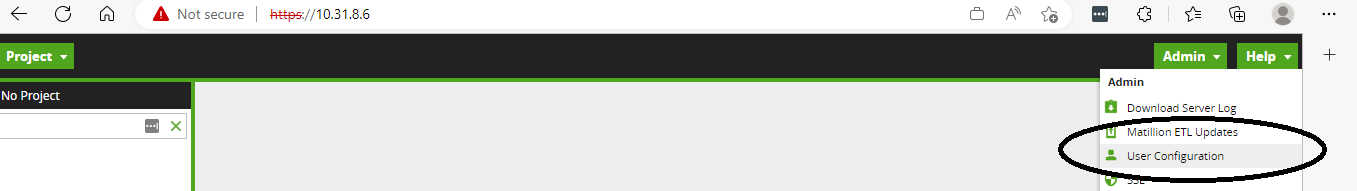
1. Launch Matillion Instance and verify status=licensed



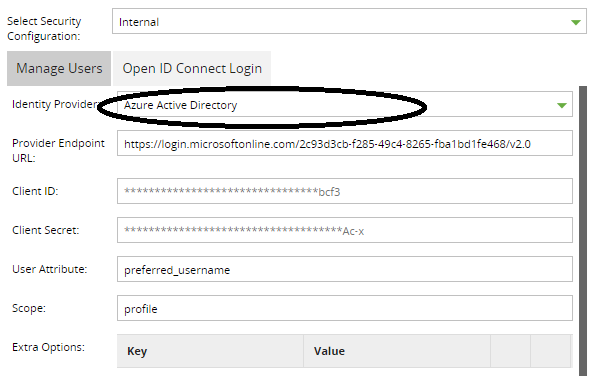


# Matillion Active Directory Setup

1. Matillion user authentication is established through CLIENT X Azure Active Directory.
2. The users will have single sign on capability.
3. Login to Matillion <https://10.31.8.6/> and go to Admin🡪user configuration



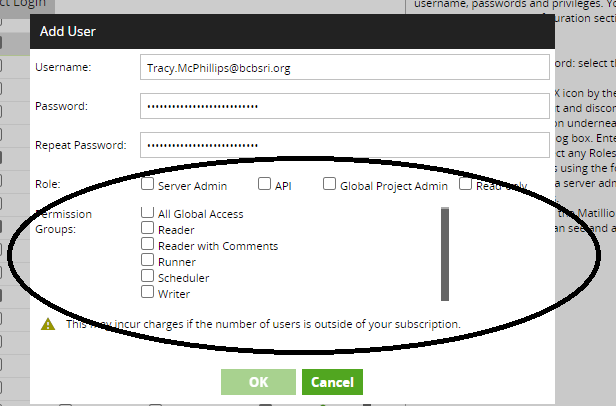
1. Click on OpenID Connect Login and Provide Endpoint URL, Client ID, Client Secret and click ok



1. Restart Matillion Instance to effect the changes

# Matillion User Access

1. Matillion User access and Permissions are granted via Matillion Tool
2. Login to Matillion <https://10.31.8.6/>, Click on Admin🡪user configuration🡪click on +icon to add a user with appropriate privileges.



# Matillion User Accounts For ETL Project setup

1. Create a Generic SNOW request to CLIENT X-Dell-Apps-GenDBA with the following Details.

Create Snowflake TEST Minor Matillion service accounts SA\_MAT\_EDR\_SUPPLY\_MINOR and SA\_MAT\_EDR\_DEMAND\_MINOR,

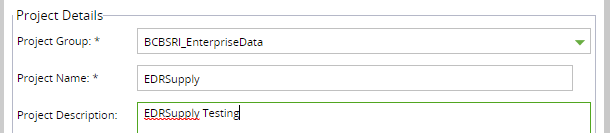
Assign Appropriate roles and Permissions.

# Matillion ETL Project Setup

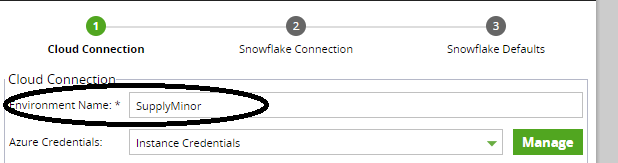
1. Login to Matillion with Admin Privileges, Click on Projects🡪switch Projects🡪Create Projects



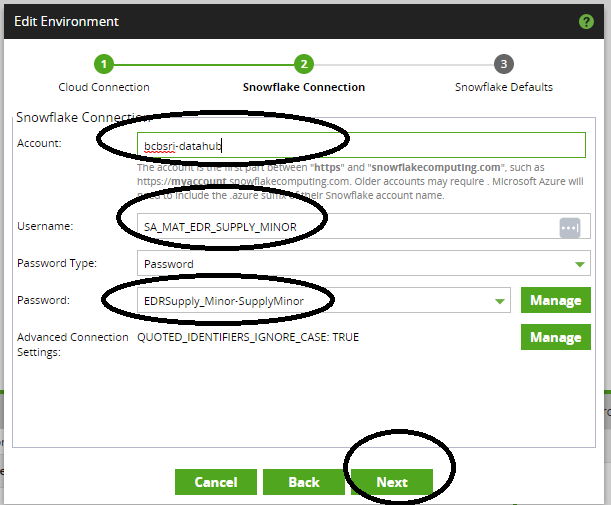
1. Provide Project Details



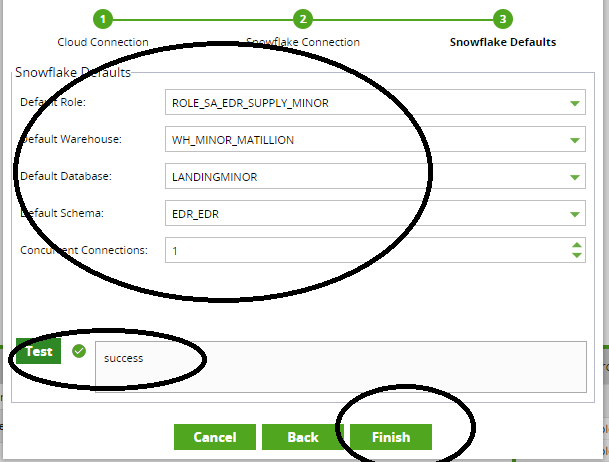
1. Provide Cloud Connection Details



1. Provide Snowflake Connection Details

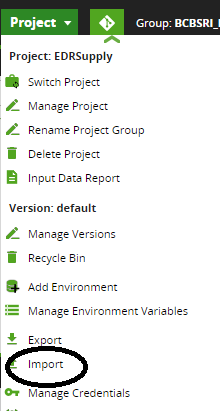


1. Provide Snowflake Database Details and Test



# Import ETL Projects

1. Click on Projects🡪Import

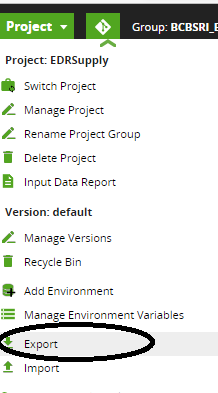


1. Import the ETL jobs as below

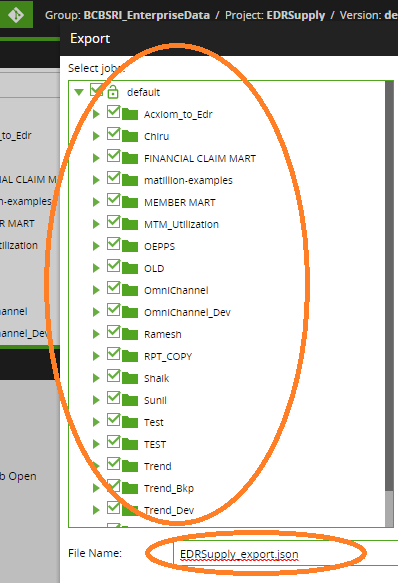


# Export ETL jobs

1. Click on ETL Projects🡪Exports



1. Export the ETL jobs as below

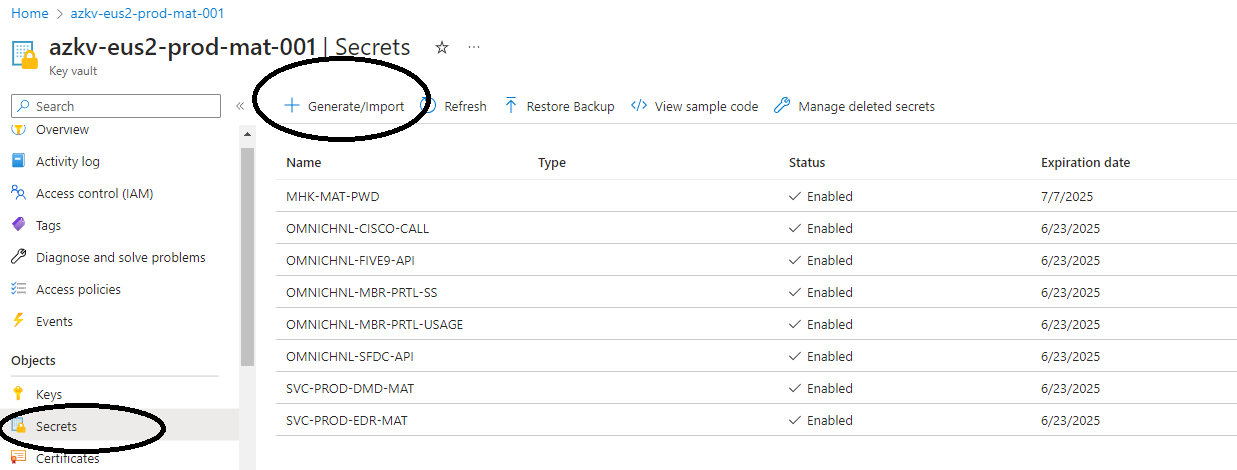


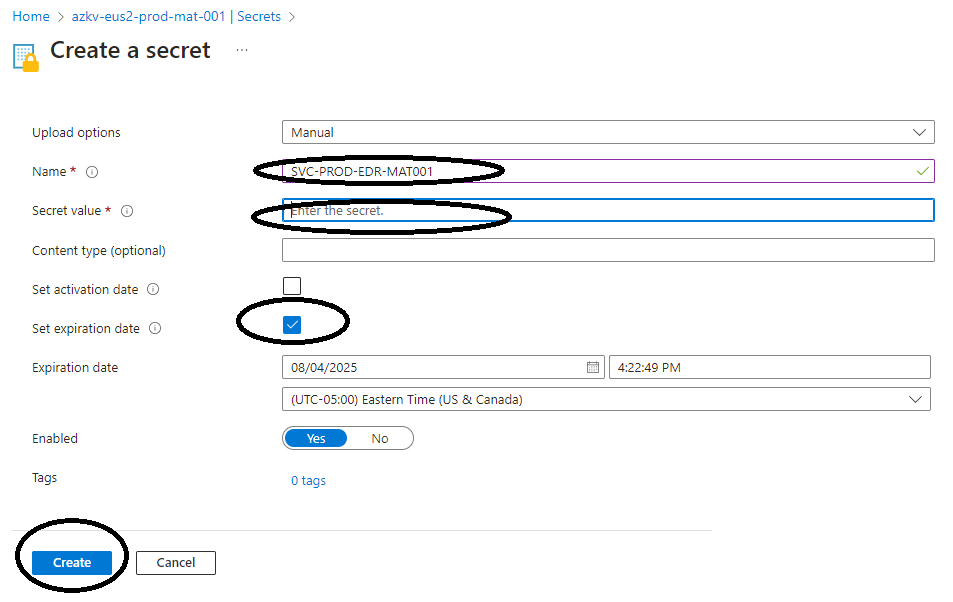
# Key Vault Integration

1. Raise a SNOW Azure Resource Creation to create key vaults with appropriate resource group on Microsoft Azure portal.
2. Raise a SNOW Cloud Access Form Request to Get a contributor Role for Matillion administrator on Newly Created key Vaults
3. Raise a SNOW Cloud Access Form Request to get a contributor role for Matillion VM’s to key vaults

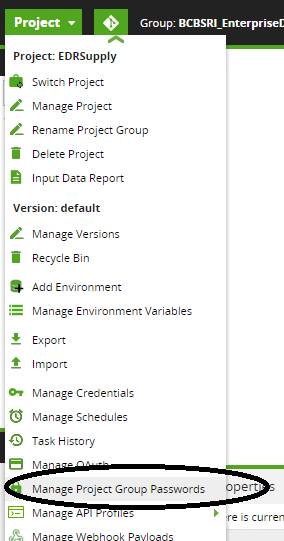
For e.g Need contributor role for VM bazeu2tmilmat01 to Key Vaults azkv-eus2-tmi-mat-001

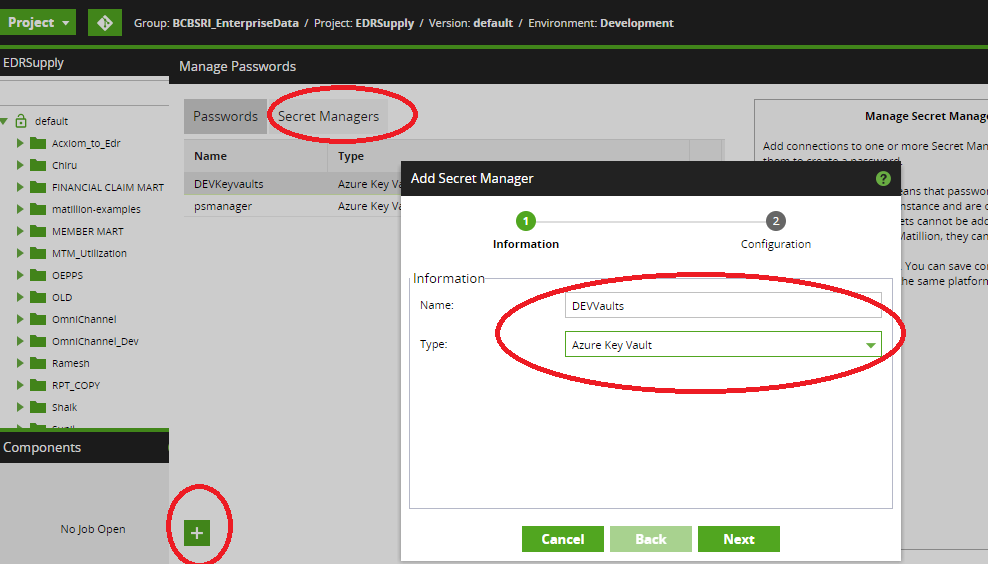
1. Login to azure portal and create the key vaults



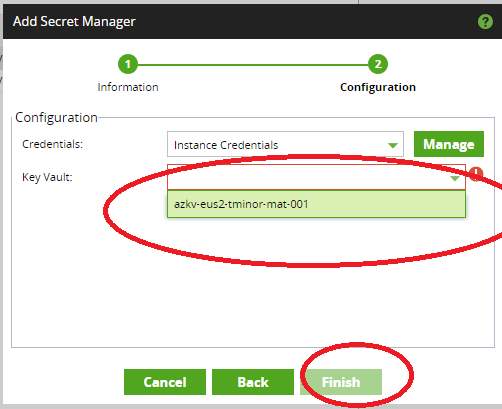


1. Matillion Key vault set up, choose ETL Project🡪click on Projects🡪 Manage Project Group Passwords

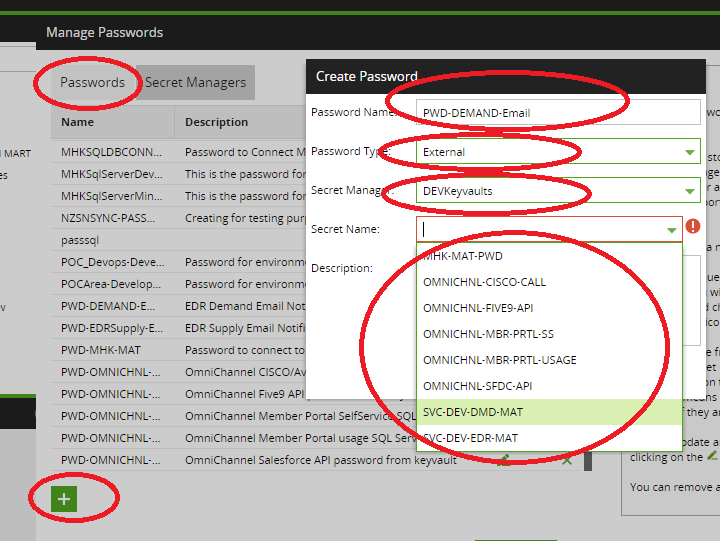




1. Choose key vault and finish



1. Fetch key-vaults defined in Azure portal - Choose a ETL Project🡪click on Projects🡪 Manage Project Group Passwords

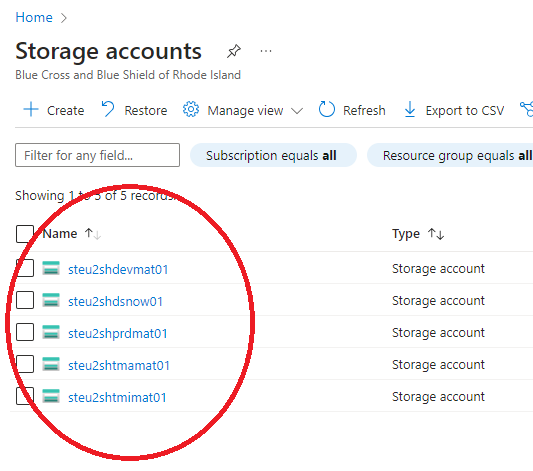


# Blob Storage Integration

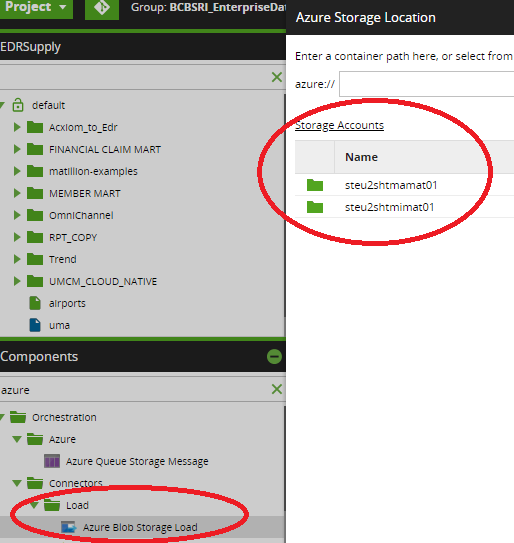
1. Raise a SNOW Azure Resource Creation to create Blob Storage with appropriate resource group on Microsoft Azure portal.
2. Raise a SNOW Cloud Access Form Request to Get a contributor Role for Matillion administrator on Newly Created Blob Storage
3. Raise a SNOW Cloud Access Form Request to get a contributor role for Matillion VM’s to k Blob Storage

e.g. Need contributor role for VM bazeu2plmat01 to storage account steu2shprdmat01

1. Login to azure portal and Verify Blob storage account



1. Verify Blob Storage in Matillion via component Azure Blob Storage Load



# Matillion DR

* We had conversation with Matillion Vendor regarding DR solution. They don’t have DR Solution for Cross Region zone out of the Box
* Possible solution for Cross Region DR is to implement Azure to Azure disaster recovery Architecture to achieve 99.9% SLA

## Architecture Diagram of Azure to Azure DR Setup, Active Passive mode

A screenshot of a computer

Description automatically generated

## Steps for DR Setup

1. Enable DR on Central US Region, Choose the US east VM and Click on Disaster recovery

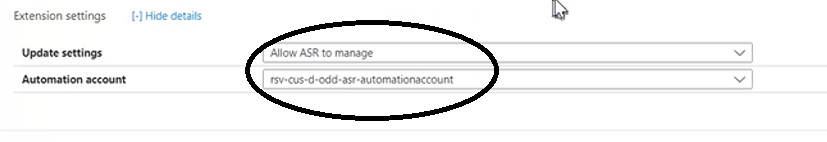
A screenshot of a computer

Description automatically generated with medium confidence

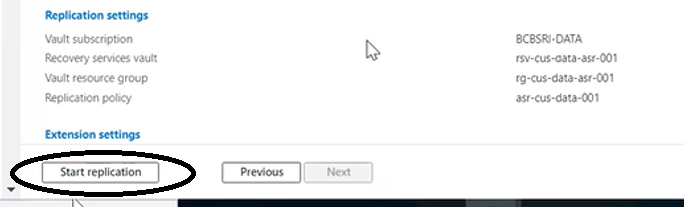
1. DR Replication Settings by Providing US central Vnet and Subnet’s



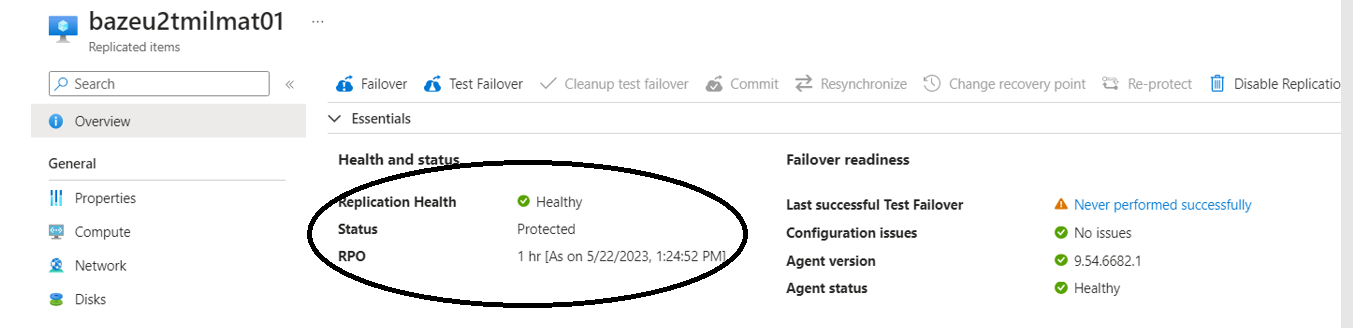




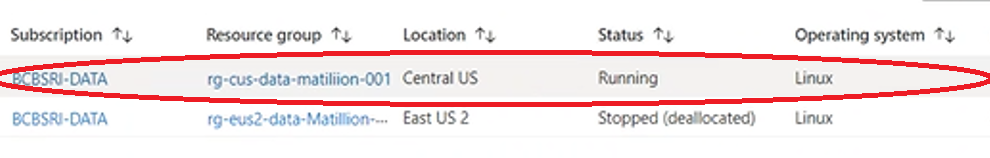
1. Enable Replication



1. DR Replication Health Check



1. Verify Matillion instance is UP and Running in US Central



## Firewall Rules

* Firewall Request to allow Matillion port on DR site (10.36.2.0/26)

## DNS Entry and Load Balancer

* DNS team to create a DNS Entry Called [https://matillion.Client X.org](https://matillion.bcbsri.org)

[Matillion DR- Technical Recovery Guide](https://bcbsri-my.sharepoint.com/personal/umashankar_annappa_bcbsri_org/Documents/Matillion/DR/Matillion%20DR-%20Technical%20Recovery%20Guide.xlsx?web=1)

## DR Testing

* DR Fail over Process from US East to US central
* DR Fail over Process from US Central to US East

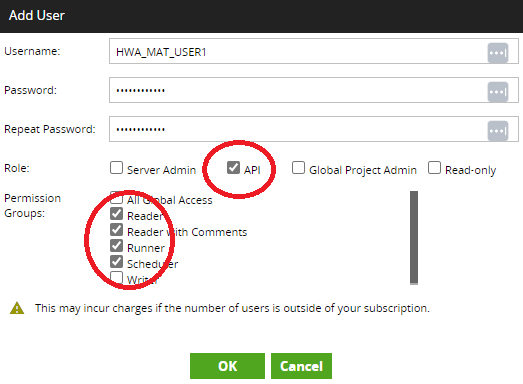
# Technical Steps for Password Rotation

## Architecture Diagram

A screenshot of a computer

Description automatically generated with medium confidence

* Get AD service accounts Created by GAM team
* Grant highlighted Permission to service account on Matillion ETL tool as below



* Provide the AD service account to HWA Team
* Request Centrify team to create a Matillion Password Utility

# Matillion DevOps

# Introduction

# Purpose of the document

Low Level Design (LLD) also known as Program Specifications contains the internal details of each component. The programmers prior to coding need these details, for developing the components. LLD / Program Specifications describe the inputs, outputs, local data, global data, and the algorithm for each program.

The main objectives of Low Level Design document are to:

* Translate the requirements stated in HLD & FS into a format amenable for coding.
* Form the basis for Unit Test Plan.

The LLD should be reviewed internally prior to commencement of subsequent phases of development.

# Scope

This document captures the Solution Design for DevOps process - Matillion GIT branching strategy and automated deployments using CI/CD pipelines.

The scope also covers how the pipelines are created, triggered and executed along with details of approval gates designed. The automated deployments are done to Dev, Minor, Major and Prod environments.

|  |  |  |
| --- | --- | --- |
| **Environment** | **Scope** | **Comments** |
| Dev Environment | Yes | Deployment process implemented using Azure DevOps using Yaml build pipelines, Classic release pipelines, Deployment gates, Python, Azure key vault, PowerShell scripting and Shell scripting |
| Minor Environment | Yes |
| Major Environment | Yes |
| Prod Environment | Yes |

# Tools/Services Used

# Azure SaaS

* 1. Azure DevOps
     + Azure Repos
     + Azure Pipelines

# Azure PaaS

* 1. Azure Key Vault

# Other Software

* 1. Python

# Intended Audience

* NTT DevOps developers and support team
* BIDW D&A Supply and Demand Developers
* Matillion Admins

# References

[API v1 - Git Integration | Matillion ETL Docs](https://documentation.matillion.com/docs/5971755)

<https://www.Matillion.com/data-productivity-cloud>

[https://docs.matillion.com/metl/docs/2974180/](https://docs.matillion.com/metl/docs/2974180/" \o "https://docs.matillion.com/metl/docs/2974180/" \t "_blank)

# DevOps Solution Architecture for Matillion

# Solution Architecture

DevOps solution is built using Azure DevOps services. This enables source code management (storage, track changes, multi user development), build and deploy code into different environments with a set of policies defined.

# Architecture Components

# Azure DevOps

Azure DevOps is a suite of products that support a collaborative culture and set of processes that bring together developers, project managers, and contributors to develop software. It allows organizations to create and improve products at a faster pace than they can with traditional software development approaches.

DevOps is a software development practice that promotes collaboration between development and operations, resulting in faster and more reliable software delivery. Commonly referred to as a culture, DevOps connects people, process, and technology to deliver continuous value.

# Azure Repos

Azure Repos is a set of version control tools that you can use to manage your code. Whether your software project is large or small, using version control as soon as possible is a good idea. Version control systems are software that help you track changes you make in your code over time.

Even if you're just a single developer, version control helps you stay organized as you fix bugs and develop new features. Version control keeps a history of your development so that you can review and even roll back to any version of your code with ease.

# Azure Artifacts

Azure Artifacts enables developers to share their code efficiently and manage all their packages from one place.

# Azure Pipeline

Azure Pipelines automatically builds and deploy code projects. Azure pipeline usually consists of CI/CD – Continuous Integration/Continuous Delivery

# CI – Continuous Integration

Continuous Integration (CI) is the practice used by development teams of automating, merging, and testing code. CI helps to catch bugs early in the development cycle, which makes them less expensive to fix.

# Repo Details

Below 2 Repos are created for Supply and Demand projects respectively.

[https://dev.azure.com/CLIENT X/BIDW/\_git/repo\_supply\_Matillion](https://dev.azure.com/BCBSRI/BIDW/_git/repo_supply_matillion)

[https://dev.azure.com/CLIENT X/BIDW/\_git/repo\_demand\_Matillion](https://dev.azure.com/BCBSRI/BIDW/_git/repo_demand_matillion)

# Environment and Matillion server details

|  |  |  |
| --- | --- | --- |
| **Environment** | **Server ID** | **Branch Integrated** |
| Development | 10.31.8.7 | mat\_dev |
| Minor | 10.31.8.6 | mat\_minor |
| Major | 10.31.8.9 | mat\_major |
| Production | 10.31.8.10 | mat\_prod |

# Branching Strategy

A diagram of a computer

Description automatically generated

* The user branches are created (with a defined naming standard) based on main branch to have sync up with the current running code in Production environment.
* The release branches are created under release folder with a defined naming standards
* The developer created Pull Request to merge code from users branch to release branch
* Through approval gates the code in release branch is first merged into individual environment branch (mat\_dev, mat\_minor, mat\_major, mat\_prod) before deployment to respective environments. Code gets merged in each environment branch. Environment branches are needed in Matillion as complete branch gets deployed in the respective environments; hence merging of code with previously deployed code is imperative.

A screenshot of a computer

Description automatically generated

# Branch Policies

Below are the policies defined and code merge process is followed

1. Developers to create user branch under users/<individual name>/ with a separate folder for each project
2. Each project folder to be created in the format <TOPAZID>\_<TOPAZ DESC>
3. Developers to create a branch under release branch as rel\_<TOPAZID>\_<TOPAZ DESC>
4. Code from individual user branch is merged with respective release branch using a Pull Request
5. Team lead to review the Pull Request and take action to merge the code into release branch
6. From release branch, the code is merged into mat\_dev branch through approvals gates from team lead and then code is deployed into dev environment
7. Post deployment of code into development environment, unit Testing and IIT is done by developer
8. Any additional modifications of code identified (addition, deletion or modification), the new code is committed to the same user branch and steps 4 to 7 are repeated
9. Post successful unit testing and IIT, team lead and Matillion admin to give approvals (through approval gates) to merge the code into mat\_minor branch
10. Post deployment of code into minor environment, testing team to complete the required unit, integration and regression testing
11. Post successful completion of testing, team lead, test lead and Matillion admin to give approvals (through approval gates) to merge the code into mat\_major branch
12. Post deployment to major environment, testing team to complete required regression testing
13. Post successful completion of testing, team lead, EDR support team and Matillion admin to give approvals (through approval gates) to merge the code into mat\_prod branch
14. Approval gates are defined at each task which need Lead’s review and approval and also Matillion Admin’s review and approval

A screenshot of a computer

Description automatically generated

# Matillion GIT Integration Set Up

Source Control Management (SCM) API in Matillion ETL. Matillion ETL uses Git for its version control, allowing teams to track changes (both small and large) to a project or a shared job with speed and efficiency.

Below are the steps for One time User Configuration settings for GIT integration for Matillion pipeline

1. Create a service account user in Matillion

A screenshot of a computer

Description automatically generated

1. Add API access privileges to service account

A screenshot of a computer

Description automatically generated

1. Navigate to Matillion Git

A computer screen with a white screen

Description automatically generated

1. Click on the Init Local Repository. Code in the default version will be added to Master branch of Matillion local Git repository. Commit the code to the Master branch.

A computer screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Click on Clone Remote Repository and establish connection to remote repository (Azure Repo)

A computer screen shot of a computer

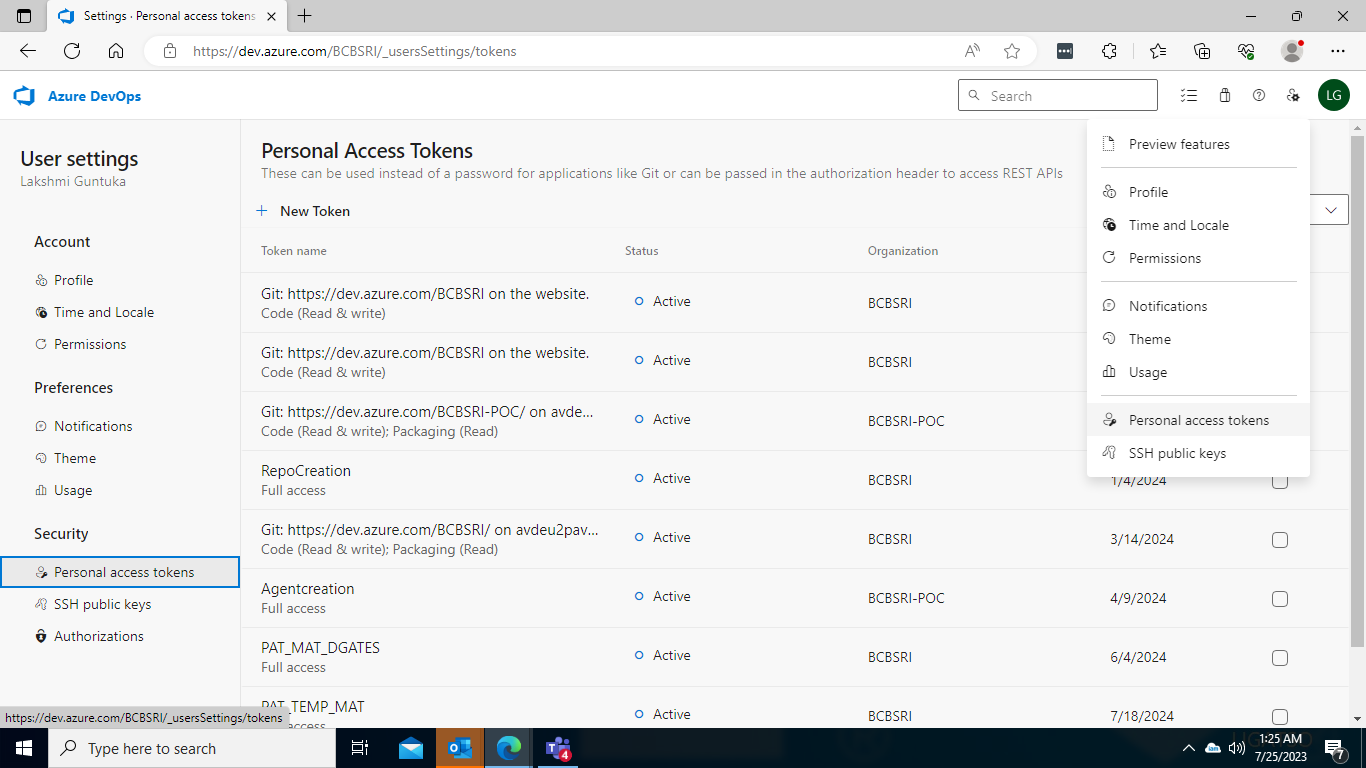
Description automatically generated

1. Provide service account credentials to connect with remote repository (PAT token for service account generated from Azure DevOps)

A screenshot of a computer

Description automatically generated

1. The service account credentials include password credentials as PAT token which are created in azure repo to that particular service account



1. Configuration of Azure Repo URL in settings

A screenshot of a computer

Description automatically generated

# Steps for Developer

The individual developer works on their own version in Matillion. As a first step the developer fetches all the latest updates from remote repo. Then the developer checks out the latest production code from mat\_prod into his/her version. Post this the developer starts working on his/her changes. Once all the changes are completed and unit tested, the developer commits the code into local repository by creating a new user branch as per the branch policies and pushes his/her changes into the Repository. The developer then creates a release branch as per the standard set and creates a Pull Request to merge his/her code from user branch to release branch.

1. Individual developer Version Creation in Matillion for developer

* Developer clicks on Manage Version from the Project Drop Down and selects Manage Version

A screenshot of a computer

Description automatically generated

* Developer selects the respective version and selects Switch Version and clicks on OK

A screenshot of a computer

Description automatically generated

1. Each developer works on his/her own version of code by checking out the production version of code from mat\_prod branch

* Developer Fetches all branches by clicking on GIT and by checking both options as shown below

A screenshot of a computer

Description automatically generated

* Developer checks out the mat\_prod branch to his/her version

A screenshot of a computer

Description automatically generated

1. Developer commits the code into new user branch

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Developer pushes the changes into remote branch

A screenshot of a computer

Description automatically generated

1. Developer creates Pull Request to merge code from user branch to release branch

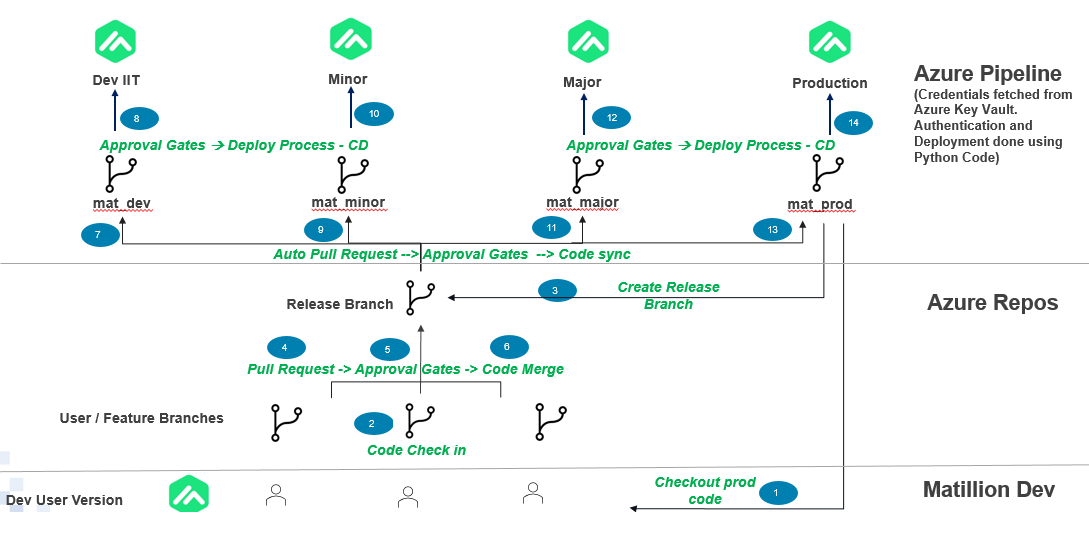
A close up of words

Description automatically generated

# Continuous Delivery

Continuous delivery (CD) is the process of automating build, test, configuration, and deployment from a build to multiple environments

# Multi Environment Deployment Diagram



**Step 1:**

The feature and release branches are cloned from mat\_prod branch to have sync up with the latest changes.

**Step 2:**

Developers develop their code and commit it to feature branch i.e., feature1, feature2 and feature3 under users/RSS/ folder

**Step 3:**

The release branches i.e., release/rel\_\* are mapped to release pipeline with repository information as artifact to the release pipeline. The below diagram depicts the pipeline configuration set up for reference.

A screenshot of a computer

Description automatically generated

**Step 4:**

In deployment job the **Linux prod agent** is configured with python 3.8 version. Below is the screenshot of agent mapping to the release pipeline.

A screenshot of a computer

Description automatically generated

**Step 5:**

In Sync with Mat\_Dev, Sync with Mat\_Minor, Sync with Mat\_Major, Sync with Mat\_Prod jobs windows-latest Microsoft hosted agent is used. Tasks are configured to create Pull Request with the API’s. Below is the screenshot for the reference.

A screenshot of a computer

Description automatically generated

1. The above job contains two tasks. First to Create Pull Request and second to execute PowerShell Script. Create Pull Request task creates the pull request automatically once code merge into release/ branch is completed
2. PowerShell Script task performs a check whether the pull request creation is successful by fetching the pull request id. Below is the screenshot for the reference.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Invoke REST API’s is configured to check the approval of Auto Pull Request created every 5 minutes. This configuration is setup in the post deployment gates. Only upon approval of the PR, the merge to mat\_dev branch is triggered. Below is the screenshot for the reference

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. The REST API’s is configuration showing PR completion status check and re-evaluation of the status every 5 minutes with 3 days wait time

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Step 6:**

The deployment job in the release pipeline includes three tasks:

* + - * 1. Azure Key Vault configuration
        2. Command Line Script
        3. Execute Python script.

**Release pipeline Configuration link:**

[https://dev.azure.com/CLIENT X/BIDW/\_releaseDefinition?definitionId=23&\_a=definition-tasks&environmentId=117](https://dev.azure.com/BCBSRI/BIDW/_releaseDefinition?definitionId=23&_a=definition-tasks&environmentId=117)

**Step 7:**

The service connection for the release pipeline is **azsc-eus2-data-Matillion-001** and it is for resource group(s) which holds key vault resources.

In Azure portal for respective key vault App registration of the Azuredevops service connection needs to be setup. so that key vault can be access through python script. The key vault name for the dev stage task is **azkv-eus2-dev-mat-001,** for minor stage task **azkv-eus2-tminor-mat-001**, for major stage task **azkv-eus2-tmajor-mat-001** and for prod stage task **azkv-eus2-prod-mat-001**. Please find the below screenshot on dev stage for the reference.

A screenshot of a computer

Description automatically generated

**Step 8:**

**Azure Key Vault: azkv-eus2-dev-mat-001** - This task will connect to azure subscription azsc-eus2-data-Matillion**-**001 and key vault azkv-eus2-dev-mat-001. In this secret filter is defined as **SVC-DEV-DMD-MAT** in which password of the Matillion server resides. Below is the screenshot for the reference.

A screenshot of a computer

Description automatically generated

**Step 9:**

**Command Line Script –** This task stores the Matillion server password into secret.txt file. The screenshot is attached below for the reference.

A screenshot of a computer

Description automatically generated

**Step 9:**

**Run a Python script –** This task executes python script with the arguments defined in the task section. It uses the below three API’s to deploy the code.

1. SCM Fetch – From Azure repository it will fetching the code and put it into Matillion agent server
2. SCM getstate – This will be finding all the commit ids
3. SCM switch commits – This will be switching latest commit id with the deployment code branch

The code is deployed into the Dev stage. In the similar way release branch code is merged to mat\_minor through automated PR. mat\_minor code then gets deployed into minor environment. Similar way of deployment is performed for both major and prod environments.

Below is the screenshot for reference.

A screenshot of a computer

Description automatically generated

**Python script used in Pipeline:  
  
**

# Monitoring / Alerting

**Scope:**

This covers the monitoring of the Matillion instance, its jobs, and the underlying Azure VM, as well as the setup of alerts for specific incidents.

**4. Procedure:**

**4.1 Setting up Monitoring for Azure VM:**

**1. Azure Monitor Setup:**

- Navigate to the Azure portal.

- Go to 'Monitoring' > 'Azure Monitor'.

- Under the 'Getting started' section, click on 'Virtual machines'.

- Choose your Matillion VM and enable monitoring.

**2. Metrics to Monitor:**

- CPU Usage

- Memory Usage

- Disk I/O

- Network In/Out

3**. Log Analytics:**

- Integrate with Azure Log Analytics to centralize logs.

- Create log queries to track specific events or conditions.

**4.2 Setting up Monitoring for Matillion Jobs:**

**1. Matillion Logging:**

- Within Matillion, navigate to 'Project' > 'Settings' > 'Logging'.

- Set the logging level to 'Detailed' to capture all job-related activities.

**2. Metrics to Monitor:**

- Job Execution Time

- Job Failures

- Transformation Errors

**4.3 Setting up Alerting:**

**1. Azure VM Alerts:**

- In Azure Monitor, under the 'Alerts' section, click on '+ New alert rule'.

- Define the alert criteria based on the metrics (e.g., CPU Usage > 80% for 10 minutes).

- Set up the action group to define what actions need to be taken when the alert is triggered (e.g., send an email, trigger a webhook, etc.).

**2. Matillion Job Alerts:**

- Use Matillion's in-built notification components like 'SNS Notify' or integrate with third-party solutions like Slack or Email.

- Set up alerts for job failures or when specific conditions are met during job execution.

**3. Alert Maintenance:**

- Regularly review and update the alerting thresholds to ensure relevance.

- Test alerts periodically to confirm they are working as expected.

# Automation



**Short Summary:**

The script (when fully implemented) is designed to facilitate the deployment of infrastructure in Azure tailored for Matillion ETL for Snowflake. Here are the main actions and resources involved:

**1. Azure Authentication:**

- Authenticates with Azure using `DefaultAzureCredential()` which provides a seamless authentication experience when running the script in different Azure environments.

**2. Virtual Machine (VM) Deployment:**

- Deploys a Linux Virtual Machine.

- The VM uses an image provided by Matillion designed for their ETL process for Snowflake.

- The VM has specific configurations related to its location, OS, hardware, storage, and network properties.

3**. Blob Storage Creation**

- Sets up Azure Blob Storage accounts.

- These storage accounts are named: `steu2shdevmat01`, `steu2shdsnow01`, `steu2shprdmat01`, `steu2shtmamat01`, and `steushtmimat01`.

- These Blob Storage accounts can be used by Matillion for tasks like storing intermediate ETL data, logs, or configuration backups.

4. **Key Vault Creation :**

- Creates two Azure Key Vaults named `azkv-eus2-tmi-mat-001` and `azkv-eus2-prod-mat-001`.

- Azure Key Vault is a cloud service for securely storing and accessing secrets. Secrets can be passwords, database connection strings, API keys, etc.

- Matillion can be integrated with Azure Key Vault to fetch necessary secrets for its operations, enhancing security by avoiding hard-coded or exposed credentials.

5. **Integration of Matillion with Blob Storage and Key**

- Once the Blob Storage and Key Vaults are set up, the intention is to configure Matillion to utilize these resources. This might involve setting up connection configurations within Matillion to read from or write to Blob Storage and fetch secrets from Key Vault.

By the end of this setup, you'll have a Matillion instance running in Azure with its own storage mechanism and a secure way to handle secrets, both critical components for robust ETL operations.

**User Instructions:**

**1. Azure SDK Installation:**

- Before you can run the script, you need to have the required Azure SDK packages installed. You can do this using pip:

**pip install azure-mgmt-compute azure-mgmt-network azure-identity**

`

**2. Azure Authentication:**

- If you're running this script locally (not within Azure Cloud Shell or on an Azure VM), you need to authenticate using Azure CLI. First, install the Azure CLI and then run:

**az login**

This will open a browser window asking you to log in to your Azure account. Once logged in, the script will be able to use this authentication.

**3. Update the Script Variables:**

- Open the script in an editor and replace placeholders like `<YOUR\_AZURE\_SUBSCRIPTION\_ID>`, `<YOUR\_RESOURCE\_GROUP\_NAME>`, and others with your actual Azure details.

**4. Networking Pre-requisite:**

- Ensure you have a network interface (`network\_interface\_id` in the script) ready to be attached to the VM. If not, create one using the Azure Portal, Azure CLI, or another method.

**5. Execute the Script:**

- Run the script using Python:

**python <script\_name>.py**

Replace `<script\_name>` with the name you've saved the script as.

# Appendix

