



Developer Guide

Version: 9.7

Doc Build Date: 12/31/2022

Disclaimers

Except as otherwise provided in an express written agreement, Alteryx Inc. (“ Alteryx”) makes no representations or warranties with respect to the software and documentation contained herein and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Furthermore, Alteryx reserves the right to revise the software and documentation from time to time without the obligation of Alteryx to notify any person of such revisions or changes.

Copyright and Trademark Notices

© 2013 - 2022 Alteryx Inc. All Rights Reserved.

Alteryx Inc.

17200 Laguna Canyon Road

Irvine, CA 92618

Phone: +1 888 836 4274

Alteryx and Trifacta are registered trademarks of Alteryx Inc.

Credits

For third-party license information, please select **About Trifacta** from the Resources menu.

See also [Third-Party License Information](#).

1. Developer	4
1.1 User-Defined Functions	5
1.1.1 Java UDFs	8
1.2 Create Custom Data Types Using RegEx	18
1.3 API Reference	23
1.3.1 Manage API Access Tokens	25
1.3.2 API Endpoints for Designer Cloud Powered by Trifacta Enterprise	28
1.3.3 API Tasks	36
1.3.3.1 API Task - Develop a Flow	37
1.3.3.2 API Task - Deploy a Flow	45
1.3.3.3 API Task - Run Job	56
1.3.3.4 API Task - Run Job on Dataset with Parameters	70
1.3.3.5 API Task - Run Plan	81
1.3.3.6 API Task - Define Deployment Import Mappings	86
1.3.3.7 API Task - Run Deployment	100
1.3.3.8 API Task - Publish Results	107
1.3.3.9 API Task - Swap Datasets	112
1.3.3.10 API Task - Manage Outputs	117
1.3.3.11 API Task - Manage AWS Configurations	132
1.3.3.12 API Task - Wrangle Output to Python	141
1.3.4 API Documentation Versions	144
1.4 Python SDK	145

Developer

Contents:

- *REST APIs*
 - *User-Defined Functions*
 - *Topics*
-

The Designer Cloud powered by Trifacta® platform provides multiple toolsets to empower developers to leverage the platform's capabilities within their data pipelines.

Use of the features documented in this section requires programming skills.

REST APIs

The Designer Cloud powered by Trifacta platform provides a robust set of REST APIs to automate your data transformation processes. The list of REST APIs continues to expand with each release. See *API Reference*.

User-Defined Functions



Feature Availability: This feature may not be available in all product editions. For more information on available features, see *Compare Editions*.

You can expand the capabilities of Wrangle (a domain-specific language for data transformation) to support functions that you need for your environment.

See *User-Defined Functions*.

Topics

User-Defined Functions

Contents:

- *UDF Service*
 - *Supported UDF Language Frameworks*
 - *Running a UDF within the Platform*
-

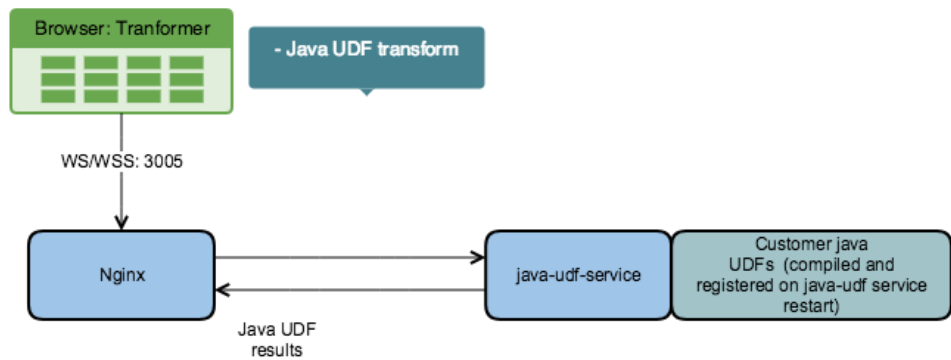
The Designer Cloud powered by Trifacta® platform enables the creation of user-defined functions (UDFs) for use in your Trifacta deployment. A **user-defined function** is a way to specify a custom process or transformation for use in your specific Trifacta solution, using familiar development languages and third-party libraries. Through UDFs, you can apply enterprise- or industry-specific expertise consistently into your data transformations. A user-defined function is a custom function that is created in one of the supported language frameworks. Each user-defined function has a defined set of inputs and generates a single output.

UDF Service

The following diagram provides a high-level overview of the UDF service which provides integration of user-defined functions into recipe execution.

- Diagram 1: The figure illustrates execution of a UDF in interactive mode, where a user interacts with the Transformer grid.
- Diagram 2: This feature illustrates how UDFs interact with the cluster at job execution time.

Java UDFs in Transformer Grid



Java UDFs on Hadoop

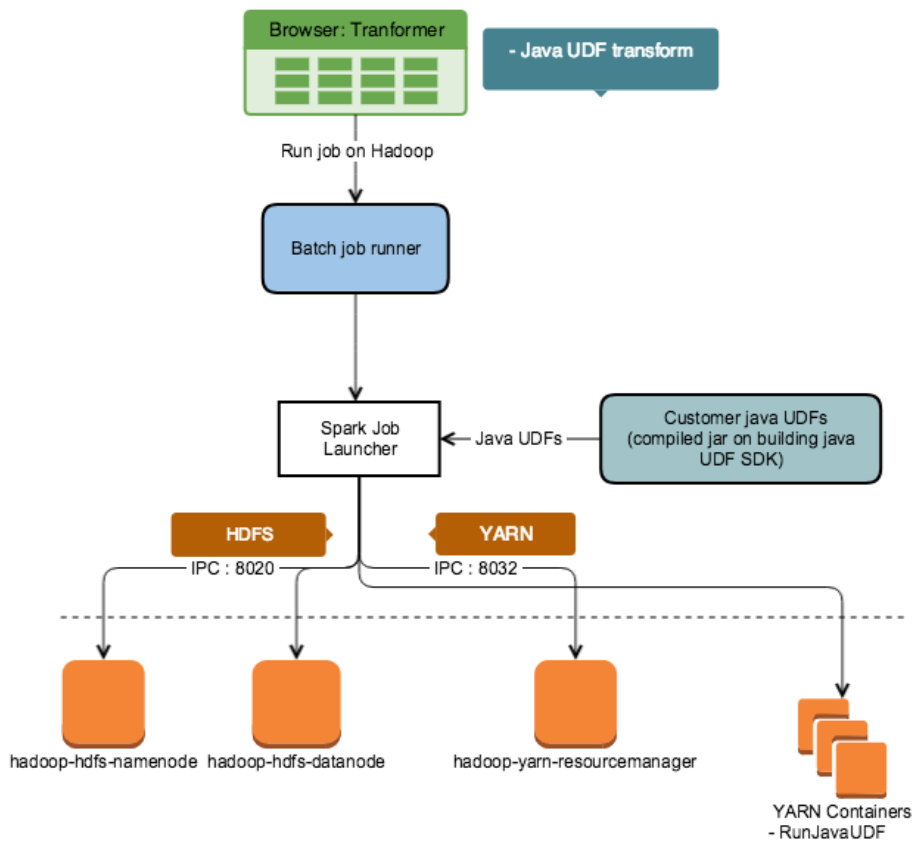


Figure: User-Defined Service

Supported UDF Language Frameworks

Please use the following links to enable the creation of user-defined functions in the listed language.

- *Java UDFs*

Running a UDF within the Platform

After you have created and tested your UDF, you can execute it by entering `udf` in the Search panel and populating the rest of the step in the Transform Builder.

In this example, the `AdderUDF` function is added:

Transformation Name	Invoke external function
Parameter: Column	cola
Parameter: Arguments	100
Parameter: New column name	myAdderUDFColumn

Notes:

- After entering `udf`, your UDF should appear in a drop-down list. If not, please verify that it has been properly created, compiled, and registered and that the `udf-service` has been restarted.
- The Column parameter is a comma-separated list of the source data to be used as inputs to the `exec` method.
- The Argument parameter is a string of comma-separated values used as inputs to the `init` method.
- Optionally, The New column name parameter can be used to provide a specific name to the generated column. If it is not used, a column name is generated.

NOTE: When a recipe containing a user-defined function is applied to text data, any non-printing (control) characters cause records to be truncated by the Spark running environment during job execution. In these cases, please execute the job on the Photon running environment.

For more information, see *Invoke External Function*.

NOTE: Running user-defined functions for an external service, such as Hive, is not supported from within a recipe step. As a workaround, you may be able to execute recipes containing such external UDFs on the Photon running environment. Performance issues should be expected on larger datasets.

See *Transformer Page*.

Java UDFs

Contents:

- *Prerequisites*
 - *Overview*
 - *Known Limitations*
 - *Enable Service*
 - *Deployment*
 - *Creating a UDF*
 - *UDF Requirements*
 - *Example - Concatenate strings*
 - *Example - Add by constant*
 - *Error Handling*
 - *Testing the UDF*
 - *Compiling the UDF*
 - *JDK version mismatches*
 - *Registering the UDF*
 - *Running Your UDF*
 - *Troubleshooting*
 - *"Websocket Receive()" error in Transformer page UI*
 - *Photon crashes during execution of UDF*
 - *Databricks cluster has either stale or unknown libraries error*
 - *Build timestamp is missing in <filename>.JAR*
-

This section describes how to create and deploy Java-based user-defined functions (UDFs) into your Trifacta® deployment.

Creation of UDFs requires development experience and access to an integrated development environment (IDE).

Prerequisites

1. Access to the Trifacta deployment
2. IDE
3. The Java UDF is stored in the Trifacta deployment in the following location: `libs/custom-udfs-sdk/build/distributions/java-custom-udf-sdk.zip`

NOTE: `custom-udf-sdk.zip` is required for compilation and executing of the unit test. Any JAR files present in `custom-udf-sdk.zip`, such as `trifacta-base-udf.jar`, do not need to be packaged in the custom UDF JAR.

NOTE: If you are installing custom UDFs and the Trifacta node does not have an Internet connection, you should download the Java UDF SDK in an Internet-accessible location, build your customer UDF JAR there, and then upload the JAR to the Trifacta node.

Overview

Each UDF requires at least one input and produces a single output value (map only).

Inputs and outputs must be one of the following types:

- Bool
- String
- Long
- Double

NOTE: If your UDF does not require an input value, you must create a dummy input as part of your UDF definition.

Known Limitations

- In the Designer Cloud application , previews are not available for user-defined functions.
- Retaining state information across the exec method is unstable. More information is provided below.

NOTE: When a recipe containing a user-defined function is applied to text data, any null characters cause records to be truncated by the running environment during Trifacta Photon job execution. In these cases, please execute the job in the Spark running environment.

Enable Service

You must enable the Java UDF service in the Designer Cloud powered by Trifacta platform .

Steps:

1. You can apply this change through the *Admin Settings Page* (recommended) or `trifacta-conf.json`. For more information, see *Platform Configuration Methods*.
2. Enable the correct flag:

```
"feature.enableUDFTransform.enabled": true,
```

3. Save your changes.

Deployment

Steps:

1. Unzip `java-custom-udf-sdk.zip`.
2. Within the unzipped directory, execute the install command. The following is specific to the Eclipse IDE:

```
gradlew eclipse
```

3. Import the project into your IDE.

Creating a UDF

UDF Requirements

All UDFs must implement the `TrifactaUDF` interface. This interface adds the four methods that each UDF must override: `init`, `exec`, `inputSchema`, and `finish`.

1. **init method:** Used for setting private variables in the UDF. This method may be a no-op function if no variables must be set. See the *Example - Concatenate strings* below.

Tip: In this method, perform your data validation on the input parameters, including count, data type, and other constraints.

NOTE: The `init` method must be specified but can be empty, if there are no input parameters.

2. **exec method:** Contains functionality of the UDF. The output of the `exec` method must be one of the supported types. It is also must match the generic as described. In the following example, `TrifactaUDF<String>` implements a `String`. This method is run on each record.

Tip: In this method, you should check the number of input columns.

Keep state that varies across calls to the `exec` method can lead to unexpected behavior. One-time initialization, such as initializing the regex compiler, is safe, but do not allow state information to mutate across calls to `exec`. This is a known issue.

3. **inputSchema method:** The `inputSchema` method describes the schema of the list on which the `exec` method is acting. The classes in the schema must be supported. Essentially, you should support the I/O types described earlier.
4. **finish method:** The `finish` method is run at the end of UDF. Typically, it is a no-op.

NOTE: If you are executing your UDF on the Spark running environment, the `finish` method cannot be invoked at this point. Instead, it is invoked as part of the shutdown of the Java VM. This later execution may result in the `finish` method failing to be invoked in situations like a JVM crash.

Example - Concatenate strings

The following code example concatenates two input strings in the `List<Object>`. This UDF can be easily modified to concatenate more strings by modifying the `inputSchema` function.

Example UDF: ConcatUDF

```
package com.trifacta.trifactaudfs;
import java.io.IOException;
import java.util.List;

/**
 * Example UDF that concatenates two columns
 */
public class ConcatUDF implements TrifactaUDF<String> {
    @Override
    public String exec(List<Object> inputs) throws IOException {
        if (inputs == null) {
            return null;
        }
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < inputSchema().length; i += 1) {
            if (inputs.get(i) == null) {
                return null;
            }
            sb.append(inputs.get(i));
        }
        return sb.toString();
    }
    @SuppressWarnings("rawtypes")
    public Class[] inputSchema() {
        return new Class[]{String.class, String.class};
    }
    @Override
    public void finish() throws IOException {
    }
    @Override
    public void init(List<Object> initArgs) {
    }
}
```

Notes:

- The first line indicates that the function is part of the `com.trifacta.trifactaudfs` package.
- The defined UDF class implements the `TrifactaUDF` class, which is the base interface for UDFs.
 - It is parameterized with the return type of the UDF (a Java `String` in this case).
 - The input into the function is a list with input parameters in the order they are passed to the function within the Designer Cloud powered by Trifacta platform . See *Running Your UDF* below.
- The UDF checks the input data for null values, and if any nulls are detected, returns a null.
- The `inputSchema` describes the input list passed into the `exec` method.
 - An error is thrown if the type of the data that is passed into the UDF does not match the schema.
 - The UDF must handle improper data. See *Error Handling* below.

Example - Add by constant

In this example, the input value is added by a constant, which is defined in the `init` method.

- The `init` method consumes a list of objects, each of which can be used to set a variable in the UDF. The input into the `init` function is a list with parameters in the order they are passed to the function within the Designer Cloud powered by Trifacta platform . See *Running Your UDF* below.

Example UDF: AdderUDF

```
package com.trifacta.trifactaudfs;
import java.io.IOException;
import java.util.List;

/**
 * Example UDF. Adds a constant amount to an Integer column.
 */
public class AdderUDF implements TrifactaUDF<Long> {
    private Long _addAmount;
    @Override
    public void init(List<Object> initArgs) {
        if (initArgs.size() != 1) {
            System.out.println("AdderUDF takes in exactly one init argument");
        }
        Long addAmount = (Long) initArgs.get(0);
        _addAmount = addAmount;
    }
    @Override
    public Long exec(List<Object> input) {
        if (input == null) {
            return null;
        }
        if (input.size() != 1) {
            return null;
        }
        return (Long) input.get(0) + _addAmount;
    }
    @SuppressWarnings("rawtypes")
    public Class[] inputSchema() {
        return new Class[] {Long.class};
    }
    @Override
    public void finish() throws IOException {
    }
}
```

Error Handling

The UDF must handle any error that should occur when processing the function. Two ways of dealing with errors:

1. For null data generated in the exec method, a null value can be returned. It appears in the final generated column.
2. Any errors that cause the UDF to stop in the init or exec methods cause an IOException to be thrown. This error signals the platform that an issue occurred with the UDF.

Tip: You can add to the Trifacta logs through Logger. Annotate your exceptions at the appropriate logging level.

Testing the UDF

JUnit can be used to test the UDF. Below are examples of testing the two example UDFs.

Example - JUnit test for Concatenate strings:

ConcatUDF Test

```
@Test
public void concatUDFTest() throws IOException {
    ConcatUDF concat = new ConcatUDF();
    ArrayList<Object> input = new ArrayList<Object>();
    input.add("hello");
    input.add("world");
    String result = concat.exec(input);
    String expected = "helloworld";
    assertEquals(expected, result);
}
```

Example - JUnit test for Add by constant:

AdderUDF Test

```
@Test
public void adderUDFTest() {
    AdderUDF add = new AdderUDF();
    ArrayList<Object> initArgs = new ArrayList<Object>(1);
    initArgs.add(1L);
    add.init(initArgs);
    ArrayList<Object> inputs1 = new ArrayList<Object>();
    inputs1.add(1L);
    long result = add.exec(inputs1);
    long expected = 2L;
    assertEquals(expected, result);

    ArrayList<Object> inputs2 = new ArrayList<Object>();
    inputs2.add(9000L);
    result = add.exec(inputs2);
    expected = 9001L;
    assertEquals(expected, result);
}
```

Compiling the UDF

After writing the UDF, it must be compiled and included in a JAR before registering it with the platform. To compile and package the function, run the following command from the root directory:

```
gradlew build
```

The UDF code is assembled, and unit tests are executed. If all is well, the following JAR file is created in `build/libs`.

NOTE: Custom UDFs should be compiled to one or more JAR files. Avoid using the example JAR filename, which can be overwritten on upgrade.

JDK version mismatches

To avoid an `Unsupported major.minor version` error during execution, the JDK version used to compile the UDF JAR file should be less than or equal to the JDK version on the Hadoop cluster.

If this is not possible, then set the value of the `Compatibility` properties in the local `build.gradle` file to the JDK version on the Hadoop cluster prior to building the JAR file.

Example:

If the Hadoop cluster is on JDK 1.8, then add the following to the `build.gradle` file:

```
targetCompatibility = '1.8'
sourceCompatibility = '1.8'
```

Registering the UDF

After a function is compiled it must be registered with the platform.:

1. Enable user-defined functions (if not done so already)
2. Path to the JAR file that was generated in the previous steps.
3. The `udfPackages` value should contain the package name where the UDFs can be found.

Example configuration:

To apply this configuration change, login as an administrator to the Trifacta node. Then, edit `trifacta-conf.json`. For more information, see *Platform Configuration Methods*.

Example Config

```
...
"feature": {
  "enableUDFTransform": {
    "enabled": true
  }
},
"udf-service": {
  "classpath": "%(topOfTree)s/services/udf-service/build/libs/udf-service.jar:%(topOfTree)s/services/udf-service/build/dependencies/*",
  "additionalJars": [
    "/vagrant/libs/custom-udfs-sdk/build/libs/custom-udfs-example.jar"
  ],
  "udfPackages": [
    "com.trifacta.trifactaudfs"
  ]
},
...
```

Notes:

- Set `enableUDFTransform.enabled` to `true`, which enables UDFs in general.
- Under `udf-service`:
 - specify the full path to the JAR under `additionalJars`
 - append the paths of any extra JAR dependencies that your UDFs require under `classpath`

NOTE: Do not include any extra JAR dependencies in the `udf-service/build/dependencies` directory, as this directory may be purged at build time.

- specify the fully qualified package names under `udfPackages`
 - This list contains all fully qualified names of your UDFs.

- For example, if your UDF is `com.company.ourudfs.MyUDF`, then the package name is the following: `com.company.ourudfs`

Steps:

After modifying the config, the `udf-service` needs to be restarted.

- a. If you created a new UDF, restart the Designer Cloud application :

```
service trifacta restart
```

- b. If you have modified an existing UDF, restart the UDF service:

NOTE: For an existing UDF, you must rebuild the JAR first. Otherwise, the changes are not recognized during service re-initialization.

```
service java-udf-service restart
```

2. As part of the restart, any newly added Java UDFs are registered with the application.

Running Your UDF

For more information on executing your UDF in the Transformer page, see *User-Defined Functions*.

For examples, see *Invoke External Function*.

Troubleshooting

"Websocket Receive()" error in Transformer page UI

If you execute a Java UDF, you may see an error similar to the following in the Transformer page:

```
Please reload page (query execution failed).pp::WebSocket::Receive() error: Unspecified failure.
```

When you check the `udf.log` file on the server, the following may be present:

```
UDFWebsocket closed with status: CloseStatus[code=1009, reason=The decoded text message was too big for the output buffer and the endpoint does not support partial messages]
```

Solution

The above issue is likely to be caused by the Trifacta Photon running environment sending too much data through the buffer of the UDF's Websocket service. By default, this buffer size is set to 1048576 bytes (1 MB).

The Trifacta Photon running environment processes data through the Websocket service in 1024 (1 K) rows at a time for the input and output columns of the UDF. If the data in the input columns to the UDF or output columns from the UDF exceeds 1 KB (1024 characters) in total size for each row, the default size of the buffer is too small, since the Trifacta Photon running environment processed 1K records at a time (1 K characters * 1 K rows > 1048576). The query then fails.

When setting a new buffer size:

- Assume that 1024 rows are processed from the buffer each time.
- Identify the input columns and output columns for the UDF that is failing.

- Identify the dataset that has the widest columns for both inputs and outputs here.

Tip: You can use the `LEN` function to do string-based computations of column width. See *LEN Function*.

- Perform the following estimate on the widest set of input and output columns that you are processing:
 - Estimate the total expected number of characters for the input columns of the UDF.
 - Add a 20% buffer to the above estimate.
 - Repeat the above estimate for the widest output columns for the UDF.
 - Set your buffer size to the larger of the two estimates (input columns' width or output columns' width).
- Example: A UDF takes two inputs and produces one output:
 - If each input column is 256 characters, then the size of 1K rows of input would be $256 \text{ bytes} * 2 \text{ (input cols)} * 1024 \text{ rows} = 0.5 \text{ MB}$.
 - If the output of the UDF per row is estimated to be 1024 characters, then the output estimate would be $1024 \text{ bytes} * 1024 \text{ rows} = 1 \text{ MB}$.
 - So, set the buffer size to be 1 MB + 20% buffer over the larger estimate between input and output. In this example, the buffer size should be 1.2 MB or 1258291 Bytes.

Steps:

1. You can apply this change through the *Admin Settings Page* (recommended) or `trifacta-conf.json`. For more information, see *Platform Configuration Methods*.
2. Change the following setting:

```
"udf-service.outputBufferSize": 1048576,
```

3. Save your changes and restart the platform.

Photon crashes during execution of UDF

During the execution of a UDF, the Photon client can crash. Possible errors include:

```
Error in changeCurrentEdit Error: Transformation engine has crashed. Please reload your browser (exit code: null; message ID: 161)
```

Solution:

This crash can be caused by a number of issues. You can try the following:

1. You can apply this change through the *Admin Settings Page* (recommended) or `trifacta-conf.json`. For more information, see *Platform Configuration Methods*.
2. Bump the value for `udf-service.udfCommunicationTimeout` setting. Raise this value a bit at a time to see if that allows the UDF to execute.

NOTE: Avoid setting this value to high, which can cause the Java heap size to be exceeded and another Photon crash. Maximum value is 2147483646.

3. Save your changes and restart the platform.

Databricks cluster has either stale or unknown libraries error

When running a job on a Databricks cluster, you may receive an error message that the cluster has either stale or unknown libraries.

NOTE: This issue occurs only on a Databricks-based cluster during job execution.

This issue is caused by a change that was made to the custom UDFs code for Databricks. Prior to Release 7.10, custom UDF jobs were checked for use in the Designer Cloud powered by Trifacta platform based on when the JAR file was created. However, this timestamp presented inconsistencies in the following cases:

- In high availability environments, the created-at time for the file might be different between nodes on the cluster.
- If the JAR file was passed between different services, each service might apply a different time as the file was written to new locations.

Solution:

The platform now uses a built-at timestamp instead. This timestamp is assigned in the manifest file when the JAR is assembled for the first time on a developer's local desktop. That timestamp remains consistent when the JAR is delivered to the cluster or to different nodes or services.

NOTE: If you are receiving the above error, you must clean the UDF JAR files off of the cluster and rebuild your custom UDFs for use in the platform. Please see earlier instructions for how to build.

Build timestamp is missing in <filename>.JAR

After you have executed a job, you may receive an error similar to the following in the Job Summary page:

```
Build Timestamp is missing in <filename>.jar. Please rebuild your jars and try again.
```

Solution:

In this case, the referenced JAR file does not have a build timestamp in it. You must rebuild and redeploy the custom UDF JAR file. See the previous Troubleshooting section for details.

Create Custom Data Types Using RegEx

Contents:

- *Custom Types Location*
 - *Examples*
 - *Example - Days of the week*
 - *Example - Sizes*
 - *Reference*
 - *Parameters*
 - *Defining probabilities for Your Custom Data Type*
 - *Add custom types to manifest*
 - *Enable custom types*
 - *Register your custom types*
 - *Restart platform*
-

As needed, you can deploy custom data types into the Designer Cloud powered by Trifacta® platform , in which type validation is performed against regular expressions that you specify. This method is most useful for validating against patterns, as opposed to specific values.

After a custom type has been added, it cannot be removed or disabled. Please verify your regular expression before saving the type.

Custom Types Location

On the server hosting the Designer Cloud powered by Trifacta platform , type definitions are stored in the following directory:

```
/opt/trifacta/node_modules/jsdata/type-packs/trifacta
```

This directory is referenced as `$CUSTOM_TYPE_DIR` in the steps below.

Before you begin creating custom data types, you should backup the `type-packs/trifacta` directory to a location outside of your Trifacta deployment.

NOTE: The `trifacta-extras` directory in the `type-packs` directory contains experimental custom data types. These data types are not officially supported. Please use with caution.

Examples

Example - Days of the week

Each custom data type is created and stored in a separate file. The following example file contains a regular expression method for validating data against the set of days of the week:

```
{
  "name": "DayOfWeek",
  "prettyName": "Day of Week",
  "category": "Date/Time",
  "defaultProbability": 1E-15,
  "testCase": {
    "stripWhitespace": true,
    "regexes": [
      "^(monday|tuesday|wednesday|thursday|friday|saturday|sunday)$",
      "^(Monday|Tuesday|Wednesday|Thursday|Friday|Saturday|Sunday)$",
      "^(mon|tue|wed|thu|fri|sat|sun)$",
      "^(Mon|Tue|Wed|Thu|Fri|Sat|Sun)$"
    ],
    "probability": 0.001
  }
}
```

Example - Sizes

Suppose your data contains size information from Extra Small (XS) to Extra Extra Large (XXL). You can create a regular expression to test for these sizes within a column of values. These sizes could be the following:

Extra Small
Small
Medium
Large
Extra Large
Extra Extra Large
XS
S
M
L
XL
XXL
Extra-Small
Extra-Large
Extra-Extra-Large

You may have noticed that there are multiple ways of expressing sizes and multiple types of case (upper case and title case). To standardize, all values should be converted to lower case to simplify evaluation. The definition may look like the following:

```
{
  "name": "size",
  "prettyName": "Size",
  "category": "String",
  "defaultProbability": 1E-15,
  "testCase": {
    "stripWhitespace": true,
    "regexes": [
      "^(xs|s|m|l|xl|xxl)$",
      "^(extra-small|small|medium|large|extra-large|extra-extra-large)$"
    ],
  },
}
```

```

    "probability": 0.001
  }
}

```

Reference

Parameters

Parameter Name	Description
name	<p>Internal identifier for the custom type. Must be unique across all standard types and custom types.</p> <div> NOTE: You should verify that your data type's name value does not conflict with other custom data type names. </div>
prettyName	Display name for the custom type.
category	The category to assign to the type. The current categories are displayed within the data type drop-down for each column.
defaultProbability	Assign a default probability for the custom type. See below.
testCase	This block contains the regular expression specification to be applied to the column values.
stripWhiteSpace	When set to true, whitespace is removed from any value prior for purposes of validation. The original value is untouched.
regexes	<p>This array contains a set of regular expressions that are used to validate the column values. For a regex type, the column value must match with at least one value among the set of expressions.</p> <div> NOTE: Matching is case-insensitive. </div> <div> NOTE: All match types must be double-escaped in the regex expression. For example, to replicate the <code>\d</code> pattern, you must enter: <code>\\d</code>. </div> <p>Designer Cloud Powered by Trifacta Enterprise Edition implements a version of regular expressions based off of <i>RE2</i> and <i>PCRE</i> regular expressions.</p>
probability	(optional) Assign an incremental change to the probability when a match is found between a value and one of the regular expressions. See <i>Defining probabilities</i> below.

Tip: In the `types` sub-directory, you can review the regex-based types that are provided with the Designer Cloud powered by Trifacta platform. While you should not edit these files directly, they may provide some guidance and some regex tips on how to configure your own custom data types.

Defining probabilities for Your Custom Data Type

For your custom type, the probability values are used to determine the likelihood that matching values indicate that the entire column is of the custom data type.

- The `defaultProbability` value specifies the baseline probability that a match between a value and one of the regular expressions indicates that the column is the specified type. On a logarithmic scale, values are typically 1E-15 to 1E-20.

- When a value is matched to one of the regular expressions, the `probability` value is used to increment the baseline probability that the next matching value is of the specified type. This value should also be expressed on a logarithmic scale (e.g. 0.001).
- In this manner, a higher number of matching values increases the probability that the type is also a match to the custom type.

Probabilities become important primarily if you are creating a custom type that is a subset of an existing type. For example, the Email Address custom type is a subset of String type. So, matches for the patterns expressed in the Email Address definition should register a higher `probability` value than the same incremental for the String type definition.

Tip: For custom types that are subsets of other, non-String types, you should lower the `defaultProbability` of the baseline type by a factor of 10 (e.g. 1E-15 to 1E-16) and raise the same probability in the custom type by a factor of 10 (e.g. 1E-14). In this manner, you can give higher probability of matching to these subset types.

Add custom types to manifest

To the `$CUSTOM_TYPE_DIR/manifest.json` file, you must add the filenames of any custom types that you have created and stored in the `types` directory:

```
{
  "types": ["bodies-of-water.json", "dayofweek.json"],
  "dictionaries": ["oceans", "seas"]
}
```

Enable custom types

Steps:

1. You can apply this change through the *Admin Settings Page* (recommended) or `trifacta-conf.json`. For more information, see *Platform Configuration Methods*.
2. Locate the following property:

```
"feature.enableCustomTypes": true,
```

3. To enable use of your custom data types in the Designer Cloud powered by Trifacta platform, locate and edit `enabledSemanticTypes` property.

NOTE: Add your entries to the items that are already present in `enabledSemanticTypes`. Do not delete and replace entries.

NOTE: Do not use this parameter to attempt to remove specific data types. Removal of the default types is not supported.

```
"webapp.enabledSemanticTypes": [
  "<CustomTypeName1>",
  "<CustomTypeName2>",
  "<CustomTypeNameN>"
]
```

where:

- `<CustomTypeName1>` corresponds to the internal `name` value for your custom data type.

4. Save your changes and restart the platform.

Register your custom types

To add your custom types to the Designer Cloud powered by Trifacta platform , run the following command from the `js-data` directory:

```
node bin/load-types --manifest ${PATH_TO_MANIFEST_FILE}
```

Restart platform

Restart services. See *Start and Stop the Platform*.

Check for the availability of your types in the column drop-down. See *Column Menus*.

API Reference

This section contains reference information on the REST APIs that are made available by Designer Cloud Powered by Trifacta® Enterprise Edition.

Access to API docs locally

NOTE: URLs to API endpoint documentation are case-sensitive.

To access the API documentation for each API endpoint and method that is available to your specific user account, select **Resources menu > API documentation** in the Designer Cloud application .

NOTE: This API documentation portal displays only the API endpoints based on your specific user account and the features enabled in your Trifacta instance. Additional API endpoint documentation may be available at <https://api.trifacta.com>. For more information on the differences between these documentation portals, see *API Documentation Versions*.

Enable Access

Access tokens required

If the API documentation is not available, a workspace administrator must enable the use of API access tokens.

API tokens enable users and processes to access the REST APIs available through the platform.

Tip: Individual users do not need personal API access tokens to use the API documentation. The feature must be enabled.

For more information, see *Workspace Settings Page*.

Enable access through the menu

To enable the Resources menu option and access to the API documentation, the following parameter must be enabled.

Steps:

1. Login to the application as an administrator.
2. You can apply this change through the *Admin Settings Page* (recommended) or `trifacta-conf.json`. For more information, see *Platform Configuration Methods*.
3. Locate the following parameter and set it to `true`:

```
"webapp.apiDoc.enabled": true,
```

4. Save your changes and restart the platform.

API Endpoint Documentation

You can access API reference documentation through the Designer Cloud application . In the left navigation bar, select **Resources menu > API documentation**.

Manage API Access Tokens

Contents:

- *Enable*
 - *Enable individual access*
 - *Generate New Token*
 - *Via API*
 - *Via UI*
 - *Use Token*
 - *List Tokens*
 - *Renew Token*
 - *Delete Token*
-

This section provides some task information for how to use API access tokens as part of your API projects in Designer Cloud Powered by Trifacta® Enterprise Edition. An **access token** is a hashed string that enables authentication when submitted to any endpoint. Access tokens limit exposure of clear-text authentication values and provide an easy method of managing authentication outside of the browser.

Notes:

- An access token is linked to its creator and can be generated by submitting a username/password combination or another valid token from the same user.
 - If a token is created for userA, userB can be provided the token to impersonate userA.
 - You cannot create access tokens for users without their authentication credentials.
 - Changes to passwords do not affect tokens.
- After a token has been created, it cannot be modified or extended.
 - You can create an unlimited number of tokens.
- Access tokens can be used for authentication with any supported version of the APIs.

Enable

This feature must be enabled in your instance of the platform. For more information, see *Enable API Access Tokens*.

Enable individual access

When access tokens are enabled, by default only administrators are permitted to generate tokens. Optionally, workspace administrators can enable individual users in the workspace to generate and use their own API access tokens. For more information, see *Workspace Settings Page*.

Generate New Token

API access tokens must be created.

Via API

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/getApiAccessToken>

Via UI

Tokens can be generated from the web application.

Steps:

1. Login to the Designer Cloud application .
2. From the left nav bar, select **User menu > Preferences > Access Tokens**.
3. Click **Generate New Token**.
4. Specify the number of days for how long the token should live.

Tip: Depending on how your environment is configured, you may be able to enter -1 to create a non-expiring token.

5. Add a user-friendly description if desired.
6. Click **Generate**.

NOTE: Copy the value of the token to the clipboard and store it in a secure location for use with your scripts. For security reasons, the token value itself cannot be retrieved from the application after it has been created.

Tip: If you wish to manage your token via the APIs, you should copy the Token ID value, too. The Token ID can always be retrieved from the Designer Cloud application .

For more information, see *Access Tokens Page*.

Use Token

After a token has been acquired, it must be included in each request to the server, for as long as it is valid.

NOTE: After a token has been created, it cannot be extended or modified.

NOTE: API access tokens are not used by users through the Designer Cloud application .

NOTE: When using the APIs in SSO environments, API access tokens work seamlessly with platform-native versions of SAML and LDAP-AD. They do not work with the reverse proxy SSO methods. For more information, see <https://api.trifacta.com/ee/es.t/index.html#section/Authentication>

After you have acquired the token, you submit it with each API request to the platform.

Example - cURL:

The following example returns a JSON version of the list of available REST API endpoints for your environment:

```
curl http://tri.example.com:3005/v4/open-api-spec -X GET -H "Authorization: Bearer (tokenValue)"
```

- (tokenValue) is the value returned for the token when it was created.

Example - REST client:

If you are submitting your API calls through a REST client, the Authorization header must be specified as follows:

```
Authorization: Bearer (tokenValue)
```

List Tokens

NOTE: For security reasons, you cannot acquire the actual token through any of these means.

Tip: You can see all of your current and expired tokens through the Designer Cloud application . See *Access Tokens Page*.

Endpoint	Description
https://api.trifacta.com/ee/es.t/index.html#operation/listApiAccessTokens	List all access tokens for your user account.
https://api.trifacta.com/ee/es.t/index.html#operation/getApiAccessToken	List your access token for the specified token ID.

Renew Token

New tokens can be acquired at any time.

NOTE: It is the responsibility of the user to acquire a new API token before the current one expires. If a token is permitted to expire, a request for a new token must include `userId` and password information.

- See <https://api.trifacta.com/ee/es.t/index.html#operation/createApiAccessToken>
- See *Access Tokens Page*.

Delete Token

- **Via API:** Acquire the `tokenId` value for the token and use the delete endpoint. See <https://api.trifacta.com/ee/es.t/index.html#operation/deleteApiAccessToken>
- **Via UI:** In the Access Tokens page, select **Delete Token...** from the context menu for the token listing. See *Access Tokens Page*.

API Endpoints for Designer Cloud Powered by Trifacta Enterprise

The following endpoints are available for Designer Cloud Powered by Trifacta® Enterprise Edition.

To access the reference documentation for each available API endpoint and method, select **Resources menu > API documentation** in the Designer Cloud application .

ApiAccessToken

Path	Description	Documentation URL
POST /v4/apiAccessTokens	Create api access token	https://api.trifacta.com/ee/es.t/index.html#operation/createApiAccessToken
GET /v4/apiAccessTokens	List api access tokens	https://api.trifacta.com/ee/es.t/index.html#operation/listApiAccessTokens
GET /v4/apiAccessTokens/:tokenId	Get api access token	https://api.trifacta.com/ee/es.t/index.html#operation/getApiAccessToken
DELETE /v4/apiAccessTokens/:tokenId	Delete api access token	https://api.trifacta.com/ee/es.t/index.html#operation/deleteApiAccessToken

AwsConfig

Path	Description	Documentation URL
POST /v4/awsConfigs	Create AWS Config	https://api.trifacta.com/ee/es.t/index.html#operation/createAwsConfig
GET /v4/awsConfigs	List AWS configs	https://api.trifacta.com/ee/es.t/index.html#operation/listAwsConfigs
PUT /v4/awsConfigs/:id	Update AWS Config	https://api.trifacta.com/ee/es.t/index.html#operation/updateAwsConfig

AwsRole

Path	Description	Documentation URL
POST /v4/awsRoles	Create AWS role	https://api.trifacta.com/ee/es.t/index.html#operation/createAwsRole
GET /v4/awsRoles	List AWS roles	https://api.trifacta.com/ee/es.t/index.html#operation/listAwsRoles
PUT /v4/awsRoles/:id	Update AWS role	https://api.trifacta.com/ee/es.t/index.html#operation/updateAwsRole
DELETE /v4/awsRoles/:id	Delete AWS role	https://api.trifacta.com/ee/es.t/index.html#operation/deleteAwsRole

Connection

Path	Description	Documentation URL
POST /v4/connections	Create connection	https://api.trifacta.com/ee/es.t/index.html#operation/createConnection
GET /v4/connections	List connections	https://api.trifacta.com/ee/es.t/index.html#operation/listConnections
GET /v4/connections/count	Count connections	https://api.trifacta.com/ee/es.t/index.html#operation/countConnections
GET /v4/connections/:id	Get connection	https://api.trifacta.com/ee/es.t/index.html#operation/getConnection
DELETE /v4/connections/:id	Delete connection	https://api.trifacta.com/ee/es.t/index.html#operation/deleteConnection
GET /v4/connections/:id/status	Get connection status	https://api.trifacta.com/ee/es.t/index.html#operation/getConnectionStatus

ConnectorMetadata

Path	Description	Documentation URL
	Create overrides for connector	

POST /v4/connectorMetadata/:connector/overrides		https://api.trifacta.com/ee/es.t/index.html#operation/updateConnectorOverrides
GET /v4/connectorMetadata/:connector/overrides	Get overrides for connector	https://api.trifacta.com/ee/es.t/index.html#operation/getConnectorOverrides
DELETE /v4/connectorMetadata/:connector/overrides	Delete all custom overrides for a connector	https://api.trifacta.com/ee/es.t/index.html#operation/deleteConnectorOverrides
GET /v4/connectorMetadata/:connector	Get connector metadata information	https://api.trifacta.com/ee/es.t/index.html#operation/getConnectorConfig
GET /v4/connectorMetadata/:connector/defaults	Get default connector metadata information	https://api.trifacta.com/ee/es.t/index.html#operation/getConnectorDefaults

EnvironmentParameter

Path	Description	Documentation URL
POST /v4/environmentParameters	Create environment parameter	https://api.trifacta.com/ee/es.t/index.html#operation/createEnvironmentParameter
GET /v4/environmentParameters	List environment parameters	https://api.trifacta.com/ee/es.t/index.html#operation/listEnvironmentParameters
POST /v4/environmentParameters/package	Import environment parameters package	https://api.trifacta.com/ee/es.t/index.html#operation/importEnvironmentParametersPackage
GET /v4/environmentParameters/package	Export environment parameters list	https://api.trifacta.com/ee/es.t/index.html#operation/getEnvironmentParametersPackage
GET /v4/environmentParameters/:id	Get environment parameter	https://api.trifacta.com/ee/es.t/index.html#operation/getEnvironmentParameter
DELETE /v4/environmentParameters/:id	Delete environment parameter	https://api.trifacta.com/ee/es.t/index.html#operation/deleteEnvironmentParameter

Flow

Path	Description	Documentation URL
POST /v4/flows	Create flow	https://api.trifacta.com/ee/es.t/index.html#operation/createFlow
GET /v4/flows	List flows	https://api.trifacta.com/ee/es.t/index.html#operation/listFlows
POST /v4/flows/package	Import Flow package	https://api.trifacta.com/ee/es.t/index.html#operation/importPackage
POST /v4/flows/package/dryRun	Import Flow package - Dry run	https://api.trifacta.com/ee/es.t/index.html#operation/importPackageDryRun
POST /v4/flows/:id/copy	Copy Flow	https://api.trifacta.com/ee/es.t/index.html#operation/copyFlow
POST /v4/flows/:id/run	Run Flow	https://api.trifacta.com/ee/es.t/index.html#operation/runFlow
GET /v4/flows/count	Count flows	https://api.trifacta.com/ee/es.t/index.html#operation/countFlows
GET /v4/flows/:id	Get flow	https://api.trifacta.com/ee/es.t/index.html#operation/getFlow
PATCH /v4/flows/:id	Patch flow	https://api.trifacta.com/ee/es.t/index.html#operation/patchFlow
DELETE /v4/flows/:id	Delete flow	https://api.trifacta.com/ee/es.t/index.html#operation/deleteFlow
GET /v4/flows/:id/package	Export flow	https://api.trifacta.com/ee/es.t/index.html#operation/getFlowPackage
GET /v4/flows/:id/package/dryRun	Export flow - Dry run	https://api.trifacta.com/ee/es.t/index.html#operation/getFlowPackageDryRun
GET /v4/flowsLibrary	Flow Library (list)	https://api.trifacta.com/ee/es.t/index.html#operation/listFlowsLibrary
GET /v4/flowsLibrary/count	Flow Library (count)	https://api.trifacta.com/ee/es.t/index.html#operation/countFlowsLibrary
GET /v4/flows/:id/inputs	List Flow inputs	https://api.trifacta.com/ee/es.t/index.html#operation/getFlowInputs
GET /v4/flows/:id/outputs	List Flow outputs	https://api.trifacta.com/ee/es.t/index.html#operation/getFlowOutputs

GET /v4/folders/:id/flows	List flows in folder	https://api.trifacta.com/ee/es.t/index.html#operation/getFlowsForFolder
GET /v4/folders/:id/flows/count	Count flows in folder	https://api.trifacta.com/ee/es.t/index.html#operation/getFlowCountForFolder
PATCH /v4/flows/:id/replaceDataset	Replace dataset	https://api.trifacta.com/ee/es.t/index.html#operation/replaceDatasetInFlow

FlowNode

Path	Description	Documentation URL
POST /v4/flowNodes/:id/commitEdges	Create new edges	https://api.trifacta.com/ee/es.t/index.html#operation/commitEdges

FlowNotificationSettings

Path	Description	Documentation URL
POST /v4/flowNotificationSettings	Create flow notification settings	https://api.trifacta.com/ee/es.t/index.html#operation/createFlowNotificationSettings

FlowPermission

Path	Description	Documentation URL
POST /v4/flows/:id/permissions	Share Flow	https://api.trifacta.com/ee/es.t/index.html#operation/shareFlow

FlowRun

Path	Description	Documentation URL
GET /v4/flowRuns/:id	Get flow run	https://api.trifacta.com/ee/es.t/index.html#operation/getFlowRun
GET /v4/flowRuns/:id/status	Get Flow Run Status	https://api.trifacta.com/ee/es.t/index.html#operation/getFlowRunStatus
GET /v4/flowRuns/:id/jobGroups	Get JobGroups for Flow Run	https://api.trifacta.com/ee/es.t/index.html#operation/getFlowRunJobGroups

FlowRunParameterOverride

Path	Description	Documentation URL
POST /v4/flowRunParameterOverrides	Create flow run parameter override	https://api.trifacta.com/ee/es.t/index.html#operation/createFlowRunParameterOverride
GET /v4/flowRunParameterOverrides/:id	Get flow run parameter override	https://api.trifacta.com/ee/es.t/index.html#operation/getFlowRunParameterOverride
PATCH /v4/flowRunParameterOverrides/:id	Patch flow run parameter override	https://api.trifacta.com/ee/es.t/index.html#operation/patchFlowRunParameterOverride
DELETE /v4/flowRunParameterOverrides/:id	Delete flow run parameter override	https://api.trifacta.com/ee/es.t/index.html#operation/deleteFlowRunParameterOverride

Folder

Path	Description	Documentation URL
POST /v4/folders	Create folder	https://api.trifacta.com/ee/es.t/index.html#operation/createFolder
GET /v4/folders	List folders	https://api.trifacta.com/ee/es.t/index.html#operation/listFolders
GET /v4/folders/count	Count folders	https://api.trifacta.com/ee/es.t/index.html#operation/countFolders
PATCH /v4/folders/:id	Patch folder	https://api.trifacta.com/ee/es.t/index.html#operation/patchFolder
DELETE /v4/folders/:id	Delete folder	https://api.trifacta.com/ee/es.t/index.html#operation/deleteFolder

ImportedDataset

Path	Description	Documentation URL
POST /v4/importedDatasets	Create imported dataset	https://api.trifacta.com/ee/es.t/index.html#operation/createImportedDataset
POST /v4/importedDatasets/:id/addToFlow	Add Imported Dataset to Flow	https://api.trifacta.com/ee/es.t/index.html#operation/addImportedDatasetToFlow
POST /v4/importedDatasets/:id/copy	Copy imported dataset	https://api.trifacta.com/ee/es.t/index.html#operation/copyDataSource
POST /v4/importedDatasets/:id/asyncRefreshSchema	Fetch and update latest datasource schema	https://api.trifacta.com/ee/es.t/index.html#operation/asyncRefreshSchema
GET /v4/importedDatasets/:id	Get imported dataset	https://api.trifacta.com/ee/es.t/index.html#operation/getImportedDataset
PUT /v4/importedDatasets/:id	Update imported dataset	https://api.trifacta.com/ee/es.t/index.html#operation/updateImportedDataset
PATCH /v4/importedDatasets/:id	Patch imported dataset	https://api.trifacta.com/ee/es.t/index.html#operation/patchImportedDataset
DELETE /v4/importedDatasets/:id	Delete imported dataset	https://api.trifacta.com/ee/es.t/index.html#operation/deleteImportedDataset
GET /v4/datasetLibrary	List Datasets	https://api.trifacta.com/ee/es.t/index.html#operation/listDatasetLibrary
GET /v4/datasetLibrary/count	Count Datasets	https://api.trifacta.com/ee/es.t/index.html#operation/countDatasetLibrary

Job

Path	Description	Documentation URL
GET /v4/jobs/:id/status	Get Job Status	https://api.trifacta.com/ee/es.t/index.html#operation/getJobStatus

JobGroup

Path	Description	Documentation URL
POST /v4/jobGroups	Run Job Group	https://api.trifacta.com/ee/es.t/index.html#operation/runJobGroup
GET /v4/jobGroups	List job groups	https://api.trifacta.com/ee/es.t/index.html#operation/listJobGroups
POST /v4/jobGroups/:id/cancel	Cancel Job Group	https://api.trifacta.com/ee/es.t/index.html#operation/cancelJobGroup
GET /v4/jobGroups/:id	Get job group	https://api.trifacta.com/ee/es.t/index.html#operation/getJobGroup
GET /v4/jobGroups/:id/profile	Get Profile Information for Job Group	https://api.trifacta.com/ee/es.t/index.html#operation/getProfilingInformationForJobGroup
GET /v4/jobGroups/:id/profileResults	Get Profile Information for Job Group As a Map	https://api.trifacta.com/ee/es.t/index.html#operation/getProfilingInformationForJobGroupConsistent
GET /v4/jobGroups/:id/pdfResults	Get PDF Results for Job Group	https://api.trifacta.com/ee/es.t/index.html#operation/getJobGroupPdfResults
GET /v4/jobGroups/:id/status	Get JobGroup Status	https://api.trifacta.com/ee/es.t/index.html#operation/getJobGroupStatus
GET /v4/jobGroups/:id/inputs	Get Job Group Inputs	https://api.trifacta.com/ee/es.t/index.html#operation/getJobGroupInputs
GET /v4/jobGroups/:id/outputs	Get Job Group Outputs	https://api.trifacta.com/ee/es.t/index.html#operation/getJobGroupOutputs
GET /v4/jobLibrary	List Job Groups	https://api.trifacta.com/ee/es.t/index.html#operation/listJobLibrary

GET /v4/jobLibrary/count	Count Job Groups	https://api.trifacta.com/ee/es.t/index.html#operation/countJobLibrary
GET /v4/jobGroups/:id/jobs	Get Jobs for Job Group	https://api.trifacta.com/ee/es.t/index.html#operation/getJobsForJobGroup
GET /v4/jobGroups/:id/publications	Get Publications for Job Group	https://api.trifacta.com/ee/es.t/index.html#operation/getPublicationsForJobGroup

Macro

Path	Description	Documentation URL
POST /v4/macros/package	Import Macro Package	https://api.trifacta.com/ee/es.t/index.html#operation/importMacroPackage
GET /v4/macros/:id/package	Get Macro Package	https://api.trifacta.com/ee/es.t/index.html#operation/getMacroPackage

Misc

Path	Description	Documentation URL
GET /v4/open-api-spec	Get OpenAPI specification	https://api.trifacta.com/ee/es.t/index.html#operation/getOpenApiSpec

OutputObject

Path	Description	Documentation URL
POST /v4/outputObjects	Create output object	https://api.trifacta.com/ee/es.t/index.html#operation/createOutputObject
GET /v4/outputObjects	List output objects	https://api.trifacta.com/ee/es.t/index.html#operation/listOutputObjects
POST /v4/outputObjects/:id/wrangleToPython	Generate python script for wrangle recipe linked to an output object	https://api.trifacta.com/ee/es.t/index.html#operation/getPythonScriptForOutputObjectInput
GET /v4/outputObjects/count	Count output objects	https://api.trifacta.com/ee/es.t/index.html#operation/countOutputObjects
GET /v4/outputObjects/:id	Get output object	https://api.trifacta.com/ee/es.t/index.html#operation/getOutputObject
PATCH /v4/outputObjects/:id	Patch output object	https://api.trifacta.com/ee/es.t/index.html#operation/patchOutputObject
DELETE /v4/outputObjects/:id	Delete output object	https://api.trifacta.com/ee/es.t/index.html#operation/deleteOutputObject
GET /v4/outputObjects/:id/inputs	List inputs for Output Object	https://api.trifacta.com/ee/es.t/index.html#operation/getInputsForOutputObject

Person

Path	Description	Documentation URL
GET /v4/people/current	Get Current Person	https://api.trifacta.com/ee/es.t/index.html#operation/getCurrentPerson
GET /v4/people/:id	Get person	https://api.trifacta.com/ee/es.t/index.html#operation/getPerson
GET /v4/people	List people	https://api.trifacta.com/ee/es.t/index.html#operation/listPerson

Plan

Path	Description	Documentation URL
POST /v4/plans	Create plan	https://api.trifacta.com/ee/es.t/index.html#operation/createPlan
GET /v4/plans	List plans	https://api.trifacta.com/ee/es.t/index.html#operation/listPlans
POST /v4/plans/:id/run	Run plan	https://api.trifacta.com/ee/es.t/index.html#operation/runPlan

POST /v4/plans/:id/permissions	Share Plan	https://api.trifacta.com/ee/es.t/index.html#operation/sharePlan
GET /v4/plans/:id/permissions	List permissions for plan	https://api.trifacta.com/ee/es.t/index.html#operation/getPlanPermissions
POST /v4/plans/package	Import plan package	https://api.trifacta.com/ee/es.t/index.html#operation/importPlanPackage
GET /v4/plans/count	Count plans	https://api.trifacta.com/ee/es.t/index.html#operation/countPlans
GET /v4/plans/:id/runParameters	List run parameters	https://api.trifacta.com/ee/es.t/index.html#operation/planRunParameters
GET /v4/plans/:id/full	Read plan with all attributes	https://api.trifacta.com/ee/es.t/index.html#operation/readFull
GET /v4/plans/:id/schedules	List plan schedules	https://api.trifacta.com/ee/es.t/index.html#operation/getSchedulesForPlan
GET /v4/plans/:id/package	Export plan	https://api.trifacta.com/ee/es.t/index.html#operation/getPlanPackage
PATCH /v4/plans/:id	Update plan	https://api.trifacta.com/ee/es.t/index.html#operation/updatePlan
DELETE /v4/plans/:id	Delete plan	https://api.trifacta.com/ee/es.t/index.html#operation/deletePlan
DELETE /v4/plans/:id/permissions/:subjectId	Delete plan permissions for a user	https://api.trifacta.com/ee/es.t/index.html#operation/deletePlanPermissions

PlanNode

Path	Description	Documentation URL
POST /v4/planNodes	Create plan node	https://api.trifacta.com/ee/es.t/index.html#operation/createPlanNode
GET /v4/planNodes/:id/runParameters	List run parameters for a plan node	https://api.trifacta.com/ee/es.t/index.html#operation/getPlanNodeRunParameters
DELETE /v4/planNodes/:id	Delete plan node	https://api.trifacta.com/ee/es.t/index.html#operation/deletePlanNode

PlanOverride

Path	Description	Documentation URL
POST /v4/planOverrides	Override a parameter in a plan	https://api.trifacta.com/ee/es.t/index.html#operation/createPlanOverride
PUT /v4/planOverrides/:id	Update the value of a parameter override in a plan	https://api.trifacta.com/ee/es.t/index.html#operation/updatePlanOverride

PlanSnapshotRun

Path	Description	Documentation URL
POST /v4/planSnapshotRuns/:id/cancel	Cancel a plan execution	https://api.trifacta.com/ee/es.t/index.html#operation/cancelPlanSnapshotRun
GET /v4/planSnapshotRuns	List plan snapshot runs	https://api.trifacta.com/ee/es.t/index.html#operation/listPlanSnapshotRuns
GET /v4/planSnapshotRuns/count	Count plan snapshot runs	https://api.trifacta.com/ee/es.t/index.html#operation/countPlanSnapshotRuns
GET /v4/planSnapshotRuns/:id	Get plan snapshot run	https://api.trifacta.com/ee/es.t/index.html#operation/getPlanSnapshotRun
GET /v4/planSnapshotRuns/:id/schedule	Get schedule for plan run	https://api.trifacta.com/ee/es.t/index.html#operation/getScheduleForPlanRun

Publication

Path	Description	Documentation URL
POST /v4/publications	Create publication	https://api.trifacta.com/ee/es.t/index.html#operation/createPublication
GET /v4/publications	List publications	https://api.trifacta.com/ee/es.t/index.html#operation/listPublications
GET /v4/publications/count	Count publications	https://api.trifacta.com/ee/es.t/index.html#operation/countPublications
GET /v4/publications/:id	Get publication	https://api.trifacta.com/ee/es.t/index.html#operation/getPublication
PATCH /v4/publications/:id	Patch publication	https://api.trifacta.com/ee/es.t/index.html#operation/patchPublication
DELETE /v4/publications/:id	Delete publication	https://api.trifacta.com/ee/es.t/index.html#operation/deletePublication

Schedule

Path	Description	Documentation URL
POST /v4/schedules	Create a schedule	https://api.trifacta.com/ee/es.t/index.html#operation/createSchedule
GET /v4/schedules	List schedules	https://api.trifacta.com/ee/es.t/index.html#operation/listSchedules
POST /v4/schedules/:id/enable	Enable schedule	https://api.trifacta.com/ee/es.t/index.html#operation/enableSchedule
POST /v4/schedules/:id/disable	Disable schedule	https://api.trifacta.com/ee/es.t/index.html#operation/disableSchedule
GET /v4/schedules/count	Count schedules	https://api.trifacta.com/ee/es.t/index.html#operation/countSchedules
GET /v4/schedules/:id	Get schedule	https://api.trifacta.com/ee/es.t/index.html#operation/getSchedule
PUT /v4/schedules/:id	Update a schedule	https://api.trifacta.com/ee/es.t/index.html#operation/updateSchedule
DELETE /v4/schedules/:id	Delete schedule	https://api.trifacta.com/ee/es.t/index.html#operation/deleteSchedule

SqlScript

Path	Description	Documentation URL
POST /v4/sqlScripts	Create sql script	https://api.trifacta.com/ee/es.t/index.html#operation/createSqlScript
GET /v4/sqlScripts	List sql scripts	https://api.trifacta.com/ee/es.t/index.html#operation/listSqlScripts
GET /v4/sqlScripts/count	Count sql scripts	https://api.trifacta.com/ee/es.t/index.html#operation/countSqlScripts
GET /v4/sqlScripts/:id	Get sql script	https://api.trifacta.com/ee/es.t/index.html#operation/getSqlScript
PATCH /v4/sqlScripts/:id	Patch sql script	https://api.trifacta.com/ee/es.t/index.html#operation/patchSqlScript
DELETE /v4/sqlScripts/:id	Delete sql script	https://api.trifacta.com/ee/es.t/index.html#operation/deleteSqlScript

WebhookFlowTask

Path	Description	Documentation URL
POST /v4/webhookFlowTasks	Create webhook	https://api.trifacta.com/ee/es.t/index.html#operation/createWebhookFlowTask
POST /v4/webhooks/test	Test webhook settings	https://api.trifacta.com/ee/es.t/index.html#operation/testWebhook
GET /v4/webhookFlowTasks/:id	Read webhook	https://api.trifacta.com/ee/es.t/index.html#operation/getWebhookFlowTask
DELETE /v4/webhookFlowTasks/:id	Delete webhook	https://api.trifacta.com/ee/es.t/index.html#operation/deleteWebhookFlowTask

Workspace

Path	Description	Documentation URL

POST /v4/workspaces/current/delete-configuration	Reset a configuration settings for the current workspace	https://api.trifacta.com/ee/es.t/index.html#operation/deleteCurrentWorkspaceConfigurationSettings
POST /v4/workspaces/:id/delete-configuration	Reset a workspace configuration settings	https://api.trifacta.com/ee/es.t/index.html#operation/deleteWorkspaceConfigurationSettings
GET /v4/workspaces/current/configuration	Get current workspace configuration	https://api.trifacta.com/ee/es.t/index.html#operation/getCurrentConfigurationForWorkspace
PATCH /v4/workspaces/current/configuration	Save current workspace configuration	https://api.trifacta.com/ee/es.t/index.html#operation/saveCurrentWorkspaceConfiguration
GET /v4/workspaces/:id/configuration	Get workspace configuration	https://api.trifacta.com/ee/es.t/index.html#operation/getConfigurationForWorkspace
PATCH /v4/workspaces/:id/configuration	Save workspace configuration	https://api.trifacta.com/ee/es.t/index.html#operation/saveWorkspaceConfiguration
GET /v4/workspaces/:id/configuration-schema	Get configuration schema	https://api.trifacta.com/ee/es.t/index.html#operation/getConfigurationSchema
GET /v4/workspaces/current/configuration-schema	Get current configuration schema	https://api.trifacta.com/ee/es.t/index.html#operation/getCurrentConfigurationSchema
PATCH /v4/workspaces/current/transfer	Transfer User Assets	https://api.trifacta.com/ee/es.t/index.html#operation/transferUserAssetsInCurrentWorkspace

WrangledDataset

Path	Description	Documentation URL
POST /v4/wrangledDatasets	Create wrangled dataset	https://api.trifacta.com/ee/es.t/index.html#operation/createWrangledDataset
GET /v4/wrangledDatasets	List wrangled datasets	https://api.trifacta.com/ee/es.t/index.html#operation/listWrangledDatasets
POST /v4/wrangledDatasets/:id/addToFlow	Add wrangled dataset to flow	https://api.trifacta.com/ee/es.t/index.html#operation/addWrangledDatasetToFlow
GET /v4/wrangledDatasets/count	Count wrangled datasets	https://api.trifacta.com/ee/es.t/index.html#operation/countWrangledDatasets
GET /v4/wrangledDatasets/:id	Get wrangled dataset	https://api.trifacta.com/ee/es.t/index.html#operation/getWrangledDataset
PATCH /v4/wrangledDatasets/:id	Patch Wrangled Dataset	https://api.trifacta.com/ee/es.t/index.html#operation/patchWrangledDataset
DELETE /v4/wrangledDatasets/:id	Delete wrangled dataset	https://api.trifacta.com/ee/es.t/index.html#operation/deleteWrangledDataset
GET /v4/wrangledDatasets/:id/primaryInputDataset	Get Input Dataset	https://api.trifacta.com/ee/es.t/index.html#operation/getInputDataset
PUT /v4/wrangledDatasets/:id/primaryInputDataset	Swap Input Dataset	https://api.trifacta.com/ee/es.t/index.html#operation/updateInputDataset

WriteSetting

Path	Description	Documentation URL
POST /v4/writeSettings	Create writesetting	https://api.trifacta.com/ee/es.t/index.html#operation/createWriteSetting
GET /v4/writeSettings	List write settings	https://api.trifacta.com/ee/es.t/index.html#operation/listWriteSettings
GET /v4/writeSettings/count	Count write settings	https://api.trifacta.com/ee/es.t/index.html#operation/countWriteSettings
GET /v4/writeSettings/:id	Get write setting	https://api.trifacta.com/ee/es.t/index.html#operation/getWriteSetting
PATCH /v4/writeSettings/:id	Patch write setting	https://api.trifacta.com/ee/es.t/index.html#operation/patchWriteSetting
DELETE /v4/writeSettings/:id	Delete write setting	https://api.trifacta.com/ee/es.t/index.html#operation/deleteWriteSetting

API Tasks

In this section, you can review examples of how to execute common tasks using one or more of the available REST APIs.

API Task - Develop a Flow

Contents:

- Overview
 - Example Datasets
- Step - Create Containing Flow
- Step - Create Datasets
- Step - Wrangle Data
- Step - Create Output Objects
- Step - Run Job
- Step - Monitoring Your Job
- Step - Re-run Job

Overview

This example walks through the process of creating, identifying, and executing a job through automated methods. For this example, these tasks are accomplished using the following methods:

NOTE: This API task applies to a Development instance of the Designer Cloud powered by Trifacta® platform, which is the default platform instance type. For more information on Development and Production instance, see *Overview of Deployment Manager*.

1. **Locate or create flow.** The datasets that you wrangle must be contained within a flow. You can add them to an existing flow or create a new one through the APIs.
2. **Create dataset.** Through the APIs, you create an imported dataset from an asset that is accessible through one of the established connections. Then, you create the recipe object through the API.
 - a. For the recipe, you must retrieve the internal identifier.
 - b. Through the application, you modify the recipe for the dataset.
3. **Automate job execution.** Using the APIs, you can automate execution of the wrangling of the dataset.
 - a. As needed, this job can be re-executed on a periodic basis or whenever the source files are updated.

Example Datasets

In this example, you are attempting to wrangle monthly point of sale (POS) data from three separate regions into a single dataset for the state. This monthly data must be enhanced with information about the products and stores in the state. So, the example has a combination of transactional and reference data, which must be brought together into a single dataset.

Tip: To facilitate re-execution of this job each month, the transactional data should be stored in a dedicated directory. This directory can be overwritten with next month's data using the same filenames. As long as the new files are structured in an identical manner to the original ones, the new month's data can be processed by re-running the API aspects of this task.

Example Files:

The following files are stored on HDFS:

Path and Filename	Description

hdfs:///user/pos/POS-r01.txt	Point of sale transactions for Region 1.
hdfs:///user/pos/POS-r02.txt	Point of sale transactions for Region 2.
hdfs:///user/pos/POS-r03.txt	Point of sale transactions for Region 3.
hdfs:///user/ref/REF_PROD.txt	Reference data on products for the state.
hdfs:///user/ref/REF_CAL.txt	Reference data on stores in the state.

NOTE: The reference and transactional data are stored in separate directories. In this case, you can assume that the user has read access through his Trifacta account to these directories, although this access must be enabled and configured for real use cases.

Base URL:

For purposes of this example, the base URL for the Trifacta platform is the following:

```
http://www.example.com:3005
```

Step - Create Containing Flow

To begin, you must locate a flow or create a flow through the APIs to contain the datasets that you are importing.

NOTE: You cannot add datasets to the flow through the `flows` endpoint. Moving pre-existing datasets into a flow is not supported in this release. Create or locate the flow first and then when you create the datasets, associate them with the flow at the time of creation.

- See <https://api.trifacta.com/ee/es.t/index.html#operation/createImportedDataset>
- See <https://api.trifacta.com/ee/es.t/index.html#operation/createWrangledDataset>

Locate:

NOTE: If you know the display name value for the flow and are confident that it is not shared with any other flows, you can use the APIs to retrieve the flowId. See <https://api.trifacta.com/ee/es.t/index.html#operation/listFlows>

1. Login through the application.
2. In the Flows page, select or create the flow to contain the above datasets.
3. In the Flow Details page for that flow, locate the flow identifier in the URL:

Flow Details URL	http://www.example.com:3005/flows/10
Flow Id	10

4. Retain this identifier for later use.

Create:

1. Through the APIs, you can create a flow using the following call:

Endpoint	http://www.example.com:3005/v4/flows
Authentication	Required

Method	POST
Request Body	<pre>{ "name": "Point of Sale - 2013", "description": "Point of Sale data for state" }</pre>

- The response should be status code 201 - Created with a response body like the following:

```
{
  "id": 10,
  "updatedAt": "2017-02-17T17:08:57.848Z",
  "createdAt": "2017-02-17T17:08:57.848Z",
  "name": "Point of Sale - 2013",
  "description": "Point of Sale data for state",
  "creator": {
    "id": 1
  },
  "updater": {
    "id": 1
  },
  "workspace": {
    "id": 1
  }
}
```

- Retain the flow identifier (10) for later use.

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/createFlow>

Checkpoint: You have identified or created the flow to contain your dataset or datasets.

Step - Create Datasets

To create datasets from the above sources, you must:

- Create an imported dataset for each file.
- For each imported dataset, create a recipe, which can be used to transform the imported dataset.

The following steps describe how to complete these actions via API for a single file.

Steps:

- To create an imported dataset, you must acquire the following information about the source. In the above example, the source is the `POS-r01.txt` file.
 - uri
 - name
 - description
 - bucket (if a file stored on S3)
- Construct the following request:

Endpoint	<code>http://www.example.com:3005/v4/importedDatasets</code>
Authentication	Required
Method	POST
Request Body	

```
{
  "uri": "hdfs:///user/pos/POS-r01.txt",
  "name": "POS-r01.txt",
  "description": "POS-r01.txt"
}
```

3. You should receive a 201 - Created response with a response body similar to the following:

```
{
  "id": 8,
  "size": "281032",
  "uri": "hdfs:///user/pos/POS-r01.txt",
  "dynamicPath": null,
  "bucket": null,
  "isSchematized": true,
  "isDynamic": false,
  "disableTypeInference": false,
  "updatedAt": "2017-02-08T18:38:56.640Z",
  "createdAt": "2017-02-08T18:38:56.560Z",
  "parsingScriptId": {
    "id": 14
  },
  "runParameters": {
    "data": []
  },
  "name": "POS-r01.txt",
  "description": "POS-r01.txt",
  "creator": {
    "id": 1
  },
  "updater": {
    "id": 1
  },
  "connection": null
}
```

4. You must retain the id value so you can reference it when you create the recipe.
 5. See <https://api.trifacta.com/ee/es.t/index.html#operation/createImportedDataset>
 6. Next, you create the recipe. Construct the following request:

Endpoint	http://www.example.com:3005/v4/wrangledDatasets
Authentication	Required
Method	POST
Request Body	<pre>{ "name": "POS-r01", "importedDataset": { "id": 8 }, "flow": { "id": 10 } }</pre>

7. You should receive a 201 - Created response with a response body similar to the following:

```
{
  "id": 23,
  "wrangled": true,
  "updatedAt": "2018-02-06T19:59:22.735Z",
  "createdAt": "2018-02-06T19:59:22.698Z",
  "name": "POS-r01",
  "active": true,
}
```



```

    "referenceInfo": null,
    "activeSample": {
      "id": 23
    },
    "creator": {
      "id": 1
    },
    "updater": {
      "id": 1
    },
    "recipe": {
      "id": 23
    },
    "flow": {
      "id": 10
    }
  }
}

```

8. From the recipe, you must retain the value for the `id`. For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/createWrangledDataset>
9. Repeat the above steps for each of the source files that you are adding to your flow.

Checkpoint: You have created a flow with multiple imported datasets and recipes.

Step - Wrangle Data

After you have created the flow with all of your source datasets, you can wrangle the base dataset to integrate all of the source into it.

Steps for Transactional data:

1. Open the `POS-r01` dataset. It's loaded in the Transformer page.
2. To chain together the other transactional data into this dataset, you use a `union` transform. In the Search panel, enter `union` in the textbox and press `ENTER`.
3. In the Union page:
 - a. Click **Add datasets**.
 - b. Select the other two transactional datasets: `POS-r02` and `POS-r03`.

NOTE: When you join or union one dataset into another, changes made in the joined dataset are automatically propagated to the dataset where it has been joined.

- c. Add the datasets and align by name.
- d. Check the dataset names and fields. If all looks well, click **Add to Recipe**.

Steps for reference data:

In the columns `Store_Nbr` and `Item_Nbr` are unique keys into the `REF_CAL` and `REF_PROD` datasets, respectively. Using the Join window, you can pull in the other fields from these reference datasets based on these unique keys.

1. Open the `POS-r01` dataset.
2. In Search panel, enter `join datasets` for the transform. The Join window opens.
3. Select the `RED_PROD` dataset. Click **Accept**. Click **Next**.
4. Review the two keys to verify that they are the proper columns on which to structure the join. Click **Next**.
5. Click the All tab. Select all fields to add. Click **Review**.
6. After reviewing your join, click **Add to Recipe**.
7. For each `Item_Nbr` value that has a matching `ITEM_NBR` value in the reference dataset, all of the other reference fields are pulled into the `POS-r01` dataset.

You can repeat the above general process to integrate the reference data for stores.

Checkpoint: You have created a flow with multiple datasets and have integrated all of the relevant data into a single dataset.

Step - Create Output Objects

Before you run a job, you must define output objects, which specify the following:

- Running environment where the job is executed
- Profiling on or off
- outputObjects have the following objects associated with them:
 - **writeSettings:** These objects define the file-based outputs that are produced for the output object
 - **publications:** These objects define the database target, table, and other settings for publication to a relational datastore.

NOTE: You can continue with this task without creating outputObjects yet. In this task, overrides are applied during the job definition, so you don't have to create the outputObjects and writeSettings at this time.

For more information on creating outputObjects, writeSettings, and publications, see *API Task - Manage Outputs*.

Step - Run Job

Through the APIs, you can specify and run a job. In the above example, you must run the job for the terminal dataset, which is `POS-r01` in this case. This dataset contains references to all of the other datasets. When the job is run, the recipes for the other datasets are also applied to the terminal dataset, which ensures that the output reflects the proper integration of these other datasets into `POS-r01`.

NOTE: In the following example, writeSettings have been specified as overrides in the job definition. These overrides are applied for this job run only. If you need to re-run the job with these settings, you must either 1) re-apply the overrides or 2) create the writeSettings objects. For more information, see *API Task - Manage Outputs*.

Steps:

1. Acquire the internal identifier for the recipe for which you wish to execute a job. In the previous example, this identifier was 23.
2. Construct a request using the following:

Endpoint	<code>http://www.example.com:3005/v4/jobGroups</code>
Authentication	Required
Method	POST

Request Body:

```
{
  "wrangledDataset": {
    "id": 23
  },
}
```

```

"overrides": {
  "execution": "photon",
  "profiler": true,
  "writesettings": [
    {
      "path": "hdfs://hadoop:50070/trifacta/queryResults/admin@example.com/POS-r01.csv",
      "action": "create",
      "format": "csv",
      "compression": "none",
      "header": false,
      "asSingleFile": false
    }
  ]
},
"ranfrom": null
}

```

3. In the above example, the specified job has been launched for recipe 23 to execute on the Trifacta Photon running environment with profiling enabled.
 - a. Output format is CSV to the designated path. For more information on these properties, see <https://api.trifacta.com/ee/es.t/index.html#operation/runJobGroup>
 - b. Output is written as a new file with no overwriting of previous files.
4. A response code of 201 - Created is returned. The response body should look like the following:

```

{
  "sessionId": "79276c31-c58c-4e79-ae5e-fed1a25ebca1",
  "reason": "JobStarted",
  "jobGraph": {
    "vertices": [
      21,
      22
    ],
    "edges": [
      {
        "source": 21,
        "target": 22
      }
    ]
  },
  "id": 3,
  "jobs": {
    "data": [
      {
        "id": 21
      },
      {
        "id": 22
      }
    ]
  }
}

```

5. Retain the `id` value, which is the job identifier, for monitoring.

Step - Monitoring Your Job

You can monitor the status of your job through the following endpoint:

Endpoint	<code>http://www.example.com:3005/v4/jobGroups/<id>/status</code>
Authentication	Required
Method	GET

Request Body	None.
---------------------	-------

When the job has successfully completed, the returned status message is the following:

```
"Complete"
```

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/runJobGroup>

Step - Re-run Job

In the future, you can re-run the job exactly as you specified it by executing the following call:

Tip: You can swap imported datasets before re-running the job. For example, if you have uploaded a new file, you can change the primary input dataset for the dataset and then use the following API call to re-run the job as specified. See <https://api.trifacta.com/ee/es.t/index.html#operation/updateInputDataset>

Endpoint	<code>http://www.example.com:3005/v4/jobGroups</code>
Authentication	Required
Method	POST
Request Body	<pre>{ "wrangledDataset": { "id": 23 }, "overrides": { "execution": "photon", "profiler": true, "writesettings": [{ "path": "hdfs://hadoop:50070/trifacta/queryResults/admin@example.com/POS-r01.csv", "action": "create", "format": "csv", "compression": "none", "header": false, "asSingleFile": false }] }, "ranfrom": null }</pre>

The job is re-run as it was previously specified.

API Task - Deploy a Flow

Contents:

- Overview
 - Prerequisites
 - Task
- Step - Get Flow Id
- Step - Export a Flow
- Step - Create Deployment
- Step - Create Connection
- Step - Create Import Rules
- Step - Import Package to Create Release
- Step - Activate Release
- Step - Run Deployment
- Step - Iterate
- Step - Set up Production Schedule

Overview

In this task, you learn how to deploy a flow in development to a production instance of the platform. After you have created and finished a flow in a Development (Dev) instance, you can deploy it to an environment designed primarily for production execution of jobs for finished flows (Prod instance). For more information on managing these deployments, see *Overview of Deployment Manager*.

Prerequisites

Finished flow: This example assumes that you have finished development of a flow with the following characteristics:

- Single dataset imported from a table through a Redshift connection
- Single JSON output

Separate Dev and Prod instances: Although it is possible to deploy flows to the same instance in which they are developed, this example assumes that you are deploying from a Dev instance to a completely separate Prod instance. The following implications apply:

- Separate user accounts to access Dev (User1) and Prod (Admin2) instances.

Tip: You should do all of your recipe development and testing in Dev/Test. Avoid making changes in a Prod environment.

NOTE: Although these are separate user accounts, the assumption is that the same admin-level user is using these accounts through the APIs.

- New connections must be created in the Prod instance to access the production version of the database table.

Task

In this example, your environment contains separate Dev and Prod instances, each of which has a different set of users.

Item	Dev	Prod

Environment	http://wrangle-dev.example.com:3005 Tip: Dev environment work can be done through the UI, which may be easier.	http://wrangle-prod.example.com:3005
User	User1 NOTE: User1 has no access to Prod.	Admin2
Source DB	devWrangleDB	prodWrangleDB
Source Table	Dev-Orders	Prod-Orders
Connection Name	Dev Redshift Conn	Prod Redshift Conn

Example Flow:

User1 is creating a flow, which is used to wrangle weekly batches of orders for the enterprise. The flow contains:

- A single imported dataset that is created from a Redshift database table.
- A single recipe that modifies the imported dataset.
- A single output to a JSON file.
- Production data is hosted in a different Redshift database. So, the Prod connection is different from the Dev connection.

Steps:

1. **Build in Dev instance:** User1 creates the flow and iterates on building the recipe and running jobs until a satisfactory output can be generated in JSON format.
2. **Export:** When User1 is ready to push the flow to production, User1 exports the flow and downloads the export package ZIP file to the local desktop.
3. **Deploy to Prod instance:**
 - a. Admin2 creates a new deployment in the Prod instance.
 - b. Admin2 creates a new connection (Prod Redshift Conn) in the Prod instance.
 - c. Admin2 creates new import rules in the Prod instance to map from the old connection (Dev Redshift Conn) to the new one (Prod Redshift Conn).
 - d. Admin2 uploads the export ZIP package.
4. **Test deployment:** Through Flow View in the Prod instance, Admin2 runs a job. The results look fine.
5. **Set schedule:** Using cron, Admin2 sets a schedule to run the active release for this deployment once per week.
 - a. Each week, the Prod-Orders table must be refreshed with data.
 - b. The dataset is now operational in the Prod environment.

Step - Get Flow Id

The first general step is for the Dev user (User1) to get the flowId and export the flow from the Dev instance.

Steps:

Tip: If it's easier, you can gather the flowId from the user interface in Flow View. In the following example, the flowId is 21:

```
http://www.wrangle-dev.example.com:3005/flows/21
```

1. Through the APIs, you can create a flow using the following call:

Endpoint	http://www.wrangle-dev.example.com:3005/v4/flows
Authentication	Required
Method	GET
Request Body	None.

2. The response should be status code 200 – OK with a response body like the following:

```
{  "data": [
    {
      "id": 21,
      "name": "Intern Training",
      "description": "null",
      "createdAt": "2019-01-08T18:14:37.851Z",
      "updatedAt": "2019-01-08T18:57:26.824Z",
      "creator": {
        "id": 2
      },
      "updater": {
        "id": 2
      },
      "folder": {
        "id": 1
      },
      "workspace": {
        "id": 1
      }
    },
    {
      "id": 19,
      "name": "example Flow",
      "description": null,
      "createdAt": "2019-01-08T17:25:21.392Z",
      "updatedAt": "2019-01-08T17:30:30.959Z",
      "creator": {
        "id": 2
      },
      "updater": {
        "id": 2
      },
      "folder": {
        "id": 4
      },
      "workspace": {
        "id": 1
      }
    }
  ]
}
```

3. Retain the flow identifier (21) for later use.

Checkpoint: You have identified the flow to export.

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/listFlows>

Step - Export a Flow

Export the flow to your local desktop.

Tip: This step may be easier to do through the UI in the Dev instance.

Steps:

1. Export flowId=21:

Endpoint	http://www.wrangle-dev.example.com:3005/v4/flows/21/package
Authentication	Required
Method	GET
Request Body	None.

2. The response should be status code 200 - OK. The response body is the flow itself.
3. Download and save this file to your local desktop. Let's assume that the filename you choose is flow-WrangleOrders.zip.

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/getFlowPackage>

Step - Create Deployment

In the Prod environment, you can create the deployment from which you can manage the new flow. Note that the following information has changed for this environment:

Item	Prod env value
userId	Admin2
baseURL	http://www.wrangle-prod.example.com:3005

Steps:

1. Through the APIs, you can create a deployment using the following call:

Endpoint	http://www.wrangle-prod.example.com:3005/v4/deployments
Authentication	Required <div>NOTE: Username and password credentials must be submitted for the Admin2 account.</div>
Method	POST
Request Body	<pre>{ "name": "Production Orders" }</pre>

2. The response should be status code 201 - Created with a response body like the following:

```
{
  "id": 3,
  "name": "Production Orders",
```



```

    "updatedAt": "2017-11-27T23:48:54.340Z",
    "createdAt": "2017-11-27T23:48:54.340Z",
    "creator": {
      "id": 1
    },
    "updater": {
      "id": 1
    }
  }
}

```

3. Retain the deploymentId (3) for later use.

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/createDeployment>

Step - Create Connection

When a flow is exported, its connections are not included in the export. Before you import the flow into a new environment:

- Connections must be created or recreated in the Prod environment. In some cases, you may need to point to production versions of the data contained in completely different databases.
- Rules must be created to remap the connection to use in the imported flow.

This section and the following step through these processes.

Steps:

1. From the Dev environment, you collect the connection information for the flow:

Endpoint	http://www.wrangle-dev.example.com:3005/v4/connections
Authentication	Required <div> NOTE: Username and password credentials must be submitted for the User1 account. </div>
Method	GET
Request Body	None.

2. The response should be status code 200 - Ok with a response body like the following:

```

{
  "data": [
    {
      "id": 9,
      "host": "dev-redshift.example.com",
      "port": 5439,
      "vendor": "redshift",
      "params": {
        "connectStrOpts": "",
        "defaultDatabase": "devWrangleDB",
        "extraLoadParams": "BLANKSASNULL EMPTYASNULL TRIMBLANKS TRUNCATECOLUMNS"
      },
      "ssl": false,
      "vendorName": "redshift",
      "name": "Dev Redshift Conn",
      "description": "",
      "type": "jdbc",
      "isGlobal": true,
      "credentialType": "iamRoleArn",
      "credentialsShared": true,
      "uuid": "b8014610-ce56-11e7-9739-27deec2c3249",
      "disableTypeInference": false,
    }
  ]
}

```

```

    "createdAt": "2017-11-21T00:55:50.770Z",
    "updatedAt": "2017-11-21T00:55:50.770Z",
    "credentials": [
      {
        "user": "devDBuser"
      }
    ],
    "creator": {
      "id": 2
    },
    "updater": {
      "id": 2
    },
    "workspace": {
      "id": 1
    }
  }
},
"count": {
  "owned": 1,
  "shared": 0,
  "count": 1
}
}

```

3. You retain the above information for use in Production.
4. In the Prod environment, you create the new connection using the following call:

Endpoint	http://www.wrangle-prod.example.com:3005/v4/connections
Authentication	Required <div> NOTE: Username and password credentials must be submitted for the Admin2 account. </div>
Method	POST
Request Body	<pre> { "host": "prod-redshift.example.com", "port": 1433, "vendor": "redshift", "params": { "connectStrOpts": "", "defaultDatabase": "prodWrangleDB", "extraLoadParams": "BLANKSASNULL EMPTYASNULL TRIMBLANKS TRUNCATECOLUMNS" }, "vendorName": "redshift", "name": "Redshift Conn Prod", "description": "", "isGlobal": true, "type": "jdbc", "ssl": false, "credentialType": "iamRoleArn", "credentials": [{ "username": "prodDBUser", "password": "<password>", "iamRoleArn": "iam:aws:12345" }] } </pre>

5. The response should be status code 201 - Created with a response body like the following:

```
{
  "id": 12,
  "host": "prod-redshift.example.com",
  "port": 5439,
  "vendor": "redshift",
  "params": {
    "connectStrOpts": "",
    "defaultDatabase": "prodWrangleDB",
    "extraLoadParams": "BLANKSASNULL EMPTYASNNULL TRIMBLANKS TRUNCATECOLUMNS"
  },
  "ssl": false,
  "name": "Redshift Conn Prod",
  "description": "",
  "type": "jdbc",
  "isGlobal": true,
  "credentialType": "iamRoleArn",
  "credentialsShared": true,
  "uuid": "fa7e06c0-0143-11e8-8faf-27c0392328c5",
  "disableTypeInference": false,
  "createdAt": "2018-01-24T20:20:11.181Z",
  "updatedAt": "2018-01-24T20:20:11.181Z",
  "credentials": [
    {
      "username": "prodDBUser"
    }
  ],
  "creator": {
    "id": 2
  },
  "updater": {
    "id": 2
  }
}
```

- When you hit the `/v4/connections` endpoint again, you can retrieve the `connectionId` for this connection. In this case, let's assume that the `connectionId` value is 12.

See <https://api.trifacta.com/ee/es.t/index.html#operation/createConnection>

Step - Create Import Rules

Now that you have defined the connection to use to acquire the production data from within the production environment, you must create an import rule to remap from the Dev connection to the Prod connection within the flow definition. This rule is applied during the import process to ensure that the flow is working after it has been imported.

In this case, you must remap the `uuid` value for the Dev connection, which is written into the flow definition, with the connection `Id` value from the Prod instance.

For more information on import rules, see *API Task - Define Deployment Import Mappings*.

Steps:

- From the Dev environment, you collect the connection information for the flow:

Endpoint	<code>http://www.wrangle-dev.example.com:3005/v4/connections</code>
Authentication	Required
	<div>NOTE: Username and password credentials must be submitted for the <code>User1</code> account.</div>
Method	GET

Request Body	None.
---------------------	-------

2. The response should be status code 200 - Ok with a response body like the following:

```
{
  "data": [
    {
      "id": 9,
      "host": "dev-redshift.example.com",
      "port": 5439,
      "vendor": "redshift",
      "params": {
        "connectStrOpts": "",
        "defaultDatabase": "devWrangleDB",
        "extraLoadParams": "BLANKSASNULL EMPTYASNULL TRIMBLANKS TRUNCATECOLUMNS"
      },
      "ssl": false,
      "vendorName": "redshift",
      "name": "Dev Redshift Conn",
      "description": "",
      "type": "jdbc",
      "isGlobal": true,
      "credentialType": "iamRoleArn",
      "credentialsShared": true,
      "uuid": "b8014610-ce56-11e7-9739-27deec2c3249",
      "disableTypeInference": false,
      "createdAt": "2017-11-21T00:55:50.770Z",
      "updatedAt": "2017-11-21T00:55:50.770Z",
      "credentials": [
        {
          "user": "devDBuser"
        }
      ],
      "creator": {
        "id": 2
      },
      "updater": {
        "id": 2
      },
      "workspace": {
        "id": 1
      }
    }
  ],
  "count": {
    "owned": 1,
    "shared": 0,
    "count": 1
  }
}
```

3. From the above information, you retain the following, which uniquely identifies the connection object, regardless of the instance to which it belongs:

```
"uuid": "b8014610-ce56-11e7-9739-27deec2c3249",
```

4. Against the Prod environment, you now create an import mapping rule:

Endpoint	http://www.wrangle-prod.example.com:3005/v4/deployments/3/objectImportRules
Authentication	Required
Method	PATCH

Request Body:

```
[{"tableName": "connections", "onCondition": {"uuid": "b8014610-ce56-11e7-9739-27deec2c3249"}, "withCondition": {"id": 12}}]
```

5. The response should be status code 200 - Ok with a response body like the following:

```
{
  "deleted": []
}
```

Since the method is a `PATCH`, you are updating the rules set that applies to all imports for this deployment. In this case, there were no pre-existing rules, so the response indicates that nothing was deleted. If another set of import rules is submitted, then the one you just created is deleted.

See <https://api.trifacta.com/ee/es.t/index.html#operation/updateObjectImportRules>

See <https://api.trifacta.com/ee/es.t/index.html#operation/updateValueImportRules>

Step - Import Package to Create Release

You are now ready to import the package to create the release.

Steps:

1. Against the Prod environment, you now import the package:

Endpoint	http://www.wrangle-prod.example.com:3005/v4/deployments/3/releases	
Authentication	Required	
Method	POST	
Request Body	The request body must include the following key and value combination submitted as form data:	
	key	value
	data	"@path-to-flow-WrangleOrders.zip"

2. The response should be status code 201 - Created with a response body like the following:

```
{
  "importRuleChanges": {
    "object": [{"tableName": "connections", "onCondition": {"uuid": "b8014610-ce56-11e7-9739-27deec2c3249"}, "withCondition": {"id": 12}}],
    "value": []
  },
  "flowName": "Wrangle Orders"
}
```

See <https://api.trifacta.com/ee/es.t/index.html#operation/importPackageForDeployment>

Step - Activate Release

When a package is imported into a release, the release is automatically set as the active release for the deployment. If at some point in the future, you need to change the active release, you can use the following endpoint to do so.

Steps:

1. Against the Prod environment, use the following endpoint:

Endpoint	http://www.wrangle-prod.example.com:3005/v4/releases/5
Authentication	Required
Method	PATCH
Request Body	<pre>{ "active": true }</pre>

2. The response should be status code 200 - OK with a response body like the following:

```
{
  "id": 3,
  "updater": {
    "id": 3
  },
  "updatedAt": "2017-11-28T00:06:12.147Z"
}
```

See <https://api.trifacta.com/ee/es.t/index.html#operation/patchRelease>

Step - Run Deployment

You can now execute a test run of the deployment to verify that the job executes properly.

NOTE: When you run a deployment, you run the primary flow in the active release for that deployment. Running the flow generates the output objects for all recipes in the flow.

NOTE: For datasets with parameters, you can apply parameter overrides through the request body through the following API call. For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/runDeployment>

Steps:

1. Against the Prod environment, use the following endpoint:

Endpoint	http://www.wrangle-prod.example.com:3005/v4/deployments/3/run
Authentication	Required
Method	POST
Request Body	None.

2. The response should be status code 201 - Created with a response body like the following:

```
{
  "data": [
    {
      "reason": "JobStarted",
      "sessionId": "dd6a90e0-c353-11e7-ad4e-7f2dd2ae4621",
      "id": 33
    }
  ]
}
```

See <https://api.trifacta.com/ee/es.t/index.html#operation/runDeployment>

Step - Iterate

If you need to make changes to fix issues related to running the job:

- Recipe changes should be made in the Dev environment and then passed through export and import of the flow into the Prod deployment.
- Connection issues:
 - Check Flow View in the Prod instance to see if there are any red dots on the objects in the package. If so, your import rules need to be fixed.
 - Verify that you can import data through the connection.
- Output problems could be related to permissions on the target location.

Step - Set up Production Schedule

When you are satisfied with how the production version of your flow is working, you can set up periodic schedules using a third-party tool to execute the job on a regular basis.

The tool must hit the Run Deployment endpoint and then verify that the output has been properly generated.

API Task - Run Job

Contents:

- *Run Job Endpoints*
 - *Run job*
 - *Run flow*
 - *Run deployment*
- *Prerequisites*
- *Step - Run Job*
- *Step - Monitoring Your Job*
- *Step - Re-run Job*
- *Step - Run Job with Overrides - Files*
 - *Input file overrides*
 - *Output file overrides*
- *Step - Run Job with Overrides - Tables*
- *Step - Run Job with Overrides - Webhooks*
- *Step - Run Job with Parameter Overrides*
- *Step - Spark Job Overrides*
- *Step - Databricks Job Overrides*
 - *General example*
 - *Databricks job overrides reference*

This section describes how to run a job using the APIs available in Designer Cloud Powered by Trifacta® Enterprise Edition.

A note about API URLs:

In the listed examples, URLs are referenced in the following manner:

```
<protocol>://<platform_base_url>/
```

In your product, these map references map to the following:

```
<http or https>://<hostname>:<port_number>/
```

For more information, see *API Reference*.

Run Job Endpoints

Depending on the type of job that you are running, you must use one of the following endpoints:

Run job

Run a job to generate the outputs from a single recipe in a flow.

Tip: This method is covered on this page.

Endpoint	/v4/jobGroups/:id
----------	-------------------

Method	POST
Reference documentation	https://api.trifacta.com/ee/es.t/index.html#operation/runJobGroup

Run flow

Run all outputs specified in a flow. Optionally, you can run all scheduled outputs.

Endpoint	/v4/flows/:id/run
Method	POST
Reference documentation	https://api.trifacta.com/ee/es.t/index.html#operation/runFlow

Run deployment

Run the primary flow in the active release of the specified deployment.

Deployments are available only through the Deployment Manager. For more information, see *Overview of Deployment Manager*.

Endpoint	/v4/deployments/:id/run
Method	POST
Reference documentation	https://api.trifacta.com/ee/es.t/index.html#operation/runDeployment

Prerequisites

Before you begin, you should verify the following:

1. **Get authentication credentials.** As part of each request, you must pass in authentication credentials to the platform. For more information, see *Manage API Access Tokens*.

For more information, see <https://api.trifacta.com/ee/es.t/index.html#section/Authentication>

2. **Verify job execution.** Run the desired job through the Designer Cloud application and verify that the output objects are properly generated.

NOTE: By default, when scheduled or API jobs are executed, no validations are performed of any writesettings objects for file-based outputs. Issues with these objects may cause failures during transformation or publishing stages of job execution. Jobs of these types should be tested through the Designer Cloud application first. A workspace administrator can disable the skipping of these validations.

3. **Acquire recipe (wrangled dataset) identifier.** In Flow View, click the icon for the recipe whose outputs you wish to generate. Acquire the numeric value for the recipe from the URL. In the following, the recipe Id is 28629:

```
http://<platform_base_url>/flows/5479?recipe=28629&tab=recipe
```

4. **Create output object.** A recipe must have at least one output object created for it before you can run a job via APIs. For more information, see *Flow View Page*.

If you wish to apply overrides to the inputs or outputs of the recipe, you should acquire those identifiers or paths now. For more information, see "Run Job with Parameter Overrides" below.

Step - Run Job

Through the APIs, you can specify and run a job. To run a job with all default settings, construct a request like the following:

NOTE: A `wrangledDataset` is an internal object name for the recipe that you wish to run. Please see previous section for how to acquire this value.

Endpoint	<protocol>://<platform_base_url>/v4/jobGroups
Authentication	Required
Method	POST
Request Body	<pre>{ "wrangledDataset": { "id": 28629 } }</pre>
Response Code	201 - Created
Response Body	<pre>{ "sessionId": "79276c31-c58c-4e79-ae5e-fed1a25ebca1", "reason": "JobStarted", "jobGraph": { "vertices": [21, 22], "edges": [{ "source": 21, "target": 22 }] }, "id": 961247, "jobs": { "data": [{ "id": 21 }, { "id": 22 }] } }</pre>

If the 201 response code is returned, then the job has been queued for execution.

Tip: Retain the `id` value in the response. In the above, 961247 is the internal identifier for the job group for the job. You will need this value to check on your job status.

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/runJobGroup>

Checkpoint: You have queued your job for execution.

Step - Monitoring Your Job

You can monitor the status of your job through the following endpoint:

Endpoint	<protocol>://<platform_base_url>/v4/jobGroups/<id>/
Authentication	Required
Method	GET
Request Body	None.
Response Code	200 - Ok
Response Body	<pre>{ "id": 961247, "name": null, "description": null, "ranfrom": "ui", "ranfor": "recipe", "status": "Complete", "profilingEnabled": true, "runParameterReferenceDate": "2019-08-20T17:46:27.000Z", "createdAt": "2019-08-20T17:46:28.000Z", "updatedAt": "2019-08-20T17:53:17.000Z", "workspace": { "id": 22 }, "creator": { "id": 38 }, "updater": { "id": 38 }, "snapshot": { "id": 774476 }, "wrangledDataset": { "id": 28629 }, "flowRun": null }</pre>

When the job has successfully completed, the returned status message includes the following:

```
"status": "Complete",
```

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/getJobGroup>

Tip: You have executed the job. Results have been delivered to the designated output locations.

Step - Re-run Job

In the future, you can re-run the job using the same, simple request:

Endpoint	<protocol>://<platform_base_url>/v4/jobGroups
Authentication	Required
Method	POST
Request Body	<pre>{ "wrangledDataset": { "id": 28629 } }</pre>

The job is re-run as it was previously specified.

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/createJobGroup>

Step - Run Job with Overrides - Files

As needed, you can specify runtime overrides for any of the settings related to the job definition or its outputs. For file-based jobs, these overrides include:

- Data sources
- Execution environment
- profiling
- Output file, format, and other settings

Input file overrides

You can override the file-based data sources your job run. In the following example, two datasets are overridden with new files.

NOTE: Overrides for data sources apply only to file-based sources. File-based sources that are converted during ingestion, such as Microsoft Excel files, cannot be swapped in this manner.

NOTE: Overrides must be applied to the entire file path. As part of this overrides, you can redefine the bucket from which the source data is taken.

Endpoint	<protocol>://<platform_base_url>/v4/jobGroups
Authentication	Required
Method	POST
Request Body	<pre>{ "wrangledDataset": { "id": 28629 }, "overrides": { "datasources": { "airlines - region 1": ["s3://my-new-bucket/test-override-input/airlines3.csv", "s3://my-new-bucket/test-override-input/airlines4.csv", "s3://my-new-bucket/test-override-input/airlines5.csv"] } } }</pre>

```

    ],
    "airlines - region 2": [
      "s3://my-new-bucket/test-override-input/airlines1.csv",
    ]
  }
}

```

The job specified for recipe 28629 is re-run using the new data sources.

Notes:

- The names of the datasources (airlines - region 1 and airlines - region 2) refer to the display name values for the datasets that are the sources for the wrangledDataset (recipe) in the flow.
- You can use this API method to overwrite the bucket name for your source, but you must replace the entire path.
 - The parameterized list of files can be from different folders, too.
- File type and size information is not displayed in the Job Details page for these overridden jobs.
- No validation is performed on the existence of these files prior to execution. If the files do not exist, the job fails.

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/createJobGroup>

Output file overrides

NOTE: Override values applied to a job are not validated. Invalid overrides may cause your job to fail.

- See *API Task - Manage Outputs*.
 - See <https://api.trifacta.com/ee/es.t/index.html#operation/getWriteSetting>
1. Acquire the internal identifier for the recipe for which you wish to execute a job. In the previous example, this identifier was 28629.
 2. Construct a request using the following:

Endpoint	<protocol>://<platform_base_url>/v4/jobGroups
Authentication	Required
Method	POST

Request Body:

```

{
  "wrangledDataset": {
    "id": 28629
  },
  "overrides": {
    "profiler": true,
    "execution": "spark",
    "writesettings": [
      {
        "path": "<new_path_to_output>",
        "format": "csv",
        "header": true,
        "asSingleFile": true,
        "includeMismatches": true
      }
    ]
  }
}

```

```

    ]
  },
  "ranfrom": null
}

```

3. In the above example, the job has been launched with the following overrides:
- Job will be executed on the Spark cluster. Other supported values depend on your product edition and available running environments:

Value for <code>overrides.execution</code>	Description
photon	Running environment on Trifacta node
spark	Spark on integrated cluster, with the following exceptions.
databricksSpark	Spark on Azure Databricks
emrSpark	Spark on AWS EMR
dataflow	Dataflow

- Job will be executed with profiling enabled.
- Output is written to a new file path.
- Output format is CSV to the designated path.
- Output has a header and is generated as a single file.
- Output will include values if they are mismatched for the column's data type.

NOTE: `includeMismatches` is `false` by default. You can set it to `true` as an override or as part of the output object definition.

4. A response code of 201 - Created is returned. The response body should look like the following:

```

{
  "sessionId": "79276c31-c58c-4e79-ae5e-fed1a25ebca1",
  "reason": "JobStarted",
  "jobGraph": {
    "vertices": [
      21,
      22
    ],
    "edges": [
      {
        "source": 21,
        "target": 22
      }
    ]
  },
  "id": 962221,
  "jobs": {
    "data": [
      {
        "id": 21
      },
      {
        "id": 22
      }
    ]
  }
}

```

5. Retain the `id` value, which is the job identifier, for monitoring.

Step - Run Job with Overrides - Tables

You can also pass job definition overrides for table-based outputs. For table outputs, overrides include:

- Path to database to which to write (must have write access)
- Connection to write to the target.

Tip: This identifier is for the connection used to write to the target system. This connection must already exist. For more information on how to retrieve the identifier for a connection, see

<https://api.trifacta.com/ee/es.t/index.html#operation/listConnections>

- Name of output table
- Target table type

Tip: You can acquire the target type from the `vendor` value in the connection response. For more information, see

<https://api.trifacta.com/ee/es.t/index.html#operation/listConnections>

- action:

Key value	Description
create	Create a new table with each publication.
createAndLoad	Append your data to the table.
truncateAndLoad	Truncate the table and load it with your data.
dropAndLoad	Drop the table and write the new table in its place.

- Identifier of connection to use to write data.
- See *API Task - Manage Outputs*.
- See <https://api.trifacta.com/ee/es.t/index.html#operation/getPublication>

1. Acquire the internal identifier for the recipe for which you wish to execute a job. In the previous example, this identifier was 28629.
2. Construct a request using the following:

Endpoint	<protocol>://<platform_base_url>/v4/jobGroups
Authentication	Required
Method	POST

Request Body:

```
{
  "wrangledDataset": {
    "id": 28629
  },
  "overrides": {
    "publications": [
      {
        "path": [
          "prod_db"
        ],

```

```

        "tableName": "Table_CaseFctn2",
        "action": "createAndLoad",
        "targetType": "postgres",
        "connectionId": 3
      }
    ],
    "ranfrom": null
  }
}

```

3. In the above example, the job has been launched with the following overrides:

NOTE: When overrides are applied to publishing, any publications that are already attached to the recipe are ignored.

- a. Output path is to the `prod_db` database, using table name is `Table_CaseFctn2`.
 - b. Output action is "create and load." See above for definitions.
 - c. Target table type is a PostgreSQL table.
4. A response code of 201 - Created is returned. The response body should look like the following:

```

{
  "sessionId": "79276c31-c58c-4e79-ae5e-fed1a25ebca1",
  "reason": "JobStarted",
  "jobGraph": {
    "vertices": [
      21,
      22
    ],
    "edges": [
      {
        "source": 21,
        "target": 22
      }
    ]
  },
  "id": 962222,
  "jobs": {
    "data": [
      {
        "id": 21
      },
      {
        "id": 22
      }
    ]
  }
}

```

5. Retain the `id` value, which is the job identifier, for monitoring.

Step - Run Job with Overrides - Webhooks

When you execute a job, you can pass in a set of parameters as overrides to generate a webhook message to a third-party application, based on the success or failure of the job.

For more information on webhooks, see *Create Flow Webhook Task*.

1. Acquire the internal identifier for the recipe for which you wish to execute a job. In the previous example, this identifier was 28629.
2. Construct a request using the following:

Endpoint	<protocol>://<platform_base_url>/v4/jobGroups
Authentication	Required
Method	POST

Request Body:

```
{
  "wrangledDataset": {
    "id": 28629
  },
  "overrides": [
    {
      "webhooks": [
        {
          "name": "webhook override",
          "url": "http://example.com",
          "method": "post",
          "triggerEvent": "onJobFailure",
          "body": {
            "text": "override"
          },
          "headers": {
            "testHeader": "val1"
          },
          "sslVerification": true,
          "secretKey": "123"
        }
      ]
    }
  ]
}
```

3. In the above example, the job has been launched with the following overrides:

Override setting	Description
name	Name of the webhook.
url	URL to which to send the webhook message.
method	The HTTP method to use. Supported values: POST, PUT, PATCH, GET, or DELETE. Body is ignored for GET and DELETE methods.
triggerEvent	Supported values: onJobFailure - send webhook message if job fails onJobSuccess - send webhook message if job completes successfully onJobDone - send webhook message when job fails or finishes successfully
body	(optional) The value of the text field is the message that is sent. <div>NOTE: Some special token values are supported. See <i>Create Flow Webhook Task</i>.</div>
header	(optional) Key-value pairs of headers to include in the HTTP request.
sslVerification	(optional) Set to true if SSL verification should be completed. If not specified, the value is true.
secretKey	(optional) If enabled, this value should be set to the secret key to use.

4. A response code of 201 - Created is returned. The response body should look like the following:

```
{
  "sessionId": "79276c31-c58c-4e79-ae5e-fed1a25ebca1",
  "reason": "JobStarted",
  "jobGraph": {
    "vertices": [
      21,
      22
    ],
    "edges": [
      {
        "source": 21,
        "target": 22
      }
    ]
  },
  "id": 962222,
  "jobs": {
    "data": [
      {
        "id": 21
      },
      {
        "id": 22
      }
    ]
  }
}
```

5. Retain the `id` value, which is the job identifier, for monitoring.

Step - Run Job with Parameter Overrides

You can pass overrides of the default parameter values as part of the job definition. You can use the following mechanism to pass in parameter overrides of the following types:

- Datasets with parameters (variable type)
- Output object parameters
- Flow parameters

The syntax is the same for each type.

1. Acquire the internal identifier for the recipe for which you wish to execute a job. In the previous example, this identifier was 28629.
2. Construct a request using the following:

Endpoint	<protocol>://<platform_base_url>/v4/jobGroups
Authentication	Required
Method	POST

Request Body:

```
{
  "wrangledDataset": {
    "id": 28629
  },
  "overrides": {
    "runParameters": {
      "overrides": {
        "data": [
          {
            "key": "varRegion",
```

```

        "value": "02"
      }
    ]
  },
  "ranfrom": null
}

```

3. In the above example, the specified job has been launched for recipe `28629`. The run parameter `varRegion` has been set to `02` for this specific job. Depending on how it's defined in the flow, this parameter could influence change either of the following:
 - a. The source for the imported dataset.
 - b. The path for the generated output.
 - c. A flow parameter reference in the recipe
 - d. For more information, see *Overview of Parameterization*.
4. A response code of `201 - Created` is returned. The response body should look like the following:

```

{
  "sessionId": "79276c31-c58c-4e79-ae5e-fed1a25ebca1",
  "reason": "JobStarted",
  "jobGraph": {
    "vertices": [
      21,
      22
    ],
    "edges": [
      {
        "source": 21,
        "target": 22
      }
    ]
  },
  "id": 962223,
  "jobs": {
    "data": [
      {
        "id": 21
      },
      {
        "id": 22
      }
    ]
  }
}

```

5. Retain the `id` value, which is the job identifier, for monitoring.

Step - Spark Job Overrides

When it is enabled, you can submit overrides to a specific set of Spark properties for your job.

This feature and the Spark properties to override must be enabled. For more information on enabling this feature, see *Enable Spark Job Overrides*.

The following example, shows how to run a job for a specified recipe with Spark property overrides applied to it. This example assumes that the job has already been configured to be executed on Spark (`"execution": "spark"`):

Endpoint	<code><protocol>://<platform_base_url>/v4/jobGroups</code>
Authentication	Required

Method	POST
---------------	------

Request Body:

```
{
  "wrangledDataset": {
    "id": 28629
  },
  "overrides": {
    "sparkOptions": [
      {
        "key": "spark.executor.cores",
        "value": "2"
      },
      {
        "key": "spark.executor.memory",
        "value": "4GB"
      }
    ]
  }
}
```

Step - Databricks Job Overrides

You can submit overrides to a specific set of Databricks properties for your job execution. These overrides can be applied to AWS Databricks or Azure Databricks.

General example

The following example shows how to run a job on Databricks for a specified recipe with several property overrides applied to it:

Endpoint	https://www.example.com/v4/jobGroups
Authentication	Required
Method	POST

Request Body:

```
{
  "wrangledDataset": {
    "id": 60
  },
  "overrides": {
    "execution": "databricksSpark",
    "profiler": true,
    "databricksOptions": [
      { "key": "maxWorkers", "value": 8 },
      { "key": "poolId", "value": "pool-123456789" },
      { "key": "enableLocalDiskEncryption", "value": true }
    ]
  }
}
```

The above overrides do the following:

- Sets the maximum number of worker nodes on the cluster to 8. Databricks is permitted to adjust the number of nodes for job execution up to this limit.
- Instructs the Databricks cluster to use worker pool `pool-123456789` for the job.
- Enables encryption on the local Databricks cluster node of temporary job files for additional security.

Databricks job overrides reference

The following properties can be overridden for AWS Databricks and Azure Databricks jobs:

```
{
  "wrangledDataset": {"id": 60},
  "overrides": {
    "databricksOptions": [
      "autoterminationMinutes" : <integer_override_value>,
      "awsAttributes.availability" : "<string_override_value>",
      "awsAttributes.availabilityZone" : "<string_override_value>",
      "awsAttributes.ebsVolume.count" : <integer_override_value>,
      "awsAttributes.ebsVolume.size" : <integer_override_value>,
      "awsAttributes.ebsVolume.type" : "<string_override_value>",
      "awsAttributes.firstOnDemandInstances" : <integer_override_value>,
      "awsAttributes.instanceProfileArn" : "<string_override_value>",
      "awsAttributes.spotBidPricePercent" : <decimal_override_value>,
      "clusterMode" : "<string_override_value>",
      "clusterPolicyId" : "<string_override_value>",
      "driverNodeType" : "<string_override_value>",
      "enableAutotermination" : <boolean_override_value>,
      "enableLocalDiskEncryption" : <boolean_override_value>,
      "logsDestination" : "<string_override_value>",
      "maxWorkers" : <integer_override_value>,
      "minWorkers" : <integer_override_value>,
      "poolId" : "<string_override_value>",
      "poolName" : "<string_override_value>",
      "driverPoolId" : "<string_override_value>",
      "driverPoolName" : "<string_override_value>",
      "serviceUrl" : "<string_override_value>",
      "sparkVersion" : "<string_override_value>",
      "workerNodeType" : "<string_override_value>",
    ]
  }
}
```

NOTE: Overrides that begin with `awsAttributes` apply to AWS Databricks only.

NOTE: If a Databricks cluster policy is used, all job-level overrides except for `clusterPolicyId` are ignored.

For more information:

- *Configure for Azure Databricks*
- *Configure for AWS Databricks*

API Task - Run Job on Dataset with Parameters

Contents:

- Overview
 - Example Datasets
 - Step - Create Containing Flow
 - Step - Create Datasets with Parameters
 - Example 1 - Dataset with Datetime parameter
 - Example 2 - Dataset with Variable
 - Example 3 - Dataset with pattern parameter
 - Example 4 - Dataset with parameterized bucket name
 - Step - Wrangle Data
 - Step - Run Job
 - Example 1 - Dataset with Datetime parameter
 - Example 2 - Dataset with Variable
 - Example 3 - Dataset with pattern parameter
 - Example 4 - Dataset with parameterized bucket name
 - Step - Monitoring Your Job
 - Step - Re-run Job
-

Overview

This example task describes how to run jobs on datasets with parameters through Designer Cloud Powered by Trifacta® Enterprise Edition.

A **dataset with parameters** is a dataset in which some part of the path to the data objects has been parameterized. Since one or more of the parts of the path can vary, you can build a dataset with parameters to capture data that spans multiple files. For example, datasets with parameters can be used to parameterize serialized data by region or data or other variable. For more information on datasets with parameters, see *Overview of Parameterization*.

Basic Task

The basic method by which you build and run a job for a dataset with parameters is very similar to the non-parameterized dataset method with a few notable exceptions. The steps in this task follow the same steps for the standard task. Where the steps overlap links have been provided to the non-parameterized task. For more information, see *API Task - Develop a Flow*.

Example Datasets

This example covers three different datasets, each of which features a different type of dataset with parameters.

Example Number	Parameter Type	Description
1	Datetime parameter	In this example, a directory is used to store daily orders transactions. This dataset must be defined with a Datetime parameter to capture the preceding 7 days of data. Jobs can be configured to process all of this data as it appears in the directory.
2	Variable	This dataset segments data into four timezones across the US. These timezones are defined using the following text values in the path: <code>pacific</code> , <code>mountain</code> , <code>central</code> , and <code>eastern</code> . In this case, you can create a parameter called <code>region</code> , which can be overridden at runtime to be set to one of these four values during job execution.
3	Pattern parameter	

		This example is a directory containing point-of-sale transactions captured into individual files for each region. Since each region is defined by a numeric value (01, 02, 03), the dataset can be defined using a pattern parameter.
4	Environment parameter	An environment parameter is defined by an admin and is available for every user of the project or workspace. In particular, environment parameters are useful for defining source bucket names, which may vary between environments in the same organization.

Step - Create Containing Flow

You must create the flow to host your dataset with parameters.

In the response, you must capture and retain the flow Identifier. For more information, see *API Task - Develop a Flow*.

Step - Create Datasets with Parameters

NOTE: When you import a dataset with parameters, only the first matching dataset is used for the initial file. If you want to see data from other matching files, you must collect a new sample within the Transformer page.

Example 1 - Dataset with Datetime parameter

Suppose your files are stored in the following paths:

```
MyFiles/1/Datetime/2018-04-06-orders.csv
MyFiles/1/Datetime/2018-04-05-orders.csv
MyFiles/1/Datetime/2018-04-04-orders.csv
MyFiles/1/Datetime/2018-04-03-orders.csv
MyFiles/1/Datetime/2018-04-02-orders.csv
MyFiles/1/Datetime/2018-04-01-orders.csv
MyFiles/1/Datetime/2018-03-31-orders.csv
```

When you navigate to the directory through the application, you mouse over one of these files and select **Parameterize**.

In the window, select the date value (e.g. YYYY-MM-DD) and then click the Datetime icon.

Datetime Parameter:

- Format: YYYY-MM-DD
- Date Range: Date is last 7 days.
- Click **Save**.

The Datetime parameter should match with all files in the directory. Import this dataset and wrangle it.

After you wrangle the dataset, return to its flow view and select the recipe. You should be able to extract the flowId and recipelId values from the URL.

For purposes of this example, here are some key values:

- flowId: 35
- recipelId: 127

Example 2 - Dataset with Variable

Suppose your files are stored in the following paths:

```
MyFiles/1/variable/census-eastern.csv
MyFiles/1/variable/census-central.csv
MyFiles/1/variable/census-mountain.csv
MyFiles/1/variable/census-pacific.csv
```

When you navigate to the directory through the application, you mouse over one of these files and select **Parameterize**.

In the window, select the region value, which could be one of the following depending on the file: `eastern`, `central`, `mountain`, or `pacific`. Click the Variable icon.

Variable Parameter:

- Name: `region`
- Default Value: Set this default to `pacific`.
- Click **Save**.

In this case, the variable only matches one value in the directory. However, when you apply runtime overrides to the `region` variable, you can set it to any value.

Import this dataset and wrangle it.

After you wrangle the dataset, return to its flow view and select the recipe. You should be able to extract the `flowId` and `recipeId` values from the URL.

For purposes of this example, here are some key values:

- `flowId`: 33
- `recipeId`: 123

Example 3 - Dataset with pattern parameter

Suppose your files are stored in the following paths:

```
MyFiles/1/pattern/POS-r01.csv
MyFiles/1/pattern/POS-r02.csv
MyFiles/1/pattern/POS-r03.csv
```

When you navigate to the directory through the application, you mouse over one of these files and select **Parameterize**.

In the window, select the two numeric digits (e.g. `02`). Click the Pattern icon.

Pattern Parameter:

- Type: `Pattern`
- Matching regular expression: `{digit}{2}`
- Click **Save**.

In this case, the `Pattern` should match any sequence of two digits in a row. In the above example, this expression matches: `01`, `02`, and `03`, all of the files in the directory.

Import this dataset and wrangle it.

After you wrangle the dataset, return to its flow view and select the recipe. You should be able to extract the `flowId` and `recipeId` values from the URL.

For purposes of this example, here are some key values:

- flowId: 32
- recipeId: 121

Checkpoint: You have created flows for each type of dataset with parameters.

Example 4 - Dataset with parameterized bucket name

You can parameterize part or all of the bucket name in your source or target paths.

Suppose you have multiple workspaces that use different S3 buckets for sources of data. For example, your environments might look like the following:

Environment	S3 Bucket Name
Dev	myco-dev
Prod	myco-prod

For your datasources, you can parameterize the name of the bucket, so that if you migrate your flow between these environments, the references to datasources are updated based on the parameterized value for the bucket in the new environment.

Create environment parameter

Parameterized buckets are a good use for environment parameters. An **environment parameter** is a parameter that is available for use by every user in the project or workspace. In this case, the bucket name can be referenced for all datasets in the project or workspace, so turning that value into a parameter makes managing your datasources much more efficient.

You can use the following example to create environment parameter called `env.bucketName`, with a value of `myco-dev`. This environment parameter would be created in your Dev environment:

NOTE: The `overrideKey` value, which is the name of the environment parameter, must begin with `env..`

Endpoint	<code>http://www.example.com:3005/v4/environmentParameters</code>
Authentication	Required
Method	POST
Request Body	<pre>{ "overrideKey": "env.bucketName", "value": { "variable": { "value": "myco-dev" } } }</pre>
Response	<pre>{ "id": 1, "overrideKey": "env.bucketName", "value": { "variable": { "value": "myco-dev" } } }</pre>

```

    },
    "createdAt": "2021-06-24T14:15:22Z",
    "updatedAt": "2021-06-24T14:15:22Z",
    "deleted_at": "2021-06-24T14:15:22Z",
    "usageInfo": {
      "runParameters": 1
    }
  }
}

```

For more information on creating environment parameters, see <https://api.trifacta.com/ee/es.t/index.html#operation/createEnvironmentParameter>

Create dataset with parameterized bucket name

The following example creates an imported dataset with two parameters:

Parameter Name	Parameter Type	Environment Parameter?	Description
myPath	path	No	<p>The parameterized part of the path.</p> <p>The static value is /.</p> <p>The default value is /dummy.</p> <p>In this case, for the job run, the value is overridden with /dummy2.</p>
env. bucketName	bucket	Yes	<p>The parameterized part of the bucket path.</p> <p>The static value is myco-.</p> <p>In this case, for the job run, the value dev is inserted after the fifth character in the variable.</p>

Endpoint	http://www.example.com:3005/v4/importedDatasets
Authentication	Required
Method	POST
Request Body	<pre> { "name": "Dummy Dataset", "uri": "/path", "description": "My S3 parameterized dataset", "type": "S3", "isDynamic": true, "runParameters": [{ "type": "path", "overrideKey": "myPath", "insertionIndices": [{ "index": 1, "order": 0 }], "value": { "variable": { "value": "dummy2" } } }], } </pre>

```

{
  "type": "bucket",
  "overrideKey": "env.bucketParam",
  "insertionIndices": [
    {
      "index": 5,
      "order": 0
    }
  ],
  "value": {
    "variable": {
      "value": "dev"
    }
  }
},
{
  "dynamicBucket": "myco-",
  "dynamicPath": "/"
}

```

Response

```

{
  "visible": true,
  "numFlows": 0,
  "path": "/dummy",
  "bucket": "",
  "type": "s3",
  "isDynamic": true,
  "runParameters": [
    {
      "type": "path",
      "overrideKey": "myPath",
      "insertionIndices": [
        {
          "index": 1,
          "order": 0
        }
      ],
      "value": {
        "variable": {
          "value": "dummy2"
        }
      },
      "isEnvironmentParameter": false
    },
    {
      "type": "bucket",
      "overrideKey": "env.bucketParam",
      "insertionIndices": [
        {
          "index": 5,
          "order": 0
        }
      ],
      "value": {
        "variable": {
          "value": "dev"
        }
      },
      "isEnvironmentParameter": true
    }
  ],
  "dynamicBucket": "myco-",
  "dynamicPath": "/"
}

```

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/createImportedDataset>

Step - Wrangle Data

After you have created your dataset with parameter, you can wrangle it through the application. For more information, see *Transformer Page*.

Step - Run Job

Below, you can review the API calls to run a job for each type of dataset with parameters, including relevant information about overrides.

Example 1 - Dataset with Datetime parameter

NOTE: You cannot apply overrides to these types of datasets with parameters. The following request contains overrides for write settings but no overrides for parameters.

1. Endpoint	http://www.example.com:3005/v4/jobGroups
Authentication	Required
Method	POST
Request Body	<pre>{ "wrangledDataset": { "id": 127 }, "overrides": { "execution": "photon", "profiler": true, "writesettings": [{ "path": "MyFiles/queryResults/joe@example.com/2018-04-03-orders.csv", "action": "create", "format": "csv", "compression": "none", "header": false, "asSingleFile": false }] }, "runParameters": {} }</pre>

2. In the above example, the job has been launched for recipe 127 to execute on the Trifacta Photon running environment with profiling enabled.
 - a. Output format is CSV to the designated path. For more information on these properties, see <https://api.trifacta.com/ee/es.t/index.html#operation/runJobGroup>
 - b. Output is written as a new file with no overwriting of previous files.
3. A response code of 201 – Created is returned. The response body should look like the following:

```
{
  "sessionId": "79276c31-c58c-4e79-ae5e-fed1a25ebca1",
  "reason": "JobStarted",
  "jobGraph": {
    "vertices": [
      21,
      22
    ],
    "edges": [
```

```

        {
            "source": 21,
            "target": 22
        }
    ],
    "id": 29,
    "jobs": {
        "data": [
            {
                "id": 21
            },
            {
                "id": 22
            }
        ]
    }
}

```

4. Retain the jobId=29 value for monitoring.

Example 2 - Dataset with Variable

In the following example, the `region` variable has been overwritten with the value `central` to execute the job on `orders-central.csv`:

1. Endpoint	<code>http://www.example.com:3005/v4/jobGroups</code>
Authentication	Required
Method	POST
Request Body	<pre> { "wrangledDataset": { "id": 123 }, "overrides": { "execution": "photon", "profiler": true, "writesettings": [{ "path": "MyFiles/queryResults/joe@example.com/region-eastern.csv", "action": "create", "format": "csv", "compression": "none", "header": false, "asSingleFile": false }] }, "runParameters": { "overrides": { "data": [{ "key": "region", "value": "central" }] } } } </pre>

2. In the above example, the job has been launched for recipe 123 to execute on the Trifacta Photon running environment with profiling enabled.
 - a. Output format is CSV to the designated path. For more information on these properties, see <https://api.trifacta.com/ee/es.t/index.html#operation/runJobGroup>

- b. Output is written as a new file with no overwriting of previous files.
3. A response code of 201 – Created is returned. The response body should look like the following:

```
{
  "sessionId": "79276c31-c58c-4e79-ae5e-fed1a25ebca1",
  "reason": "JobStarted",
  "jobGraph": {
    "vertices": [
      21,
      22
    ],
    "edges": [
      {
        "source": 21,
        "target": 22
      }
    ]
  },
  "id": 27,
  "jobs": {
    "data": [
      {
        "id": 21
      },
      {
        "id": 22
      }
    ]
  }
}
```

4. Retain the jobgroupId=27 value for monitoring.

Example 3 - Dataset with pattern parameter

NOTE: You cannot apply overrides to these types of datasets with parameters. The following request contains overrides for write settings but no overrides for parameters.

1. Endpoint	http://www.example.com:3005/v4/jobGroups
Authentication	Required
Method	POST
Request Body	<pre>{ "wrangledDataset": { "id": 121 }, "overrides": { "execution": "photon", "profiler": false, "writesettings": [{ "path": "hdfs://hadoop:50070/trifacta/queryResults/admin@example.com/POS-r02.txt", "action": "create", "format": "csv", "compression": "none", "header": false, "asSingleFile": false }] } }</pre>

```
"runParameters": {}
}
```

2. In the above example, the job has been launched for recipe 121 to execute on the Trifacta Photon running environment with profiling enabled.
 - a. Output format is CSV to the designated path. For more information on these properties, see <https://api.trifacta.com/ee/es.t/index.html#operation/runJobGroup>
 - b. Output is written as a new file with no overwriting of previous files.
3. A response code of 201 – Created is returned. The response body should look like the following:

```
{
  "sessionId": "79276c31-c58c-4e79-ae5e-fed1a25ebca1",
  "reason": "JobStarted",
  "jobGraph": {
    "vertices": [
      21,
      22
    ],
    "edges": [
      {
        "source": 21,
        "target": 22
      }
    ]
  },
  "id": 28,
  "jobs": {
    "data": [
      {
        "id": 21
      },
      {
        "id": 22
      }
    ]
  }
}
```

4. Retain the jobId=28 value for monitoring.

Example 4 - Dataset with parameterized bucket name

The following example contains a parameterized bucket reference, with a specified override value. Administrators and project owners can specify the default value for environment parameters, and users can specify overrides for these values at job execution time.

Endpoint	http://www.example.com:3005/v4/jobGroups
Authentication	Required
Method	POST
Request Body	<pre>{ "wrangledDataset": { "id": 121 }, "runParameters": { "overrides": { "data": [{ "key": "env.bucketName", "value": "myco-dev2" }] } } }</pre>

```

    }
  }
}

```

In the above example, the job has been launched for recipe 121 to execute with the `env.bucketName` override value (`myco-dev2`) for the environment parameter.

For more information on these properties, see <https://api.trifacta.com/ee/es.t/index.html#operation/runJobGroup>

Step - Monitoring Your Job

After the job has been created and you have captured the `jobGroup` Id, you can use it to monitor the status of your job. For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/getJobGroup>

Step - Re-run Job

If you need to re-run the job as specified, you can use the `wrangledDataset` identifier to re-run the most recent job.

Tip: When you re-run a job, you can change any variable values as part of the request.

Example request:

Endpoint	<code>http://www.example.com:3005/v4/jobGroups</code>
Authentication	Required
Method	POST
Request Body	<pre> { "wrangledDataset": { "id": 123 }, "runParameters": { "overrides": { "data": [{ "key": "region", "value": "central" }] } } } </pre>

For more information, see *API Task - Develop a Flow*.

API Task - Run Plan

Contents:

- *Prerequisites*
 - *Step - Run Plan*
 - *Step - Run Plan with Overrides*
 - *Step - Monitoring Your Plan Run*
 - *Step - Add Flow Messages*
-

This section describes how to run a plan using the APIs available in Designer Cloud Powered by Trifacta® Enterprise Edition.

- A plan is a scheduled sequence of tasks based on a trigger that you define.
 - When a plan is executed via API, the request is the trigger, and the plan is executed immediately.
- Plans can be designed in the Designer Cloud application . For more information, see *Plans Page*.
- For more information on plans in general, see *Overview of Operationalization*.

A note about API URLs:

In the listed examples, URLs are referenced in the following manner:

```
<protocol>://<platform_base_url>/
```

In your product, these map references map to the following:

```
<http or https>://<hostname>:<port_number>/
```

For more information, see *API Reference*.

Prerequisites

Before you begin, you should verify the following:

1. **Get authentication credentials.** As part of each request, you must pass in authentication credentials to the platform.

Tip: The recommended method is to use an access token, which can be generated from the Designer Cloud application . For more information, see *Access Tokens Page*.

For more information, see <https://api.trifacta.com/ee/es.t/index.html#section/Authentication>

2. **Verify plan and its flows and outputs:**
 - a. You must create a plan first. See *Plan View Page*.
 - b. As part of creating that plan, you must verify that all referenced flows and output objects are properly defined and can be executed independently.

NOTE: In a flow, all recipes that you wish to have executed by the corresponding task must have a defined output object. For each output object, you must create at least one write settings or publication object. During plan runs, these objects are not validated, and tasks fail without them.

- c. Any applicable parameters are applied to the tasks at the time of execution. Parameter overrides are not supported in plans.
- d. See *Flow View Page*.
3. **Verify plan execution.** Run the desired plan through the Designer Cloud application and verify that the output objects are properly generated. See *Plan View Page*.
4. **Acquire plan identifier.** In Plan View, acquire the numeric value for the plan from the URL. In the following, the plan Id is 1234:

```
http://<platform_base_url>/plans/1234
```

Step - Run Plan

Through the APIs, you can run a plan. Construct a request like the following, where:

- `<id>` is the plan identifier that you already extracted from the Plan View URL.

Endpoint	<code><protocol>://<platform_base_url>/v4/plans/<id>/run</code>
Authentication	Required
Method	POST
Request Body	None.
Response Code	201 - Created
Response Body	<pre>{ "validationStatus": "Valid", "planSnapshotRunId": 2 }</pre>

If the 201 response code is returned, then the plan has been queued for execution.

Tip: Retain the `id` value in the response. In the above, 2 is the internal identifier for the plan run, which is referenced via the generated snapshot of the corresponding flows in the plan's tasks. You will need this value to check on your plan run status.

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/runPlan>

Checkpoint: You have queued your plan for execution.

Step - Run Plan with Overrides

When you run your plan, you can apply overrides to any of the parameters that are sourced in flow tasks within the plan. Overrides are applied in the body request when submitting to the plan run API endpoint.

Endpoint	<code><protocol>://<platform_base_url>/v4/plans/<id>/run</code>
Authentication	Required
Method	POST
Request Body	

Request Body	<pre>{ "planNodeOverrides": [{ "handle": "ax", "overrideKey": "region", "value": { "variable": { "value": "02" } } }, { "handle": "cq", "overrideKey": "state", "value": "CA" }] }</pre>
Response Code	201 - Created
Response Body	<pre>{ "validationStatus": "Valid", "planSnapshotRunId": 2 }</pre>

Request Body Attribute	Description
handle	<p>This value corresponds to the identifier for the task node in Plan View. In the Designer Cloud application ,</p> <p>Tasks are label in the following format:</p> <pre><task_type>-<handleId></pre> <p>where:</p> <ul style="list-style-type: none"> • <task_type> - is a string literal: <ul style="list-style-type: none"> • flowtask denotes a flow task. • http denotes an HTTP task. • slack denotes a Slack task. • delete denotes a Delete task. • <handleId> - a lowercase identifier for the task. Handle value must be two lowercase letters, at a minimum. Value must be unique to the tasks of the plan. This value is used as the handle value. <p>Tip: You can retrieve this value by selecting the task in Plan View, which is listed at the top of the task icon.</p>
override Key	The name of the parameter to override.
value	The override value to apply to the parameter. This value can be specified as a String value or as a JSON object. See the previous examples.

If the 201 response code is returned, then the plan has been queued for execution.

Tip: Retain the `id` value in the response. In the above, 2 is the internal identifier for the plan run, which is referenced via the generated snapshot of the corresponding flows in the plan's tasks. You will need this value to check on your plan run status.

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/runPlan>

Step - Monitoring Your Plan Run

You can monitor the status of your plan run through the following endpoint, where:

- `<id>` is the plan snapshot identifier for your run that you retained from the previous step.

Endpoint	<code><protocol>://<platform_base_url>/v4/planSnapshotRuns/<id></code>
Authentication	Required
Method	GET
Request Body	None.
Response Code	200 - Ok
Response Body	<pre>{ "id": 2, "status": "InProgress", "scheduleHistoryId": null, "startedAt": "2020-04-23T17:53:33.466Z", "finishedAt": null, "submittedAt": "2020-04-23T17:53:32.993Z", "executionId": null, "createdAt": "2020-04-23T17:53:33.312Z", "updatedAt": "2020-04-23T17:53:33.499Z", "plan": { "id": 1 } }</pre>

When the plan run has successfully completed, the returned status message includes the following:

```
"status": "Complete",
```

For more information, see <https://api.trifacta.com/ee/es.t/index.html#tag/PlanSnapshotRun> You can also review your plan runs through the Designer Cloud application at the following URL:

```
<protocol>://<platform_base_url>/plans/<planId>/runs/<planSnapshotRunId>
```

Tip: You have executed the plan run. Results have been delivered to the designated output locations.

Step - Add Flow Messages

You can configure flow webhooks and email notifications to deliver to stakeholders through the individual flows that are referenced in your plans.

NOTE: These features may require enablement and configuration in your environment.

For more information on these messaging types, see *Overview of Operationalization*.

A flow **webhook** is a REST API-based message that is triggered on the success or failure of generating an output from a flow. When the output referenced in a plan is generated, any webhook messages for the output are also triggered.

NOTE: You can define the equivalent of a webhook in your plan. HTTP tasks execute similar requests to a flow webhook and are an integrated part of plans. For more information, see *Create HTTP Task*.

Some uses:

- You can configure webhooks to deliver messages for each output referenced in the flow. Based on the schedule for your flow, you can review these messages to determine if the flow executed properly.
- You can configure a final output in the final task that is executed after the upstream recipes in the same flow.
 - All of the upstream recipes in the flow feed into a final recipe, which generates an unused output.
- When you create a flow webhook based on this final output, you can send a message that the final task has been executed.
 - For more information, see *Create Flow Webhook Task*.

An **email notification** is an email that is sent through the configured SMTP server to stakeholders based on the successful or failed execution of an output. You can define email notifications for your individual flows, and these messages get delivered as part of the flow execution that is part of the plan.

Tip: When an email notification is sent as part of task execution, the internal plan identifier is included as part of the message.

For more information on email notification, see *Manage Flow Notifications Dialog*.

API Task - Define Deployment Import Mappings

Contents:

- *Import Rules*
 - *Notes on import rules*
- *Import Rule Requirements*
- *Import Rule Types*
 - *Object Mapping Types*
 - *Value Mapping Types*
- *Examples*
 - *Example - Replace a connection*
 - *Example - Remap an HDFS location*
 - *Example - Remap an S3 location*
 - *Example - Remap a WASB location*
 - *Example - Remap an ADLS Gen2 location*
 - *Example - Remap an ADLS Gen1 location*
 - *Example - Remap a relational datasource*
- *Import Dry-Run*

Before you import a packaged flow into a Production environment, you may need to apply import rules to remap objects and locations from the source instance to the new instance. Import mapping rules are not required when importing into the same environment, although they may be helpful in some cases.

Tip: If you are importing a flow that references file-based sources and wish to use the original files in your imported file, you may find it easier to configure the importing user's permissions to access the appropriate directories of the sources and then to swap datasets as needed after you complete the import. This method is suitable and easier to do across a fewer number of flows.

NOTE: Import mapping rules apply to deployments in a Production instance under deployment management. You cannot apply import mapping rules between two Dev instances.

NOTE: Import mapping rules do not apply to any SQL referenced in your flows or datasets. In addition to remapping the connection identifiers, you should store your database names and table names as environment parameters, which apply to the entire workspace. These variables can be exported from one workspace and imported into another, where they can be updated to the correct value for the workspace. For more information, see *Overview of Parameterization*.

NOTE: Import mapping rules require the use of the APIs made available from the Designer Cloud powered by Trifacta® platform . API usage is considered a developer-level skill.

- For more information on creating an export package, see *Export Flow*.
- For more information on how to import, see *Import Flow*.

You can apply the following types of remappings:

Type	Description
Value	For value remappings, you can specify rules to match on specific values or patterns of values in the import package and remap those values for use in the new instance.

	<p>NOTE: In this release, value remapping is supported only for S3 bucket names and paths to imported datasets and output locations. Examples are provided below.</p>
Object	<p>For object remappings, you can specify rules to match a value listed in the import package and remap that value to a defined object in the new instance.</p> <p>NOTE: In this release, object remapping is supported only for connections. An example is provided below.</p>

Import Rules

When a flow is imported, references in the flow definition that apply in the source instance may not apply in the target instance. For example, the location paths to the source datasets may need to be rewritten to point to a different location in the target instance.

Before you import your flow definition, you need to define rules for any value or object remapping that must be done in the target environment.

Notes on import rules

1. Value and object remapping rules should be completed before you import the flow. The flow may be non-functional until the rules are applied.

Tip: After you create your import rules, you can perform via API a dry run of the import. Any errors are reported in the response. Details are provided below.

2. Value and object remapping rules are applied at the time of import. If you add new rules, they are not retroactively applied to release packages that have already been imported.
3. When changing rules:
 - a. Any previously applied rules to the same import object are deleted.
 - b. You can apply multiple rules in the same change.
 - c. Rules are applied in the order in which they are listed in the request. Rules listed later in the request must be compatible with expected changes applied by the earlier rules.
4. Value and object remapping must be completed via API. API usage is considered a developer-level skill. Examples are provided below.

NOTE: Import mapping rules do not work for parameterized datasets. If the imported dataset with parameters is still accessible, you should be able to run jobs from it.

Import Rule Requirements

- If you are importing into the same instance from which you exported (Dev/Test/Prod on the same instance):
 - Import rules are not required.
 - If you want to use a different source of data in your Prod flow, you must create import rules.
- If you are importing into a different instance from which you exported (Dev and Prod on different instances):
 - Import rules are required, except in unusual cases.

Import Rule Types

The following types of rules can be applied to import mappings.

NOTE: Depending on the type of mapping, some of these rules may not apply. Please be sure to review the Examples below.

Object Mapping Types

Type	Description
tableName	Set this value to <code>connections</code> . You must then specify the <code>uuid</code> of the connection identifier in the imported flow and replace it with the internal identifier of the connection in the importing instance.

Value Mapping Types

Type	Description
fileLocation	This type is used to remap paths to files. <div>NOTE: <code>filelocation</code> rules apply to both input and output paths. Paths and their rules should be defined with care.</div>
s3Bucket	(AWS) Name of the S3 to remap.
dbName	(relational source) Name of the table to remap.
dbPath	(relational source) Path to the database table. This value is an array.
host	(Azure) Depending on the Azure datastore, this rule replaces: <ul style="list-style-type: none">WASB: blobhost nameADLS Gen2: storage accountADLS Gen1: datastore in the datalake
userinfo	(Azure) Depending on the Azure datastore, this rule replaces: <ul style="list-style-type: none">WASB: container nameADLS Gen2: filesystem name

Examples

The following are some example import rules to address specific uses.

Example - Replace a connection

In this following example, you must remap the connection from the source instance of the platform to the corresponding connection in the instance where you are importing.

First, you must be able to uniquely identify the connection from the source that you wish to remap.

- While the connection Id may work in a limited scope, that identifier is unlikely to be unique within your environment.
- If you do know the connect Id from the source system, you can skip the first step below.

In the API response in a connection definition, you can acquire the `uuid` value for the connection, which is a unique identifier for the connection object across all instances of the platform:

Item	v4 APIs
API Endpoint	From the source instance: <div><code>/v4/connections</code></div>

Method	GET
Request Body	None.
Response Body	<pre> { "data": [{ "connectParams": { "vendor": "redshift", "vendorName": "redshift", "host": "redshift.example.com", "port": "5439", "extraLoadParams": "BLANKSASNULL EMPTYASNULL TRIMBLANKS TRUNCATECOLUMNS", "defaultDatabase": "test" }, "id": 2, "host": "redshift.example.com", "port": 5439, "vendor": "redshift", "params": { "extraLoadParams": "BLANKSASNULL EMPTYASNULL TRIMBLANKS TRUNCATECOLUMNS", "defaultDatabase": "test" }, "ssl": false, "vendorName": "redshift", "name": "redshift", "description": null, "type": "jdbc", "isGlobal": true, "credentialType": "iamRoleArn", "credentialsShared": true, "uuid": "097c2300-2f6a-11e9-a585-57562e0d9cd6", "disableTypeInference": false, "createdAt": "2019-02-13T08:33:28.368Z", "updatedAt": "2019-02-13T08:33:28.381Z", "credentials": [{ "iamRoleArn": "arn:aws:iam:something", "username": "UserName" }], "creator": { "id": 1 }, "updater": { "id": 1 }, "workspace": { "id": 1 } }, { "connectParams": { "vendor": "hive", "vendorName": "hive", "host": "hadoop", "port": "10000", "jdbc": "hive2", "defaultDatabase": "default" }, "id": 1, "host": "hadoop", "port": 10000, "vendor": "hive", "params": { </pre>

	<pre> "jdbc": "hive2", "connectStringOptions": "", "defaultDatabase": "default" }, "ssl": false, "vendorName": "hive", "name": "hive", "description": null, "type": "jdbc", "isGlobal": true, "credentialType": "conf", "credentialsShared": true, "uuid": "08a1a180-2f6a-11e9-b2b2-85d2b0b67f5e", "disableTypeInference": false, "createdAt": "2019-02-13T08:33:26.936Z", "updatedAt": "2019-02-13T08:33:26.952Z", "credentials": [], "creator": { "id": 1 }, "updater": { "id": 1 }, "workspace": { "id": 1 } }, { "count": 2 }] }</pre>
Documentation	See https://api.trifacta.com/ee/es.t/index.html#operation/getConnection

In the above, you identify that the connection used for the exported flow is the Redshift one. This object has the following unique identifier:

```
"uuid": "097c2300-2f6a-11e9-a585-57562e0d9cd6"
```

In the target system, you must now create a rule in the deployment into which you are importing that searches for this unique value. In the following example:

- The deploymentId is known to be 4.
- The connectionId for the equivalent Redshift connection in the target system is 1.

The `uuid` field in the import package is searched for the matching string. If it is found, the connection in the import package is replaced with the connection in the target system with an Id of 1:

Item	v4 APIs
API Endpoint	<code>/v4/deployments/4/objectImportRules</code>
Method	PATCH
Request Body	<pre> [{ "tableName": "connections", "onCondition": { "uuid": "097c2300-2f6a-11e9-a585-57562e0d9cd6" }, "withCondition": { </pre>

	<pre> "id": 1 } }] </pre>
Status Code - Success	200 – OK
Response Body	<p>When the new rules are applied, all previously existing rules for the object in the deployment are deleted. The response body contains any rules that have been deleted as part of this request.</p> <p>In the following example, there were no rules, so nothing was deleted:</p> <pre> { "deleted": { "data": [] } } </pre>
Documentation	See https://api.trifacta.com/ee/es.t/index.html#operation/updateObjectImportRules

To test your rule, perform a dry run of the import. See below.

Example - Remap an HDFS location

In this example, your import rule must remap the path to the source from your Dev paths to your Prod paths. Suppose the pattern looks like this:

Dev Path	hdfs://datasets/dev/1/164e0bca-8c91-4e3c-9d0a-2a85eedec817/myData.csv
Prod Path	hdfs://datasets/prod/1/164e0bca-8c91-4e3c-9d0a-2a85eedec817/myData-Prod.csv

Note the differences:

- The `/dev/` part of the path has been replaced by `/prod/`.
- The filename is different.

You can use the following value import rules to change the path values. In the following example, the rules are applied separately.

NOTE: You can specify multiple rules in a single request. Rules are applied in the order that they are listed. Latter rules must factor the results of earlier rules.

Request:

Item	v4 APIs
API Endpoint	<code>/v4/deployments/4/valueImportRules</code>
Method	PATCH

Request Body:

```
[
  { "type": "fileLocation", "on": "\\dev\\", "with": "/prod/" },
  { "type": "fileLocation", "on": "\\([a-zA-Z0-9_]*).csv", "with": "$1-Prod.csv" }
]
```

Response:

Item	v4 APIs
Status Code - Success	200 - OK
Response Body	<p>When the new rules are applied, all previously existing rules for the object in the deployment are deleted. The response body contains any rules that have been deleted as part of this request.</p> <p>In the following example, there were no rules, so nothing was deleted:</p> <pre>{ "deleted": { "data": [] } }</pre>
Documentation	See https://api.trifacta.com/ee/es.t/index.html#operation/updateValueImportRules

To test your rule, perform a dry run of the import. See below.

Example - Remap an S3 location

For S3 sources, you can apply remapping rules including changing to a new S3 bucket.

In this example, your import rule must remap the path to the source from your Dev paths to your Prod paths. Suppose the pattern looks like this:

Dev S3 Bucket Name	wrangle-dev
Dev Path	/projs/tweets/v04/tweets_month.csv
Prod S3 Bucket Name	wrangle-prod
Prod Path	/tweets/tweets_month.csv

You can use the following value import rules to change the bucket name and path values.

NOTE: You can specify multiple rules in a single request. Rules are applied in the order that they are listed. Latter rules must factor the results of earlier rules.

s3Bucket name rule: This rule replaces the name of the S3 bucket to use with the new one: wrangle-prod.

fileLocation rule: This rule uses regular expressions to match each segment of the path in the bucket's paths.

- Files are located at a consistent depth in the source bucket.
- Path segments and filename use only alphanumeric values and underscores (_).
- The replacement path is shortened to contain only the parent name (\$2) and the filename (\$4) in the path.
- This rule applies to both input and output object file paths.

Request:

Item	v4 APIs
API Endpoint	<code>/v4/deployments/4/valueImportRules</code>
Method	PATCH

Request Body:

```
[
  {
    "type": "s3Bucket", "on": "wrangle-dev", "with": "wrangle-prod",
    {
      "type": "fileLocation", "on": "\\([a-zA-Z0-9_]*\\)\\([a-zA-Z0-9_]*\\)\\([a-zA-Z0-9_]*\\)\\.csv/", "
      with": "$2/$4.csv"
    }
  }
]
```

Response:

Item	v4 APIs
Status Code - Success	200 - OK
Response Body	<p>When the new rules are applied, all previously existing rules for the object in the deployment are deleted. The response body contains any rules that have been deleted as part of this request.</p> <p>In the following example, there were no rules, so nothing was deleted:</p> <pre>{ "deleted": { "data": [] } }</pre>
Documentation	See https://api.trifacta.com/ee/es.t/index.html#operation/updateValueImportRules

To test your rule, perform a dry run of the import. See below.

Example - Remap a WASB location

For WASB sources, you can apply remapping rules during import.

In this example, your import rule must remap the blob host, container, and file location:

Dev Blobhost	<code>storage-wasb-account-dev.blob.core.windows.net</code>
Dev Container	<code>container-dev</code>
Dev File Location	<code>/projs/work/orders.csv</code>
Prod Blobhost	<code>storage-wasb-account-prod.blob.core.windows.net</code>
Prod Container	<code>container-prod</code>

Prod File Location	/2003/transactions/orders.csv
---------------------------	-------------------------------

You can use the following value import rules to change the blobhost, container, and file paths.

NOTE: You can specify multiple rules in a single request. Rules are applied in the order that they are listed. Latter rules must factor the results of earlier rules.

host rule: This rule replaces the blobhost name to use with the new one: `storage-wasb-account-prod.blob.core.windows.net`.

userinfo rule: This rule replaces the container name to use with the new one: `container-prod`.

fileLocation rule: This rule performs a text substitution to replace the file path. This rule applies to both input and output object file paths.

Request:

Item	v4 APIs
API Endpoint	<code>/v4/deployments/4/valueImportRules</code>
Method	PATCH

Request Body:

```
[
  {
    "type": "host", "on": "storage-wasb-account-dev.blob.core.windows.net", "with": "storage-wasb-account-prod.blob.core.windows.net",
  },
  {
    "type": "userinfo", "on": "container-dev", "with": "container-prod",
  },
  {
    "type": "fileLocation", "on": "/projs/work/orders.csv", "with": "/2003/transactions/orders.csv"
  }
]
```

Response:

Item	v4 APIs
Status Code - Success	200 - OK
Response Body	<p>When the new rules are applied, all previously existing rules for the object in the deployment are deleted. The response body contains any rules that have been deleted as part of this request.</p> <p>In the following example, there were no rules, so nothing was deleted:</p> <pre>{ "deleted": { "data": [] } }</pre>
Documentation	See https://api.trifacta.com/ee/es.t/index.html#operation/updateValueImportRules

To test your rule, perform a dry run of the import. See below.

Example - Remap an ADLS Gen2 location

For ADLS Gen2 sources, you can apply remapping rules during import.

In this example, your import rule must remap the storage account, filesystem, and file location:

Dev Storage Account	storage-adlsgen2-account-dev.blob.core.windows.net
Dev Filesystem	filesystem-dev
Dev File Location	/projs/work/orders.csv
Prod Storage Account	storage-adlsgen2-account-prod.blob.core.windows.net
Prod Filesystem	filesystem-prod
Prod File Location	/2003/transactions/orders.csv

You can use the following value import rules to change the storage account, filesystem, and file paths.

NOTE: You can specify multiple rules in a single request. Rules are applied in the order that they are listed. Latter rules must factor the results of earlier rules.

host rule: This rule replaces the storage account name to use with the new one: storage-adlsgen2-account-prod.blob.core.windows.net.

userinfo rule: This rule replaces the filesystem name to use with the new one: filesystem-prod.

fileLocation rule: This rule performs a text substitution to replace the file path. This rule applies to both input and output object file paths.

Request:

Item	v4 APIs
API Endpoint	<code>/v4/deployments/4/valueImportRules</code>
Method	PATCH

Request Body:

```
[
  {
    "type": "host",
    "on": "storage-adlsgen2-account-dev.blob.core.windows.net",
    "with": "storage-adlsgen2-account-prod.blob.core.windows.net"
  },
  {
    "type": "userinfo",
    "on": "filesystem-dev",
    "with": "filesystem-prod"
  },
  {
    "type": "fileLocation",
    "on": "/projs/work/orders.csv",
    "with": "/2003/transactions/orders.csv"
  }
]
```

Response:

Item	v4 APIs

Status Code - Success	200 - OK
Response Body	<p>When the new rules are applied, all previously existing rules for the object in the deployment are deleted. The response body contains any rules that have been deleted as part of this request.</p> <p>In the following example, there were no rules, so nothing was deleted:</p> <pre>{ "deleted": { "data": [] } }</pre>
Documentation	See https://api.trifacta.com/ee/es.t/index.html#operation/updateValueImportRules

To test your rule, perform a dry run of the import. See below.

Example - Remap an ADLS Gen1 location

For ADLS Gen1 sources, you can apply remapping rules during import.

In this example, your import rule must remap the Azure data lake store and file location:

Dev data store	adl://storage-adlsgen1-account.azuredatalakestore.net
Dev File Location	/projs/work/orders.csv
Prod data store	adl://storage-adlsgen1-account-prod.azuredatalakestore.net
Prod File Location	/2003/transactions/orders.csv

You can use the following value import rules to change the datastore and file paths.

NOTE: You can specify multiple rules in a single request. Rules are applied in the order that they are listed. Latter rules must factor the results of earlier rules.

host rule: This rule replaces the datastore name to use with the new one: `storage-adlsgen1-account-prod.azuredatalakestore.net`.

fileLocation rule: This rule performs a text substitution to replace the file path. This rule applies to both input and output object file paths.

Request:

Item	v4 APIs
API Endpoint	<code>/v4/deployments/4/valueImportRules</code>
Method	PATCH

Request Body:

```
[
  { "type": "host", "on": "storage-adlsgen1-account-dev.azuredatalakestore.net", "with": "storage-adlsgen1-account-prod.azuredatalakestore.net" },
```



```
{ "type": "fileLocation", "on": "/projs/work/orders.csv", "with": "/2003/transactions/orders.csv" }
]
```

Response:

Item	v4 APIs
Status Code - Success	200 - OK
Response Body	<p>When the new rules are applied, all previously existing rules for the object in the deployment are deleted. The response body contains any rules that have been deleted as part of this request.</p> <p>In the following example, there were no rules, so nothing was deleted:</p> <pre>{ "deleted": { "data": [] } }</pre>
Documentation	See https://api.trifacta.com/ee/es.t/index.html#operation/updateValueImportRules

To test your rule, perform a dry run of the import. See below.

Example - Remap a relational datasource

When you migrate a relational source from a Dev instance to a Prod instance, you may need to remap your flow to use the production database and table.

NOTE: These rules can be applied to sources or publications of a flow.

In this example, you are replacing the input and output source databases and tables with the corresponding production DB values.

Item	Dev value	Prod value
Table name 1	dev_trans	prod_trans
Path value 1	dev_db2_src	prod_db2_src
Table name 2	dev_trans_out	prod_trans_out
Path value 2	dev_db2_out	prod_db2_out

In a single request, you can apply the rules changes to map the above Dev values to the Prod values.

NOTE: You can specify multiple rules in a single request. Rules are applied in the order that they are listed. Latter rules must factor the results of earlier rules.

The `on` parameter accepts regular expressions. In the following example request, the `on` parameter has been configured to use a regular expression, under the assumption that all current and future imports will respect the current pattern or database paths and table names.

dbTableName rule: This rule replaces the name of the table to use.

dbPath rule: This rule replaces the path value to database table.

NOTE: The content of a dataset or output `dbPath` is an array. The regular expression for `on` is applied to every element in the `dbPath` value. Typically, there's only one element in the `dbPath` array. In some cases, there may be multiple elements, so be careful when specifying the `on` value.

Request:

Item	v4 APIs
API Endpoint	<code>/v4/deployments/4/valueImportRules</code>
Method	PATCH

Request Body:

```
[
  {
    "type": "dbTableName",
    "on": "/dev_([a-zA-Z0-9_]*)/",
    "with": "prod_$1",
  },
  {
    "type": "dbPath",
    "on": "/dev_([a-zA-Z0-9_]*)_src/",
    "with": "prod_$1_out"
  }
]
```

Response:

Item	v4 APIs
Status Code - Success	200 - OK
Response Body	<p>When the new rules are applied, all previously existing rules for the object in the deployment are deleted. The response body contains any rules that have been deleted as part of this request.</p> <p>In the following example, there were no rules, so nothing was deleted:</p> <pre>{ "deleted": { "data": [] } }</pre>
Documentation	See https://api.trifacta.com/ee/es.t/index.html#operation/updateValueImportRules

To test your rule, perform a dry run of the import. See below.

Import Dry-Run

After you have specified a set of import rules, you can perform a dry-run of an import of an import package. This dry-run does not perform the actual import but does report any permissions errors or other issues in the response.

In this example, the `flow2import.zip` file contains the package to import into deployment 4.

Request:

Item	v4 APIs				
API Endpoint	<div>/v4/deployments/4/releases/dryRun</div>				
Method	POST				
Request Body	<div>In form data submitted with the request, you must include the following key-value pair:<table><tr><th>Key</th><th>Value</th></tr><tr><td>data</td><td>"@flow2import.zip"</td></tr></table></div>	Key	Value	data	"@flow2import.zip"
Key	Value				
data	"@flow2import.zip"				

Response:

Item	v4 APIs
Status Code - Success	200 - OK
Response Body	The response body contains any import remapping rules that have been applied during the import process.
Documentation	See https://api.trifacta.com/ee/es.t/index.html#operation/importPackageForDeploymentDryRun

After the above dry-run has been executed, the import package can be imported and is automatically connected to the appropriate connection. See <https://api.trifacta.com/ee/es.t/index.html#operation/importPackageForDeployment>

API Task - Run Deployment

Contents:

- *Run Job Endpoints*
- *Prerequisites*
- *Step - Run Deployment*
- *Step - Monitoring Your Deployment Job*
- *Step - Run Deployment with Overrides*
 - *Acquire the active outputs for the deployment*
 - *Acquire output object information*
 - *Run deployment with overrides*
- *Step - Run Deployment with Spark Overrides*

This section describes how to run a deployment using the APIs available in Designer Cloud Powered by Trifacta® Enterprise Edition.

- A **deployment** is a packaging mechanism for versioning your production-level flows.
- Deployments are created and managed through a separate interface.
- For more information, see *Overview of Deployment Manager*.

A note about API URLs:

In the listed examples, URLs are referenced in the following manner:

```
<protocol>://<platform_base_url>/
```

In your product, these map references map to the following:

```
<http or https>://<hostname>:<port_number>/
```

For more information, see *API Reference*.

Run Job Endpoints

Depending on the type of job that you are running, you must use one of the following endpoints:

Run job:

Run a job to generate the outputs from a single recipe in a flow.

Endpoint	/v4/jobGroups/:id
Method	POST
Reference documentation	https://api.trifacta.com/ee/es.t/index.html#operation/runJobGroup

Run flow:

Run all outputs specified in a flow. Optionally, you can run all scheduled outputs.

Endpoint	/v4/flows/:id/run

Method	POST
Reference documentation	https://api.trifacta.com/ee/es.t/index.html#operation/runFlow

Run deployment:

Run the primary flow in the active release of the specified deployment.

Tip: This method is covered on this page.

Deployments are available only through the Deployment Manager. For more information, see *Overview of Deployment Manager*.

Endpoint	/v4/deployments/:id/run
Method	POST
Reference documentation	https://api.trifacta.com/ee/es.t/index.html#operation/runDeployment

Prerequisites

Before you begin, you should verify the following:

1. **Get authentication credentials.** As part of each request, you must pass in authentication credentials to the Designer Cloud powered by Trifacta platform .

Tip: The recommended method is to use an access token, which can be generated from the Designer Cloud application . For more information, see *Access Tokens Page*.

See <https://api.trifacta.com/ee/es.t/index.html#operation/Authentication>

2. **Develop your flow.** Before you deploy a flow through the Deployment Manager, you should build and test your flow in a development environment. See *API Task - Develop a Flow*.
3. **Deploy your flow.** After you have developed a flow, you can deploy it. See *API Task - Deploy a Flow*.
4. **Acquire deployment identifier.** In Deployment Manager, acquire the numeric value for the deployment. See *Deployment Manager Page*.

Step - Run Deployment

Through the APIs, you can run a deployment. When a deployment is executed:

- All of the recipes that are included in the active release of the deployment are executed.
- All default values for outputs are applied.

In the following example, the deployment identifier is 2.

Endpoint	<protocol>://<platform_base_url>/v4/deployments/2/run
Authentication	Required
Method	POST
Request Body	None.
Response Code	201 - Created
Response Body	{ "data": [

```

{
  "reason": "JobStarted",
  "sessionId": "dd6a90e0-c353-11e7-ad4e-7f2dd2ae4621",
  "id": 33
}
]
}

```

If the 201 response code is returned, then the deployment job has been queued for execution.

Tip: Retain the `id` value in the response. In the above, 961247 is the internal identifier for the job group for the job. You will need this value to check on your job status.

For more information, see *API Task - Deploy a Flow*.

Checkpoint: You have queued your deployment job for execution.

Step - Monitoring Your Deployment Job

You can monitor the status of your deploy job through the following endpoint using the `id` value that was returned in the previous step:

```
<protocol>://<platform_base_url>/v4/jobGroups/<id>/status
```

For more information, see *API Task - Run Job*.

Step - Run Deployment with Overrides

When you run a deployment, you cannot apply overrides to the request. However, you can complete the following steps to apply overrides when you execute the jobs within the active release. In this task, you run jobs on the specific recipes of the active deployment, applying overrides as needed.

Suppose you are running the jobs for deployment id 2, and you want to apply some overrides.

NOTE: A deployment can trigger several different jobs within a single flow. In the following example, it is assumed that there is only one output.

Acquire the active outputs for the deployment

The first step is to acquire all of the active outputs for the deployment.

Endpoint	<code><protocol>://<platform_base_url>/v4/deployments/2/activeoutputs</code>
Authentication	Required
Method	GET
Request Body	None.

Response Code	200 - OK
Response Body	<pre>{ "data": [{ "outputObjectId": 6, "flowNodeId": 27, "recipeName": "USDA_Farmers_Market_2014" }] }</pre>

in the above response:

- The `flowNodeId` value corresponds to the recipe (wrangledDataset) identifier that you wish to modify.
- The `outputObjectId` value corresponds to the output object that is produced by default from the recipe.

Acquire output object information

The next step is to review the output object to determine what needs to be overridden. Since you are overriding a file-based publication, you can query directly for the `writeSettings` objects associated with the output object:

Endpoint	<protocol>://<platform_base_url>/v4/outputObjects/6/writeSettings
Authentication	Required
Method	GET
Request Body	None.
Response Code	200 - OK
Response Body	<pre>{ "data": [{ "delim": ",", "id": 17, "path": "hdfs://hadoop:50070/example/joe@example.com/USDA_Farmers_Market_2014.csv", "action": "create", "format": "csv", "compression": "none", "header": true, "asSingleFile": true, "prefix": null, "suffix": "_increment", "hasQuotes": true, "createdAt": "2019-11-05T18:26:31.972Z", "updatedAt": "2019-11-05T18:30:56.756Z", "creator": { "id": 2 }, "updater": { "id": 2 }, "outputObject": { "id": 6 } }, { "delim": ",", "id": 16, "path": "hdfs://hadoop:50070/example/joe@example.com/USDA_Farmers_Market_2014." }] }</pre>

```

json",
    "action": "create",
    "format": "json",
    "compression": "none",
    "header": false,
    "asSingleFile": false,
    "prefix": null,
    "suffix": "_increment",
    "hasQuotes": false,
    "createdAt": "2019-11-05T18:26:44.983Z",
    "updatedAt": "2019-11-05T18:30:56.743Z",
    "creator": {
      "id": 2
    },
    "updater": {
      "id": 2
    },
    "outputObject": {
      "id": 6
    }
  }
]
}

```

Run deployment with overrides

Now that you have access to the outputs generated by the deployment, you decide to override the following for each file:

- Filename:
 - Remove the year information at the end of the filename
 - Store in a separate folder called `final`
- Wipe the table and reload it each time (`action=overwrite`)
- Disable profiling

From the `activeOutputs` endpoint, you retrieved the `flowNodeId` (27).

Endpoint	<protocol>://<platform_base_url>/v4/jobGroups/
Authentication	Required
Method	POST
Request Body	<pre> { "wrangledDataset": { "id": 27 }, "overrides": { "profiler": false, "writesettings": [{ "path": "hdfs://hadoop:50070/example/joe@example.com/final/USDA_Farmers_Market. csv", "format": "csv", "action": "overwrite" }, { "path": "hdfs://hadoop:50070/example/joe@example.com/final/USDA_Farmers_Market. json", "format": "json", "action": "overwrite" }] } } </pre>

	<pre> }, "ranfrom": null } </pre>
Response Code	201 - Created
Response Body	<pre> { "sessionId": "b29467c3-fc59-499e-aed6-d797746a86eb", "reason": "JobStarted", "jobGraph": { "vertices": [10, 11, 12], "edges": [{ "source": 10, "target": 11 }, { "source": 10, "target": 12 }] }, "id": 4, "jobs": { "data": [{ "id": 10 }, { "id": 11 }, { "id": 12 }] } } </pre>

Checkpoint: Your job with overrides has been queued for execution.

You can use the job identifier (4) to monitor the job status.

Step - Run Deployment with Spark Overrides

When you run a deployment, you can specify override values to the Spark properties that have been made available for overrides.

NOTE: These overrides only apply if you are running the job on Spark and if the feature has been enabled in your deployment.

- This feature and the properties to override must be enabled. See *Enable Spark Job Overrides*.

- All of the recipes that are included in the active release of the deployment are executed.

All other default values are applied. In the following example, the deployment identifier is 2.

Endpoint	<protocol>://<platform_base_url>/v4/deployments/2/run
Authentication	Required
Method	POST
Request Body	<pre>{ "sparkOptions": [{ "key": "spark.executor.cores", "value": "2" }, { "key": "spark.executor.memory", "value": "4GB" }] }</pre>
Response Code	201 - Created
Response Body	<pre>{ "data": [{ "reason": "JobStarted", "sessionId": "dd6a90e0-c353-11e7-ad4e-7f2dd2ae4621", "id": 33 }] }</pre>

If the 201 response code is returned, then the deployment job has been queued for execution.

Checkpoint: You have queued your deployment job for execution.

API Task - Publish Results

Contents:

- Overview
 - Basic Task
 - Step - Create Connections
 - Redshift connection
 - Hive connection
 - Tableau Server connection
 - SQL DW connection
 - Step - Run Job
 - Step - Publish Results to Hive
 - Step - Publish Results to Tableau Server
 - Step - Publish Results to SQL DW
 - Step - Publish Results to Redshift
 - Step - Publish Results with Overrides
-

Overview

After you have run a job to generate results, you can publish those results to different targets as needed. This section describes how to automate those publishing steps through the APIs.

NOTE: This task applies to re-publishing job results after you have already generated them.

NOTE: After you have generated results and written them to one target, you cannot publish to the same target. You must configure the outputs to specify a different format and location and then run a new job.

In the application, you can publish after generating results. See *Publishing Dialog*.

Basic Task

1. Create connections to each target to which you wish to publish. Connections must support write operations.
2. Specify a job whose output meets the requirements for the target.
3. Run the job.
4. When the job completes, publish the results to the target(s).

Step - Create Connections

For each target, you must have access to create a connection to it. After a connection is created, it can be reused, so you may find it easier to create them through the application.

- Some connections can be created via API. For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/createConnection>
- Other connections must be created through the application. Links to instructions are provided below.

NOTE: Connections created through the application must be created through the Connections page, which is used for creating read/write connections. Do not create these connections through the Import Data page. See *Connections Page*.

Redshift connection

- **Required Output Format:** Avro
- **Example Id:** 2
- **Create via API:** N
- **Doc Link:** *Amazon Redshift Connections*

Other Requirements:

- Requires S3 set as the base storage layer. See *Set Base Storage Layer*.

Hive connection

- **Required Output Format:** Avro
- **Example Id:** 1
- **Create via API:** Y
- **Doc Link:** *Hive Connections*
- **Other Requirements:**
 - Requires integration with a Hadoop cluster.

Tableau Server connection

- **Required Output Format:** HYPER
- **Example Id:** 3
- **Create via API:** Y
- **Doc Link:** *Tableau Server Connections*
- **Other Requirements:**
 - None.

SQL DW connection

- **Required Output Format:** Parquet
- **Example Id:** 4
- **Create via API:** N
- **Doc Link:** *Microsoft SQL Data Warehouse Connections*
- **Other Requirements:**
 - Available only on Azure deployments. See *Configure for Azure*.

Step - Run Job

Before you publish results to a different datastore, you must generate results and store them in HDFS.

NOTE: To produce some output formats, you must run the job on the Spark running environment.

In the examples below, the following example data is assumed:

Identifier	Value
jobId	2
flowId	3

For more information on running a job, see <https://api.trifacta.com/ee/es.t/index.html#operation/runJobGroup>
For more information on the publishing endpoint, see <https://api.trifacta.com/ee/es.t/index.html#operation/publishJobGroup>

Step - Publish Results to Hive

The following uses the Avro results from the specified job (jobId = 2) to publish the results to the `test_table` table in the `default` Hive schema through `connectionId=1`.

NOTE: To publish to Hive, the targeted database is predefined in the connection object. For the `path` value in the request body, you must specify the schema in this database to use. Schema information is not available through API. To explore the available schemas, click the Hive icon in the Import Data page. The schemas are the first level of listed objects. For more information, see *Import Data Page*.

Request:

Endpoint	<code>http://www.wrangle-dev.example.com:3005/v4/jobGroups/2/publish</code>
Authentication	Required
Method	PUT
Request Body	<pre>{ "connection": { "id": 1 }, "path": ["default"], "table": "test_table", "action": "create", "inputFormat": "avro" }</pre>

Response:

Status Code	200 - OK
Response Body	<pre>{ "jobgroupId": 2, "reason": "JobStarted", "sessionId": "24862060-4fcd-11e8-8622-fda0fbf6f550" }</pre>

Step - Publish Results to Tableau Server

The following uses the HYPER results from the specified job (jobId = 2) to publish the results to the `test_table3` table in the `default` Tableau Server database through `connectionId=3`.

Request:

Endpoint	<code>http://www.wrangle-dev.example.com:3005/v4/jobGroups/2/publish</code>
Authentication	Required
Method	PUT
Request Body	

Request Body	<pre>{ "connection": { "id": 3 }, "path": ["default"], "table": "test_table3", "action": "createAndLoad", "inputFormat": "hyper" }</pre>
---------------------	--

Response:

Status Code	200 - OK
Response Body	<pre>{ "jobgroupId": 2, "reason": "JobStarted", "sessionId": "24862060-4fcd-11e8-8622-fda0fbf6f552" }</pre>

Step - Publish Results to SQL DW

The following uses the Parquet results from the specified job (jobId = 2) to publish the results to the `test_table4` table in the `dbo` SQL DW database through `connectionId=4`.

Request:

Endpoint	<code>http://www.wrangle-dev.example.com:3005/v4/jobGroups/2/publish</code>
Authentication	Required
Method	PUT
Request Body	<pre>{ "connection": { "id": 4 }, "path": ["dbo"], "table": "test_table4", "action": "createAndLoad", "inputFormat": "pgt" }</pre>

Response:

Status Code	200 - OK
Response Body	<pre>{ "jobgroupId": 2, "jobIds": 22, "reason": "JobStarted", "sessionId": "855f83a0-dc94-11e8-bd1a-f998d808020d" }</pre>

Step - Publish Results to Redshift

The following uses the Avro results from the specified job (jobId = 2) to publish the results to the `test_table2` table in the `public` Redshift schema through `connectionId=2`.

NOTE: To publish to Redshift, the targeted database is predefined in the connection object. For the `path` value in the request body, you must specify the schema in this database to use. Schema information is not available through API. To explore the available schemas, click the Redshift icon in the Import Data page. The schemas are the first level of listed objects. For more information, see *Import Data Page*.

Request:

Endpoint	<code>http://www.wrangle-dev.example.com:3005/v4/jobGroups/2/publish</code>
Authentication	Required
Method	PUT
Request Body	<pre>{ "connection": { "id": 2 }, "path": ["public"], "table": "test_table2", "action": "create", "inputFormat": "avro" }</pre>

Response:

Status Code	200 - OK
Response Body	<pre>{ "jobgroupId": 2, "reason": "JobStarted", "sessionId": "fae64760-4fc4-11e8-8cba-0987061e4e16" }</pre>

Step - Publish Results with Overrides

When you are publishing results to a relational source, you can apply overrides to the job to redirect the output or change the action applied to the target table. For more information, see *API Task - Run Job*.

API Task - Swap Datasets

Contents:

- Overview
 - Example Datasets
 - Assumptions
- Step - Import Dataset
- Step - Swap Dataset from Recipe
- Step - Rerun Job
- Step - Monitor Your Job
- Step - Schedule Your Job

Overview

After you have created a flow, imported a dataset, and created a recipe for that dataset, you may need to swap in a different dataset and run the recipe against that one. This task steps through that process via the APIs.

NOTE: If you are processing multiple parallel datasources in a single job, you should create a dataset with parameters and then run the job. For more information, see *API Task - Run Job on Dataset with Parameters*.

This task utilizes the following methods:

1. **Creating an imported dataset.** After the new file has been added to the backend datastore, you can import into Designer Cloud Powered by Trifacta® Enterprise Edition as an imported dataset.
2. **Swap dataset.** Using the ID of the imported dataset you created, you can now assign the dataset to the recipe in your flow.
3. **Run a job.** Run the job against the dataset.
4. **Monitor progress.** Monitor the progress of the job until it is complete.

Example Datasets

In this example, you are wrangling data from orders placed in different regions on a quarterly basis. When a new file drops, you want to be able to swap out the current dataset that is assigned to the recipe and swap in the new one. Then, run the job.

Example Files:

The following files are stored in HDFS:

Path and Filename	Description
hdfs:///user/orders/MyCo-orders-west-Q1.txt	Orders from West region for Q1
hdfs:///user/orders/MyCo-orders-west-Q2.txt	Orders from West region for Q2
hdfs:///user/orders/MyCo-orders-north-Q1.txt	Orders from North region for Q1
hdfs:///user/orders/MyCo-orders-north-Q2.txt	Orders from North region for Q2
hdfs:///user/orders/MyCo-orders-east-Q1.txt	Orders from East region for Q1
hdfs:///user/orders/MyCo-orders-east-Q2.txt	Orders from East region for Q2

Assumptions

You have already created a flow, which contains the following imported dataset and recipe:

NOTE: When an imported dataset is created via API, it is always imported as an unstructured dataset. Any recipe that references this dataset should contain initial parsing steps required to structure the data.

Tip: Through the UI, you can import one of your datasets as unstructured. Create a recipe for this dataset and then edit it. In the Recipe panel, you should be able to see the structuring steps. Back in Flow View, you can chain your structural recipe off of this one. Dataset swapping should happen on the first recipe.

Object Type	Name	Id
flow	MyCo-Orders-Quarter	2
Imported Dataset	MyCo-orders-west-Q1.txt	8
Recipe (wrangledDataset)	n/a	9
Job	n/a	3

Base URL:

For purposes of this example, the base URL for the platform is the following:

```
http://www.example.com:3005
```

Step - Import Dataset

NOTE: You cannot add datasets to the flow through the `flows` endpoint. Moving pre-existing datasets into a flow is not supported in this release. Create or locate the flow first and then when you create the datasets, associate them with the flow at the time of creation.

- See <https://api.trifacta.com/ee/es.t/index.html#operation/createImportedDataset>
- See <https://api.trifacta.com/ee/es.t/index.html#operation/createWrangledDataset>

NOTE: When an imported dataset is created via API, it is always imported as an unstructured dataset. Any recipe that references this dataset should contain initial parsing steps required to structure the data.

The following steps describe how to create an imported dataset and assign it to the flow that has already been created (`flowId=2`).

Steps:

1. To create an imported dataset, you must acquire the following information about the source.
 - a. path
 - b. type
 - c. name
 - d. description

- e. bucket (if a file stored on S3)
2. In this example, the file you are importing is `MyCo-orders-west-Q2.txt`. Since the files are similar in nature and are stored in the same directory, you can acquire this information by gathering the information from the imported dataset that is already part of the flow. Execute the following:

Endpoint	http://www.example.com:3005/v4/importedDatasets
Authentication	Required
Method	POST
Request Body	<pre>{ "path": "hdfs:///user/orders/MyCo-orders-west-Q2.txt", "name": "MyCo-orders-west-Q2.txt", "description": "MyCo-orders-west-Q2" }</pre>

3. The response should be a 201 - Created status code with something like the following:

```
{
  "id": 12,
  "size": "281032",
  "path": "hdfs:///user/orders/MyCo-orders-west-Q2.txt",
  "dynamicPath": null,
  "workspaceId": 1,
  "isSchematized": false,
  "isDynamic": false,
  "disableTypeInference": false,
  "createdAt": "2018-10-29T23:15:01.831Z",
  "updatedAt": "2018-10-29T23:15:01.889Z",
  "parsingRecipe": {
    "id": 11
  },
  "runParameters": [],
  "name": "MyCo-orders-west-Q2.txt.txt",
  "description": "MyCo-orders-west-Q2.txt",
  "creator": {
    "id": 1
  },
  "updater": {
    "id": 1
  },
  "connection": null
}
```

4. You must retain the `id` value so you can reference it when you create the recipe.
5. See <https://api.trifacta.com/ee/es.t/index.html#operation/createImportedDataset>

Checkpoint: You have imported a dataset that is unstructured and is not associated with any flow.

Step - Swap Dataset from Recipe

The next step is to swap the primary input dataset for the recipe to point at the newly imported dataset. This step automatically adds the imported dataset to the flow and drops the previous imported dataset from the flow.

1. Use the following to swap the primary input dataset for the recipe:

Endpoint	http://www.example.com:3005/v4/wrangledDatasets/9
----------	---

	/primaryInputDataset
Authentication	Required
Method	PUT
Request Body	<pre>{ "importedDataset": { "id": 12 } }</pre>

2. The response should be a 200 – OK status code with something like the following:

```
{
  "id": 9,
  "wrangled": true,
  "createdAt": "2019-03-03T17:58:53.979Z",
  "updatedAt": "2019-03-03T18:01:11.310Z",
  "recipe": {
    "id": 9,
    "name": "POS-r01",
    "description": null,
    "active": true,
    "nextPortId": 1,
    "createdAt": "2019-03-03T17:58:53.965Z",
    "updatedAt": "2019-03-03T18:01:11.308Z",
    "currentEdit": {
      "id": 8
    },
    "redoLeafEdit": {
      "id": 7
    },
    "creator": {
      "id": 1
    },
    "updater": {
      "id": 1
    }
  },
  "referenceInfo": null,
  "activeSample": {
    "id": 7
  },
  "creator": {
    "id": 1
  },
  "updater": {
    "id": 1
  },
  "referencedFlowNode": null,
  "flow": {
    "id": 2
  }
}
```

3. The new imported dataset is now the primary input for the recipe, and the old imported dataset has been removed from the flow.

<https://api.trifacta.com/ee/es.t/index.html#operation/updateInputDataset>

Step - Rerun Job

To execute a job on this recipe, you can simply re-run any job that was executed on the old imported dataset, since you reference the job by jobId and wrangledDataset (recipe) Id.

Endpoint	http://www.example.com:3005/v4/jobGroups
Authentication	Required
Method	POST
Request Body	<pre>{ "wrangledDataset": { "id": 9 } }</pre>

The job is re-run as it was previously specified.

If you need to modify any job parameters, you must create a new job definition.

Step - Monitor Your Job

After the job has been queued, you can track it to completion. See *API Task - Develop a Flow*.

Step - Schedule Your Job

When you are satisfied with how your flow is working, you can set up periodic schedules using a third-party tool to execute the job on a regular basis.

The tool must hit the above endpoints to swap in the new dataset and run the job.

API Task - Manage Outputs

Contents:

- Overview
 - Basic Task
 - Variations
- Step - Get Recipe ID
- Step - Create outputObject
- Step - Run a Test Job
- Step - Create writeSettings Object
- Step - Get Connection ID for Publication
- Step - Create a Publication
- Step - Apply Overrides
- Step - Apply Spark Job Overrides

Overview

Through the APIs, you can separately manage the outputs associated with an individual recipe. This task describes how to create output objects, which are associated with your recipe, and how to publish those outputs to different datastores in varying formats. You can continue to modify the output objects and their related write settings and publications independently of managing the wrangling process. Whenever you need new results, you can reference the wrangled dataset with which your outputs have been associated, and the job is executed and published in the appropriate manner to your targets.

Relevant terms:

Term	Description
outputObjects	An outputObject is a definition of one or more types of outputs and how they are generated. It must be associated with a recipe. <div>NOTE: An outputObject must be created for a recipe before you can run a job on it. One and only one outputObject can be associated with a recipe.</div>
writeSettings	A writeSettings object defines file-based outputs within an outputObject. Settings include path, format, compression, and delimiters.
publications	A publications object is used to specify a table-based output and is associated with an outputObject. Settings include the connection to use, path, table type, and write action to apply.

NOTE: If you need to make changes for purposes of a specific job run, you can add overrides to the request for the job. These overrides apply only for the current job. For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/runJobGroup>

Basic Task

Here's the basic task described in this section.

1. Get the internal identifier for the recipe for which you are building outputs.
2. Create the outputObject for the recipe.
3. Create a writeSettings object and associate it with the outputObject.
4. Run a test job, if desired.
5. For any publication, get the internal identifier for the connection to use.
6. Create a publication object and associate it with the outputObject.

7. Run your job.

Variations

If you are generating exclusively file-based or relational outputs, you can vary this task in the following ways:

For file-based outputs:

1. Get the internal identifier for the recipe for which you are building outputs.
2. Create the outputObject for the recipe.
3. Create a writeSettings object and associate it with the outputObject.
4. Run your job.

For relational outputs:

1. Get the internal identifier for the recipe for which you are building outputs.
2. Create the outputObject for the recipe.
3. For any publication, get the internal identifier for the connection to use.
4. Create a publication object and associate it with the outputObject.
5. Run your job.

Step - Get Recipe ID

To begin, you need the internal identifier for the recipe.

NOTE: In the APIs, a recipe is identified by its internal name, a **wrangled dataset**.

Request:

Endpoint	http://www.wrangle-dev.example.com:3005/v4/wrangledDatasets
Authentication	Required
Method	GET
Request Body	None.

Response:

Status Code	200 - OK
Response Body	<pre>{ "data": [{ "id": 11, "wrangled": true, "createdAt": "2018-11-12T23:06:36.473Z", "updatedAt": "2018-11-12T23:06:36.539Z", "recipe": { "id": 10 }, "name": "POS-r01", "description": null, "referenceInfo": null, "activeSample": { "id": 11 }, "creator": { "id": 1 }, "updater": {</pre>

```

      "id": 1
    },
    "flow": {
      "id": 4
    }
  },
  {
    "id": 1,
    "wrangled": true,
    "createdAt": "2018-11-12T23:19:57.650Z",
    "updatedAt": "2018-11-12T23:20:47.297Z",
    "recipe": {
      "id": 19
    },
    "name": "member_info",
    "description": null,
    "referenceInfo": null,
    "activeSample": {
      "id": 20
    },
    "creator": {
      "id": 1
    },
    "updater": {
      "id": 1
    },
    "flow": {
      "id": 6
    }
  }
]
}

```

cURL example:

```

curl -X GET \
  http://www.wrangle-dev.example.com:3005/v4/wrangledDatasets \
  -H 'authorization: Basic <auth_token>' \
  -H 'cache-control: no-cache'

```

Relevant terms:

Term	Description
URL	URL and method to execute.
authorization	Authorization taken to pass to the platform. Basic authorization works. <div>NOTE: This token must be passed with each request to the platform.</div>
cache-control	Cache control setting.
content-type	HTTP content type to send. These applications use <code>application/json</code> .

Checkpoint: In the above, let's assume that the recipe identifier of interest is `wrangledDataset=11`. This means that the flow where it is hosted is `flow.id=4`. Retain this information for later.

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/getWrangledDataset>

Step - Create outputObject

Create the outputObject and associate it with the recipe identifier. In the following request, the wrangledDataset identifier that you retrieved in the previous call is applied as the flowNodeId value.

The following example includes an embedded writeSettings object, which generates a CSV file output. You can remove this embedded object if desired, but you must create a writeSettings object before you can generate an output.

Request:

Endpoint	http://www.wrangle-dev.example.com:3005/v4/outputObjects
Authentication	Required
Method	POST
Request Body	<pre>{ "execution": "photon", "profiler": true, "isAdhoc": true, "writeSettings": { "data": [{ "delim": ",", "path": "hdfs://hadoop:50070/trifacta/queryResults/admin@example.com/POS_01. avro", "action": "create", "format": "avro", "compression": "none", "header": false, "asSingleFile": false, "prefix": null, "suffix": "_increment", "includeMismatches": true, "hasQuotes": false }] }, "flowNode": { "id": 11 } }</pre>

Response:

Status Code	201 - Created
Response Body	<pre>{ "id": 4, "execution": "photon", "profiler": true, "isAdhoc": true, "updatedAt": "2018-11-13T00:20:49.258Z", "createdAt": "2018-11-13T00:20:49.258Z", "creator": { "id": 1 }, "updater": { "id": 1 }, "flowNode": { "id": 11 } }</pre>


```
}  
}
```

cURL example:

```
curl -X POST \  
  http://www.wrangle-dev.example.com/v4/outputObjects \  
  -H 'authorization: Basic <auth_token>' \  
  -H 'cache-control: no-cache' \  
  -H 'content-type: application/json' \  
  -d '{  
    "execution": "photon",  
    "profiler": true,  
    "isAdhoc": true,  
    "writeSettings": {  
      "data": [  
        {  
          "delim": ",",  
          "path": "hdfs://hadoop:50070/trifacta/queryResults/admin@example.com/POS_01.avro",  
          "action": "create",  
          "format": "avro",  
          "compression": "none",  
          "header": false,  
          "asSingleFile": false,  
          "prefix": null,  
          "suffix": "_increment",  
          "includeMismatches": true,  
          "hasQuotes": false  
        }  
      ]  
    },  
    "flowNode": {  
      "id": 11  
    }  
  }'
```

Relevant terms:

Term	Description
URL	URL and method to execute.
authorization	Authorization taken to pass to the platform. Basic authorization works. <div>NOTE: This token must be passed with each request to the platform.</div>
cache-control	Cache control setting.
content-type	HTTP content type to send. These applications use <code>application/json</code> .

Checkpoint: You've created an `outputObject` (`id=4`) and an embedded `writeSettings` object and have associated them with the appropriate recipe `flowNodeId=11`. You can now run a job for this recipe generating the specified output.

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/createOutputObject>

Step - Run a Test Job

Now that outputs have been defined for the recipe, you can just execute a job on the specified recipe `flowNodeId=11`:

Request:

Endpoint	<code>http://www.wrangle-dev.example.com:3005/v4/jobGroups</code>
Authentication	Required
Method	POST
Request Body	<pre>{ "wrangledDataset": { "id": 11 } }</pre>

Response:

Status Code	201 - Created
Response Body	<pre>{ "sessionId": "79276c31-c58c-4e79-ae5e-fed1a25ebca1", "reason": "JobStarted", "jobGraph": { "vertices": [21, 22], "edges": [{ "source": 21, "target": 22 }] }, "id": 2, "jobs": { "data": [{ "id": 21 }, { "id": 22 }] } }</pre>

NOTE: To re-run the job against its currently specified outputs, writeSettings, and publications, you only need the recipe ID. If you need to make changes for purposes of a specific job run, you can add overrides to the request for the job. These overrides apply only for the current job. For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/runJobGroup>

To track the status of the job:

- You can monitor the progress through the application.
- You can monitor progress through the `status` field by querying the specific job. For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/getJobGroup>

Checkpoint: You've run a job, generating one output in Avro format.

Step - Create writeSettings Object

Suppose you want to create another file-based output for this `outputObject`. You can create a second `writeSettings` object, which publishes the results of the job run on the recipe to the specified location.

The following example creates settings for generating a parquet-based output.

Request:

Endpoint	<code>http://www.wrangle-dev.example.com:3005/v4/writeSettings/</code>
Authentication	Required
Method	POST
Request Body	<pre>{ "delim": ",", "path": "hdfs://hadoop:50070/trifacta/queryResults/admin@example.com/POS_r03.pqt", "action": "create", "format": "pqt", "compression": "none", "header": false, "asSingleFile": false, "prefix": null, "suffix": "_increment", "hasQuotes": false, "outputObjectId": 4 }</pre>

Response:

Status Code	201 - Created
Response Body	<pre>{ "delim": ",", "id": 2, "path": "hdfs://hadoop:50070/trifacta/queryResults/admin@example.com/POS_r03.pqt", "action": "create", "format": "pqt", "compression": "none", "header": false, "asSingleFile": false, "prefix": null, "suffix": "_increment", "hasQuotes": false, "updatedAt": "2018-11-13T01:07:52.386Z", "createdAt": "2018-11-13T01:07:52.386Z", "creator": { "id": 1 }, "updater": { "id": 1 }, "outputObject": {</pre>

```
    "id": 4
  }
}
```

cURL example:

```
curl -X POST \
  http://www.wrangle-dev.example.com/v4/writeSettings \
  -H 'authorization: Basic <auth_token>' \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -d '{
    "delim": ",",
    "path": "hdfs://hadoop:50070/trifacta/queryResults/admin@example.com/POS_r03.pqt",
    "action": "create",
    "format": "pqt",
    "compression": "none",
    "header": false,
    "asSingleFile": false,
    "prefix": null,
    "suffix": "_increment",
    "hasQuotes": false,
    "outputObject": {
      "id": 4
    }
  }
```

Relevant terms:

Term	Description
URL	URL and method to execute.
authorization	Authorization token to pass to the platform. Basic authorization works. <div>NOTE: This token must be passed with each request to the platform.</div>
cache-control	Cache control setting.
content-type	HTTP content type to send. These applications use <code>application/json</code> .

Checkpoint: You've added a new `writeSettings` object and associated it with your `outputObject` (`id=4`). When you run the job again, the Parquet output is also generated.

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/createWriteSetting>

Step - Get Connection ID for Publication

To generate a publication, you must identify the connection through which you are publishing the results.

Below, the request returns a single connection to Hive (`id=1`).

Request:

Endpoint	<code>http://www.wrangle-dev.example.com:3005/v4/connections</code>
Authentication	Required

Method	GET
Request Body	None.

Response:

Status Code	200 - OK
Response Body	<pre>{ "data": [{ "id": 1, "host": "hadoop", "port": 10000, "vendor": "hive", "params": { "jdbc": "hive2", "connectStringOptions": "", "defaultDatabase": "default" }, "ssl": false, "vendorName": "hive", "name": "hive", "description": null, "type": "jdbc", "isGlobal": true, "credentialType": "conf", "credentialsShared": true, "uuid": "28415970-e6c4-11e8-82be-9947a31ecdd5", "disableTypeInference": false, "createdAt": "2018-11-12T21:44:39.816Z", "updatedAt": "2018-11-12T21:44:39.842Z", "credentials": [], "creator": { "id": 1 }, "updater": { "id": 1 }, "workspace": { "id": 1 } }], "count": 1 }</pre>

cURL example:

```
curl -X GET \
  http://www.wrangle-dev.example.com/v4/connections \
  -H 'authorization: Basic <auth_token>' \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json'
```

Relevant terms:

Term	Description
URL	URL and method to execute.
authorization	Authorization taken to pass to the platform. Basic authorization works.

	NOTE: This token must be passed with each request to the platform.
cache-control	Cache control setting.
content-type	HTTP content type to send. These applications use application/json.

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/listConnections>

Step - Create a Publication

Example - Hive:

You can create publications that publish table-based outputs through specified connections. In the following, a Hive table is written out to the `default` database through `connectionId = 1`. This publication is associated with the `outputObject id=4`.

Request:

Endpoint	<code>http://www.wrangle-dev.example.com:3005/v4/publications</code>
Authentication	Required
Method	POST
Request Body	<pre>{ "path": ["default"], "tableName": "myPublishedHiveTable", "targetType": "hive", "action": "create", "outputObject": { "id": 4 }, "connection": { "id": 1 } }</pre>

Response:

Status Code	201 - Created
Response Body	<pre>{ "path": ["default"], "id": 3, "tableName": "myPublishedHiveTable", "targetType": "hive", "action": "create", "updatedAt": "2018-11-13T01:25:39.698Z", "createdAt": "2018-11-13T01:25:39.698Z", "creator": { "id": 1 }, "updater": { "id": 1 }, "outputObject": { "id": 4 }, }</pre>

```

"connection": {
  "id": 1
}

```

cURL example:

```

curl -X POST \
  http://example.com:3005/v4/publications \
  -H 'authorization: Basic <auth_token>' \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -d '{
    "path": [
      "default"
    ],
    "tableName": "myPublishedHiveTable",
    "targetType": "hive",
    "action": "create",
    "outputObject": {
      "id": 4
    },
    "connection": {
      "id": 1
    }
  }'

```

Relevant terms:

Term	Description
URL	URL and method to execute.
authorization	Authorization taken to pass to the platform. Basic authorization works. NOTE: This token must be passed with each request to the platform.
cache-control	Cache control setting.
content-type	HTTP content type to send. These applications use <code>application/json</code> .

Example - Snowflake:

In the following, a Snowflake table is updated through `connectionId = 4` using an `upsert` action. This action updates the values in the table based on matching column values between source and the corresponding columns in the target table. When these key values match, the columns that you specify in the request are updated with values from the source data. Additional options are listed under Notes below.

This publication is associated with the `outputObject id=6`.

Request:

Endpoint	<code>http://www.wrangle-dev.example.com:3005/v4/publications</code>
Authentication	Required
Method	POST
Request Body	

```
{
  "tableName": "snowflake_table_merge",
  "path": [
    "tri-dev",
    "demo_test"
  ],
  "targetType": "snowflake",
  "connectionId": 4,
  "action": "upsert",
  "outputObjectId": 6,
  "runParameters": [],
  "parameters": {
    "mergeJoinKeys": "[\"colA\"]",
    "colsToUpdate": "[\"colB\", \"colC\"]",
    "deleteUnmatchedRowsInTarget": "false",
    "externalTableLocation": null,
    "insertChecked": "true",
    "isDeltaTable": "false",
    "isExternalTable": "false",
    "matchedRowsAction": "update"
  }
}
```

Response:

Status Code	201 - Created
Response Body	<pre>{ "path": ["default"], "id": 3, "tableName": "snowflake_table_merge", "targetType": "snowflake", "action": "upsert", "updatedAt": "2018-11-13T01:25:39.698Z", "createdAt": "2018-11-13T01:25:39.698Z", "creator": { "id": 1 }, "updater": { "id": 1 }, "outputObject": { "id": 6 }, "connection": { "id": 4 } }</pre>

Notes:

Property	Description
action	Set this value to upsert to update the target table using the merge (upsert) action.
connection.id	This value is in the internal identifier for the Snowflake connection through which to publish the table.
mergeJoinKeys	The set of one or more columns whose values are used to determine if a row in the source data that you are publishing matches a row in the target table.
colsToUpdate	

	<p>If the join keys above do match, then the <code>colsToUpdate</code> columns are updated with the values from the source data.</p> <div> NOTE: These column values are updated if <code>matchedRowsAction</code> is set to <code>update</code>. </div>
<code>deleteUnmatchedRowsInTarget</code>	When <code>true</code> , rows in the target table that do not match the <code>mergeJoinKeys</code> columns are deleted. Only matching rows remain.
<code>externalTableLocation</code>	This parameter can be used to include a path to the external table. Otherwise, this value is null.
<code>insertChecked</code>	When <code>true</code> and the join keys do not match, the non-matching rows are written to the table.
<code>isDeltaTable</code>	When <code>true</code> , the target table is written as a delta table, which means data is stored as changes from the previous version of the table.
<code>matchedRowsAction</code>	<p>This parameter defines the action taken on the table when rows are matched for the join keys:</p> <ul style="list-style-type: none"> • <code>update</code> - (default) update the columns to update with the values from the source data • <code>delete</code> - delete the matching rows in the target

For more information, see

<https://api.trifacta.com/ee/es.t/index.html#operation/createPublication>

Checkpoint: You're done.

You have done the following:

1. Created an output object:
 - a. Embedded a `writeSettings` object to define an Avro output.
 - b. Associated the `outputObject` with a recipe.
2. Added another `writeSettings` object to the `outputObject`.
3. Added a table-based publication object to the `outputObject`.

You can now generate results for these three different outputs whenever you run a job (create a `jobgroup`) for the associated recipe.

Step - Apply Overrides

When you are publishing results to a relational source, you can optionally apply overrides to the job to redirect the output or change the action applied to the target table. For more information, see *API Task - Run Job*.

Step - Apply Spark Job Overrides

You can optionally submit override values for a predefined set of Spark properties on the output object. These overrides are applied each time that the `outputObject` is used to generate a set of results.

NOTE: This feature and the Spark properties available for override must be configured by a workspace administrator. For more information, see *Enable Spark Job Overrides*.

Tip: You can apply Spark job overrides to the job itself, instead of applying overrides to the `outputObject`. For more information, see *API Task - Run Job*.

In the following example, an existing outputObject (id=4) is modified to include override values for the default set of Spark overrides. Each Spark property and its value as specified as a key-value pair in the request:

Request:

Endpoint	http://www.wrangle-dev.example.com:3005/v4/outputObjects/4
Authentication	Required
Method	PUT
Request Body	<pre>{ "execution": "spark", "outputObjectSparkOptions": [{ "key": "spark.driver.memory", "value": "10G" }, { "key": "spark.executor.memory", "value": "10G" }, { "key": "spark.executor.cores", "value": "5" }, { "key": "transformer.dataframe.checkpoint.threshold", "value": "450" }] }</pre>

Response:

Status Code	200 - Ok
Response Body	<pre>{ "id": 4, "updater": { "id": 1 }, "updatedAt": "2020-03-21T00:27:00.937Z", "createdAt": "2020-03-20T23:30:42.991Z" }</pre>

cURL example:

```
curl -X PUT \
  http://www.wrangle-dev.example.com/v4/outputObjects/4 \
  -H 'authorization: Basic <auth_token>' \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -d '{
    "execution": "spark",
    "outputObjectSparkOptions": [
      {
        "key": "spark.driver.memory",
        "value": "10G"
      },
    ],
  }'
```

```
{
  "key": "spark.executor.memory",
  "value": "10G"
},
{
  "key": "spark.executor.cores",
  "value": "5"
},
{
  "key": "transformer.dataframe.checkpoint.threshold",
  "value": "450"
}
]
}'
```

Relevant terms:

Term	Description
URL	URL and method to execute.
authorization	Authorization taken to pass to the platform. Basic authorization works. <div>NOTE: This token must be passed with each request to the platform.</div>
cache-control	Cache control setting.
content-type	HTTP content type to send. These applications use <code>application/json</code> .

API Task - Manage AWS Configurations

Contents:

- Overview
 - *Per-User Authentication Methods*
 - *Basic Steps*
- *Step - Acquire information*
- *Step - Locate user*
- *Step - Create awsConfig object*
- *Step - Verify Authentication*
- *Step - For Method 2, assign new IAM role to awsConfig object*
- *Step - Switching Persons or Workspaces for an awsRole*
- *Step - Switching Authentication Methods*

Overview

The Designer Cloud powered by Trifacta® platform supports multiple methods of authenticating to AWS resources. At the topmost level, authentication can be broken down into two modes: system and user.

- **System mode:** One set of credentials is used for each user of the platform to authenticate to AWS.
- **User mode:** Individual user accounts must be configured with AWS credentials.

NOTE: This section covers how to manage AWS credentials through the APIs for individual users (user mode). When in system mode, please manage AWS configuration through the application.

Per-User Authentication Methods

To connect to AWS resources and access S3 data, the following information is required for each user, depending on the method of authentication.

NOTE: Since instance credentials are provided at the system level, use of the instance credential provider type in AWS configuration objects is not supported.

Method 1 - AWS Key and Secret

If users are providing key-secret combinations, the following information is required.

Item	Description
key/secret	(credential provider type is default) The AWS key and secret for the user to authenticate
default bucket	The default S3 bucket where the user can upload data and store generated results
extra buckets	Any extra S3 buckets to which the user should have access

Method 2 - AWS IAM Role ARNs

Users can access AWS resources by assigning an awsConfig object to the account.

Tip: This method is recommended.

The following information is required:

Item	Description

IAM role	(credential provider type is <code>temporary</code>) The IAM role to use to authenticate. NOTE: If this information is not immediately available, a placeholder one is created when you create the configuration object. You can assign roles later. More information is provided below.
default bucket	The default S3 bucket where the user can upload data and store generated results
extra buckets	Any extra S3 buckets to which the user should have access

Authentication objects

For each authentication method, the above pieces of information must be provided for each user.

These pieces of information are defined in an `awsConfig` object. An **awsConfig** object is a set of AWS configuration properties that can be created, modified, and assigned to individual users via API.

For Method 2, the `awsConfig` object maps to an `awsRole` object. An **awsRole object** references an IAM role and an `awsConfig` object. When you create an `awsConfig` object and its credential provider is set to `temporary`, the `awsRole` object is automatically created for you:

- Each `awsRole` object maps to a single IAM role.
- Each `awsRole` object is mapped to an `awsConfig` object.
- The `awsConfig` object is then assigned to individual users.
- Through this mechanism, you have more flexibility in assigning the active role to users.
- As needed, the `awsConfig` object can be mapped at a later time to another `awsRole` object through the `role` attribute.

This approach steps through the process for all these methods.

Platform roles

Your account must have one of the following roles:

- Workspace admin
- Trifacta admin

For more information, see

<https://api.trifacta.com/ee/es.t/index.html#section/Authentication>

Basic Steps

1. Choose your method of authentication.
2. Locate the internal identifier for the user to which to assign the configuration object.
3. Create an `awsConfig` object, assigning the object to the user as part of the process.
4. Verify that the assignment is working.

Step - Acquire information

Acquire all of the information listed above for the `awsConfig` object you wish to create.

Step - Locate user

Now, you need to locate the internal identifier for the user to which you wish to assign this AWS configuration.

Request:

Endpoint	http://www.wrangle-dev.example.com:3005/v4/people
Authentication	Required
Method	GET
Request Body	None.

Response:

Status Code	200 - Ok
Response Body	<pre>{ "data": [{ "id": 3, "email": "4070250@example.com", "name": "Test User4070250", "ssoPrincipal": null, "hadoopPrincipal": null, "isAdmin": false, "isDisabled": false, "forcePasswordChange": false, "state": "active", "lastStateChange": null, "createdAt": "2019-04-16T16:27:51.143Z", "updatedAt": "2019-04-16T16:27:56.630Z", "outputHomeDir": "/trifacta/queryResults/4070250@example.com", "fileUploadPath": "/trifacta/uploads", "awsConfigId": 2 }, { "id": 2, "email": "32870@example.com", "name": "Test User32870", "ssoPrincipal": null, "hadoopPrincipal": null, "isAdmin": false, "isDisabled": false, "forcePasswordChange": false, "state": "active", "lastStateChange": null, "createdAt": "2019-04-16T16:27:19.511Z", "updatedAt": "2019-04-16T16:27:26.703Z", "outputHomeDir": "/trifacta/queryResults/32870@example.com", "fileUploadPath": "/trifacta/uploads", "awsConfigId": 1 }, { "id": 1, "email": "<admin_email>", "name": "Administrator", "ssoPrincipal": null, "hadoopPrincipal": null, "isAdmin": true, "isDisabled": false, "forcePasswordChange": false, "state": "active", "lastStateChange": null, "createdAt": "2019-04-16T07:44:04.299Z", "updatedAt": "2019-04-16T16:28:16.379Z", "outputHomeDir": "/trifacta/queryResults/admin@example.com", "fileUploadPath": "/trifacta/uploads", "awsConfigId": 3 }] }</pre>

```
}  
}
```

Checkpoint: In the above, you noticed that `userId=2` is associated with `awsConfig` object `id=1`, which is the one you are replacing. This is the user to modify. Retain this value for later.

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/getPerson>

Step - Create `awsConfig` object

Create the AWS configuration object.

NOTE: Optionally, the `personId` value can be inserted into the request to assign the AWS configuration object to a specific user at create time, when it is created by an admin user. If it is created by a non-admin user, the object is assigned to the user who created it.

NOTE: For Method 2, an `awsRole` object is automatically created for you when you create the `awsConfig` object. It is mapped to the `awsConfig` object.

Request:

Endpoint	<code>http://www.wrangle-dev.example.com:3005/v4/awsConfigs</code>
Authentication	Required
Method	POST
Request Body	<p>Method 1: AWS key-secret combination</p> <pre>{ "credentialProvider": "default", "personId": 2, "key": "<my_key>", "secret": "<my_secret>", "defaultBucket": "main_bucket", "extraBuckets": ["extra-bucket1", "extra-bucket2"] }</pre> <p>Method 2: IAM role</p> <pre>{ "credentialProvider": "temporary", "personId": 2, "role": "<my_iam_role_object>", "defaultBucket": "main_bucket", "extraBuckets": ["extra-bucket1", "extra-bucket2"] }</pre>

Response for Method 2:

Status Code	201 - Created
-------------	---------------

Response Body**Method 2 example:**

```
{
  "extraBuckets": [
    "extra-bucket1",
    "extra-bucket2"
  ],
  "id": 6,
  "defaultBucket": "main_bucket",
  "credentialProvider": "temporary",
  "externalId": null,
  "activeRoleId": "4",
  "updatedAt": "2019-04-16T23:06:32.049Z",
  "createdAt": "2019-04-16T23:06:32.047Z",
  "credential": null
}
```

Checkpoint: In the above, the awsConfig object has an internal identifier (id=6).

As part of the request, this object was assigned to user 2 personId=2.

The activeRoleId attribute indicates that the internal ID of the awsRole object that was automatically created for you.

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/createAwsConfig>

Step - Verify Authentication

To verify that the above configuration works:

1. User id=2 should login to the application.
2. User uploads assets through the Import Data page.
3. User creates a short recipe that modifies these assets.
4. User runs a job on that recipe to generate output to the default S3 bucket in CSV or JSON for downloading.
5. User verifies that the results can be downloaded.

Checkpoint: Configuration and verification is complete.

Step - For Method 2, assign new IAM role to awsConfig object

If you need to change the IAM role ARN for a user, you can modify the awsConfig object for that user with the new role information.

NOTE: This section only applies if credentialProvider has been set to temporary for the object and if you are using multiple IAM role ARNs in the Designer Cloud powered by Trifacta platform .

The following request modifies the awsConfig id=6.

Request:

Endpoint	http://www.wrangle-dev.example.com:3005/v4/awsConfigs/6
Authentication	Required
Method	PATCH
Request Body	


```
{
  "role": "<my_iam_role_object_3>"
}
```

Response:

Status Code	200 - OK
Response Body	<pre>{ "extraBuckets": ["extra-bucket1", "extra-bucket2"], "id": 6, "defaultBucket": "main_bucket", "credentialProvider": "temporary", "externalId": null, "activeRoleId": "<awsRoleId>", "updatedAt": "2019-04-16T23:06:32.049Z", "createdAt": "2019-04-16T23:06:32.047Z", "credential": null }</pre>

Checkpoint: In the above step, you assigned a new IAM role to the awsConfig object. The underlying awsRole object is created for you and automatically assigned. For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/createAwsRole>

NOTE: After you have completed the above update, the previous awsRole object still exists. If the IAM role associated with it is no longer in use, you should delete the awsRole object. See <https://api.trifacta.com/ee/es.t/index.html#operation/deleteAwsRole>

Step - Switching Persons or Workspaces for an awsRole

When you create or modify an awsRole, you can optionally pass in a person or workspace identifier. When either value is provided, the Designer Cloud powered by Trifacta platform attempts to assign the awsRole to the provided identifier based on the related awsConfig object.

Acquire awsRole identifier

Via awsConfig identifier

Use the following endpoint to retrieve the awsConfig object. This one uses awsConfigId=6:

Request:

Endpoint	http://www.wrangle-dev.example.com:3005/v4/awsConfigs/6
Authentication	Required
Method	GET
Request Body	Empty.

Response:

Status Code	200 - OK
Response Body	<pre>{ "extraBuckets": ["extra-bucket1", "extra-bucket2"], "id": 6, "defaultBucket": "main_bucket", "credentialProvider": "temporary", "externalId": null, "activeRoleId": "<awsRoleId>", "updatedAt": "2019-04-16T23:06:32.049Z", "createdAt": "2019-04-16T23:06:32.047Z", "credential": null }</pre>

Acquire the value for activeRoleId.

Via awsRoles identifier

Use the following request to retrieve all of the awsRoles:

Request:

Endpoint	http://www.wrangle-dev.example.com:3005/v4/awsRoles
Authentication	Required
Method	GET
Request Body	Empty.

In the response, locate the appropriate identifier.

Reassign the awsRole

You can use the PUT method of the following endpoint to re-assign the awsRole to the specified person or workspace. The following example reassigns awsRoleId=3 to personId=6.

Request:

Endpoint	http://www.wrangle-dev.example.com:3005/v4/awsRoles/3
Authentication	Required
Method	PUT
Request Body	<pre>{ "personId": 6 }</pre>

If the request is successful, the awsRole is reassigned to the person identifier.

Tip: In the above request, you can replace "personId" with "workspaceId" to reassign the role to all users in a workspace.

Step - Switching Authentication Methods

Suppose you have created your awsConfig objects to use the AWS Key-Secret method of authenticating. You have now created a set of IAM roles that you would like to assign to your Trifacta users.

The generalized steps for completing this task is the following:

1. Acquire the identifiers for all of the awsConfigs you wish to modified. For each awsConfig, retain the `personId`, so that you can map your configuration changes to individuals. See <https://api.trifacta.com/ee/es.t/index.html#operation/listAwsConfigs>
 - a. For more information on getting your list of users, see <https://api.trifacta.com/ee/es.t/index.html#operation/getPerson>
2. For each user account (`personId`), you must identify the IAM role that you wish to assign it.
3. Use the following modification to the awsConfig object to switch to using the IAM role for the user:

Request:

Endpoint	<code>http://www.wrangle-dev.example.com:3005/v4/awsConfigs/<awsConfigId></code>
Authentication	Required
Method	PUT
Request Body	<pre>{ "credentialProvider": "temporary", "role": "<my_iam_role_object>" }</pre>

Response for Method 2:

Status Code	200 - Ok
Response Body	Method 2 example: <pre>{ "extraBuckets": ["extra-bucket1", "extra-bucket2"], "id": "<awsConfigId>", "defaultBucket": "main_bucket", "credentialProvider": "temporary", "externalId": null, "activeRoleId": "<awsRoleId>", "updatedAt": "2019-04-16T23:06:32.049Z", "createdAt": "2019-04-16T23:06:32.047Z", "credential": null }</pre>

Notes:

Item	Description
<code>credentialProvider</code>	To use IAM roles, this attribute must be updated to be <code>temporary</code> .
<code>role</code>	The IAM role to assign to the configuration.
	If needed, you can change the person (user) to which this awsConfig is applied. Note that the former user of the

personId	configuration cannot access AWS resources until you create a new configuration object for the user's account.
activeRoleId	(response) Internal identifier of the awsRole object that was created for you and assigned to this awsConfig object.

NOTE: The above request must be applied to each awsConfig object that you wish to remap to using an IAM role.

API Task - Wrangle Output to Python

NOTE: As of Release 9.7, Wrangle to Python conversion has been deprecated. For more information, please see *End of Life and Deprecated Features*.

EXPERIMENTAL FEATURE: This feature is intended for demonstration purposes only. This feature may be modified or removed from the Designer Cloud powered by Trifacta platform without warning in a future release. It should not be deployed in a production environment.

Contents:

- *Alpha Release*
- *Enable*
- *Limitations*
- *v4 OutputObjects WrangleToPython Create*
 - *Required Permissions*
 - *Request*
 - *Response*
 - *Reference*

You can enable an API endpoint that converts the your recipe steps to generate a specific output into Python Pandas code. When executed, this generated Python Pandas code applies transformations to your specified dataset and generates the specified output.

Alpha Release

This endpoint is the beginning of enabling the Designer Cloud powered by Trifacta® platform to integrate with pre-existing Python data pipelines.

This documentation provides information on how to explore the capabilities of the Designer Cloud powered by Trifacta platform to generate Python code for your external data pipelines.

Enable

To enable generation of Python Pandas code, please complete the following:

Steps:

1. You apply this change through the *Workspace Settings Page*. For more information, see *Platform Configuration Methods*.
2. Locate the following setting and set it to *Enabled*:

Wrangle to Python Conversion

3. The feature is now enabled.

Limitations

- This endpoint does not currently support multi-dataset operations.
- The generated Python code does not yield readable columns in the output code.

- Conversion of Wrangle script to Python code is supported for CSV files only.

v4 OutputObjects WrangleToPython Create

This section contains reference documentation on the API endpoint. This endpoint method is applied to a specified outputObject. This outputObject is the result of execution of a specific recipe (wrangledDataset). That recipe has references to its source importedDatasets and connections.

For more information on supported versions of the APIs, see <https://api.trifacta.com/ee/es.t/index.html#section/Overview/Versioning-and-Endpoint-Lifecycle>.

This API enables generation of Python Pandas code for the Wrangle recipe associated with an output object.

Version: v4

Relevant terms:

Term	Description
outputObjects	<p>An outputObject is a definition of one or more types of outputs and how they are generated. It must be associated with a recipe.</p> <div> <p>NOTE: An outputObject must be created for a recipe before you can run a job on it. One and only one outputObject can be associated with a recipe.</p> </div>
writeSettings	A writeSettings object defines file-based outputs within an outputObject. Settings include path, format, compression, and delimiters.
publications	A publications object is used to specify a table-based output and is associated with an outputObject. Settings include the connection to use, path, table type, and write action to apply.

Required Permissions

NOTE: Each request to the Alteryx® Analytics Cloud must include authentication credentials. See <https://api.trifacta.com/ee/es.t/index.html#section/Authentication>.

Request

Request Type: POST

Endpoint:

```
/v4/outputObjects/<id>/wrangleToPython
```

Request URI - Example:

```
/v4/outputObjects/3/wrangleToPython
```

Request Body:

The following defines the running environment used for the outputObject.

```
{
  "execution": "spark"
}
```

Response

Response Status Code - Success: 200 - OK

Response Body Example:

The generated response is the Python Pandas code:

```
{
from trifacta.transform_functions.function_definitions import Replace
import pandas as pd
import numpy as np

def run_transforms(df0=None):
    if (df0 is None):
        df0 = pd.read_csv('input.csv', skip_blank_lines=False, lineterminator='\n', dtype=str, encoding='UTF-8')
        replace1 = Replace('x', 'u', False, False)
        replace2 = Replace('x', 'u', True, False)
        replace3 = Replace('x', 'u', True, True)
        replace4 = Replace('x', '\\\\\\\\', True, False)
        df3 = pd.DataFrame({'new_column1': df0['coll'].apply((lambda x: replace1.exec(x))), 'new_column2': df0
['coll'].apply((lambda x: replace2.exec(x))), 'new_column3': df0['coll'].apply((lambda x: replace3.exec(x))),
'new_column4': df0['coll'].apply((lambda x: replace4.exec(x)))})
        return df3
    }
```

Reference

For more information , see

<https://api.trifacta.com/ee/es.t/index.html#operation/getPythonScriptForOutputObjectInput>

API Documentation Versions

Depending on where you access the API documentation, you may be presented different sets of available endpoints for the following reasons.

API Reference Docs

Location: api.trifacta.com

At api.trifacta.com, you can select the API reference documentation for your product edition. When selected, you can review all possible API endpoints and methods that are supported in your product edition.

This portal is available on the public Internet.

Tip: Review this content to see what is possible for your product edition.

API Deployment Docs

Location: In the Designer Cloud application , select **Resources menu > API documentation**.

This instance of the API documentation filters the preceding API reference documentation based on the following:

- Features that are enabled or disabled in your Trifacta deployment
- Endpoints that are accessible based on your API access token
- Any roles that may be applicable to your user account

This portal is available only for registered users of the workspace from which it is accessed.

Tip: Review this version to see the API endpoints that you can use with your current account in your specific Trifacta deployment.

Python SDK

NOTE: As of Release 9.7, Wrangle to Python conversion has been deprecated. For more information, please see *End of Life and Deprecated Features*.

Contents:

- *Prerequisites*
 - *Trifacta prerequisites*
 - *Python prerequisites*
- *Limitations*
- *Download and Install*
- *Examples*
- *Wrangle function reference*
 - *Trifacta module functions*
 - *WrangleFlow module functions*
 - *Data profiling functions*

The Trifacta® Python SDK enables you to integrate the Designer Cloud application into your Python pipelines. When your Python environment has been integrated with the Designer Cloud application, you can leverage the visual tools in the application to rapidly construct your transformation steps on example data that you upload. When you have finished building your recipe, you can invoke a function in your Python environment to download the recipe as Python Pandas code for use in your data pipelines.

Basic task:

1. Through your Python notebook:
 - a. Upload example data to your Trifacta workspace.
 - b. Launch the Designer Cloud application.
2. In the Designer Cloud application :
 - a. Use the transformation tools in the application to transform your example data using a series of recipe steps.
 - b. Iterate on your recipe. Generate results through the Designer Cloud application to verify that you have transformed your data correctly.
3. In your Python notebook:
 - a. Invoke a function to translate the recipe into Python Pandas and download it to your local Python environment.
 - b. Deploy this recipe into other Python pipelines to transform other datasets as needed.

Prerequisites

Trifacta prerequisites

- A workspace administrator must enable the Python to Wrangle feature in your workspace. For more information, see *Workspace Settings Page*.
- You must have a valid API access token. For more information, see *Manage API Access Tokens*.

Python prerequisites

NOTE: If you receive the following error message: `PermissionError: You must setup a trifacta configuration, use tfconfig.setup_configuration(user, pwd)`, then you must deploy `.trifacta.py.conf` in the directory where the software is located.

Please see the installation instructions available at the download URL listed below.

Limitations

NOTE: This is an Alpha release. Do not use the Python SDK in a production environment.

- Some Wrangle functions and transformations are not supported by Python Pandas. Known limitations:
 - NUMFORMAT function
 - String comparison functions
- Transformations that use Array or Map data types are not supported for Python Pandas generation.
- Uploaded files must be in CSV file format.

Download and Install

For more information on downloading and installing the Python SDK, see <https://pypi.org/project/trifacta/>.

Examples

For a basic example, please see <https://pypi.org/project/trifacta/>.

Wrangle function reference

The following wrangling functions are available through the SDK.

Trifacta module functions

`tf` is an alias to the Trifacta module.

Function Name	Description	Arguments
<code>tf.wrangle(*datasets)</code>	Upload one or more datasets to the Designer Cloud application and create a flow for it. This flow is then available through the Designer Cloud application, where you can transform the dataset through the user interface. See https://pypi.org/project/trifacta/ .	*datasets: Pandas DataFrames to be wrangled. It could also be a tuple, where the first element in the tuple is a Pandas DataFrame, and second element is the reference name (string) for the DataFrame.

WrangleFlow module functions

All the below functions are available for the `WrangleFlow` object in your Python environment. So, you must call them using a `WrangleFlow` object.

`wf` is a reference to the `WrangleFlow` object.

Function Name	Description	Arguments
<code>wf.add_datasets(*datasets)</code>	Add Pandas DataFrames to a flow, where <code>datasets</code> is a list of DataFrames.	*datasets: Pandas DataFrames to be added to a flow. It could also be a tuple, where the first element in the tuple is a Pandas DataFrame, and second element is the reference name (string) for the DataFrame.
<code>get_pandas(add_to_next_cell=False, recipe_name="<my_recipe>")</code>	Generates Python Pandas code for your Wrangle recipe.	add_to_next_cell: Set it to True, if you're using Jupyter Notebook and would like to add the generated Pandas code to be added to next cell. If False, the Pandas code is returned as string.

		recipe_name: Recipe for which you want to get the Pandas code. If not specified, the default recipe is used. Use <code>wf.recipe_names()</code> to retrieve available recipes.
<code>wf.run_job(pbar=None, execution='photon', recipe_name=None)</code>	Run a job for a specified recipe.	pbar: can be ignored. execution: Running environment in Designer Cloud powered by Trifacta platform where you want to execute the job. Possible values: <code>photon</code> or <code>emrSpark</code> . recipe_name: Recipe for which you want to execute the job. If set to <code>None</code> , input is the default recipe.
<code>wf.profile(recipe_name=None)</code>	Generate a profile for a specified recipe.	recipe_name: Recipe for which you want to generate profile. If set to <code>None</code> , input is the default recipe.
<code>wf.recipe_names()</code>	Lists the recipe names for the recipe present in Trifacta a flow.	N/A
<code>wf.open_profile(recipe_name=None)</code>	Open a profile that you have previously generated for the specified recipe.	recipe_name: Recipe for which you want to open the profile. If set to <code>None</code> , input is the default recipe.

Data profiling functions

Function Name	Description	Arguments
<code>wf.summary(recipe_name=None)</code>	Returns a table of summary statistics per column	recipe_name: Recipe name for which you want to generate the summary. If set to <code>None</code> , input is the default recipe.
<code>wf.dqBars(show_types=True, recipe_name=None)</code>	Returns the valid/invalid/missing ratio per column	show_types: Show column types information along with data quality bars for the column. recipe_name: Recipe name for which you want to generate the data quality bar. If set to <code>None</code> , input is the default recipe.
<code>wf.colTypes(recipe_name=None)</code>	Lists the inferred data type for each column	recipe_name: Recipe name for which you want to infer data types for each column. If set to <code>None</code> , input is the default recipe.
<code>wf.barsDfList(recipe_name=None)</code>	Returns a list of dataframes, one per column, representing a bar-chart for that column	recipe_name: Recipe name for which you want to generate the bar-chart. If set to <code>None</code> , input is the default recipe.
<code>wf.pdfProfile(filename=None, recipe_name=None)</code>	Returns a snazzy PDF report with all the statistics	filename: Name of the file to which PDF profile results are written. If set to <code>None</code> , results are returned back from the function. recipe_name: Recipe for which you want to generate PDF profile results. If set to <code>None</code> , results are generated for the default recipe.



Copyright © 2022 - Trifacta, Inc.
All rights reserved.