Deployment Options in Matillion ETL

This article describes the second of three commonly-used choices for how to manage and deploy your Matillion solution between multiple environments: for example development – test – production. Note that in this series we're looking exclusively at options which are available through Matillion ETL's web user interface. Additional options are available using Matillion's REST API.

WITH THIS METHOD, YOU CREATE MULTIPLE PROJECTS. FOR EXAMPLE:

ONE PROJECT FOR DEVELOPMENT
ONE PROJECT FOR QA
ONE PROJECT FOR PRODUCTION

YOU THEN PROMOTE CODE BETWEEN PROJECTS WHEN IT HAS BEEN DEVELOPED AND TESTED.

– Using Multiple Projects

In order to describe the details, first a few definitions. What do we actually mean by "Matillion code"? And what's a Project as opposed to an "Environment"?

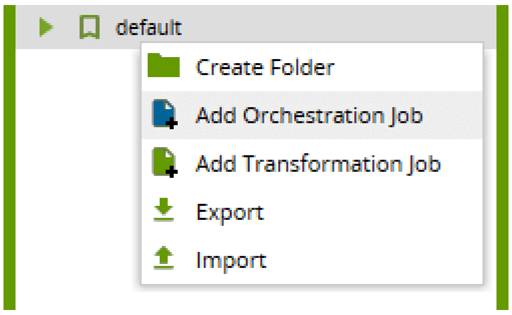
Orchestration and Transformation Jobs

Matillion is an ELT tool. It does two main things on your behalf:

1. Data ingestion – in other words loading data from external sources into the database
2. Data transformation – getting your data ready for visualization or analysis, for example, integration, reshaping, applying business rules, aggregation, and deriving calculated fields.

Deployment Options in Matillion ETL

These two things are implemented by Orchestration Jobs and Transformation Jobs respectively.



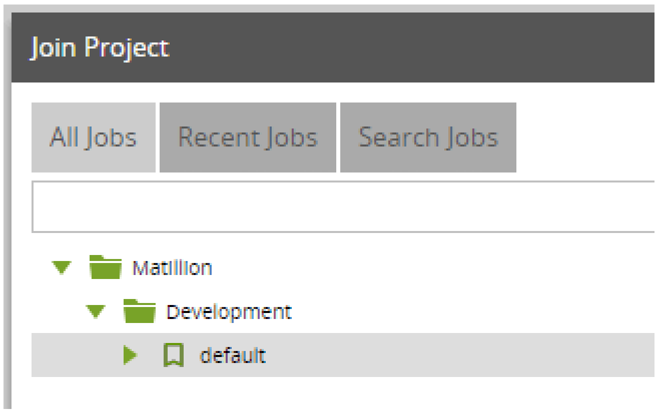
Orchestration jobs fulfill an additional command-and-control function: they define the overall sequence of events and can be scheduled. Orchestration jobs can call Transformation jobs as part of their work.

“Matillion Code” means the definition of the Orchestration and Transformation Jobs that you are using. Matillion Jobs always exist inside a Project.

Matillion Projects

A Project is simply a collection of metadata. When you’re working in Matillion you are always within the context of a Project. When you log in, it’s the first dialog that pops up after the login credentials.

Deployment Options in Matillion ETL



You'll need to select a Group, a Project and a Version. In the above screenshot, these are:

- Group: **Matillion**
- Project: **Development**
- Version: **default**

Every Group can contain multiple projects. You can define more than one Group although it's often simplest to stick with just one. A good choice for the Group name is your company name.

One Project can contain multiple Versions. You can create your own version snapshot at any time, but there's always a *current* one named "default".

Among other things, Projects contain:

- Orchestration and Transformation Job definitions
- Folder structure
- One or more Environments

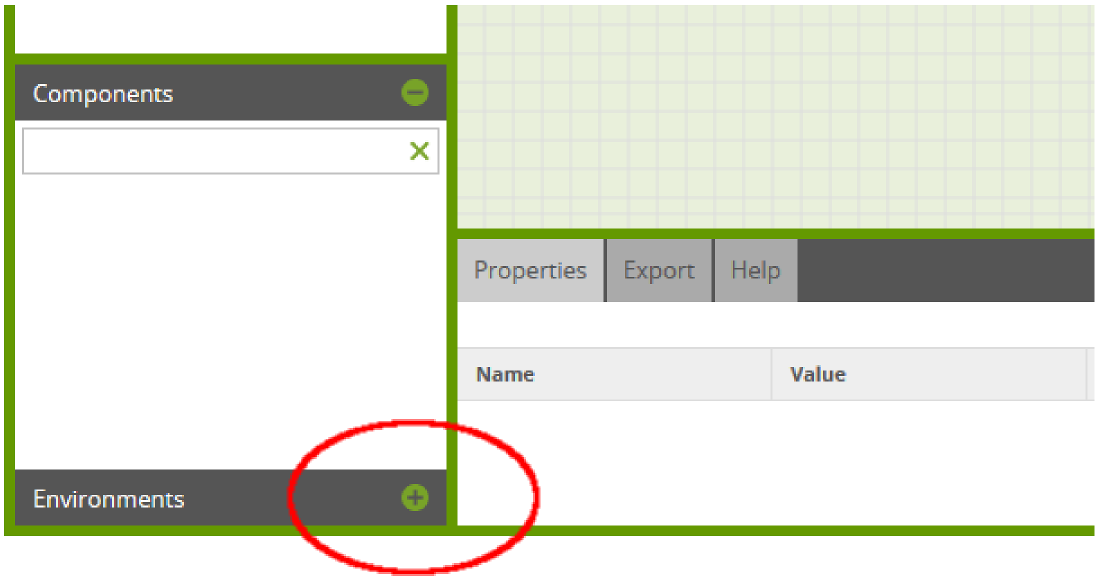
It's the "Environments" which define where the Matillion Jobs can be executed at runtime.

What's a Matillion Environment?

Deployment Options in Matillion ETL

A Matillion Environment defines a target data warehouse.

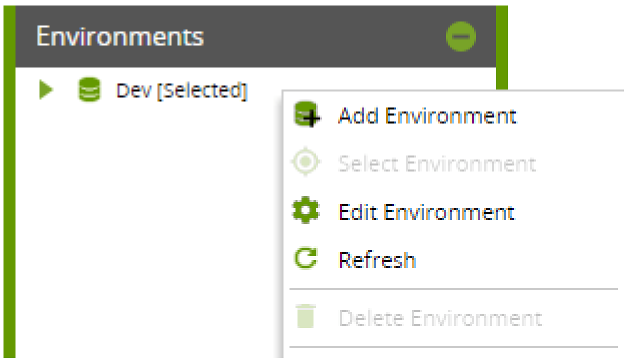
To manage your Environments you'll need to expand the panel at the bottom left of the screen, which is minimized by default.



When you first launched Matillion, you went through an initial configuration screen. This asked for details of the target data warehouse plus a couple of other configuration items. For this reason, you'll always have at least one Environment.

You can manage environments through the right-click context menu. The Edit Environment option will take you back to that initial configuration screen, where you can change the settings if necessary.

Deployment Options in Matillion ETL



You can add, edit and remove Environments using these options.

Note that:

1. Matillion has an overall cap on the total number of Environments that may exist within your installation. The cap depends upon the instance size. When you use multiple Projects then each Project will have at least one Environment, and they all count towards the cap. You'll need to switch between Projects to count the total number.
2. You are not permitted to delete the last Environment within a Project.

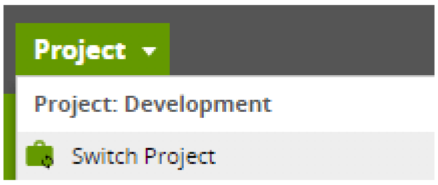
Solution Overview

With this code deployment option:

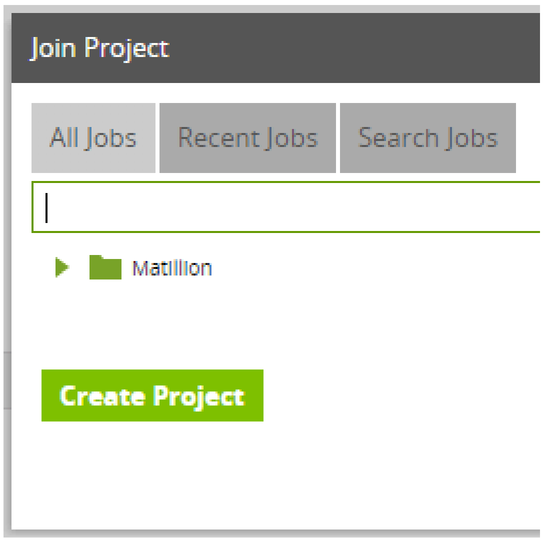
- You define multiple Projects (development, test, production)
- Every Project owns one corresponding Environment (development, test, production)
- You copy the code from one Project into another once it's ready to be promoted

If you started out with just one "Development" project, you might now add a "Production" project. Choose Switch Project from the Project menu.

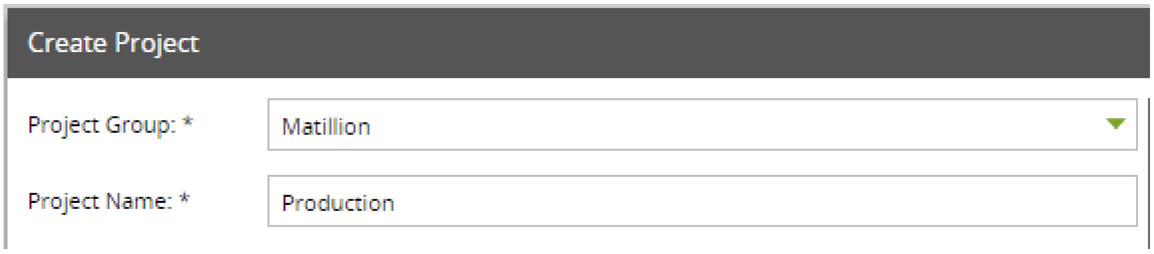
Deployment Options in Matillion ETL



Then press the **Create Project** button.



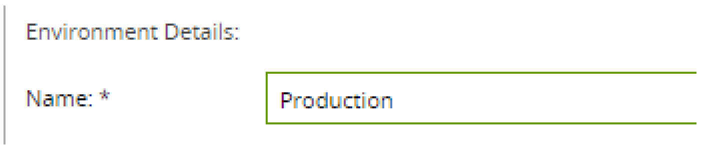
Be sure to choose the same Group name (your company name) from the **Project Group** dropdown list, and name your new **Project Name** "Production":



Project Group: *	Matillion
Project Name: *	Production

Deployment Options in Matillion ETL

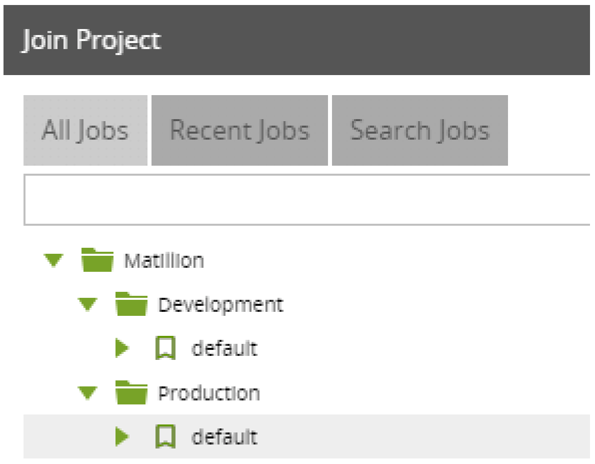
You're also creating a new Environment at this point (remember, every Project must always have at least one Environment), and it's a good idea to name this "Production" too:


Environment Details:

Name: * Production

Now when you follow Project / Switch Project you should find two Projects:

- A "Development" project, which owns an Environment named "Dev", pointing to the Development target data warehouse.
- A "Production" project, which owns an Environment named "Production", pointing to the Production target data warehouse.



Deployment Options in Matillion ETL

Developers would log onto the Development project to do their work. When finished, they would export their work ready for a DBA or DevOps person to log onto the Production project to import and schedule the jobs.

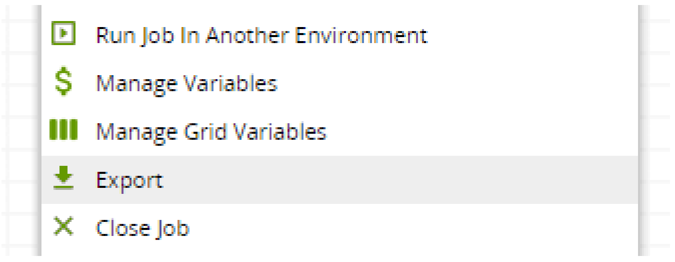
You may extrapolate this scenario to fit any combination of environments that you need. Some examples are:

- Dev → Production
- Dev → System Test → Production
- Dev → System Test → UAT → Production

Exporting and Importing Jobs

Matillion's mechanism for copying job metadata from one Project into another is via the Export/Import feature.

While editing a Job, you can find the Export option from the right-click context menu:



You can also get to the same dialog by following the Project / Export menu. The dialog allows you to select one or more Jobs you want to save and download as a JSON file. You can put this file into external version control.

From the Project / Import menu, you can import a Matillion-generated JSON file, and choose which jobs to import into the target Project.

BY DEFAULT, THE JOBS WILL BE ADDED INTO THE SAME FO
STRUCTURE AS THEY WERE AT THE SOURCE.

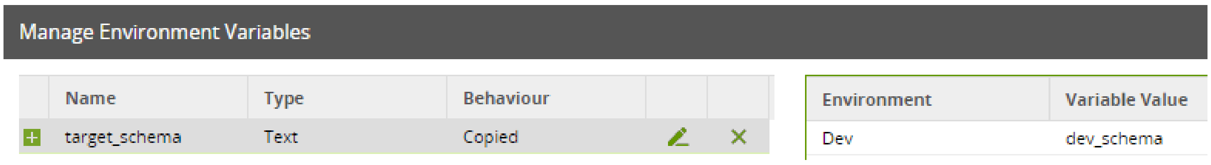
YOU'LL FIND IT EASIEST TO MAINTAIN EXACTLY THE SA
FOLDER STRUCTURE IN DEVELOPMENT AS IN PRODUCTIO

Matillion also has various REST API endpoints for exporting and importing Jobs. These are used for automation, although be aware that API-generated metadata formats are not compatible with the UI-based export and import.

A note on Environment Variables

Environment variables have a very useful feature in that you can set a different default per Environment.

While logged into the Development Project, you might have a variable named ***target_schema***. Its default value (for the Dev environment) is set to ***dev_schema***.

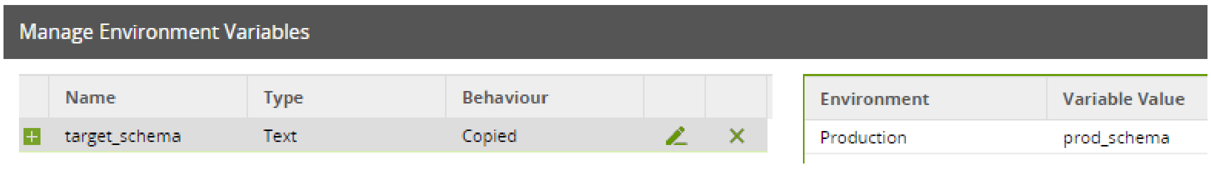

Manage Environment Variables

	Name	Type	Behaviour		
+	target_schema	Text	Copied	✎	✕

Environment	Variable Value
Dev	dev_schema

Similarly, while logged into the Production Project, the same variable can exist, but with a default value (for the Production environment this time) set to ***prod_schema***.

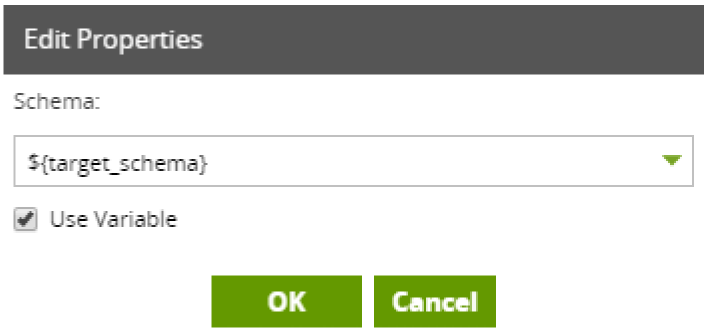
Deployment Options in Matillion ETL


Manage Environment Variables

	Name	Type	Behaviour		
+	target_schema	Text	Copied	✎	✕

Environment	Variable Value
Production	prod_schema

So, instead of hardcoding the schema name of a Table Input component, for example, you could set it to ${target_schema}


Edit Properties

Schema:

${target_schema}

☑ Use Variable

Then, in Development the Table Input would automatically read from the ***dev_schema***, and once deployed into the Production Project it would automatically start to read from the ***prod_schema*** with no code change required.

Incidentally, this is why when you export an Environment Variable, the default value is not among the properties that are saved into the JSON export file.

Summary

When you use multiple Projects to manage code deployment, you set up multiple parallel copies of the codebase in a different Project, and each copy pointing to a different target data warehouse (dev, test or production).

Deployment Options in Matillion ETL

You control exactly when and what code you promote to the different environments. Also, you can take advantage of the Environment Variable feature which allows you to automatically customize job behavior according to where it is running.

Backup mechanisms work as normal and are still recommended.

Be aware of Matillion ETL's cap on the total number of Environments that you can create because this applies across all projects simultaneously.

Some metadata items, including user logins, OAuth credentials, and the Password Manager are not Project-specific. These are server-wide and automatically apply to every Project. More on this subject in the third article in this series.

Reference : https://www.matillion.com/blog/deployment-options-in-matillion-etl-using-multiple-projects

Deployment Options in Matillion ETL

Using Environments

This is the first blog in a three-part series on Matillion ETL deployment options. This article describes the first of three commonly-used choices for how to manage and deploy your Matillion solution between multiple environments: for example development - test - production. Note that in this series we're looking exclusively at options which are available through Matillion ETL's web user interface. Additional options are available using Matillion's REST API.

WITH THIS METHOD, YOU DON'T ACTUALLY MOVE ANY CO
AT ALL!

INSTEAD, YOU SIMPLY RUN A SINGLE VERSION OF THE
MATILLION CODE AGAINST THE ENVIRONMENT OF YOUR
CHOICE.

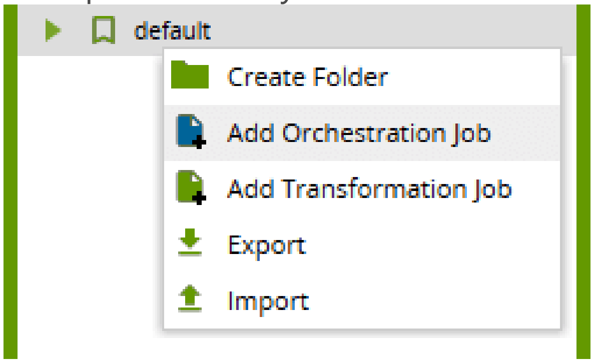
In order to describe the details, first a few definitions. What do we actually mean by "Matillion code" and, what's an "Environment"?

Orchestration and Transformation Jobs

Matillion is an ELT tool. it does two main things on your behalf:

1. Data ingestion - in other words loading data from external sources into the database
2. Data transformation - getting your data ready for visualization or analysis; for example, integration, reshaping, applying business rules, aggregation, and deriving calculated fields.

These two things are implemented by Orchestration Jobs and Transformation Jobs respectively.



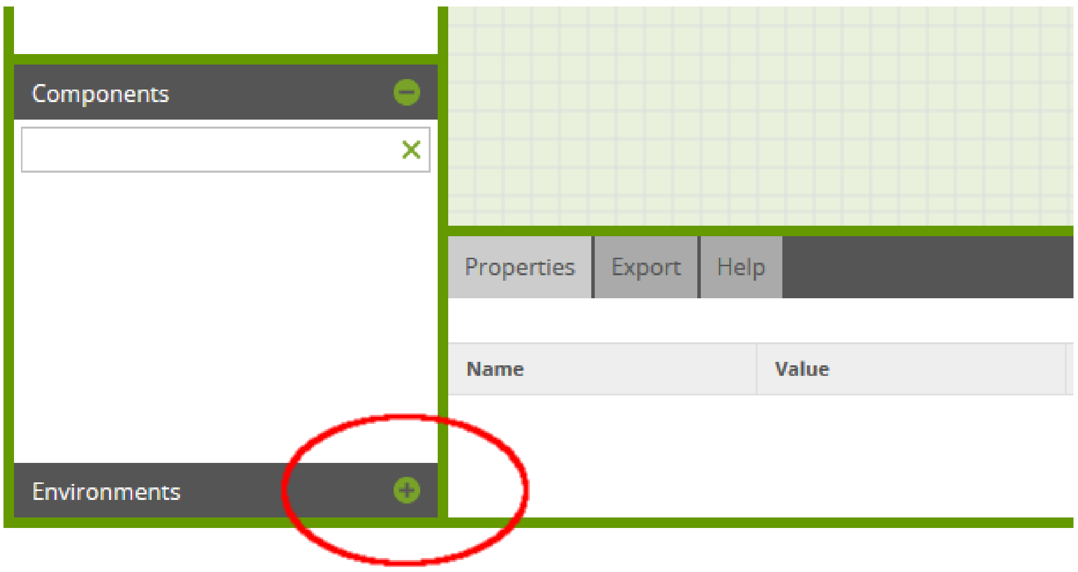
Orchestration jobs

Deployment Options in Matillion ETL

fulfill an additional command-and-control function: they define the overall sequence of events and can be scheduled. Orchestration jobs can call Transformation jobs as part of their work. "Matillion Code" means the definition of the Orchestration and Transformation Jobs that you are using. So, at runtime, where does a Matillion Job actually execute? Against an "Environment".

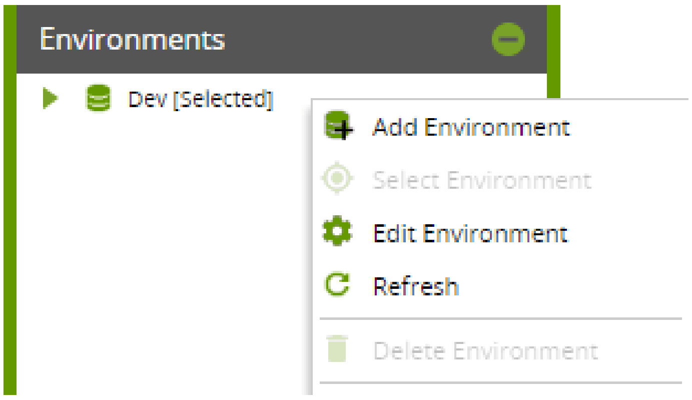
What's a Matillion Environment?

A Matillion Environment defines a target data warehouse. To manage your Environments you'll need to expand the panel at the bottom left of the screen, which is minimized by
default.



When you first launched Matillion, you went through an initial configuration screen. This asked for details of the target data warehouse plus a couple of other configuration items. For this reason, you'll always have at least one Environment. You can manage environments through the right-click context menu. The Edit Environment option will take you back to that initial configuration screen, where you can change the settings if

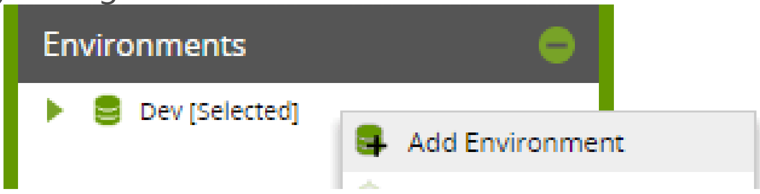
Deployment Options in Matillion ETL

necessary. 

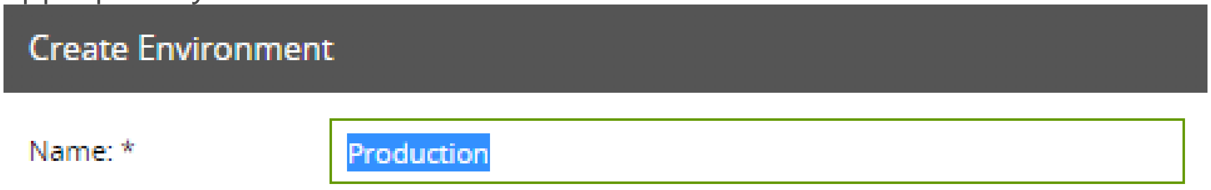
You can add, edit and remove Environments using these options. Note that:

1. Matillion has an overall cap on the total number of environments that may exist within your installation. The cap depends upon the instance size. If you have used multiple Projects then each Project will have at least one Environment, and they all count towards the cap even if they are actually pointing at the same target. There's more on this subject in the second post of this series.
2. You are not permitted to delete the last Environment within a Project.

Solution Overview

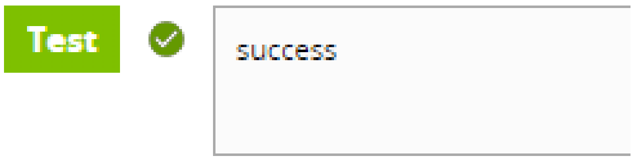
With this simple code deployment option, you don't move the Matillion code at all: you just run it against different Environments. If you already had a "Dev" environment, you might add a "Production" environment. 

Then choose the name, and fill in the details of your Production target data warehouse appropriately.



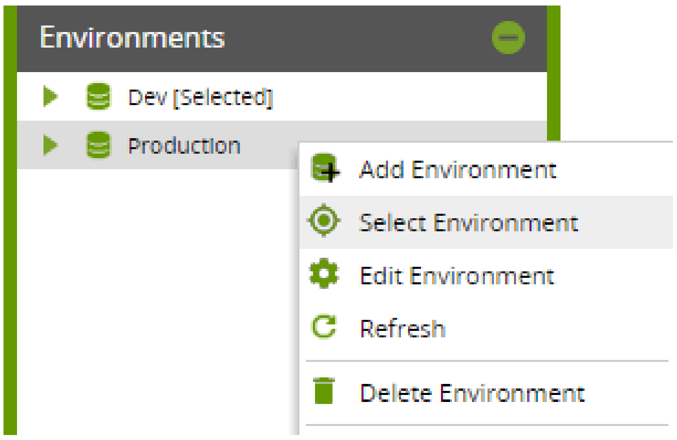
Name: * Production

Whenever you create or edit an Environment, always press the **Test** button, and ensure that you see a ‘Success’



message!

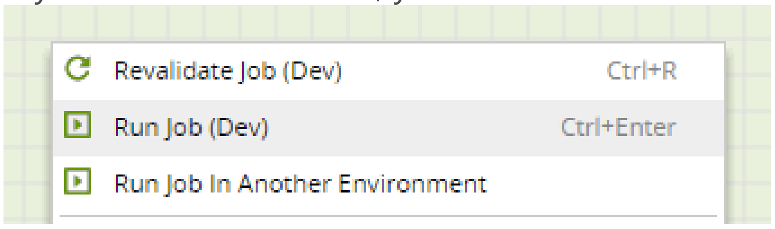
Provided you’re still within the total number of permitted Environments, you’ll now have two entries listed in the Environments panel.



Note that:

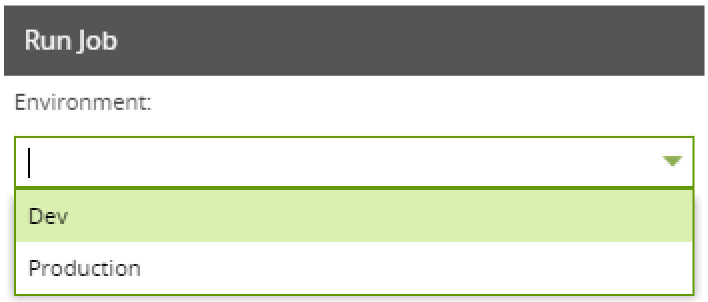
- One Environment is always marked [Selected], and you can change this by using the “Select Environment” context menu option.
- You can delete Environments if you have created more than one

Now when you are in a Job editor, your context menu will have additional options:

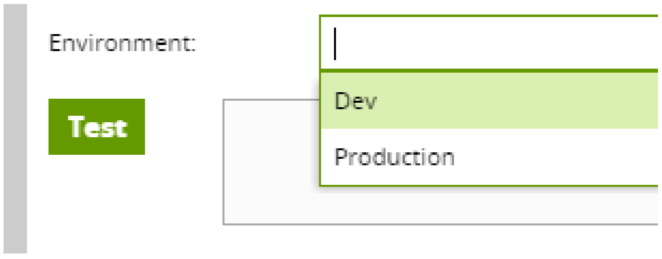


- Run Job (Dev) - means to run the job against the Environment which is marked as [Selected]
- Run Job in Another Environment - opens another dialog which allows you to choose which Environment to run the job against

Deployment Options in Matillion ETL



You'll also see the same drop-down menu in the Maintain Schedule



editor.

Summary

There is no code movement at all with this method of "code deployment". Instead, you take advantage of the fact that Matillion ETL can have more than one target environment defined - for example, development, test, and production. At runtime, you choose which one to use. Normal backup mechanisms still work and are still recommended. This method of code deployment is the simplest of the three but correspondingly offers the least governance.

Reference : https://www.matillion.com/blog/deployment-options-in-matillion-etl-using-environments

Deployment Options in Matillion ETL

Using Multiple Instances

This is the third blog in a three-part series on Matillion ETL deployment options. This article describes the third of three commonly-used choices for how to manage and deploy your Matillion solution between multiple environments, for example, development – test – production. Note that in this series we're looking exclusively at options which are available through Matillion ETL's web user interface. Additional options are available using Matillion's REST API.

WITH THIS METHOD, YOU CREATE MULTIPLE MATILLION INSTANCES. FOR EXAMPLE:

ONE INSTANCE FOR DEVELOPMENT
ONE INSTANCE FOR QA
ONE INSTANCE FOR PRODUCTION

EVERY MATILLION INSTANCE CONTAINS A PARALLEL, INDEPENDENT SET OF METADATA.

YOU PROMOTE CODE BETWEEN INSTANCES ONCE IT HAS BEEN DEVELOPED AND TESTED.

In order to describe the details, first a few definitions. What's a Matillion "instance"? What do we actually mean by "Matillion code"? And what's a "Project" as opposed to an "Environment"?

Definition of a Matillion ETL Instance

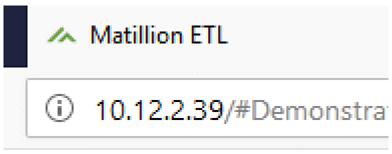
When you purchase Matillion ETL, you actually have to do two things:

1. Take the option to launch the product, by accepting the EULA.
2. Launch the product.

Deployment Options in Matillion ETL

Matillion ETL is a Virtual Machine (VM) which, when launched, runs in your own cloud environment (it's NOT a SaaS solution, which would be hosted on your behalf by some third party).

You can stop and start your VM whenever you want. You connect to the Instance using a web browser, by navigating to the address supplied by the cloud infrastructure provider (for example AWS, Google or Azure).



Every running Matillion ETL VM is one **Instance**.

By launching the product repeatedly you can choose to have more than one Matillion Instance running at the same time, and that's the code deployment scenario being described in this article.

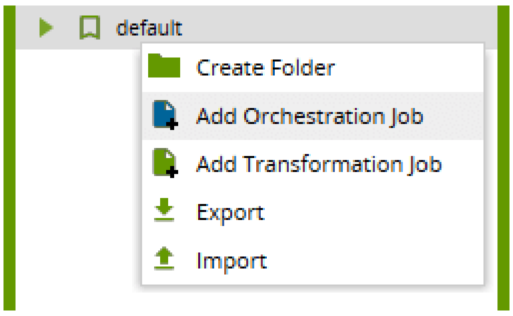
Orchestration and Transformation Jobs

Every Matillion Instance is an ELT tool: it does two main things on your behalf:

1. Data ingestion – in other words loading data from external sources into the database
2. Data transformation – getting your data ready for visualization or analysis, for example, integration, reshaping, applying business rules, aggregation, and deriving calculated fields.

These two things are implemented by Orchestration Jobs and Transformation Jobs respectively.

Deployment Options in Matillion ETL

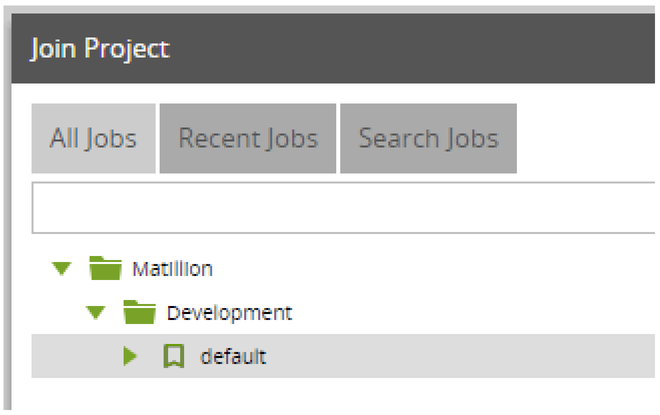


Orchestration jobs fulfill an additional command-and-control function: they define the overall sequence of events and can be scheduled. Orchestration jobs can call Transformation jobs as part of their work.

“Matillion Code” means the definition of the Orchestration and Transformation Jobs that you are using. Matillion Jobs always exist inside a Project.

Matillion Projects

A Project is simply a collection of metadata. When you’re working in a Matillion ETL Instance you are always within the context of a Project. When you log in, it’s the first dialog that pops up after the login credentials.



You’ll need to select a Group, a Project and a Version. In the above screenshot, these are:

- Group: **Matillion**
- Project: **Development**
- Version: **default**

Deployment Options in Matillion ETL

A Matillion Instance can contain multiple Groups, and every Group can, in turn, contain multiple projects. It's often simplest to stick with just one Group, and a good choice for the Group name is your company name.

One Project can contain multiple Versions. You can create your own version snapshot at any time, but there's always a current one named "default".

Among other things, Projects contain

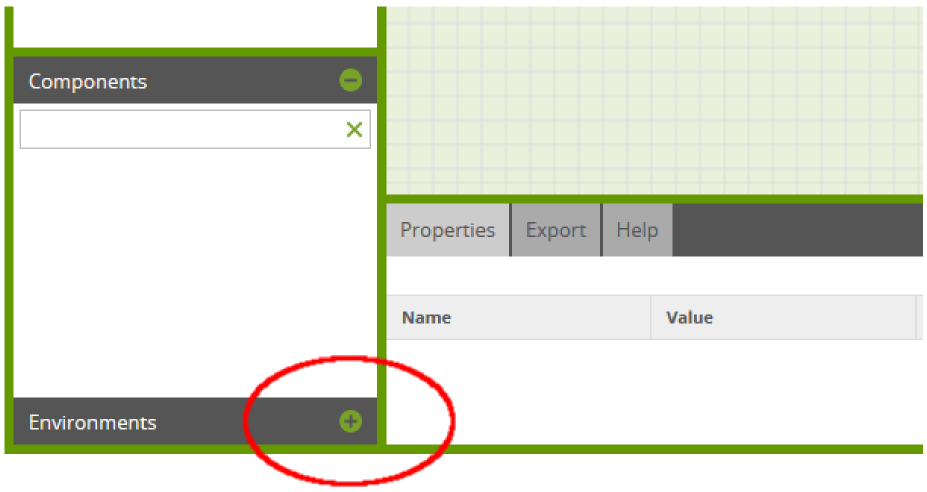
- Orchestration and Transformation Job definitions
- Folder structure
- One or more Environments

It's the "Environments" which define where the Matillion Jobs can be executed at runtime.

What's a Matillion Environment?

A Matillion Environment defines a target data warehouse.

To manage your Environments you'll need to expand the panel at the bottom left of the screen, which is minimized by default.

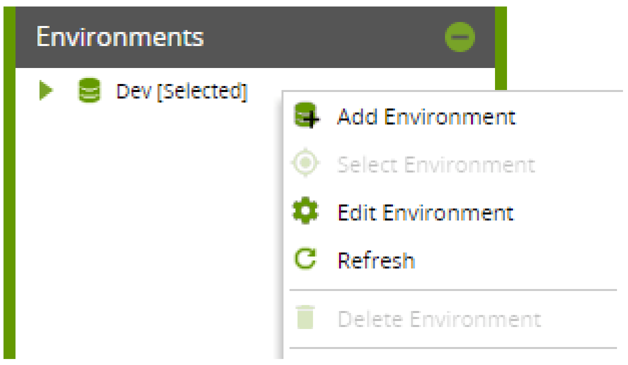


Deployment Options in Matillion ETL

When you first launched and connected to a new Matillion ETL Instance, you went through an initial configuration screen. This asked for details of the target data warehouse plus a couple of other configuration items. For this reason, you'll always have at least one Environment.

When deploying code through multiple Matillion ETL instances, it's highly recommended that you create one Environment per Project, pointing to the target data warehouse for which the instance was created (development, test or production).

You can manage environments through the right-click context menu. The Edit Environment option will take you back to that initial configuration screen, where you can change the settings if necessary.



You can add, edit and remove Environments using these options.

Note that:

1. Every Matillion ETL Instance has an overall cap on the total number of environments that may exist. The cap depends upon the instance size. If you have used multiple Projects then each Project will have at least one Environment, and they all count towards the cap (even if they are actually pointing at exactly the same target).
2. You are not permitted to delete the last Environment within a Project.

Deployment Options in Matillion ETL

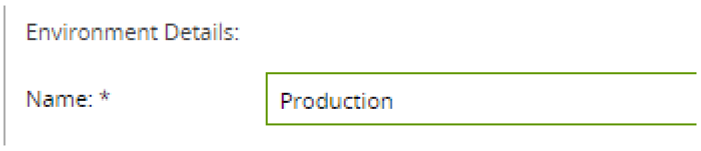
Solution Overview

With this code deployment option:

- You launch multiple Matillion ETL Instances (one for development, one for test, one for production).
- Every Instance contains the same, parallel Groups and Projects (e.g. Company Name and Project Name).
- Every Project owns one corresponding Environment (development, test, production).
- You copy the code from one Instance into another once it's ready to be promoted.

If you started out with a "Development" Matillion ETL instance, you might now launch a new one for "Production". It will take a few minutes to launch and log into the web interface. Once connected you should set it up with the same Group name and the same Project name that exist on the Development instance.

You're also creating a new Environment at this point, and as explained in the previous section it's a good idea to name it "Production" this time:



Environment Details:

Name: * Production

Once completed, you'll have two running Matillion instances, both with the same Project Group (ideally named after your company), and the same-named Project:

- A "Development" instance, in which the project owns an Environment named "Dev", pointing to the Development target data warehouse.

- A “Production” instance, in which the project owns an Environment named “Production”, pointing to the Production target data warehouse.

Developers would log onto the Development instance to do their work. When finished, they would export their work ready for a DBA or DevOps person to log onto the Production instance to import and schedule the jobs.

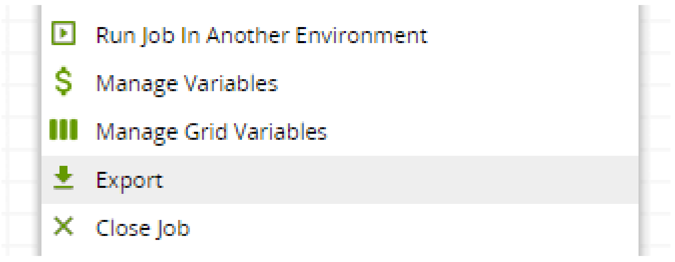
You may extrapolate this scenario to fit any combination of environments that you need. Some examples are:

- Dev → Production
- Dev → System Test → Production
- Dev → System Test → UAT → Production

Exporting and Importing Jobs

Matillion ETL’s mechanism for copying job metadata from one Instance (or Project) into another is via the Export/Import feature.

While editing a Job, you can find the Export option from the right-click context menu:



You can also get to the same dialog by following the Project / Export menu. The dialog allows you to select one or more Jobs to be saved and downloaded as a JSON file. You can put this file into external version control.

Deployment Options in Matillion ETL

From the Project / Import menu, you can import a Matillion-generated JSON file, and choose which jobs to import into the target Project.

BY DEFAULT, THE JOBS WILL BE ADDED INTO THE SAME FOLDER STRUCTURE AS THEY WERE AT THE SOURCE.

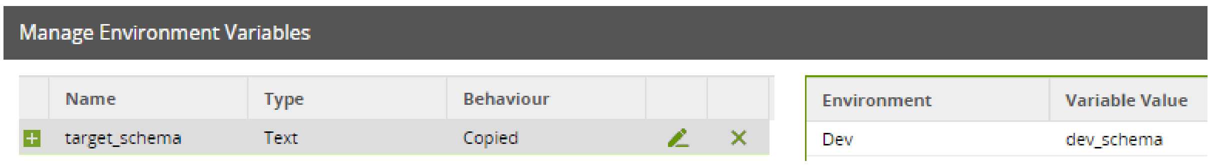
YOU'LL FIND IT EASIEST TO MAINTAIN EXACTLY THE SAME FOLDER STRUCTURE IN THE DEVELOPMENT INSTANCE AS IN THE PRODUCTION INSTANCE.

Matillion also has various REST API endpoints for exporting and importing Jobs. These are used for automation, although be aware that API-generated metadata formats are not compatible with the UI-based export and import.

A note on Environment Variables

Environment variables have a very useful feature in that you can set a different default per Environment.

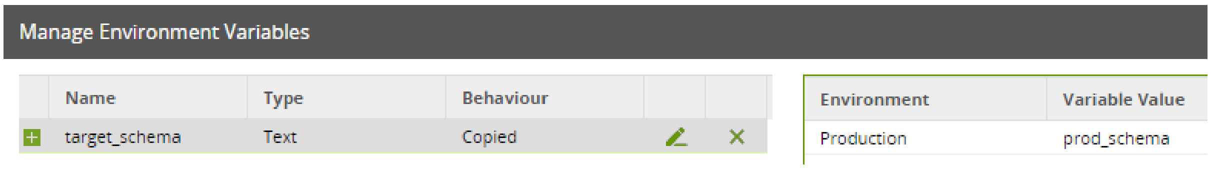
While logged into the Development instance, your Project might have a variable named ***target_schema***. Its default value (for the Dev environment) is set to ***dev_schema***.


Manage Environment Variables

Name	Type	Behaviour
target_schema	Text	Copied

Environment	Variable Value
Dev	dev_schema

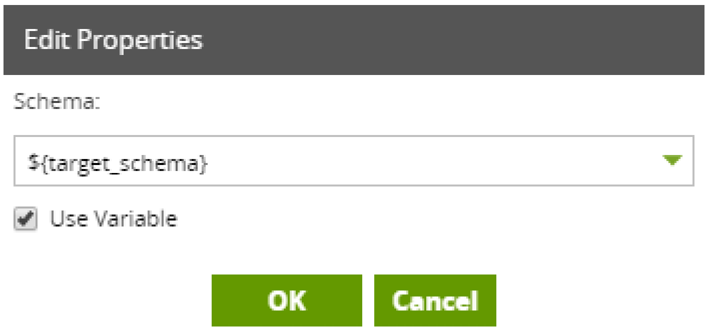
Similarly, while logged into the Production instance, the same variable can exist, but with a default value (for the Production environment this time) set to ***prod_schema***.


Manage Environment Variables

	Name	Type	Behaviour		
+	target_schema	Text	Copied	✎	✕

Environment	Variable Value
Production	prod_schema

So, instead of hardcoding the schema name of a Table Input component, for example, you could set it to ${target_schema}


Edit Properties

Schema:

${target_schema}

☑ Use Variable

Then, in Development the Table Input would automatically read from the ***dev_schema***, and once deployed into the Production instance it would automatically start to read from the ***prod_schema*** with no code change required.

Incidentally, this is why when you export an Environment Variable, the default value is not among the properties that get saved into the JSON export file.

Summary

When you use multiple Matillion Instances to manage code deployment, you effectively set up multiple parallel copies of the codebase, each copy pointing to a different target data warehouse (dev, test or production).

You control exactly when and what code gets promoted into the different environments. Also, you can take advantage of the Environment Variable feature which allows you to automatically customize job behavior according to where it is running.

Normal backup mechanisms still work and are still recommended.

Metadata items such as user logins, OAuth credentials, and the Password Manager are instance-wide and automatically apply to every Project. You can take advantage of this for example to:

- Restrict access to the Production instance to only DevOps personnel
- Automatically have Development data taken from Sandbox data sources, and Production data coming from live sources

This method of code deployment is the most sophisticated of the three, allowing the greatest degree of flexibility and governance options. However, you will need to budget for having multiple Matillion ETL instances running in parallel.

Reference : matillion.com/blog/deployment-options-in-matillion-etl-using-multiple-instances