TRIFACTA

Developer Guide

Version: 5.0
Doc Build Date: 04/10/2018

Developer

This section contains topics of interest to data engineers and other developers.

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

Topics:

User-Defined Functions

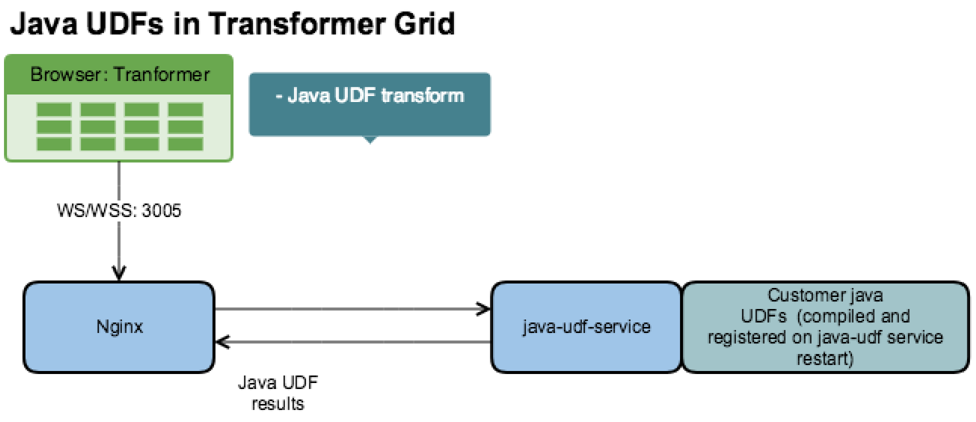
Contents:

The 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 Hadoop at job execution time.


Java UDFs in Transformer Grid
Browser: Tranformer
- Java UDF transform
WS/WSS: 3005
Nginx
java-udf-service
Customer java UDFs (compiled and registered on java-udf service restart)
Java UDF results

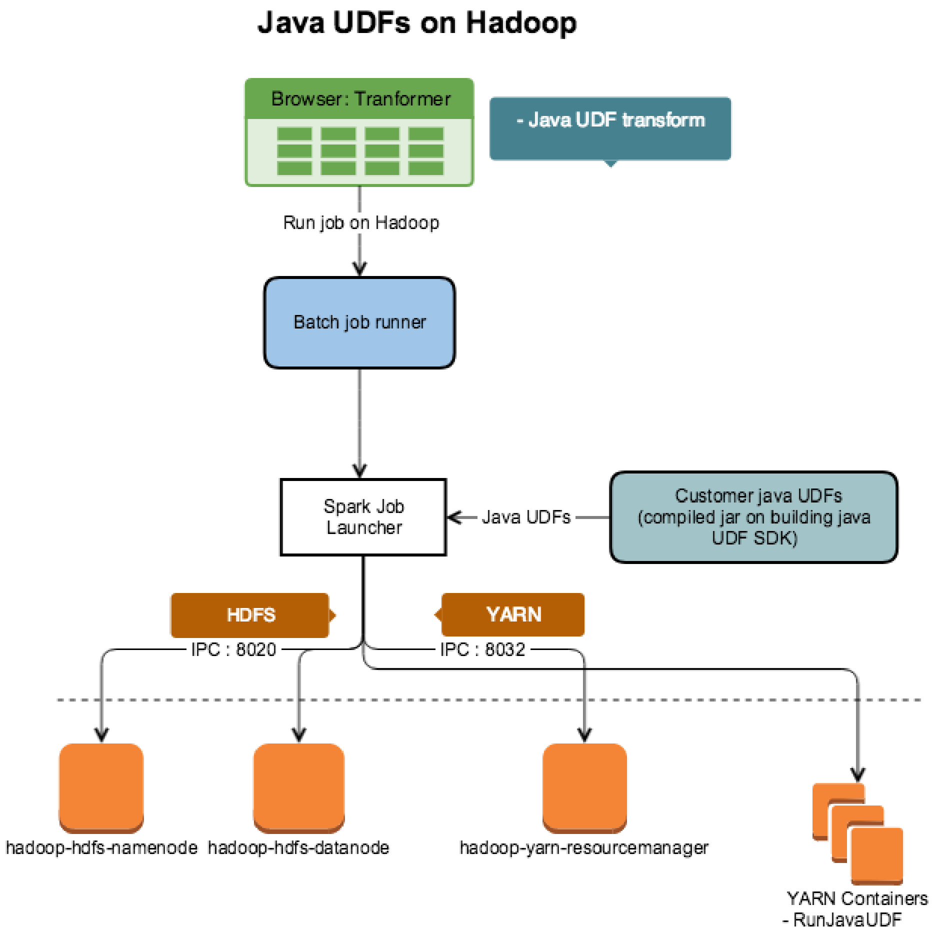


Figure: User-Defined Service

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 executed:

```
udf name:'AdderUDF' col:column1 args:'1' as:'udf_output'
```

Notes:

- The `udf` command causes the named UDF to run.
- After you type `name`, 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 `col` argument is a comma-separated list of the source data to be used as inputs to the exec method.
- The `args` argument is a string of comma-separated values used as inputs to the init method.
- Optionally, the `as` 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 running environment during Hadoop job execution. In these cases, please execute the job on the Trifacta Server.

See *Transformer Page*.

Java UDFs

Contents:

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).

Pre-requisites

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: 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 can take one or more inputs and produces a single output value (map only).

Inputs and outputs must be one of the following types:

- Bool
- String
- Long
- Double

Known Limitations

- In the Trifacta 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 Server job execution. In these cases, please execute the job on Hadoop.

Enable Service

You must enable the Java UDF service in the 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.

> **NOTE:** Each UDF requires at least one input parameter.

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.
    a. This method is run on each record.

> **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 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 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`. Some of these settings may not be available through the *Admin Settings Page*. 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.

1. a. **If you created a new UDF,** restart the Trifacta 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*.

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 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 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 Photon 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 bytes * 2 (input cols) * 1024 rows = 0.5 MB.
  - If the output of the UDF per row is estimated to be 1024 characters, then the output estimate would be 1024 bytes * 1024 rows = 1MB.
  - 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.

Create Custom Data Types Using RegEx

Contents:

As needed, you can deploy custom data types into the 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.

- If your custom data type contains a pre-defined set of values, you can create the custom type using a dictionary file for validation. See *Create Custom Data Types*.

Custom Types Location

On the server hosting the Trifacta platform, type definitions such as dictionaries and custom data types are stored in the following directory:

```
/opt/trifacta/js-data/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.

Directory contents:

- The `dictionaries` sub-directory contains user-defined dictionaries.

  > **NOTE:** Please use the user interface to interact with your dictionaries. See *Custom Type Dialog*.

- The `types` sub-directory contains individual custom data type definitions, each in a separate file.
- The `manifest.json` file contains a JSON manifest of all of the custom dictionaries and types in the system.

Examples

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)$",
      "^(mon|tue|wed|thu|fri|sat|sun)$"
    ],
    "probability": 0.001
  }
}
```

Parameters:

Parameter Name	Description
`name`	Internal identifier for the custom type. Must be unique across all standard types and custom types. **NOTE:** You should verify that your data type's `name` value does not conflict with other custom data type names.
`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`	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. **NOTE:** All match types must be double-escaped in the regex expression. For example, to replicate the `\d` pattern, you must enter: `\\d`. For more information on regular expressions, see *https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions*.
`probability`	(optional) Assign an incremental change to the probability when a match is found between a value and one of the regular expressions. See *Defining probabilities* below.

Tip: In the `types` sub-directory, you can review the regex-based types that are provided with theTrifacta 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 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

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

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

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

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

where:

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

Register your custom types

To add your custom types to the 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 *Create Custom Data Types*.

Command Line Interface

The Trifacta® command line interface (CLI) enables scripted execution of jobs and management of users and connections for the Trifacta platform. This section provides documentation on how to install and deploy the command line tools and includes example commands for each supported action.

Topics:

- *Install CLI Tools*
- *CLI for Connections*
- *CLI for Jobs*
- *CLI for User Admin*
- *CLI Config File*

Logging

The CLI submits requests to the platform through the Trifacta Application, which writes its logging information to the following file:

```
/opt/trifacta/logs/webapp.log
```

In the above log file, some CLI requests, such as job execution, can be located by searching for the following:

```
"ranfrom": "cli"
```

Tip: From the output of the CLI, you should get in the habit of capturing the job, dataset, flow, or other object identifier that the request is creating, modifying, or removing. These IDs are useful for parsing the log file or locating the object in the application.

Administrators can download log files through the Trifacta node operating system or through the web interface for the platform. For more information, see *System Services and Logs*

Log Levels

By default, the logging level for the web application is set to `INFO`.

If you are attempting to debug an issue related to the CLI, you can change the logging level.

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

The log level is defined in the following parameter:

```
"webapp.loggerOptions.level": "INFO",
```

For more information, see *Admin Settings Page*.

Install CLI Tools

Contents:

By default, the Trifacta® Command Line Interface (CLI) tools are installed on the Trifacta node during installation. You can use them from there.

Optionally, you can install the CLI tools on a separate server. For example, you might want to create a dedicated server from which you can run a set of predefined jobs on a periodic basis.

NOTE: The location from where you are running the CLI tools must be able to access the pre-installed instance of the Trifacta platform.

This section describes how to download and install the CLI tools on a dedicated server.

Download

The Trifacta CLI installer is available through a separate file next to the software distribution provided by Trifacta.

- *.RPM for CentOS/RHEL
- *.DEB for Ubuntu

The appropriate file should be downloaded to the server where you are installing the tools. For more information, see *Trifacta Support*.

Install

Steps:

1. On the node where you are installing, execute the command.
   a. For CentOS/RHEL6:

      ```
      rpm -Uvh trifacta-cli-X.Y.Z-AAAA.el6.x86_64.rpm
      ```

      where:
      `X.Y.Z` = the three-digit release number.
      `AAAA` = internal build number.
   b. For CentOS/RHEL7:

      ```
      rpm -Uvh trifacta-cli-X.Y.Z-AAAA.el7.x86_64.rpm
      ```

      where:
      `X.Y.Z` = the three-digit release number.
      `AAAA` = internal build number.
   c. For Ubuntu 14.04 (Trusty):

      ```
      sudo dpkg -i trifacta-cli_X.Y.Z-AAAA~trusty_amd64.deb
      ```

      where:
      `X.Y.Z` = the three-digit release number.
      `AAAA` = internal build number.
   d. For Ubuntu 16.04 (Xenial):

```
sudo dpkg -i trifacta-cli_X.Y.Z-AAAA~xenial_amd64.deb
```

where:
`X.Y.Z` = the three-digit release number.
`AAAA` = internal build number.

2. When the installation is complete, you can begin using the tools. The tools are installed in the following directory:

```
/opt/trifacta/bin/
```

Upgrade

When you upgrade to a new version of Trifacta Wrangler Enterprise, you must complete the following steps to ensure that your CLI tools and scripts are upgraded:

> **NOTE:** There is no guarantee of compatibility between versions of Trifacta Wrangler Enterprise CLI tools. You should re-install the tools with each upgrade.

1. Download and install the new version of the CLI tools. See earlier in this section.
2. Unless changes are required, you can try to run your CLI scripts using the CLI packages that you downloaded from the previous version. If your scripts fail when running jobs, then you should try to re-download the packages from the Transformer page. For more information, see *Recipe Panel*.

For more information, see *Changes to the Command Line Interface*.

CLI for Connections

Contents:

> **NOTE:** This feature requires developer-level skills to enable and use.

The command line references lets you manage connections between the Trifacta® platform and various types of datastores. You can also use this CLI for the following:

- Create, edit, or delete connections

  > **NOTE:** In this release, you cannot create Redshift or SQL DW connections via the CLI. This known issue will be fixed in a future release.

- Get information on all connections

> **NOTE:** Sharing of connections is not supported through the command line interface.

Requirements

- The CLI must have access to a running Trifacta instance. You can specify the host and port of this instance.
- For each connection that you create, the Trifacta node must be able to access it through the listed host and port.

Command Reference

The CLI tools are stored in the following directory:

```
/opt/trifacta/bin/
```

For creating or modifying connections, execute the following command:

```
./trifacta_cli.py (parameters)
```

Parameters are specified below.

Parameters

Common

These parameters are common to job or connection actions.

Parameter	Description	Applicable CLI Commands
`command_type`	(Required) The type of CLI command to execute. Accepted values: • `create_connection` - Create a new connection object. • `edit_connection` - Edit an existing connection object. • `list_connections` - List all connection objects for the specified user. • `delete_connection` - Delete a connection object. See *Examples* below. For more information on the following commands, see *CLI for Jobs*. • `run_job` - Execute a specified job on the specified running environment. • `get_job_status` - Get job status information. • `get_publications` - Acquire publication information for a specified job. • `publish` - Publish a completed job to the specified database table, which has not been created yet. • `load_data` - Load data into the database table, to which a schema has already been applied. Use to append to existing table. • `truncate_and_load` - Overwrite data in specified table.	All
`user_name`	(Required) Trifacta username of the user to execute the job. Please specify the full username. **NOTE:** In the response, this value is listed as `user`.	All

`password`	(Required) Trifacta password for the username If no password is specified, you are prompted to enter one. **NOTE:** If you have enabled Kerberos-based access to the Trifacta platform, you do not need to provide a password. To enable, additional configuration is required. See *Set up for a Kerberos-enabled Hadoop cluster*. **NOTE:** Passwords can be stored in an external file. See *CLI Config File*.	All
`cli_output_path`	(Optional) Defines the client-side path where the JSON output is stored for all commands. Default value is `cli_results.out.` **NOTE:** The user issuing the command must also have execute permissions on all parent folders in the specified `cli_output_path`.	All
`disable_ssl_certification`	(Optional) When communicating over HTTPS, this setting can be used to override the default behavior of validating the server certificate before executing the command. **NOTE:** If you have stored a self-signed certificate on the Trifacta node, please set the `REQUESTS_CA_BUNDLE` environment variable to point to the directory that contains the trusted server's certificate(s). The CLI will verify against these certs. In this case, the `disable_ssl_certificate` parameter is not needed.	All
`conn_ssl`	(Optional) Connect to the datastore over SSL. **NOTE:** You must modify the `host` parameter value to include the appropriate port number for the SSL connection. **NOTE:** SSL connections are not supported for Hive, Redshift, or SQL Server.	All

Params for managing connections

The following parameters apply to managing connection objects only. Some of the preceding parameters may be required for connection actions.

Parameter	Description	Applicable CLI Command

`conn_type`	The type of connection. **NOTE:** After the connection has been created, you cannot change its type. **Tip:** For a list of supported connection types, enter the following at the command line: `./trifacta_cli.py create_connection -h` These connection types can be created by Trifacta admin users only: **NOTE:** These connections must be created through the CLI and must be created as public connections. Include the `conn_is_global` flag in the command. **NOTE:** Only 1 Hive and 1 Redshift connection is permitted per Trifacta deloyment. • Hadoop Hive • Amazon Redshift **NOTE:** A Redshift connection requires S3 as your base storage layer. See *Set Base Storage Layer*. These connection types can be created by any user with appropriate permissions: **NOTE:** Jobs using sources from these connections cannot be executed on Spark. • Microsoft SQL Server • PostgreSQL Database • Oracle Database • Teradata Database For more information on the supported connection types and the tokens to insert for this parameter, see *Connection Types*.	`create_connection`
`conn_name`	Internal name of the connection. This name is referenced in your CLI scripts. It should be a single value without spaces. **NOTE:** This value must be unique among your connection names.	`create_connection`, `edit_connection` , `list_connection`, `delete_connection`

`conn_id`	The internal identifier for the connection. When a connection is created, it is assigned an internal numeric identifier. This ID or the connection_name can be used to reference the connection in future commands. **Tip:** This value is available when you hover over a connection in the application. See *Flows Page*.	`edit_connection` , `list_connection` , `delete_connection`
`conn_host`	Host of the datastore to which you are connecting.	`create_connection`, `edit_connection`
`conn_port`	Port number to access the datastore.	`create_connection`, `edit_connection`
`conn_description`	This text value is displayed to users when they create or edit connections of this type through the Trifacta Application.	`create_connection`, `edit_connection`
`conn_credential_type`	The type of credentials to create. Supported values: • `basic` - Simple username/password to be provided in `conn_credential_location`. Used for JDBC database connections. • `aws` - AWS-specific credentials to be provided in `conn_credential_location`. Used for Redshift connections. • `trifacta_service` - Uses the Trifacta credentials specified in `trifacta-conf.json`. Used for Hive connections.	`create_connection`, `edit_connection`
`conn_credential_location`	The path to a JSON file containing the credentials for your connection, as consistent with the `conn_credential_type`. For more information on the expected format, see *Credentials file* below. **NOTE:** A credential file is not needed if the credential type is `trifacta_service`.	`create_connection, edit_connection`
`conn_params_location`	When you create a connection, you can reference a JSON file containing parameters to apply during the creation of any connection of this type. See *Params file* below.	`create_connection`, `edit_connection`
`conn_skip_test`	If this parameter is added to the command, the connection is not tested. The default is to test the connection. This flag requires no value. **Tip:** After creation, you can test and modify the connection through the application. See *Flows Page*.	`create_connection`, `edit_connection`
`conn_is_global`	If this parameter is added, the connection is public and is available to all Trifacta users after it has been created. This flag requires no value. **NOTE:** To use this option, the executing user must be a Trifacta admin. Hive and Redshift connections require this parameter. **NOTE:** After a connection has been made public, it cannot be made private again. It must be deleted and recreated.	`create_connection`, `edit_connection`

For documentation on the CLI parameters, run:

```
./trifacta_cli.py --help
```

Additional documentation might be available for individual commands using the following:

```
./trifacta_cli.py <commmand> --help
```

Credentials file

You can store connection login credentials in a file on the Trifacta node. When managing connections, you can reference this JSON credentials file in the command, which forces the use of encrypted versions of the credentials stored in the file. Examples are provided below.

Example - Basic credentials:

This example applies for relational connection types: Oracle, PostGreSQL, SQL Server, and Teradata.

```
{
  "username": "<your_username>",
  "password": "<your_password>"
}
```

Example - AWS credentials:

This example applies to connections of AWS type (Redshift).

```
{
 "username": "<your_user>",
 "password": "<your_password>"
  "iamRoleArn": "<your_IAM_role_ARN>"
}
```

NOTE: `iamRoleArn` is optional. For more information, see *Configure for EC2 Role-Based Authentication*.

Params file

In an external file, you can create a set of parameters to pass to any object for which you are creating a connection. For example, when you create a connection to a database, you may need to reference a default database to which any instance of the connection connects.

The following parameters are supported for each vendor.

Vendor	JSON Parameter	Description	Required
Hive	`defaultDatabase`	Name of the default database	No
Redshift	`defaultDatabase`	Name of the default database	Yes
PostgreSQL	`database`	Name of the database.	Yes
Oracle	`service`	Service to use for the connection	Yes
SQL Server	None.		
Teradata	None.		

Additional parameters:

Except for Redshift connections, you can submit additional configuration parameters using the `ConnectStrOpts` key-value pair in the parameters file. Example:

```
"connectStrOpts": ";transportMode=http;httpPath=cliservice"
```

NOTE: Each vendor uses a specific separator between the connection URL and the connection string options. For example, if you are creating or editing a Teradata connection and are submitting `ConnectStrOpts` parameters, the string value must begin with a comma:

```
"connectStrOpts": ",Key1=Value1,Key2=Value2"
```

For more information, see the documentation provided with your database product.

Redshift uses the `extraLoadParams` method, which is described below.

Arbitrary parameters for JDBC connections:

For any supported JDBC connection type, you can include arbitrary parameters specific to the JDBC database as part of your connection string options.

With the exception of Oracle, all JDBC vendor support the following example. The database to which you are connecting supports a parameter (`myView`) which for the database has a value of `custom1`. To extend the preceding Teradata example, your `connectStrOpts` value would be the following, which begins with a comma (`,`):

```
"connectStrOpts": ",Key1=Value1,Key2=Value2?myView=custom1"
```

When the connection is created and used, the connection string might look like the following:

```
jdbc:teradata://example.com:1025/DatabaseServerName?myView=custom1
```

For submitting arbitrary parameters to Oracle, please see the example below.

Example - Hive params:

NOTE: By default, the Hive connection is defined to use TCP. If you are using HTTP to connect to Hive, additional configuration is required, including insertion of additional parameters in your params file. See *Configure for Hive*.

NOTE: If you are connecting to a Kerberos-enabled cluster, you must include the Kerberos principal for Hive as part of the `connectStrOpts` value. See *Configure for Hive*.

```
{
  "connectStrOpts": ";<depends_on_deployment>",
  "defaultDatabase": "default",
  "jdbc": "hive2"
}
```

For more information on connection string options for Hive, see *Configure for Hive*.

Example - Redshift params:

```
{
 "defaultDatabase":"<your_database>",
 "extraLoadParams": "BLANKSASNULL EMPTYASNULL TRIMBLANKS TRUNCATECOLUMNS"
}
```

The first parameter defines the default database.

The second parameter is used when you publish results to Redshift. For more information on these values, see *http://docs.aws.amazon.com/redshift/latest/dg/copy-parameters-data-conversion.html*.

Example - PostgreSQL params:

```
{
  "database":"<your_database>"
}
```

Example - Oracle params:

```
{
  "service":"orcl"
}
```

For submitting arbitrary parameters to Oracle, the arbitrary string must follow the ORA format, in which most of the connection string is replaced by parameters. For example:

```
{
  "connectStrOpts":
"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521))(CONNECT_
DATA=(SERVICE_NAME=orcl)))",
  "service":"orcl"
}
```

In this case, the generated connection string might look like the following:

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(POR
T=1521))(CONNECT_DATA(SERVICE_NAME=orcl)))
```

The original host, port, and service name values specified in the connection are ignored and replaced by these values.

Examples

At the command line, all jobs must be executed through connection objects. For each datastore to which the Trifacta platform is connected, you must create at least one connection object and then reference it in any job execution tasks.

Create connection

NOTE: For Hive and Redshift, connections must be created through the CLI by a Trifacta admin user and must be created as public connections (include the `--conn_is_global` flag). You can only create one connection of each of these types.

For more information on creating a Redshift connection through the CLI, see *Create Redshift Connections*.

For more information on creating a Hive connection through the CLI, see *Configure for Hive*.

Command

Example (all one command):

```
./trifacta_cli.py create_connection --user_name <trifacta_user> --password
<trifacta_password>
--conn_type microsoft_sqlserver --conn_name aSQLServerConnection
--conn_description "This is my connection."
--conn_host example.com --conn_port 1234
--conn_credential_type basic
--conn_credential_location ~/.trifacta/config_conn.json
--conn_params_location ~/.trifacta/p.json
--cli_output_path ./conn_create.out
```

Output

```
Success: Connection aSQLServerConnection created
JSON results written to conn_create.out.
```

JSON Response

Output is written to `./conn_create.out`.

```
{
    "conn_credential_location": "~/.trifacta/config_conn.json",
    "conn_credential_type": "basic",
    "conn_host": "example.com",
    "conn_id": 9,
    "conn_name": "aSQLServerConnection",
    "conn_params_location": "~/.trifacta/p.json",
    "conn_port": "1234",
    "conn_type": "microsoft_sqlserver",
    "host": "http://example.com:3005",
    "results": {
        "createdAt": "2016-06-30T21:53:58.977Z",
        "createdBy": 3,
        "credential_type": "basic",
        "credentials": [
            {
                "username": "<trifacta_user>"
            }
        ],
        "deleted_at": null,
        "description": null,
        "host": "example.com",
        "id": 9,
        "is_global": false,
        "name": "aSQLServerConnection",
        "port": 1234,
        "type": "microsoft_sqlserver",
        "updatedAt": "2016-06-30T21:53:58.977Z",
        "updatedBy": 3
    },
    "status": "success",
    "user_name": "<trifacta_user>"
}
```

Edit connection

Command

In the following command, all parameters specified within angled brackets are optional settings that can be changed. The other ones are required to perform any edit.

You must specify the `conn_name` or the `conn_id`.

NOTE: If you are editing the connection's credentials, you must specify the `conn_credential_type` in the command, which is required if you are changing any credential parameter. This step completely replaces the old credentials, so you must specify all connection parameters in the command.

Example (all one command):

```
./trifacta_cli.py edit_connection --user_name <trifacta_user> --password
<trifacta_password>
--conn_name aSQLServerConnection <--conn_type microsoft_sqlserver>
<--conn_description "This is my connection.">
<--conn_host mynewhost.com> <--conn_port 1234>
<--conn_credential_type basic> <--conn_credential_location
~/.trifacta/config_conn.json>
<--cli_output_path ./conn_edit.out>
```

Output

Following assumes that only the above values for `host` and `cli_output_path` contain new values:

```
Success: Updated connection aSQLServerConnection
JSON results written to conn_edit.out.
```

JSON Response

Output is written to `./conn_edit.out`.

```
{
    "conn_description": "This is my connection.",
    "conn_id": 9,
    "conn_name": "aSQLServerConnection",
    "conn_params_location": "~/.trifacta/p.json",
    "host": "http://nynewhost.com:3005",
    "results": {
        "createdAt": "2016-06-30T22:08:47.016Z",
        "createdBy": 3,
        "credential_type": "basic",
        "credentials": [
            {
                "username": "<trifacta_user>"
            }
        ],
        "deleted_at": null,
        "description": "This is my connection.",
        "host": "mynewhost.com",
        "id": 9,
        "is_global": false,
        "name": "aSQLServerConnection",
        "port": 1234,
        "type": "microsoft_sqlserver",
        "updatedAt": "2016-06-30T22:09:03.670Z",
        "updatedBy": 3
    },
    "status": "success",
    "user_name": "<trifacta_user>"
}
```

List connections

Command

Example (all one command):

```
./trifacta_cli.py list_connections --host dev.redshift.example.com
--user_name <trifacta_user> --password <trifacta_password>
--cli_output_path ./conn_list.out
```

Tip: You can specify a `conn_name` or `conn_id` to return the information about a connection.

Output

```
Listing connections
Found 2 connections for params {'noLimit': 'true'}.
Redshift:
  description: None
  host: dev.redshift.example.com
  credentials: ["{u'username': u'<trifacta_user>'}"]
  port: 5439
  is_global: True
  name: Redshift
  id: 2
  credential_type: custom
  params:
     extraLoadParams: BLANKSASNULL EMPTYASNULL TRIMBLANKS TRUNCATECOLUMNS
    defaultDatabase: dev
  type: amazon_redshift
Hive:
  description: None
  host: dev.hive.example.com
  credentials: ["{u'username': u'<trifacta_user>'}"]
  port: 10000
  is_global: True
  name: Hive
  id: 1
  credential_type: conf
  params:
     jdbc: hive2
    connectStrOpts:
    defaultDatabase: default
  type: hadoop_hive
JSON results written to conn_list.out.
```

JSON Response

Output is written to `./conn_list.out`.

```
{
    "connections": [
        {
            "conn_createdAt": "2016-06-01T21:12:59.383Z",
            "conn_createdBy": 2,
            "conn_credential_type": "custom",
            "conn_credentials": [
                {
                    "username": "<trifacta_user>"
                }
            ],
            "conn_deleted_at": null,
            "conn_description": null,
            "conn_host": "dev.redshift.example.com",
            "conn_id": 2,
            "conn_is_global": true,
            "conn_name": "Redshift",
            "conn_params": {
                "extraLoadParams": "BLANKSASNULL EMPTYASNULL TRIMBLANKS
TRUNCATECOLUMNS",
                "defaultDatabase": "dev"
            },
            "conn_port": 5439,
            "conn_type": "amazon_redshift",
            "conn_updatedAt": "2016-06-01T21:33:38.672Z",
            "conn_updatedBy": 2
        },
        {
            "conn_createdAt": "2016-06-01T21:11:41.222Z",
            "conn_createdBy": 2,
            "conn_credential_type": "conf",
            "conn_credentials": [
                {
                    "username": "<trifacta_user>"
                }
            ],
            "conn_deleted_at": null,
            "conn_description": null,
            "conn_host": "dev.hive.example.com",
            "conn_id": 2,
            "conn_is_global": true,
            "conn_name": "Hive",
            "conn_params": {
                "jdbc": "hive2",
                "connectStrOpts": "",
                "defaultDatabase": "default"
            },
            "conn_port": 10000,
            "conn_type": "hadoop_hive",
            "conn_updatedAt": "2016-06-01T21:39:58.090Z",
            "conn_updatedBy": 2
        }
    ],
    "host": "http://localhost:3005",
    "status": "success",
    "user_name": "<trifacta_user>"
}
```

Delete connection

NOTE: You cannot delete a connection that is in use by a dataset.

Tip: You can delete a connection by using its internal connection identifier (`conn_id`), instead of its connection name.

Command

Example (all one command):

```
./trifacta_cli.py delete_connection --user_name <trifacta_user> --password
<trifacta_password>
--conn_name aSQLServerConnection --cli_output_path ./conn_delete.out
```

Output

```
Success. Deleted connection with id: 9
JSON results written to conn_delete.out.
```

JSON Response

Output is written to `./conn_delete.out`.

```
{
    "conn_name": "aSQLServerConnection",
    "host": "http://localhost:3005",
    "status": "success",
    "user_name": "<trifacta_user>"
}
```

CLI for Jobs

Contents:

NOTE: This feature requires developer-level skills to enable and use.

The Command Line Interface for Jobs enables programmatic control over a variety of operations on the platform. You can use the CLI to execute any of the following types of commands:

- Run a job

  NOTE: In this release, you cannot run jobs using datasets imported from Redshift or SQL DW connections via the CLI. This known issue will be fixed in a future release.

- Check job status
- Publish a completed job to other datastores asynchronously

  NOTE: In this release, you cannot publish results to Redshift or SQL DW connections via the CLI. This known issue will be fixed in a future release.

- Get information on publications of a job
- Load data into a new or existing table in Redshift
- (Hive only) Clear an existing table and load with new data

Requirements

- The CLI must have access to a running instance of the Trifacta® platform. You can specify the host and port of this instance.
- If you are running jobs for a dataset with parameters, the downloaded assets reference only the first matching file of the dataset. To run the job across all files in the dataset with parameters, you must build the matching logic within your CLI script. For more information on datasets with parameters, see *Overview of Parameterization*.

Command Reference

Execute the following command from the top-level Trifacta directory. The Python script references`script.cli`and `datasources.tsv` as parameters.

For repeat executions of the same `script.cli` file, you can parameterize the values in the `datasources.tsv`.

The CLI tools are stored in the following directory:

```
/opt/trifacta/bin/
```

For executing jobs, specify parameters for the following:

```
./trifacta_cli.py (parameters)
```

Parameters

Common

These parameters are common to job or connection actions.

Parameter	Description	Applicable CLI Commands

`command_type`	The type of CLI command to execute. Accepted values: • `run_job` - Execute a specified job on the specified running environment. • `get_job_status` - Get job status information. • `get_publications` - Acquire publication information for a specified job. • `publish` - Publish a completed job to the specified database table, which has not been created yet. • `load_data` - Load data into the database table, to which a schema has already been applied. Use to append to existing table. • `truncate_and_load` - Overwrite data in specified table. See *Examples* below. For more information on the following commands, see *CLI for Connections*. • `create_connection` - Create a new connection object. • `edit_connection` - Edit an existing connection object. • `list_connections` - List all connection objects for the specified user. • `delete_connection` - Delete a connection object.	All
`user_name`	(Required) Trifacta username of the user to execute the job. Please specify the full username. **NOTE:** In the response, this value is listed as `user`.	All
`password`	Trifacta password for the username If no password is specified, you are prompted to enter one. **NOTE:** If you have enabled Kerberos-based access to the Trifacta platform, you do not need to provide a password. To enable, additional configuration is required. See *Set up for a Kerberos-enabled Hadoop cluster*. **NOTE:** You can store username and password information in an external file so that they don't need to be included in each command. For more information, see *CLI Config File*.	All
`cli_output_path`	Defines the client-side path where the JSON output is stored for all commands. Default value is`cli_results.out` **NOTE:** The user issuing the command must also have execute permissions on all parent folders in the specified `cli_output_path`.	All
`disable_ssl_certification`	(Optional) When communicating over HTTPS, this setting can be used to override the default behavior of validating the server certificate before executing the command. **NOTE:** If you have stored a self-signed certificate on the Trifacta node, please set the `REQUESTS_CA_BUNDLE` environment variable to point to the directory that contains the trusted server's certificate(s). The CLI will verify against these certs. In this case, the `disable_ssl_certificate` parameter is not needed.	All commands

`conn_ssl`	(Optional) Connect to the datastore over SSL. **NOTE:** You must modify the `host` parameter value to include the appropriate port number for the SSL connection. **NOTE:** SSL connections are not supported for Hive, Redshift, or SQL Server.	All commands

Params for managing jobs

These parameters apply to managing jobs only.

Parameter	Description	Applicable CLI Commands
`host`	(Required) The server and port number of theTrifacta instance. Replace this value with the host and port of the running Trifacta instance. If it is not provided, `localhost:3005` is assumed. **NOTE:** In some environments, the `http://` or `https://` protocol identifier may be required as part of the `host` value.	All
`conn_name`	Internal name of the connection. This name is referenced in your CLI scripts. It should be a single value without spaces. **NOTE:** This value must be unique among your connection names.	`load_data`, `publish`, `truncate_and_load`
`conn_id`	The internal identifier for the connection. When a connection is created, it is assigned an internal numeric identifier. This ID or the `connection_name` can be used to reference the connection in future commands. **Tip:** This value is available when you hover over a connection in the application. See *Flows Page*.	`publish` , `load_data`, `truncate_and_load`
`job_type`	The execution environment in which to run the job: `photon` = Run on Photon running environment on Trifacta Server . **NOTE:** If the `job_type` parameter is not specified, CLI jobs are run on the Photon running environment. `hadoop` = Run in the default running environment for your Hadoop cluster. **NOTE:** When this job type is applied, your CLI scripts automatically transition to running jobs in Spark. `spark` = Run on the Spark running environment in Hadoop.	`run_job`
`job_id`	The internal identifier for the job. This value can be retrieved from the output of a completed `run_job` command.	`get_job_status`, `publish`,`get_publications`,`load_data`
`profiler`	When `on`, profiling of your job is enabled. Default is `off`.	`run_job`

data	Full UNIX path to the source TSV file. This file contains a URL pointing to the actual Hive or HDFS source: one TSV file for each job run. Executing user must have access to this file.	run_job
script	Full UNIX path from the Trifacta root directory to the CLI script file. Executing user must have access.	run_job
publish_action	(Optional) Defines the action taken on second and subsequent publish operations: • create - (default) A new file is created with each publication. Filename is numeric identifier of the job ID. • append - Each publication appends to the existing output file. Filename is consistent across publications. **NOTE:** Compression of published files is not supported through the command line interface. **NOTE:** When publishing single files to S3, the append operation is not supported. • replace - Subsequent publications replace the same file with each execution.	run_job
header	(Optional), The output for a CSV job with append or create publishing action includes the column headers as the first row. Default is false. **NOTE:** If you use the header option, you must also include the single_file option, or this setting is ignored.	run_job
single_file	(Optional) When true, CSV or JSON outputs are written to a single file. Default is false.	run_job

output_path	(Required) Defines the fully qualified URI to where the job results are written, as in the following examples:
	`hdfs://host:port/path/filename.csv`
	`s3://bucketName/path/filename.csv`
	NOTE: The output_path must include the protocol identifier or host and port number (if applicable).
	This parameter specifies the base filename. If you are publishing files, the publish_action parameter value may change the exact filename that is written.
	Protocol is set in webapp.storageProtocolin trifacta-conf.json.
		run_job

output_format	Accepted values:csv,json, pqt (Parquet), and avro (Avro). **NOTE:** For pqt format, job_type=spark is required. For job_type=photon, you may generate csv, json, and avro results.	run_job
database	Name of Redshift or Hive database to which you are publishing or loading.	publish,load_data
table	The table of the database to which you are publishing or loading.	publish,load_data
publish_format	The format of the output file from which to publish to Hive or Redshift tables. Accepted values: csv, json, pqt (Parquet), oravro (Avro).	publish, get_publications

`publish_opt_file`	Path to file containing definitions for multiple file or table targets to which to write the job's results. For more information, see *CLI Publishing Options File*.	`run_job`
`skip_publish_validation`	By default, the CLI automatically checks for schema validation when generating results to a pre-existing source. If this flag is set, schema validation is skipped on results output.	`run_job`

For documentation on the CLI parameters, run:

```
./trifacta_cli.py --help
```

Additional documentation may be available for individual commands using the following:

```
./trifacta_cli.py <commmand> --help
```

Examples

A key function of the CLI is to execute jobs. You can also check job status through the command line interface and then take subsequent publication actions using other commands.

Run job

This command requires a dataset and a CLI script. The CLI script is used to programmatically run a recipe produced in the Transformer page.

For example, if you receive raw data each day, you can parameterize the execution of the same recipe against daily downloads written to HDFS.

Each run of the CLI script creates a new job. A finished CLI job appears on the Jobs page.

Steps:

1. A recipe is specific to a dataset. In the Transformer page, open the Recipe Panel.
2. Click Download.
3. Select CLI Script.
4. Download to your desktop. The ZIP contains the following:
    - `script.cli` Contains the necessary code and configuration to access HDFS and the script in the Trifacta database.
    - `datasources.tsv` Contains pointers to the source storage location of your datasource(s).
        - If you are running jobs for a dataset with parameters, the downloaded assets reference only the first matching file of the dataset. To run the job across all files in the dataset with parameters, you must build the matching logic within your CLI script. For more information on datasets with parameters, see *Overview of Parameterization*.
        - For an example of how to add parameters in a local script, see *CLI Example - Parameterize Job Runs*.
    - `publishopts.json` Template file for defining one or more publishing targets for running jobs. See *CLI Publishing Options File*.
5. These files must be transferred to the Trifacta Server where you can reference them from the Trifacta root directory.

Notes on connections and jobs

In the downloaded ZIP, the `datasources.tsv` file may contain a reference to the connection used to import the dataset. However, if you are running the CLI in an Trifacta platform instance that is different from its source, this connectionId may be different in the new environment. From the new environment, please do the following:

1. Use the `list_connections` operation to acquire the list of connections available in the new environment. See *CLI for Connections*.
2. Acquire the Id value for the connection corresponding to the one used in `datasources.tsv`.

    NOTE: The user who is executing the CLI script must be able to access the connection in the new environment.

3. Edit `datasources.tsv`. Replace the connection Id value in the file with the value retrieved through the CLI.
4. When the job is executed, it should properly connect to the source through the connection in the new environment.

Command - Basic Job Run

NOTE: This method of specifying a single-file publishing action has been superseded by a newer method, which relies on an external file for specifying publishing targets. In a future release, this method may be deprecated. For more information, see *CLI Publishing Options File*.

Example (All one command):

```
./trifacta_cli.py run_job --user_name <trifacta_user> --password
<trifacta_password> --job_type spark
--output_format json --data redshift-test/datasources.tsv --script
redshift-test/script.cli
--cli_output_path ./job_info.out --profiler on --output_path
hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/42/
cleaned_table_1.json
```

Output

```
Job #42 has been successfully launched:

You may monitor the progress of your job here: http://localhost:3005/jobs
```

JSON Response

JSON response written to`job_info.out`:

```
{
  "status": "success",
  "job_outputs": {
    "other_outputs": [

"hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/42
/.profiler/profilerValidValueHistograms.json",

"hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/42
/.profiler/profilerSamples.json",

"hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/42
/.profiler/profilerTypeCheckHistograms.json",

"hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/42
/.profiler/profilerInput.json"
    ]
    "job_result_files": [

"hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/42
/cleaned_table_1.json",
    ]
  },
  "job_id": 42,
  "cli_script":
"/trifacta/queryResults/foo@trifacta.com/redshift-test/script.cli",
  "job_type": "spark",
  "profiler": "on",
  "source_data":
"/trifacta/queryResults/foo@trifacta.com/redshift-test/datasources.tsv",
  "host": "localhost:3005",
  "output_path":
"hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/42
/cleaned_table_1.json",
  "user": "foo@trifacta.com",
  "output_file_formats": [
    "json"
  ],
}
```

Command - File Publishing Options

You can specify publication options as part of your `run_job` command. In the following, a single CSV file with headers is written to a new file with each job execution.

Example (All one command):

```
./trifacta_cli.py run_job --user_name <trifacta_user> --password
<trifacta_password> --job_type spark
--output_format csv --data redshift-test/datasources.tsv --script
redshift-test/script.cli
--publish_action create --header true --single_file true
--cli_output_path ./job_info.out --profiler on --output_path
hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/43/
cleaned_table_1.csv
```

Output

```
Job #43 has been successfully launched:

You may monitor the progress of your job here: http://localhost:3005/jobs
```

JSON Response

JSON response written to`job_info.out`:

```
{
  "status": "success",
  "job_outputs": {
    "other_outputs": [

"hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/43
/.profiler/profilerValidValueHistograms.json",

"hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/43
/.profiler/profilerSamples.json",

"hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/43
/.profiler/profilerTypeCheckHistograms.json",

"hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/43
/.profiler/profilerInput.json"
    ]
    "job_result_files": [

"hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/43
/cleaned_table_1.csv",
    ]
  },
  "job_id": 43,
  "cli_script":
"/trifacta/queryResults/foo@trifacta.com/redshift-test/script.cli",
  "output_file_formats": [
    "csv",
  ],
  "job_type": "spark",
  "host": "localhost:3005",
  "job_output_path":
"/trifacta/queryResults/foo@trifacta.com/MyDataset/43/",
  "user": "foo@trifacta.com",
  "source_data":
"/trifacta/queryResults/foo@trifacta.com/redshift-test/datasources.tsv",
  "profiler": "on"
}
```

Command - publishing to multiple targets

As part of the CLI job, you can define multiple file or table targets to which to write the job results. For more information, see *CLI Publishing Options File*.

Get job status

After you queue a job through the CLI, you can review the status of the job through the application or through the CLI.

> **Tip:** You can acquire the job ID through the application as needed. For example, at some point in the future, you might decide to publish to Hive the results from a job you executed two weeks ago. It might be easiest to retrieve this job identifier from the Dataset Details page. See *Dataset Details Page*.

Command

Example (All one command):

```
./trifacta_cli.py get_job_status --user_name <trifacta_user> --password
<trifacta_password> --job_id 42
--cli_output_path ./job_info.out
```

Output

```
Job status: Complete
```

JSON Response

JSON response written to`job_info.out`:

```
{
  "status": "success",
  "job_outputs": {
    "other_outputs": [

"hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/42
/.profiler/profilerValidValueHistograms.json",

"hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/42
/.profiler/profilerSamples.json",

"hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/42
/.profiler/profilerTypeCheckHistograms.json",

"hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/42
/.profiler/profilerInput.json"
    ]
    "job_result_files": [

"hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/42
/cleaned_table_1.json",

"hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/42
/cleaned_table_1.csv",

"hdfs://localhost:8020/trifacta/queryResults/foo@trifacta.com/MyDataset/42
/cleaned_table_1.avro",
    ]
  },
  "job_id": 42,
  "cli_script":
"/trifacta/queryResults/foo@trifacta.com/redshift-test/script.cli",
  "output_file_formats": [
    "csv",
    "json",
    "avro",
  ],
  "job_type": "spark",
  "host": "localhost:3005",
  "job_output_path":
"/trifacta/queryResults/foo@trifacta.com/MyDataset/42/",
  "user": "foo@trifacta.com",
  "source_data":
"/trifacta/queryResults/foo@trifacta.com/redshift-test/datasources.tsv",
  "profiler": "on"
}
```

Publish

You can publish job results to Hive or Redshift:

- For Hive, you can publish Avro or Parquet results from HDFS or S3 to Hive.
- For Redshift, you can publish CSV, JSON, or Avro results from S3 to Redshift.

> **NOTE:** To publish to Redshift, results must be written first to S3.

> **NOTE:** By default, data is published to Redshift using the `public` schema. To publish using a different schema, preface the`t able` value with the name of the schema to use:`MySchema.MyTable`.

Publish commands can be executed as soon as the job identifier has been created. After the publish command is submitted, the publish job is queued for execution after any related transform job has been completed.

You execute one publish command for each output you wish to write to a supported database table. A new database table is created every run.

Command

Example (All one command):

```
./trifacta_cli.py publish --user_name <trifacta_user> --password
<trifacta_password> --job_id 42
--database dev --table table_job_42 --conn_name 1 --publish_format avro
--cli_output_path ./publish_info.out
```

Output

```
Publish has been successfully launched:You may monitor the progress of your
publish job here: http://localhost:3005/jobs
Upon success, you may view the results of your publish job here:
http://localhost:3005/jobs/42
```

JSON Response

JSON response written to`publish_info.out`:

```
{"status": "Job Started", "job_id": 42}
```

Get publications

You can retrieve a JSON list of all publications that have been executed for a specific job.

Command

Example (All one command):

```
./trifacta_cli.py get_publications --user_name <trifacta_user> --password
<trifacta_password> --job_id 42
--cli_output_path ./publications.out --publish_format avro
```

Output

```
Job with id 42 has 2 avro publication(s) associated with it. The list of
publications is available in "./publications.out".
```

JSON Response

JSON response written to`publications.out`:

```
{
  "publications": [
    {
      "publication_target": "redshift",
      "job_id": "42",
      "database": "dev",
      "publication_id": 69,
      "app_host": "trifacta.example.com:3005",
      "user": "foo@trifacta.com",
      "table": "table_job_42",
      "publish_format": "avro",
      "connect_str": "jdbc:redshift://dev.example.com:5439/dev"
    },
    {
      "publication_target": "hive",
      "job_id": "42",
      "database": "default",
      "publication_id": 70,
      "app_host": "trifacta.example.com:3005",
      "user": "foo@trifacta.com",
      "table": "table_job_42",
      "publish_format": "avro",
      "connect_str": "jdbc:hive2://hadoop:10000/default"
     }
  ],
}
```

Load data into table

You can load data into pre-existing Redshift tables. Data is appended after any existing rows.

NOTE: When appending data into a Redshift table, the columns displayed in the Transformer page must match the order and data type of the columns in the target table.

Command

Example (All one command):

```
./trifacta_cli.py load_data --user_name <trifacta_user> --password
<trifacta_password> --job_id 42
--database dev --table table_42 --conn_name aSQLServerConnection
--publish_format avro
--cli_output_path ./load_info.out
```

Output

```
Load data/Append has been successfully launched:You may monitor the
progress of your publish job here: http://localhost:3005/jobs
Upon success, you may view the results of your Load data/Append job here:
http://localhost:3005/jobs/42
```

JSON Response

JSON response written to`load_info.out`:

```
{"status": "Job Started", "job_id": 42}
```

Truncate and load

For existing Hive tables, you can clear them and load them with results from a job. You cannot truncate and load into Redshift tables.

Command

Example (All one command):

```
./trifacta_cli.py truncate_and_load --user_name <trifacta_user> --password
<trifacta_password> --job_id 10
--database dev --table table_43 --conn_name aSQLServerConnection
--publish_format avro
--cli_output_path ./load_and_trunc_info.out
```

Output

```
Truncate and Load has been successfully launched:You may monitor the
progress of your publish job here: http://localhost:3005/jobs
Upon success, you may view the results of your Truncate and Load job here:
http://localhost:3005/jobs/10
```

JSON Response

JSON response written to`load_and_trunc_info.out`:

```
{"status": "Job Started", "job_id": 10}
```

CLI Example - Parameterize Job Runs

You can use the following Bash script to execute parameterized job runs on the Trifacta® node. This script accepts parameters to identify the CLI package downloaded to the node and then runs the job, whose output includes an identifier for the current date. In this manner, the script can be run on a daily basis on any number of CLI packages.

The CLI package includes:

- `script.cli` - script file
- `datasources.tsv` - file containing a pointer to the storage location of the source data

These values are provided to the script as command-line parameters (`--script` and `--source`).

Based on the specified parameters, this script does the following:

- Launches the job
- Monitors job execution
- Generates error messages if the execution fails
- Generates a success message if the execution succeeds

```
#!/bin/bash

## Example script to run a scheduled job after updating the date.

## Scan the command line arguments for the script file path and for the
datasources.tsv file path
for i in "$@"
do
case $i in
    --script=*)
    ScriptParam="${i#*=}"
    ;;

    --source=*)
    SourceParam="${i#*=}"
    ;;

    *)
    # unknown option
    echo ${i} parameter is not recognized. Please provide --script and
--source values.
    echo
    exit 0
    ;;
esac
done
if [${ScriptParam} -eq ""]
then
  echo --script param is required.
  echo
  exit 0
fi
if [${SourceParam} -eq ""]
then
  echo --source param is required.
  echo
  exit 0
fi
```

```
## Get the current date.
DATE=`date +%m%d`

## Define Job parameters
AppHost=http://localhost:3005
User=user@trifacta.com
Password=password
## Hard-coded version: Script='/var/log/myDir/script.cli'
Script='${ScriptParam}'
## Hard-coded version: Data='/var/log/myDir/datasources.tsv'
Data='${SourceParam}'
JobType='spark'
OutputFormats='avro'
OutputPath="/data/common/output/$DATE"
extraArgs="--disable_server_certificate_verification"

## uncomment once daily job commences
## Change the job source date
## NEWPATH="hdfs://namenode:port/path/to/$DATE/file"
## echo $NEWPATH > $Data


## Launch job
echo "/opt/trifacta/bin/trifacta_cli.py run_job --script=$Script
--data=$Data --host=$AppHost --job_type=$JobType --user_name=$User
--password=$Password --output_formats=$OutputFormats
--output_path=$OutputPath $extraArgs"  >> stdout.txt
/opt/trifacta/bin/trifacta_cli.py run_job --script=$Script --data=$Data
--host=$AppHost --job_type=$JobType --user_name=$User --password=$Password
--output_formats=$OutputFormats --output_path=$OutputPath $extraArgs  >>
stdout.txt 2>> stderr.txt


JobLaunched=$?

if [ "$JobLaunched" -eq 1 ]
then
  echo "Failed to launch job. See stderr.txt for details"
  exit 1
fi

## Parse job id from stdout.txt
JobId=$(cat stdout.txt | sed ':a;N;$!ba;s/\n/ /g;s/.*\Job
#\([0-9]*\).*/\1/')

JobInfo=''
JobStatus='Pending'


## If jobId exists..
if [ "$JobId" -ge 0 ]
then
```

```
  echo "Job with Id " $JobId " launched"

  ## Start loop to monitor job status
  while [ "$JobStatus" != 'Complete' ] && [ "$JobStatus" != 'Failed' ]
  do
    ## Get job status from server
    command="/opt/trifacta/bin/trifacta_cli.py get_job_status
--job_id=$JobId --host=$AppHost --user_name=$User --password=$Password
$extraArgs"

    JobStatus=`$command | sed -e 's/^.*: //'`
    if [ "$JobStatus" != 'Complete' ] && [ "$JobStatus" != 'Failed' ]
    then
      echo "Waiting for job to complete..."
      sleep 20
    fi
  done
else
  echo "Failed to launch job. See stderr.txt for details"
  exit 1
fi

if [ "$JobStatus" = 'Complete' ]
then
  echo "Job "$JobId" is complete."
  echo "Output path is $OutputPath"
else
```

```
    echo "Job failed to complete. Job status = "$JobStatus
    exit 1
  fi
```

You can use the above as a basic template for execution of any type of CLI command.

CLI Publishing Options File

If needed, you can specify multiple file or table targets as part of a single CLI job. In your CLI command, the path on the Trifacta node to this JSON file is specified as the `publish_opt_file` parameter, as in the following:

```
./trifacta_cli.py run_job --user_name <trifacta_user> --password
<trifacta_password> --job_type spark --data redshift-test/datasources.tsv
--script redshift-test/script.cli --cli_output_path ./job_info.out
--profiler on --publish_opt_file /json/publish/file/publishopts.json
```

The file `publishopts.json` contains the specification of the targets.

Tip: To specify this file, you can run this job through the application. After the job has completed, download the CLI script from the Recipe panel in the Transformer page. The downloaded `publishopts.json` file contains the specification for the targets you just executed. See *Recipe Panel*.

Example `publishopts.json`file:

```
{
  "file": [
    {
      "path":
"hdfs://hadoop:50070/trifacta/queryResults/admin@trifacta.local/POS-r01.cs
v",
      "action": "create",
      "format": "csv",
      "header": true,
      "asSingleFile": true,
      "compression": "none"
    },
    {
      "path":
"hdfs://hadoop:50070/trifacta/queryResults/admin@trifacta.local/POS-r01.js
on",
      "action": "create",
      "format": "json",
      "header": false,
      "asSingleFile": false,
      "compression": "none"
    }
  ],
  "hive": [
    {
      "databaseName":"default",
      "tableName":"POS-r01",
      "action":"overwrite"
    }
  ]
}
```

NOTE: All of the following properties require valid values, unless noted.

File targets:

Property	Description
`path`	Full path to the target file. Path must include the protocol identifier, such as `hdfs://` and the port number.
`action`	The action to take on the file. Supported actions: • `create` - Create a new file with each subsequent publication. Filenames for subsequent job runs are appended with the job number identifier. • `append` - The results of each subsequent job run are appended to the existing file contents. • `replace` - The results of each subsequent job run replace the same file. Previous job run results are lost unless moved out of the location. Some limitations apply to these options. See *Run Job Page*.

`format`	Output format for the file. Supported formats: • `csv` • `json` • `avro` • `pqt`
`header`	If set to `true`, then output files in CSV format include a header row. Headers cannot be applied when compression is enabled.
`asSingleFile`	If set to `true`, then output files are written to a single file. If set to `false`, then the output files are written to multiple files as needed.
`compression`	(optional) This property can be used to specify any compression to apply to a text-based file. Supported compression formats: • `gzip` • `bzip2` • `snappy` If this is not specified, then no compression is applied to the output file.

Hive targets:

Property	Description
`databaseName`	Name of the database. **NOTE:** The database must contain at least one table.
`tableName`	Name of the table in the database to which to write.
`action`	The write action to apply to the table. Supported actions: • `create` - Create a new table with each subsequent publication. Table names for subsequent job runs are appended with a timestamp. • `append` - The results of each subsequent job run are appended to the existing table contents. • `replace` - The results of each subsequent job run are written to the same table, which has been emptied. Previous job run results are lost unless moved out of the location (dropAndLoad). • `overwrite` - The results of each subsequent job run are written to a newly created table with the same name as the output table from the previous job run (truncateAndLoad). Some limitations apply to these options. See *Run Job Page*.

CLI for User Admin

Contents:

NOTE: This feature requires developer-level skills to enable and use.

The Command Line Interface for User Administration enables administrators to perform bulk user management tasks on the platform. You can use the CLI to manage the following tasks:

- Create, edit, or delete users.
- Enable or disable an existing users.
- Retrieve individual or all user profiles, including any security details.
- Password reset.

Command Reference

The CLI tools are stored in the following directory:

```
/opt/trifacta/bin/
```

For creating or managing users, specify parameters for the following:

```
./trifacta_admin_cli.py --admin_username <trifacta_admin_username>
--admin_password <trifacta_admin_password> --create_user --user_name
joe@example.com
--password user_pwd --verbose
```

Parameters

Parameter	Description	Applicable CLI Commands
`admin_username`	Username of the admin account to be used to execute the user admin command. Pleas e specify the full username. **NOTE:** If Single Sign On is enabled, you must specify your SSO credentials here instead. See below.	All
`admin_password`	Password of the admin account. **NOTE:** If Single Sign On is enabled, you must specify your SSO credentials here instead. See below. **NOTE:** If you have enabled Kerberos-based access to the Trifacta platform, you do not need to provide a password. To enable, additional configuration is required. See *Set up for a Kerberos-enabled Hadoop cluster*. **NOTE:** Passwords can be stored in an external file, which is automatically checked during script execute. See *CLI Config File*.	All
`admin_command_type`	The type of CLI command to execute. Accepted values: • `create_user` - Create a new user account with the specified credentials. • `show_user` - Retrieve account information for the specified user account. • `edit_user` - Modify the specified user account. • `get_password_reset_url` - Generate a password reset URL for a specified user. • `delete_user` - Delete the specified user account. See Examples below.	All
`host`	(Optional) The server and port number of the Trifacta® instance. By default, this value is set to `http://localhost:3005`. Specify a new value if needed. **NOTE:** When SSO is enabled, you must specify this value to point to the Apache server port where the Trifacta node authentication gateway listens. When running the CLI on the Trifacta node, this value is typically the following: `https://localhost:2443` See below. **NOTE:** In some environments, the `http://` or `https://` protocol identifier may be required as part of the `host` value.	All

`user_name`	Username of the account to be modified. This value is the user ID. It must resolve to a valid, accessible email address. Some features of the platform fail to work correctly with invalid email addresses. **NOTE:** In the response, this value is listed as `email` .	All
`password`	Password of the account to be modified. **NOTE:** If you have enabled Kerberos-based access to the Trifacta platform, you do not need to provide a password. To enable, additional configuration is required. See *Set up for a Kerberos-enabled Hadoop cluster*.	All
`name`	Display name for the user.	`create_user` a nd `edit_user` if making changes to this parameter.
`verbose`	Generate verbose output.	All
`disable_ssl_certification`	(Optional) When communicating over HTTPS, this setting can be used to override the default behavior of validating the server certificate before executing the command. **NOTE:** If you have stored a self-signed certificate on the Trifacta node, please set the `REQUESTS_CA_BUNDLE` environment variable to point to the directory that contains the trusted server's certificate(s). The CLI will verify against these certs. In this case, the `disable_ssl_certificate` paramet er is not needed.	All
`disable`	(Optional) Put the user in a disabled state.	`create_user` a nd `edit_user`
`enable`	(Optional) Put the user in an enabled state. Default is to enable the user.	`create_user` a nd `edit_user`
`transfer_assets_to`	(Optional) When deleting a user, you can optionally transfer all of the user's assets to another user. **NOTE:** Assets cannot be transferred to another user before or after the user deletion command. If assets are not transferred, they remain unowned in the system and are not removed from any form of storage.	`delete_user`

For documentation on the CLI parameters, run:

```
./trifacta_admin_cli.py --help
```

Config file

You can store Trifacta platform username and password information in an external file. See *CLI Config File*.

User account properties

The following user account properties are exposed through the command line:

Property	Description	Editable through CLI?

`--hadoopPrincipal`	Hadoop principal value that is used to connect to the Hadoop environment. This setting applies only when secure impersonation is enabled.	Y
`--outputHomeDir`	The output home directory for the user. By default, the results of each job executed by the user are generated in a sub-directory within this one.	Y
`--name`	The display name for the user.	Y
`--isDisabled`	When set to `True`, the user account is disabled and cannot be used to login to the application.	Y
`--email`	The email address associated with the user account. The email address is also the userID for the account.	Y
`--ssoPrincipal`	The SSO principal value associated with the user account. This value only applies to environments that are integrated with an enterprise Single Sign On solution.	Y
`--enableAdmin`	When set to `True`, this user account is a system administrator account. You should limit the number of accounts that have system administrator access.	Y
`--disableAdmin`	When set to `True`, this user account is not a system administrator account. You should limit the number of accounts that have system administrator access.	Y
`--lastLoginTime`	The timestamp of when the user account was most recently used to login to the application.	N

Examples

User Admin under SSO

If you are in an SSO environment, the following properties require special values to properly authenticate with AD/LDAP. All values are required:

Property	Description
`admin_username`	Use the SSO username for the admin user issuing the command.
`admin_password`	Use the password associated with the SSO username.
`host`	This value must point to the SSO gateway on the Trifacta node and must include the port number. If you are running the CLI on the Trifacta node, use the following: `https://localhost:2443` For more information, see *Configure SSO for AD-LDAP*.
`ssoPrincipal`	In SSO environments, this parameter is required. It must be set to the SSO principal value associated with the user that is being modified.

Create user

Command

Example (all one command):

```
./trifacta_admin_cli.py --admin_username <trifacta_admin_user>
--admin_password <trifacta_admin_password>
create_user --user_name joe@example.com --password Hello2U --name Joe
```

Notes

- Add `--disable` parameter to create the user in a disabled state.

Output

```
Create user joe@example.com
```

```
Account information for joe@example.com
  hadoopPrincipal: None
  outputHomeDir: /trifacta/queryResults/joe@example.com
  name: Joe
  isDisabled: False
  email: joe@example.com
  ssoPrincipal: None
  enableAdmin: False
  lastLoginTime: None
```

Show user

Command

Example (all one command):

```
./trifacta_admin_cli.py --admin_username <trifacta_admin_user>
--admin_password <trifacta_admin_password>
show_user --user_name joe@example.com
```

Output

```
Show user joe@example.com
```

```
Account information for joe@example.com
  hadoopPrincipal: None
  outputHomeDir: /trifacta/queryResults/joe@example.com
  name: Joe
  isDisabled: False
  email: joe@example.com
  ssoPrincipal: None
  enableAdmin: False
  lastLoginTime: None
```

Edit user

Command

The following command changes the Single Sign On principal for the user to a new value. The values for other user account settings found in the response below can be inserted in the command to modify those settings.

Example (all one command):

```
./trifacta_admin_cli.py --admin_username <trifacta_admin_user>
--admin_password <trifacta_admin_password>
edit_user --user_name joe@example.com --ssoPrincipal sso_principal
```

Output

```
Edit user joe@example.com
```

```
Account information for joe@example.com
  hadoopPrincipal: None
  outputHomeDir: /trifacta/queryResults/joe@example.com
  name: Joe
  isDisabled: True
  email: joe@example.com
  ssoPrincipal: sso_principal
  enableAdmin: False
  lastLoginTime: None
```

Generate password reset URL

Command

The following command generates a URL for a specified user that enables the user to reset his or her account password.

> **NOTE:** The script returns with a URL containing the hostname with which it was invoked. You should invoke the script with a fully qualified domain name. If returned hostname is not accessible to the designated user, then the hostname must be replaced prior to passing the URL to the user for execution.

Example (all one command):

```
./trifacta_admin_cli.py --admin_username <trifacta_admin_user>
--admin_password <trifacta_admin_password>
get_password_reset_url --user_name joe@example.com
```

Output

```
Generating password reset url for user joe@example.com
Reset
url:http://localhost:3005/password-reset?email=joe@example.com&code=CD4423
2791
```

Disable user

Command

The following command disables the specified user. Disabled users can no longer login to the application and cannot execute any jobs or commands at the command line.

Example (all one command):

```
./trifacta_admin_cli.py --admin_username <trifacta_admin_user>
--admin_password <trifacta_admin_password>
edit_user --user_name joe@example.com --disable
```

Output

```
Edit user joe@example.com
```

```
Account information for joe@example.com
  hadoopPrincipal: None
  outputHomeDir: /trifacta/queryResults/joe@example.com
  name: Joe
  isDisabled: True
  email: joe@example.com
  ssoPrincipal: None
  enableAdmin: False
  lastLoginTime: None
```

Delete user

Command

Delete the user `joe@example.com` and transfer his assets to `jim@example.com`.

> **NOTE:** The transfer of the deleted user's assets is optional. If it is invoked, the user to whom the assets are assigned must have matching permissions on the datastores where the imported datasets are located.
>
> If it is not invoked, the assets remain on the datastore and cannot be managed through the Trifacta platform until someone creates imported datasets from the files or directories.

Example (all one command):

```
./trifacta_admin_cli.py --admin_username <trifacta_admin_user>
--admin_password <trifacta_admin_password>
delete_user --user_name joe@example.com --transfer_assets_to
jim@example.com
```

Output

```
Delete user joe@example.com
Transferring assets from joe@example.com to jim@example.com
```

Troubleshooting

Exceeded 30 redirects when executing Admin CLI in SSO mode

If you are executing the Admin CLI in SSO mode on the localhost, you may receive the following error message to standard output:

```
Exceeded 30 redirects
```

Solution:

This problem occurs when the CLI is run against the application, instead of the gateway proxy. Please insert the host of the gateway proxy for the `host` parameter, instead of the host of the application.

CLI Config File

As an alternative to including admin passwords in each command that is executed, you can insert a set of admin credentials into a configuration file. The file location is the following:

```
~/.trifacta/config.json
```

One or more sets of credentials can be specified in the following format:

```
{
"credentials": {
  "usernameX":"passwordX",
  "usernameY":"passwordY"
  }
}
```

In your scripts, you can specify just the value for `admin_username`, and the config file is checked for the appropriate password, which is applied to the command.

NOTE: The permissions on this config file should be set such that only the user executing the command can read the file.

API Reference

This section contains reference information on the REST APIs that are made available by the Trifacta platform.

Topics:

API Overview

Contents:

To enable programmatic control over its objects, the Trifacta platform supports a range of REST API endpoints across the objects in the platform. This section provides an overview of the API design, methods, and supported use cases.

Supported operations:

- **Connections:** Get information about connections
- **Datasets:** Create, list, update, and delete operations on datasets
    - Swap datasets
- **Jobs and Results:**
    - Launch job
    - Get job status
    - Publish job results
    - Create dataset from results
- **Get profile metadata:**
    - Quality bar status
    - Schema (column names and types)
- **Users:** Create, list, delete

Uses:

- Can be used for automation of resource management for end-to-end workflow
- Can be used to integrate wrangling experience in third-party application
- See *Use Cases* below.

Design Overview

URL Format

```
<http/https>://<my_server>:<port_number>/<version>/<endpoint>/[resource_id
]/[association][?args]
```

Elements in square brackets `[brackets]` are optional.

Element	Description	Example
`<http/https>`	HTTP protocol identifier. The protocol should be `https` in a production environment.	`https`
`<my_server>`	Name of the Trifacta node	`wrangler.example.com`
`<port_number>`	Port number over which you access the Trifacta platform. By default, this value is `3005`.	`3005`
`<version>`	API version number. **NOTE:** Unless stated otherwise, the versions for all API endpoints is `v3`.	`v3`
`<endpoint>`	Name of the API endpoint to use.	`/connections`
`[resource_id]`	Internal identifier for the specific resource requested from the endpoint. This value defines the object against which the requested operation is performed.	`/10`
`[association]`	If applicable, the **association** identifiers the API endpoint that is requested using the context determined by the `<endpoint>` and the `[resource_id]`. Associations can also be referenced by query parameter. See *Embedding Associations* below.	`/jobGroups`
`[?args]`	In some cases, arguments can be passed to the endpoint in the form of query parameters.	`?arg1=value1&arg2=value2`

Naming Conventions

- Resource names are plural and expressed in camelCase.
- Resource names are consistent between main URL and URL parameter.

v4 conventions

The following conventions apply to v4 and later versions of the APIs:

- Parameter lists are consistently enveloped in the following manner:

```
{ "data": [
    {
       ...
    }
  ]
}
```

- Field names are in `camelCase` and are consistent with the resource name in the URL or with the `embed` URL parameter.
- From early API versions, foreign keys have been replaced with identifiers like the following:

v3 and earlier	v4 and later
`"createdBy": 1,`	`"creator": { "id": 1 },`
`"updatedBy": 2,`	`"updater": { "id": 2 },`

Operations and Methods

Support for basic CRUD (Create, Read, Update, and Delete) operations across most platform objects.

NOTE: Some of these specific operations may not be supported in the current release. For a complete list, see *API Endpoints*.

Operation	HTTP Method	Example URL	Notes
Create	`POST`	`/v3/people`	
	`POST`	`/v3/jobResults`	
Read	`GET`	`/v3/people/1`	`1` = internal user Id
	`GET`	`/v3/jobResults/10`	`10` = internal job Id
	`GET`	`/v3/people/1/jobGroups`	
	`GET`	`/v3/jobGroups/4/flowNode`	`flowNode` is a singular reference. Most resource names are plural.
List	`GET`	`/v3/people`	
	`GET`	`/v3/jobResults`	
Update	`PATCH`	`/v3/people/1`	Partial replacement
	`PATCH`	`/v3/jobResults/10`	Partial replacement
	`PUT`	`/v3/people/1`	Full replacement
	`PUT`	`/v3/jobResults/10`	Full replacement
Delete	`DELETE`	`/v3/people/1`	

	DELETE	/v3/jobResults/10	

Embedding Associations

An association can be referenced using the above URL structuring or by applying the `embed` query parameter as part of the reference to the specific resource. Example:

```
https:/wrangler.example.com/v3/jobGroups/6?embed=flowNode
```

Example response:

```
{
  "id": 6,
  "description": "A nifty job group",
  "flowNode": {
    "id": 1,
    "script": {
      "id": 1
    },
    "terminal": true
    ...
  }
}
```

The `id` value of the association is always included in the response.

Media Type Headers

NOTE: Some endpoints may accept and return a custom media type. These endpoints are documented.

Action	Header	Required?
Client request that expects a response body	request header: `Accept: application/json`	should include
Client request that includes a request body	request header: `Content-Type: application/json`	required
Server response that includes a response body	response header: `Content-Type: application/json`	required

Authentication

The REST APIs use the same authentication methods as the UI. Each call to an API endpoint must include authentication credentials for a user with access to the requested objects. See *API Authentication*.

SSL

If SSL has been enabled for the Trifacta platform, requests to URL endpoints are automatically redirected to the HTTPS equivalent.

Upload

Single-file upload is supported.

Multi-file upload is not supported.

Versioning and Endpoint Lifecycle

> **NOTE:** API versioning is not synchronized to specific releases of Trifacta Wrangler Enterprise. For example, some API endpoints for v4 may be updated, while v3 instances of the API endpoint are still supported. APIs are designed to be backward compatible.

APIs are designed to be backward compatible so that scripts and other tooling built on a previous version of an endpoint remain valid until the previous version has reached end-of-life. Each API is supported across a window of Trifacta Wrangler Enterprise releases, after which you must reference a newer version of the API.

API endpoint routes support a consistent structuring and do not contain business logic.

Version information is available at the following endpoint:

```
<http/https>://<my_server>:<port_number>/<version>/version
```

For more information, see *API Version Support Matrix*.

HTTP Status Codes and Errors

Request Method	Request Endpoint	HTTP Status Code (success)
`POST`	`/v3/<resource>`	201 Created
`GET`	`/v3/<resource>`	200 OK
`GET`	`/v3/<resource>/<id>`	304 Not Modified when client has cached version. See *Caching* below.
`PATCH`	`/v3/<resource>/<id>`	200 OK
`PUT`	`/v3/<resource>/<id>`	200 OK
`DELETE`	`/v3/<resource>/<id>`	204 No Content

The following error codes can apply to any of the above requests:

> **NOTE:** 5xx status codes may be generated by the server.

HTTP Status Code (client errors)	Notes
400 Bad Request	Potential reasons: • Resource doesn't exist. • Request is incorrectly formatted. • Request contains invalid values.
403 Forbidden	Incorrect permissions to access the Resource.
404 Not Found	Resource cannot be found.
410 Gone	Resource has been previously deleted.
415 Unsupported Media Type	Incorrect `Accept` header.

Caching

When a resource has been cached in the client, the client may set an `If-Modified-Since` header in HTTP date format on the request. If so:

General Response	HTTP Status Code
Returns full modified resource	200 OK
Returns an empty response for unmodified resource	304 Not Modified

Use Cases

REST API Tasks

By chaining together sequences of calls to API endpoints, you can create, read, update, and delete objects using identifiers accessible through the returned JSON. For more information, see *API Endpoints*.

For more information on endpoint workflows, see *API Workflows*.

UI Integrations

The REST APIs can also be used for integrating the core transformation experience of the Trifacta platform into a third-party application. Using a series of URL-based calls, you can retrieve and display specified datasets in the Transformer page, where authenticated users can wrangle datasets controlled by the third-party application. See *API - UI Integrations*.

About This Documentation

- Unless otherwise noted, the documentation and examples apply to version 3 (v3) of the Trifacta platform APIs.
- Examples may require modification to work in your environment.

API Authentication

Contents:

The Trifacta REST APIs support the following methods of authentication.

> **NOTE:** You must submit authentication credentials with each request to the platform.

Required Permissions

Authenticating user must be a valid user of the deployed instance of the Trifacta platform.

Basic Authentication

As request parameters, you can submit username/password under Basic Auth to any REST API endpoint.

> **NOTE:** The user must have permissions to execute the endpoint action.

Example:

This example submits authentication requirements over HTTP, including the username and password (`me@example.com:me_pwd`):

```
$ curl  -u me@example.com:me_pwd \
    -b ~/cookies.txt -c ~/cookies.txt \
    http://<platform_host>:<platform_port_number>/v3/<endpoint>
```

where:

Parameter	Description
`-u me@example.com:me_pwd`	Required username and password.
`-b` and `-c`	Required paths and filenames for storage of send and receive HTTP cookies.
`<platform_host>`	Fully qualified name of the host of the Trifacta platform
`<platform_port_number>`	Port number through which to access the Trifacta platform. Default is `3005`.

SSO Authentication

In a single-sign on environment, you can use basic authentication to interact with the APIs.

NOTE: Enabling SSO integration with the Trifacta platform requires additional configuration. See*Configure SSO for AD-LDAP*.

However, some changes are required:

- Basic authentication to the gateway must be enabled as part of the configuration for the reverse proxy. This feature is enabled by default, but please verify that it has not been explicitly disabled in your environment. For more information, see *Configure SSO for AD-LDAP*.
- You must authenticate using the SSO principal as the username and the LDAP or AD password associated with that user.
- You must authenticate to the SSO gateway. In the Trifacta platform, this value corresponds to the `<platform_host>:<platform_port_number>` value.

Example:

```
$ curl  -u myUser@example.com:foobar -x
http://<platform_host>:<platform_port_number> \
    -b ~/cookies.txt -c ~/cookies.txt \
    http://<platform_host>:<platform_port_number>/v3/<endpoint>
```

NOTE: For the protocol identifier, you can also use `https` if SSL is enabled. See *Install SSL Certificate*.

Parameter	Description
`myUser@example.com:foobar`	LDAP principal and password associated with that username.

For more information, see *Configure SSO for AD-LDAP*.

Kerberos Authentication

In a Kerberos environment, credentials must be submitted with each request using the SPNEGO Auth method.

- Kerberos is a network authentication protocol for client/server applications.
- SPNEGO provides a mechanism for extending Kerberos to Web applications through HTTP.

- For more information on the differences, see *https://msdn.microsoft.com/en-us/library/ms995330.aspx#http-sso-2_topic2*.

Credentials are authenticated by the KDC for each request.

> **NOTE:** SPNEGO must be enabled and configured for your REST client or programming library.

Example 1 - Embedded in Java:

SPNEGO requires a custom client. The following SPNEGO client enables submission of URL-based authentication parameters from within Java: *http://docs.oracle.com/javase/6/docs/technotes/guides/security/jgss/lab/part5.html*

Example 2 - Using cURL:

To use cURL:

1. Verify that your version of cURL supports GSS:

```
$ curl -V
curl 7.51.0 (x86_64-apple-darwin16.0) libcurl/7.51.0 SecureTransport
zlib/1.2.8
Protocols: dict file ftp ftps gopher http https imap imaps ldap ldaps
pop3 pop3s rtsp smb smbs smtp smtps telnet tftp
Features: AsynchDNS IPv6 Largefile GSS-API Kerberos SPNEGO NTLM
NTLM_WB SSL libz UnixSockets
```

2. Verify that `GSS-API` and `SPNEGO` are in the output.
3. Run `kinit` and authenticate using the hadoop principal:

```
$ kinit
Please enter the password for [hadoop.user.principal]@localhost:
$
```

4. Access using `cURL`:

```
$ curl --negotiate -u anything \
    -b ~/cookies.txt -c ~/cookies.txt \
    http://<platform_host>:<platform_port_number>/v3/<endpoint>
```

where:

Parameter	Description
`--negotiate`	Enables SPNEGO use in cURL. This option requires a library built with GSS-API or SSPI support. If this option is used several times, only the first one is used. Use `--proxy-negotiate` to enable Negotiate (SPNEGO) for proxy authentication.
`-u anything`	Required username. However, this username is ignored. Instead, the principal used in `kinit` is applied.

For more information:

- *https://hadoop.apache.org/docs/r2.7.3/hadoop-auth/Examples.html*
- *https://msdn.microsoft.com/en-us/library/ms995329.aspx*

Logout

Since each request requires credentials, logging out is not required.

API Endpoints

The following endpoints are available in this release of Trifacta Wrangler Enterprise. Please verify that you are referring to the correct version of the endpoint.

Topics:

v4 Endpoints

NOTE: This feature is in Beta release.

These endpoints apply to version 4 of the APIs for the Trifacta platform.

- For more information on support for this version, see *API Version Support Matrix*.

Topics:

API JobGroups Create v4

NOTE: This feature is in Beta release.

Contents:

Create a jobGroup, which launches the specified job as the authenticated user.

The request specification depends on one of the following conditions:

- Dataset has already had a job run against it and just needs to be re-run.
- Dataset has not had a job run, or the job definition needs to be re-specified.

NOTE: In this release, you cannot execute jobs sourced from datasets in Redshift or SQL DW or publish to these locations via the API. This known issue will be fixed in a future release.

Version: `v4`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `POST`

Endpoint:

```
/v4/jobGroups
```

Request Body - Re-run job:

If you are re-running a job that has already executed and do not need to modify any job settings, you can use the following simplified body to launch it:

```
{
  "wrangledDataset": {
    "id": 7
  }
}
```

Request Body - Specify job:

If you are specifying a new job or must re-run a job with new settings, you must include a version of the following request body. Required parameters are listed below:

```
{
  "wrangledDataset": {
    "id": 1
  },
  "overrides": {
    "execution": "photon",
    "profiler": false,
    "writesettings": [
      {
        "path":
"hdfs://hadoop:50070/trifacta/queryResults/admin@trifacta.local/cdr_txt.cs
v",
        "action": "create",
        "format": "csv",
        "compression": "none",
        "header": false,
        "asSingleFile": false
      }
    ]
  },
  "ranfrom": null
}
```

Request Body - Specify job for dataset with parameters:

If you have created a dataset with parameters, you can specify overrides for parameter values during execution through the APIs. Through this method, you can iterate job executions across all matching sources of a parameterized dataset. For more information on creating datasets with parameters, see *Overview of Parameterization*.

In the following, the `runParameters` override has been specified for the `varRegion`. In this case, the value `02` is inserted for the specified variable as part of the job execution.

```
{
  "wrangledDataset": {
    "id": 33
  },
  "overrides": {
    "execution": "photon",
    "profiler": false,
    "writesettings": [
      {
        "path":
"hdfs://hadoop:50070/trifacta/queryResults/admin@trifacta.local/cdr_txt.cs
v",
        "action": "create",
        "format": "csv",
        "compression": "none",
        "header": false,
        "asSingleFile": false
      }
    ],
    "runParameters": {
      "overrides": {
        "data": [{
          "key": "varRegion",
          "value": "02"
        }
      ]}
    },
  },
  "ranfrom": null
}
```

Response

Response Status Code - Success: `201 - Created`

Response Body Example:

```
{
    "reason": "JobStarted",
    "sessionId": "eb3e98e0-02e3-11e8-a819-25c9559a2a2c",
    "id": 9,
    "jobs": {
        "data": [
            {
                "id": 12
            },
            {
                "id": 13
            }
        ]
    }
}
```

Reference

Request Reference:

Property	Description
`wrangledDataset`	(required) Internal identifier for the object whose results you wish to generate. The recipes of all preceding datasets on which this dataset depends are executed as part of the job.
`overrides.execution`	(required, if first time running the job) Indicates the running environment on which the job is executed. Accepted values: • `photon` • spark For more information, see *Running Environment Options*.
`overrides.profiler`	(required, if first time running the job) When set to `true`, a visual profile of the job is generated as specified by the profiling options for the platform. See *Profiling Options*.
`overrides.writesettings`	(required, if first time running the job) These settings define the publishing options for the job. See below.
`ranfrom`	(optional) If this value is set to `null`, then the job does not show up in the Job Results page. If set to `cli`, the job appears as a CLI job. See *Job Results Page*.

writesettings Reference:

The `writesettings` values allow you to specify aspects of the publication of results to the specified `path` location.

NOTE: `writesettings` values are required if you are running this specified job for the dataset for the first time.

NOTE: To specify multiple outputs, you can include additional `writesettings` objects in the request. For example, if you want to generate output to `csv` and `json`, you can duplicate the `writesettings` object for `csv` and change the `format` value in the second one to `json`.

These settings correspond to values that you can apply through the UI or through the command line interface.

- For UI information, see *Run Job Page*.
- For CLI information, see *CLI for Jobs*.

Property	Description

`path`	(required) The fully qualified path to the output location where to write the results
`action`	(required) If the output file or directory exists, you can specify one of the following actions: • `create` - Create a new, parallel location, preserving the old results. • `append` - Add the new results to the old results. • `overwrite` - Replace the old results with the new results.
`format`	(required) Output format for the results. Specify one of the following values: • `csv` • `json` • `avro` • `pqt` **NOTE:** Parquet format requires execution on a Hadoop running environment (`overrides.execution` must be `spark`). **NOTE:** To specify multiple output formats, create additional `writesettings` object for each output format.
`compression`	(optional) For `csv` and `json` results, you can optionally compress them using `bzip2` or `gzip` compression. Default is `none`.
`header`	(optional) For `csv` results with `action` set to `create` or `append`, this value determines if a header row with column names is inserted at the top of the results. Default is `false`.
`asSingleFile`	(optional) For `csv` and `json` results, this value determines if the results are concatenated into a single file or stored as multiple files. Default is `false`.

Response reference:

Property	Description
reason	Current state of the job group at time of API call. Since this call creates the job group, this value is always `Job started` in the response to this call.
sessionId	Session identifier for the job group.
id	Internal identiifer of the job group.
jobs.data.id	Internal identifier of the individual job or jobs created as part of this job group.

v3 Endpoints

These endpoints apply to version 3 of the APIs for the Trifacta platform.

- For more information on support for this version, see *API Version Support Matrix*.

Connections

Endpoint	Action	Behavior	Documentation
`/connections`	`POST`	Create	*API Connections Create v3*
`/connections`	`GET`	List	*API Connections Get List v3*
`/connections/:id`	`GET`	Read	*API Connections Get v3*
`/connections/:id/status`	`GET`	Read Status	*API Connections Get Status v3*
`/connections`	`DELETE`	Delete	*API Connections Delete v3*

Datasets and Recipes

Endpoint	Action	Behavior	Documentation
`/importedDatasets`	`POST`	Create	*API ImportedDatasets Create v3*
`/importedDatasets`	`GET`	List	*API ImportedDatasets Get List v3*
`/importedDatasets/:id`	`GET`	Read	*API ImportedDatasets Get v3*
`/importedDatasets/:id`	`DELETE`	Delete	*API ImportedDatasets Delete v3*
`/v3/importedDatasets/:id/addToFlow`	`POST`	Create	*API ImportedDatasets Post AddToFlow v3*
`/wrangledDatasets`	`POST`	Create	*API WrangledDatasets Create v3*
`/wrangledDatasets`	`GET`	List	*API WrangledDatasets Get List v3*
`/wrangledDatasets/:id`	`GET`	Read	*API WrangledDatasets Get v3*
`/wrangledDatasets/:id`	`DELETE`	Delete	*API WrangledDatasets Delete v3*
`/wrangledDatasets/:id/primaryInputDatasets`	`GET`	Read	*API WrangledDatasets Get PrimaryInputDataset v3*
`/wrangledDatasets/:id/primaryInputDatasets`	`PUT`	Update	*API WrangledDatasets Put PrimaryInputDataset v3*

Flows

Endpoint	Action	Behavior	Documentation
`/flows`	`POST`	Create	*API Flows Create v3*
`/flows`	`GET`	List	*API Flows Get List v3*
`/flows/:id`	`GET`	Read	*API Flows Get v3*
`/flows/:id`	`PATCH`	Update	*API Flows Patch v3*
`/flows/:id`	`DELETE`	Delete	*API Flows Delete v3*

Flow import and export

Endpoint	Action	Behavior	Documentation
`/flows/package/dryRun`	`POST`	Import dry run	*API Flows Package Post DryRun v3*
`/flows/package`	`POST`	Import	*API Flows Package Post v3*
`/flows/:id/package/dryRun`	`GET`	Export dry run	*API Flows Package Get DryRun v3*
`/flows/:id/package`	`GET`	Export	*API Flows Package Get v3*

Jobgroups and Jobs

Endpoint	Action	Behavior	Documentation
`/jobGroups`	`POST`	Create	*API JobGroups Create v3*
`/jobGroups`	`GET`	List	*API JobGroups Get List v3*
`/jobGroups/:id`	`GET`	Read	*API JobGroups Get v3*
`/jobGroups/:id/jobs`	`GET`	Read	*API JobGroups Get Jobs v3*
`/jobGroups/:id/status`	`GET`	Read Status	*API JobGroups Get Status v3*
`/jobGroups/:id/publish`	`PUT`	Create	*API JobGroups Put Publish v3*
`/jobGroups/:id`	`DELETE`	Delete	*API JobGroups Delete v3*

Deployments and Releases

Endpoint	Action	Behavior	Documentation
`/deployments`	`POST`	Create	*API Deployments Create v3*
`/deployments`	`GET`	List	*API Deployments Get List v3*
`/deployments/:id`	`GET`	Read	*API Deployments Get v3*
`/deployments/:id`	`PATCH`	Update	*API Deployments Patch v3*
`/deployments:/:id/objectImportRules`	`PATCH`	Update	*API Deployments Object Import Rules Patch v3*
`/deployments:/:id/valueImportRules`	`PATCH`	Update	*API Deployments Value Import Rules Patch v3*
`/deployments/:id?embed=releases`	`GET`	List	*API Deployments Get Release List v3*
`/deployments/:id/run`	`POST`	Create	*API Deployments Run v3*
`/deployments/:id`	`DELETE`	Delete	*API Deployments Delete v3*
`/deployments/:id/releases`	`POST`	Create	*API Releases Create v3*
`/deployments/:id/releases/dryRun`	`POST`	Create	*API Releases Create DryRun v3*
`/releases:id`	`GET`	Read	*API Releases Get v3*
`/releases/:id`	`PATCH`	Update	*API Releases Patch v3*
`/releases/:id`	`DELETE`	Delete	*API Releases Delete v3*

Users

Endpoint	Action	Behavior	Documentation
`/people`	`POST`	Create	*API People Create v3*
`/people`	`GET`	List	*API People Get List v3*
`/people`	`PATCH`	Update	*API People Patch v3*
`/people/:id`	`GET`	Read	*API People Get v3*
`/people/:id`	`DELETE`	Delete	*API People Delete v3*

Miscellaneous

Endpoint	Action	Behavior	Documentation
`/session`	`GET`	Read	*API Session Get*

API Connections Create v3

Contents:

Create the specified connection.

NOTE: In this release, you cannot create Redshift or SQL DW connections via the API. Please create these connections through the application. This known issue will be fixed in a future release.

For more information on connections, see *CLI for Connections*.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `POST`

Endpoint:

```
/v3/connections/
```

Request Body - Relational Connection:

For relational connections, the request body should look like the following. All properties are required unless noted.

> **NOTE:** Relational connections require the creation and installation of an encryption key file on the Trifacta node. This file must be present before the connection is created. See *Create Encryption Key File*.

This example creates a SQL Server connection of `basic` type. A valid username/password combination must be specified in the `credentials` p roperty.

```
{
     "name": "sqlserver",
     "description": "",
     "isGlobal": false,
     "type": "jdbc",
     "host": "sqlserver.example.com",
     "port": 1433,
     "vendor": "sqlserver",
     "params": {
       "connectStrOpts": ""
     },
     "ssl": false,
     "credentialType": "basic",
     "credentials": [
        {
          "username": "<username>",
          "password": "<password>"
        }
     ]
}
```

Property	Description
name	Display name of the connection
description	(Optional) Display description for the connection
isGlobal	(Optional) If `true`, the connection is available to all users. The default is `false`.

type	For more information on the value to insert for the connection, see *Connection Types*.
host	Host name of the relational server to which to connect.
port	Port number for the relational server. The default value varies between database vendors. For more information, please see the documentation provided with your database distribution.
vendor	For more information on the value to insert for the connection, see *Connection Types*.
params	(Optional) Set of JSON parameters that are passed to the database when initializing the connection. Depending on the database vendor, you may be required to submit via this parameter the name of the default database. You can also pass in optional parameters through the `ConnecStrOpts` parameter. For more information, see *CLI for Connections*.
ssl	(Optional) If set to `true`, the connection is made over SSL. The default is `false`. **NOTE:** If you connect over SSL, you must modify the hostname value to use HTTPS. **NOTE:** SSL connections to SQL Server are not supported.
credentialType	Set this value to one of the following: • `basic` - Simple username/password to be provided in the `credentials` property. • `conf` - Use the connection credentials stored in `trifacta-conf.json`. • `custom` - Connection credentials are specified in the `params` property, such as using key-value parameters to specify access credentials.
credentials	(Optional) If `credentialType=basic`, this property must contain the username and password to use to connect to the relational source.

Request Body - Hive Connection:

You can create create only one public connection to Hive.

```
{
  "host": "hadoop",
  "port": 10000,
  "vendor": "hive",
  "params": {
      "jdbc": "hive2",
      "connectStrOpts": "",
      "defaultDatabase": ""
  },
  "ssl": false,
  "name": "hiveAPI",
  "description": "Hive conn via API",
  "type": "jdbc",
  "isGlobal": true,
  "credentialType": "conf",
  "credentialsShared": true
}
```

The following property values are specific to Hive connections.

Property	Description

`host`	Set this value to `hadoop` to integrate with the Hive instance for the Hadoop cluster to which the Trifacta platform is connected.
`params`	Connection parameters for the Hive instance. **NOTE:** The following parameter entry is required for Hive: `"jdbc": "hive2",` Others are optional.
`type`	Set this value to `jdbc`.
`isGlobal`	**NOTE:** For Hive connections, this value must be set to `true`.
`credentialType`	Set this value to `conf`.

For more information, see *Configure for Hive*.

Request Body - Redshift Connection:

```
{
  "host": "redshift.example.net",
  "port": 5439,
  "vendor": "redshift",
  "params": {
      "connectStrOpts": "",
      "defaultDatabase": "dev",
      "extraLoadParams": "BLANKSASNULL EMPTYASNULL TRIMBLANKS
TRUNCATECOLUMNS"
  },
  "ssl": true,
  "name": "redshift",
  "description": "Redshiftconn",
  "type": "jdbc",
  "isGlobal": true,
  "credentialType": "custom",
  "credentialsShared": true,
  "credentials": [
    {"key":"user","value":"<userId>"},
    {"key":"password","value":"<PWD>"},
    {"key":"iamRoleArn","value":"<IAM_role_ARN>"}
  ]
}
```

The following property values are specific to Redshift connections.

Property	Description

`params`	A default database value is required. The `extraLoadParams` value is used when you publish results to Redshift. For more information on these values, see *http://docs.aws.amazon.com/redshift/latest/dg/copy-parameters-data-conversion.html*.
`type`	Set this value to `jdbc`.
`credentialType`	Set this value to `custom`. Credentials are specified below.
`credentials`	`username` and `password` must be specified in this key-value format, although the value for either can be an empty string. **NOTE:** `iamRoleArn` is optional. For more information, see *Configure for EC2 Role-Based Authentication*.

For more information on parameters and credentials, see *Create Redshift Connections*.

Response

Response Status Code - Success: `201 - Created`

Response Body Example:

```
{
    "connectString": "jdbc:sqlserver://sqlserver.example.com:1433",
    "id": 5,
    "host": "sqlserver.example.com",
    "port": 1433,
    "vendor": "sqlserver",
    "params": {
        "connectStrOpts": ""
    },
    "ssl": false,
    "name": "sqlserver",
    "description": "",
    "type": "jdbc",
    "createdBy": 1,
    "isGlobal": false,
    "credentialType": "basic",
    "createdAt": "2017-07-05T18:00:19.165Z",
    "updatedAt": "2017-07-05T18:00:19.165Z",
    "updatedBy": 1,
    "credentials": [
        {
            "username": "<username>"
        }
    ]
}
```

Reference

For more information on the response body properties, see *API Connections Get v3*.

API Connections Delete v3

Contents:

Delete the specified connection.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `DELETE`

Endpoint:

```
/v3/connections/<id>
```

where:

Parameter	Description
`<id>`	Internal identifier for the connection

Request URI - Example:

```
/v3/connections/4
```

Request Body:

Empty.

Response

Response Status Code - Success: `204 - No Content`

Response Body Example:

Empty.

Reference

None.

API Connections Get List v3

Contents:

Get the list of accessible connections for the authenticated user.

For more information on connections, see *CLI for Connections*.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/connections
```

Endpoint with paged retrieval:

By default, this endpoint returns results in sets of 25.

You can apply query parameters to change the size of the default set and to page through result sets. The following example queries for results 100 at a time. In this case, the query asks for results 201-300:

```
/v3/connections?limit=100%offset=2
```

If the count of retrieved results is less than the limit, you have reached the end of the results.

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
  "data": [
    {
      "connectString":
"jdbc:postgresql://localhost:5432/trifacta?ssl=true",
      "id": 9,
      "host": "localhost",
      "port": 5432,
      "vendor": "postgres",
      "params": {
        "database": "trifacta"
      },
      "ssl": true,
      "name": "dc31bef0_e7ea_11e6_8c19_4f07370e0072",
      "description": null,
      "type": "jdbc",
      "createdBy": 2,
      "isGlobal": false,
      "credentialType": "basic",
      "createdAt": "2017-01-31T19:24:37.167Z",
    "updatedAt": "2017-01-31T19:24:37.167Z",
    "updatedBy": 2,
    "credentials": [
      {
        "username": "trifacta"
      }
    ]
  },
  {
    "connectString": "jdbc:postgresql://does-not-exist:5432/trifacta",
    "id": 7,
    "host": "does-not-exist",
    "port": 5432,
    "vendor": "postgres",
    "params": {
      "database": "trifacta"
    },
    "ssl": false,
    "name": "dc3197e0_e7ea_11e6_8c19_4f07370e0072",
    "description": null,
    "type": "jdbc",
    "createdBy": 2,
    "isGlobal": false,
    "credentialType": "basic",
    "createdAt": "2017-01-31T19:24:31.899Z",
    "updatedAt": "2017-01-31T19:24:31.899Z",
    "updatedBy": 2,
    "credentials": [
      {
        "username": "trifacta"
      }
    ]
```

```
        }
      ]
    }
```

Reference

For more information on the properties of a connection, see *API Connections Get v3*.

API Connections Get Status v3

Contents:

Run a connection test for the specified connection.

For more information on connections, see *CLI for Connections*.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/connections/<id>/status
```

where:

Parameter	Description
`<id>`	Internal identifier for the connection

Request URI - Example:

```
/v3/connections/10/status
```

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
  "result": "SUCCESS",
  "reason": null
}
```

Reference

Property	Description
result	Results of testing the connection. • For more information on debugging failures in relational connections, see *Enable Relational Connections*. • For more information on debugging Hive connections. see *Configure for Hive*. • For more information on debugging S3 connections, see *Enable S3 Access*.
reason	If the `result` value is not `SUCCESS`, additional information may be included here.

API Connections Get v3

Contents:

Get the specified connection.

For more information on connections, see *CLI for Connections*.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/connections/<id>
```

where:

Parameter	Description
`<id>`	Internal identifier for the connection

Request URI - Example:

```
/v3/connections/3
```

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
    "data": [
        {
            "connectParams": {
                "vendor": "postgres",
                "host": "localhost",
                "port": "5432",
                "database": "trifacta"
            },
            "id": 10,
            "host": "localhost",
            "port": 5432,
            "vendor": "postgres",
            "params": {
                "connectStrOpts": "",
                "database": "trifacta"
            },
            "ssl": false,
            "name": "postgres",
            "description": "",
            "type": "jdbc",
            "createdBy": 1,
            "isGlobal": false,
            "credentialType": "basic",
            "credentialsShared": true,
            "uuid": "7d173c90-c4e1-11e7-a768-71cd1fa636c3",
            "createdAt": "2017-11-09T00:04:00.345Z",
            "updatedAt": "2017-11-09T00:04:00.345Z",
            "updatedBy": 1,
            "credentials": [
                {
                    "username": "<userId>"
                }
            ]
        },
        {
            "connectParams": {
                "vendor": "hive",
                "host": "hadoop",
                "port": "10000",
                "jdbc": "hive2",
                "defaultDatabase": "default"
            },
            "id": 1,
            "host": "hadoop",
            "port": 10000,
            "vendor": "hive",
            "params": {
                "jdbc": "hive2",
                "connectStringOptions": "",
                "defaultDatabase": "default"
            },
            "ssl": false,
            "name": "hive",
            "description": null,
            "type": "jdbc",
            "createdBy": 1,
            "isGlobal": true,
            "credentialType": "conf",
            "credentialsShared": true,
            "uuid": "ae41c5a0-c460-11e7-8163-c3c02bb1fb0b",
            "createdAt": "2017-11-08T08:41:57.755Z",
            "updatedAt": "2017-11-08T08:41:57.755Z",
            "updatedBy": 1,
            "credentials": []
        }
    ],
    "count": {
        "owned": 2,
        "shared": 0,
        "count": 2
    }
}
```

Reference

Property	Description
connectParams.vendor	The type of connection. See *Connection Types*.
connectParams.host	Host of the source
connectParams.port	Port number for the source
connectParams.database	Name of the default database (if applicable)
id	Internal identifier for the connection
host	Host of the source
port	Port number for the source
vendor	String identifying the connection's vendor
params	This setting is populated with any parameters that are passed to the source during connection and operations. For relational sources, this setting may include the default database.
ssl	When `true`, the Trifacta platform uses SSL to connect to the source.
name	Internal name of the connection
description	User-friendly description for the connection
type	Type of connection
createdBy	Internal identifier for the user who created the connection
isGlobal	If `true`, the connection is public and available to all users. **NOTE:** After a connection has been made public, it cannot be made private again. It must be deleted and recreated. Default is `false`. A connection can be made public through the command line interface or the Connections page. See *Connections Page*.
credentialType	The type of credentials used for the connection. This value varies depending on where the credentials are stored. See *CLI for Connections*.
credentialsShared	If `true`, the credentials used for the connection are available for use by users who have been shared the connection.
uuid	A universal object identifier, which is unique across instances of the platform. This internal identifier is particularly useful when create import mapping rules. • See *API Deployments Value Import Rules Patch v3*. • See *API Deployments Object Import Rules Patch v3*.
createdAt	Timestamp for when the connection was made
updatedAt	Timestamp for when the connection was last updated
updatedBy	Internal identifier for the user who last updated the connection
credentials	If present, these values are the credentials used to connect to the database. **NOTE:** For security reasons, you can store the connection's credentials in an external file on the Trifacta Server, after which they do not appear in this setting. See *CLI for Connections*.

API Deployments Create v3

Contents:

Create a new deployment.

> **NOTE:** Deployments pertain to Production instances of the Trifacta® platform. For more information, see *Overview of Deployment Management*.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `POST`

Endpoint:

```
/v3/deployments/
```

Request Body:

```
{
    "name": "Test Deployment"
}
```

Response

Response Status Code - Success: `201 - Created`

Response Body Example:

```
{
    "id": 20,
    "name": "Test Deployment",
    "createdBy": 1,
    "updatedBy": 1,
    "updatedAt": "2017-10-12T23:48:54.340Z",
    "createdAt": "2017-10-12T23:48:54.340Z"
}
```

Reference

For more information on properties of a deployment, see *API Deployments Get v3*.

API Deployments Delete v3

Contents:

Delete the specified deployment.

Deleting a deployment removes all releases, packages, and flows underneath it. This step cannot be undone.

NOTE: Deployments pertain to Production instances of the Trifacta® platform. For more information, see *Overview of Deployment Management*.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `DELETE`

Endpoint:

```
/v3/deployments/<id>
```

where:

Parameter	Description
`<id>`	Internal identifier for the deployment

Request URI - Example:

```
/v3/deployments/4
```

Request Body:

Empty.

Response

Response Status Code - Success: `204 - No Content`

Response Body Example:

Empty.

Reference

None.

API Deployments Get List v3

Contents:

Get the list of accessible deployments for the authenticated user.

> **NOTE:** Deployments pertain to Production instances of the Trifacta® platform. For more information, see *Overview of Deployment Management*.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/deployments
```

Endpoint with paged retrieval:

By default, this endpoint returns results in sets of 25.

You can apply query parameters to change the size of the default set and to page through result sets. The following example queries for results 100 at a time. In this case, the query asks for results 201-300:

```
/v3/deployments?limit=100%offset=2
```

If the count of retrieved results is less than the limit, you have reached the end of the results.

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
    "data": [
        {
            "id": 2,
            "name": "Deployment 2",
            "createdAt": "2017-10-12T17:45:18.485Z",
            "updatedAt": "2017-10-12T17:45:18.485Z",
            "createdBy": 1,
            "updatedBy": 1
        },
        {
            "id": 1,
            "name": "My First Deployment",
            "createdAt": "2017-10-10T00:36:49.278Z",
            "updatedAt": "2017-10-10T00:36:49.278Z",
            "createdBy": 1,
            "updatedBy": 1
        }
    ],
    "count": 2
}
```

Reference

For more information on the properties of a deployment, see *API Deployments Get v3*.

API Deployments Get Release List v3

Contents:

Get the list of releases for the specified deployment for the authenticated user.

> **NOTE:** Deployments and releases pertain to Production instances of the Trifacta® platform. For more information, see *Overview of Deployment Management*.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/deployments/:id?embed=releases
```

Endpoint with paged retrieval:

By default, this endpoint returns results in sets of 25.

You can apply query parameters to change the size of the default set and to page through result sets. The following example queries for results 100 at a time. In this case, the query asks for results 201-300:

```
/v3/deployments/:id?embed=releases&limit=100%offset=2
```

If the count of retrieved results is less than the limit, you have reached the end of the results.

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
    "id": 1,
    "name": "Payment10-SteveO",
    "createdAt": "2017-09-26T07:00:00.000Z",
    "updatedAt": "2017-10-12T23:47:56.801Z",
    "createdBy": 1,
    "updatedBy": 1,
    "releases": [
        {
            "id": 2,
            "notes": "Testing with a new format",
            "packageUuid": "11d472a0-a799-11e7-9c5c-9dd7feba47aa",
            "active": null,
            "createdAt": "2017-10-02T19:07:24.311Z",
            "updatedAt": "2017-10-05T12:21:46.177Z",
            "deploymentId": 1,
            "createdBy": 1,
            "updatedBy": 1
        },
        {
            "id": 1,
            "notes": null,
            "packageUuid": "6648f8c0-a9e6-11e7-a092-8394937c7038",
            "active": true,
            "createdAt": "2017-10-05T16:01:27.881Z",
            "updatedAt": "2017-10-12T20:07:42.143Z",
            "deploymentId": 1,
            "createdBy": 1,
            "updatedBy": 1
        }
    ]
}
```

Reference

For more information on the properties of a release, see *API Releases Get v3*.

API Deployments Get v3

Contents:

Get the specified deployment.

NOTE: Deployments pertain to Production instances of the Trifacta® platform. For more information, see *Overview of Deployment Management*.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/deployments/<id>
```

where:

Parameter	Description
`<id>`	Internal identifier for the deployment

Request URI - Example:

```
/v3/deployments/3
```

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
    "id": 3,
    "name": "Payments Deployment",
    "createdAt": "2017-09-27T07:00:00.000Z",
    "updatedAt": "2017-09-30T00:17:00.079Z",
    "createdBy": 1,
    "updatedBy": 2
}
```

Reference

Property	Description
id	Internal identifier for the deployment
name	Display name for the deployment. This value appears in the user interface.
createdAt	Timestamp for when the deployment was created.
updatedAt	Timestamp for when the deployment was last updated.
createdBy	Internal identifier for the user who created the deployment.
updatedBy	Internal identifier for the user who last updated the deployment.

API Deployments Object Import Rules Patch v3

Contents:

Create a list of object-based import rules for the specified deployment. Delete all previous rules applied to the same object.

> **NOTE:** Import rules must be applied to individual deployments.

The generated rules apply to all flows that are imported into the deployment after they has been created.

> **NOTE:** Deployments pertain to Production instances of the Trifacta® platform. For more information, see *Overview of Deployment Management*.

The response contains any previously created rules that have been deleted as a result of this change.

You can also make replacements in the import package based on value mappings. See *API Deployments Value Import Rules Patch v3*.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `PATCH`

Endpoint:

```
/v3/deployments/<id>/objectImportRules
```

where:

Parameter	Description
`<id>`	Internal identifier for the deployment

Request URI - Example:

```
/v3/deployments/4/objectImportRules
```

Request Body Example: Replace connection

The following JSON array describes replacing the connection specified by the UUID, which is a field on the connection object exported from the original platform instance.   This connection reference is replaced by a reference to connection ID`1` in the local platform instance and is applied to any release uploaded into the deployment after the rule has been created:

```
[{"tableName":"connections","onCondition":{"uuid":"d75255f0-a245-11e7-8618
-adc1dbb4bed0"},"withCondition":{"id":1}}]
```

You can specify matching values using string literals.

Match Type	Example Syntax
string literal	`{"uuid":"d75255f0-a245-11e7-8618-adc1dbb4bed0"}`
string literal	`{"id":"22"}`

NOTE: Use of Trifacta patterns or regular expressions is not supported.

For more information on patterns, see *Text Matching*.

Request Body Example: Replace multiple connections

This example request includes replacements for multiple connection references.

NOTE: Rules are applied in the listed order. If you are applying multiple rules to the same object in the import package, the second rule must reference the expected changes applied by the first rule.

This type of replacement applies if the imported packages contain sources that are imported through two separate connections:

```
[

{"tableName":"connections","onCondition":{"uuid":"d75255f0-a245-11e7-8618-
adc1dbb4bed0"},"withCondition":{"id":1}},

{"tableName":"connections","onCondition":{"uuid":"d552045e0-c314-22b5-9410
-acd1bcd8eea2"},"withCondition":{"id":2}}
]
```

Response

Response Status Code - Success: `200 - OK`

The response body contains any previously created rules that have been deleted as a result of this update.

Response Body Example: All new rule, no deletions

If the update does not overwrite any previous rules, then no rules are deleted. So, the response looks like the following:

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

Response Body Example: Replace connection

If you submit the request again, the response contains the rule definition of the previous update, which has been deleted. This example applies to the one-rule change listed previously:

```
{
    "deleted": [
        {
            "onCondition": {
                "uuid": "d75255f0-a245-11e7-8618-adc1dbb4bed0"
            },
            "withCondition": {
                "id": 1
            },
            "id": 1,
            "tableName": "connections",
            "createdBy": 3,
            "updatedBy": 3,
            "createdAt": "2017-11-07T01:42:46.798Z",
            "updatedAt": "2017-11-07T01:42:46.798Z",
            "deploymentId": 4
        }
    ]
}
```

Reference

Property	Description
onCondition	The matching object identifier and the specified literal or pattern to match.
withCondition	The identifier for the object type, as specified in by the `tableName` value, which is being modified.
id	Internal identifier for the object import rule
tableName	Name of the table to which the mapping is applied. Values: • `connections` - applies to local connections
deploymentId	Internal identifier for the deployment to which to apply the import rule.

API Deployments Patch v3

Contents:

Update the specified deployment.

NOTE: Deployments pertain to Production instances of the Trifacta® platform. For more information, see *Overview of Deployment Management*.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `PATCH`

Endpoint:

```
/v3/deployments/<id>
```

where:

Parameter	Description
`<id>`	Internal identifier for the deployment

Request URI - Example:

```
/v3/deployments/2
```

Request Body: Example - Modify the deployment name

```
{
    "name": "New Deployment Name"
}
```

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
    "id": 2,
    "updatedBy": 1,
    "updatedAt": "2017-10-13T00:06:12.147Z"
}
```

Reference

For more information on the properties of a deployment, see *API Deployments Get v3*.

API Deployments Run v3

Contents:

Run the job for the active release of the specified deployment.

- At least one manual output must be specified for the main flow within the package. See *Flow View Page*.
- An active release must be specified for the deployment. See *API Releases Patch v3*.

NOTE: Deployments pertain to Production instances of the Trifacta® platform. For more information, see *Overview of Deployment Management*.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `POST`

Endpoint:

```
/v3/deployments/<id>/run
```

where:

Parameter	Description
`<id>`	Internal identifier for the deployment

Request URI - Example:

```
/v3/deployments/4/run
```

Request Body:

Empty.

Response

Response Status Code - Success: `201 - Created`

Response Body Example:

```
{
    "data": [
        {
            "reason": "JobStarted",
            "sessionId": "dd6a90e0-c353-11e7-ad4e-7f2dd2ae4621",
            "id": 33,
            "jobs": {
                "data": [
                    {
                        "id": 68
                    },
                    {
                        "id": 69
                    },
                    {
                        "id": 70
                    }
                ]
            }
        }
    ]
}
```

Reference

Property	Description
reason	Action undertaken on the endpoint.
sessionId	Internal identifier for the session of the job run
id	JobGroup identifier. For more information, see *API JobGroups Get v3*.
jobs.data.id	Internal identifier for the individual jobs that compose the job group being executed.

API Deployments Value Import Rules Patch v3

Contents:

Create a list of value-based import rules for the specified deployment. Delete any previous rules applied to the same values.

> **NOTE:** Import rules must be applied to individual deployments.

The generated rules apply to all flows that are imported into the Production instance after they have been created.

> **NOTE:** Deployments pertain to Production instances of the Trifacta® platform. For more information, see *Overview of Deployment Management*.

The response contains any previously created rules that have been deleted as a result of this change.

You can also make replacements in the import package based on object references. See *API Deployments Object Import Rules Patch v3*.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `PATCH`

Endpoint:

```
/v3/deployments/<id>/valueImportRules
```

where:

Parameter	Description
`<id>`	Internal identifier for the deployment

Request URI - Example:

```
/v3/deployments/4/valueImportRules
```

Request Body Example: Single value replacement

The following JSON array describes a single replacement rule for the S3 bucket name. In this case, the `wrangle-dev` bucket name has been replaced by the `wrangle-prod` bucket name, which means data is pulled in the Production deployment from the appropriate S3 bucket.

> **NOTE:** The executing user of any job must have access to any data source that is remapped in the new instance.

```
[{"type":"s3Bucket","on":"wrangle-dev","with":"wrangle-prod"}]
```

Request Body Example: Multiple value replacements

The following JSON array describes two replacements for the `fileLocation` values. In this case, rules are applied in succession.

> **NOTE:** Rules are applied in the listed order. If you are applying multiple rules to the same object in the import package, the second rule must reference the expected changes applied by the first rule.

```
[
  {"type":"fileLocation","on":"klamath","with":"klondike"},
  {"type":"fileLocation","on":"/\/dev\//","with":"/prod/"}
]
```

In the above:

- The first rule replaces the string `klamath` in the path to the source with the following value: `klondike`.
- The second rule performs a regular expression match on the string `/dev/`. Since the match is described using the regular expression syntax, the backslashes must be escaped. The replacement value is the following literal: `/prod/`.

You can specify matching values using the following types of matches:

Match Type	Example Syntax
string literal	`{"on":"d75255f0-a245-11e7-8618-adc1dbb4bed0"}`
regular expression	`{"on":"/[0-9a-zA-z]{8}-a245-11e7-8618-adc1dbb4bed0/"}`

NOTE: Use of Trifacta patterns is not supported.

For more information on patterns, see *Text Matching*.

Response

Response Status Code - Success: `200 - OK`

The response body contains any previously created rules that have been deleted as a result of this update.

Response Body Example: All new rule, no deletions

If the update does not overwrite any previous rules, then no rules are deleted. So, the response looks like the following:

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

Response Body Example: Replace file location, delete previous rule

If you submit the request again, the response contains the rule definition of the previous update, which has been deleted.

```
{
    "deleted": [
        {
            "on": "wrangle-dev",
            "id": 1,
            "type": "s3Bucket",
            "with": "wrangle-prod",
            "createdBy": 3,
            "updatedBy": 3,
            "createdAt": "2017-11-07T02:16:57.743Z",
            "updatedAt": "2017-11-07T02:16:57.743Z",
            "deploymentId": 1
        }
    ]
}
```

Reference

Property	Description
on	The specified literal or pattern to match.
id	Internal identifier for the value import rule
type	The type of value import rule: • `fileLocation` - the location of a specified file. • `s3Bucket` - location of the S3 bucket to modify
with	The replacement value or pattern
deploymentId	Internal identifier for the deployment to which to apply the import rule.

API Flows Create v3

Contents:

Create a new flow with specified name.

> **NOTE:** You cannot add datasets to the flow through this endpoint. Moving pre-existing datasets into a flow is not supported in this release. Create the flow first and then when you create the datasets, associate them with the flow at the time of creation.
>
> - See *API ImportedDatasets Create v3*.
> - See *API WrangledDatasets Create v3*.

Version: `v3`

Required Permissions

The authenticated user must be an admin.

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `POST`

Endpoint:

```
/v3/flows/
```

Request Body:

A `name` value is required.

```
{
  "name": "My Flow",
  "description": "This is my flow."
}
```

Response

Response Status Code - Success: `201 - Created`

Response Body Example:

```
{
  "id": 6,
  "name": "My Flow",
  "description": "This is my flow.",
  "createdBy": 1,
  "updatedBy": 1,
  "updatedAt": "2017-02-17T17:08:57.848Z",
  "createdAt": "2017-02-17T17:08:57.848Z"
}
```

Reference

For more information on the properties of a flow, see *API Flows Get v3*.

API Flows Delete v3

Contents:

Delete the specified flow.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `DELETE`

Endpoint:

```
/v3/flows/<id>
```

where:

Parameter	Description
`<id>`	Internal identifier for the flow

Request URI - Example:

```
/v3/flows/2
```

Request Body:

Empty.

Response

Response Status Code - Success: `204 - No Content`

Response Body Example:

Empty.

Reference

For more information on the properties of a user, see *API Flows Get v3*.

API Flows Get List v3

Contents:

Returns a list of all flows accessible to the authenticated user.

Version: `v3`

Required Permissions

- If you are not logged in or are logged as a non-admin user, you can retrieve only your flows.
- If you are logged in as an admin, you can retrieve all flows in the platform.

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/flows
```

Endpoint with paged retrieval:

By default, this endpoint returns results in sets of 25.

You can apply query parameters to change the size of the default set and to page through result sets. The following example queries for results 100 at a time. In this case, the query asks for results 201-300:

```
/v3/flows?limit=100%offset=2
```

If the count of retrieved results is less than the limit, you have reached the end of the results.

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Examples:

```
[
    {
        "id": 7,
        "name": "USDA_Farmers_Market_2014 Flow",
        "description": null,
        "deleted_at": null,
        "cpProject": null,
        "createdAt": "2017-11-07T19:30:29.296Z",
        "updatedAt": "2017-11-07T19:30:29.296Z",
        "createdBy": 1,
        "updatedBy": 1,
        "associatedPeople": [
            {
                "outputHomeDir":
"/trifacta/queryResults/admin@trifacta.local",
                "name": "Administrator",
                "email": "admin@trifacta.local",
                "id": 1,
                "flowpermission": {
                    "flowId": 7,
                    "personId": 1,
                    "role": "owner"
                }
            }
        ]
    },
    {
        "id": 3,
        "name": "2013 POS",
        "description": null,
        "deleted_at": null,
        "cpProject": null,
        "createdAt": "2017-11-07T17:02:34.662Z",
        "updatedAt": "2017-11-07T17:02:34.662Z",
        "createdBy": 1,
        "updatedBy": 1,
        "associatedPeople": [
            {
                "outputHomeDir":
"/trifacta/queryResults/admin@trifacta.local",
                "name": "Administrator",
                "email": "admin@trifacta.local",
                "id": 1,
                "flowpermission": {
                    "flowId": 3,
                    "personId": 1,
                    "role": "owner"
                }
            }
        ]
    }
]
```

Reference

For more information on the properties of a user, see *API Flows Get v3*.

API Flows Get v3

Contents:

Retrieve the flow information for a specified flow identifier.

Version: `v3`

Required Permissions

The authenticated user must be an admin.

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/flows/<id>
```

Parameter	Description
`<id>`	Internal identifier of the flow to retrieve.

Request URI - Example:

```
/v3/flows/10
```

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
[
   {
        "id": 10,
        "name": "2013 POS",
        "description": null,
        "deleted_at": null,
        "cpProject": null,
        "createdAt": "2017-11-07T17:02:34.662Z",
        "updatedAt": "2017-11-07T17:02:34.662Z",
        "createdBy": 1,
        "updatedBy": 1,
        "associatedPeople": [
            {
                "outputHomeDir":
"/trifacta/queryResults/admin@trifacta.local",
                "name": "Administrator",
                "email": "admin@trifacta.local",
                "id": 1,
                "flowpermission": {
                    "flowId": 3,
                    "personId": 1,
                    "role": "owner"
                }
            }
        ]
    }
]
```

Reference

Property	Description
id	Internal identifier for the flow
name	Display text for the flow
description	User-friendly description for the flow
cpProject	Not used.
createdAt	Timestamp for when the flow was created
updatedAt	Timestamp for when the flow was last modified
createdBy	Internal identifier of the user who created the flow
updatedBy	Internal identifier of the user who last updated the flow
associatedPeople.outputHomeDir	Output directory for the associated person.
associatedPeople.name	Name of the associated person.
associatedPeople.email	User ID (email address) of the associated person.
associatedPeople.id	Internal ID for the associated person.
associatedPeople.flowpermission.flowId	Internal ID for the flow.
associatedPeople.flowpermission.personId	Internal ID for the permitted user.
associatedPeople.flowpermission.role	Role for the permitted user. • `owner` - can do anything to the flow. • `collaborator` - can do most actions on the flow except change the name and delete the flow.

API Flows Package Get DryRun v3

Contents:

Performs a dry-run of generating a flow package and exporting it, which performs a check of all permissions required to export the package.

If they occur, permissions errors are reported in the response.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/flows/<id>/package/dryRun
```

Parameter	Description
`<id>`	Internal identifier of the flow to retrieve.

Request URI - Example:

```
/v3/flows/7/package/dryRun
```

Request Body:

None.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{}
```

Reference

None.

API Flows Package Get v3

Contents:

- *Required Permissions*
- *Request*
- *Response*
- *Reference*

Retrieve a package containing the definition of the specified flow.

Response body is the contents of the package. Package contents are a ZIPped version of the flow definition.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/flows/<id>/package
```

Parameter	Description
`<id>`	Internal identifier of the flow to retrieve.

Request URI - Example:

```
/v3/flows/7/package
```

Request Body:

None.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

Response body is the contents of the ZIP file. This package should be downloaded to your local environment.

Reference

None.

API Flows Package Post DryRun v3

Contents:

- *Required Permissions*
- *Request*
- *Response*
- *Reference*

Performs a dry-run of importing a flow package, which performs a check of all permissions required to import the package, as well as any specified import rules.

- For more information on import rules, see *Define Import Mapping Rules*.

If they occur, errors are reported in the response.

After you have successfully completed a dry-run, you can execute a formal import. See *API Flows Package Post v3*.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `POST`

Endpoint:

```
/v3/flows/package/dryRun
```

Request URI - Example:

```
/v3/flows/package/dryRun
```

Request Body:

The request body must include the following key and value combination submitted as form data. This path is the location of the ZIP package that you are importing.

key	value
data	"@path-to-file"

Example request - curl:

```
curl -X POST \
  http://example.com:3005/v3/flows/package/dryRun \
  -H 'authorization: Basic c29sc29uQHRyaWZhY3RhLmNvbTphZG1pbg==' \
  -H 'cache-control: no-cache' \
  -H 'content-type: multipart/form-data' \
  -F data=@response.zip
```

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
    "importRuleChanges": {
        "object": [],
        "value": []
    },
    "flowName": "[267f4340]  2013 POS"
}
```

Reference

None.

API Flows Package Post v3

Contents:

Performs an import of a flow package, which   also applies any specified import rules.

- Before you import, you can perform a dry-run to check for errors. See *API Flows Package Post DryRun v3*.
- For more information on import rules, see *Define Import Mapping Rules*.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `POST`

Endpoint:

```
/v3/flows/package/
```

Request URI - Example:

```
/v3/flows/package
```

Request Body:

The request body must include the following key and value combination submitted as form data. This path is the location of the ZIP package that you are importing.

key	value
data	"@path-to-file"

Example request - curl:

```
curl -X POST \
  http://example.com:3005/v3/flows/package \
  -H 'authorization: Basic c29sc29uQHRyaWZhY3RhLmNvbTphZG1pbg==' \
  -H 'cache-control: no-cache' \
  -H 'content-type: multipart/form-data' \
  -F data=@response.zip
```

Response

Response Status Code - Success: `201 - Created`

Response Body Example:

```
{
    "importRuleChanges": {
        "object": [],
        "value": []
    },
    "flowName": "[267f4340]  2013 POS"
}
```

Reference

None.

API Flows Patch v3

Contents:

Update an existing flow based on the specified identifier.

> **NOTE:** You cannot add datasets to the flow through this endpoint. Moving pre-existing datasets into a flow is not supported in this release. Create the flow first and then when you create the datasets, associate them with the flow at the time of creation.
>
> - See *API ImportedDatasets Create v3*.
> - See *API WrangledDatasets Create v3*.

Version: `v3`

Required Permissions

The authenticated user must be an admin.

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `PATCH`

Endpoint:

```
/v3/flows/<id>
```

Parameter	Description
`<id>`	Internal identifier of the flow to update.

Request URI - Example:

```
/v3/flows/6
```

Request Body:

You can modify the following properties.

```
{
  "name": "My Flow",
  "description": "This is my flow."
}
```

Response

Response Status Code - Success: `200 - Ok`

Response Body Example:

```
{
  "id": 6,
  "updatedBy": 1,
  "updatedAt": "2017-02-17T18:28:47.675Z"
}
```

Reference

For more information on the properties of a flow, see *API Flows Get v3*.

API ImportedDatasets Create v3

Contents:

Create an imported dataset from an available resource. Created dataset is owned by the authenticated user.

> **NOTE:** Do not create an imported dataset from a file that is being used by another imported dataset. If you delete the newly created imported dataset, the file is removed, and the other dataset is corrupted. Use a new file or make a copy of the first file first.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request and Response

Request Type: `POST`

Endpoint:

```
/v3/importedDatasets
```

Response Status Code - Success: `201 - Created`

Examples by Type

Below, you can review the basic request body for creating imported datasets for various types of sources:

- File (HDFS or S3 source)
- Hive
- Relational
- Relation with Custom SQL Query

File (HDFS and S3 sources)

Request Body - HDFS file:

Below, the `bucket` value is set to `null`. This parameter applies only to S3 sources.

> **NOTE:** The `path` value should not include the HDFS protocol, host, or port information. You only need to provide the path on HDFS.

```
{
  "path":
"/trifacta/uploads/1/4aee9852-cf92-47a8-8c6a-9ff2adeb3b4a/POS-r02.txt",
  "type": "hdfs",
  "bucket": null,
  "name": "POS-r02b.txt",
  "description": "POS-r02 - copy"
}
```

Request Body - S3 file:

For S3 sources, a bucket must be specified. Below, the `bucket` value is set to `myBucket`.

> **NOTE:** The `path` value should not include the S3 protocol, host, or port information. You only need to provide the path on S3.

```
{
  "path":
"/trifacta/uploads/1/4aee9852-cf92-47a8-8c6a-9ff2adeb3b4a/POS-r02.txt",
  "type": "s3",
  "bucket": "myBucket",
  "name": "POS-r02b.txt",
  "description": "POS-r02 - copy"
}
```

Response Body - file:

Following example is for an HDFS file. For an S3 file, `type=s3`.

```
{
  "id": 8,
  "size": "281032",
  "path":
"/trifacta/uploads/1/4aee9852-cf92-47a8-8c6a-9ff2adeb3b4a/POS-r02.txt",
  "isSharedWithAll": false,
  "type": "hdfs",
  "bucket": null,
  "isSchematized": false,
  "createdBy": 1,
  "updatedBy": 1,
  "updatedAt": "2017-02-08T18:38:56.640Z",
  "createdAt": "2017-02-08T18:38:56.560Z",
  "connectionId": null,
  "parsingScriptId": 14,
  "cpProject": null
}
```

Hive

Request Body - Hive:

Notes:

- Note that the `type=jdbc`.
- The `columns` key is optional. If not provided, all columns in the source table are included.

```
{
  "visible": true,
  "numFlows": 0,
  "size": -1,
  "type": "jdbc",
  "jdbcType": "TABLE",
  "jdbcPath": [
    "DB1"
  ],
  "jdbcTable": "MyHiveTable",
  "columns": [
    "column1",
    "column2"
  ],
  "connectionId": 16,
  "name": "My Hive Table"
}
```

Response Example - Hive:

```
{
  "jdbcTable": "MyHiveTable",
  "jdbcPath": [
    "DB1"
  ],
  "columns": [
    "column1",
    "column2"
  ],
  "filter": null,
  "raw": null,
  "isSharedWithAll": false,
  "id": 192,
  "size": "-1",
  "type": "jdbc",
  "connectionId": 16,
  "createdBy": 1,
  "updatedBy": 1,
  "updatedAt": "2017-02-17T18:09:17.745Z",
  "createdAt": "2017-02-17T18:09:16.407Z",
  "path": null,
  "bucket": null,
  "parsingScriptId": 366,
  "cpProject": null,
  "isSchematized": true
}
```

Relational

Request Body - Relational:

Notes:

- If you know the `size` value for the table, please provide. It is helpful for performance reasons and validation but is not required.
- The `columns` key is optional. If not provided, all columns in the source table are included.

```
{
  "visible": true,
  "numFlows": 0,
  "size": 65536,
  "type": "jdbc",
  "jdbcType": "TABLE",
  "jdbcPath": [
    "OracleDB_1"
  ],
  "jdbcTable": "MyOracleTable",
  "columns": [
    "I",
    "J",
    "K"
  ],
  "connectionId": 7,
  "name": "My Oracle Table"
}
```

Response Example - Relational:

```
{
  "jdbcTable": "MyOracleTable",
  "jdbcPath": [
    "OracleDB_1"
  ],
  "columns": [
    "I",
    "J",
    "K"
  ],
  "filter": null,
  "raw": null,
  "isSharedWithAll": false,
  "id": 195,
  "size": "65536",
  "type": "jdbc",
  "connectionId": 7,
  "createdBy": 1,
  "updatedBy": 1,
  "updatedAt": "2017-02-17T18:10:48.662Z",
  "createdAt": "2017-02-17T18:10:47.441Z",
  "path": null,
  "bucket": null,
  "parsingScriptId": 372,
  "cpProject": null,
  "isSchematized": true
}
```

Relational with Custom SQL Query

You can submit custom SQL queries to relational or hive connections. These custom SQLs can be used to pre-filter the data inside the database, improving performance of the query and the overall dataset.

- For more information, see *Enable Custom SQL Query*.

Request Body:

Notes:

- See previous notes on queries to relational sources.
- As part of the request body, you must submit the custom SQL query as the value for the `raw` property.

The following example is valid for Oracle databases. Note the escaping of the double-quote marks.

> **NOTE:** Syntax for the custom SQL query varies between relational systems. For more information on syntax examples, see *Create Dataset with SQL*.

```
{
  "visible": true,
  "numFlows": 0,
  "type": "jdbc",
  "jdbcType": "TABLE",
  "connectionId": 7,
  "raw": "SELECT INST#,BUCKET#,INST_LOB# FROM
\"AUDSYS\".\"CLI_SWP$7395268a$1$1\"",
  "size": -1,
  "name": "SQL Dataset 1"
}
```

Response Body:

In the response, note that the source of the data is defined by the `connectionId` value and the SQL defined in the `raw` value.

```
{
  "jdbcTable": null,
  "jdbcPath": null,
  "columns": null,
  "filter": null,
  "raw": [
    "SELECT INST#,BUCKET#,INST_LOB# FROM
\"AUDSYS\".\"CLI_SWP$7395268a$1$1\""
  ],
  "isSharedWithAll": false,
  "id": 196,
  "size": "-1",
  "type": "jdbc",
  "connectionId": 7,
  "createdBy": 1,
  "updatedBy": 1,
  "updatedAt": "2017-02-17T19:09:10.117Z",
  "createdAt": "2017-02-17T19:07:12.757Z",
  "path": null,
  "bucket": null,
  "parsingScriptId": 378,
  "cpProject": null,
  "isSchematized": true
}
```

Reference

For more information on the properties of an imported dataset, see *API ImportedDatasets Get v3*.

API ImportedDatasets Delete v3

Contents:

Delete the specified dataset.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `DELETE`

Endpoint:

```
/v3/importedDatasets/<id>
```

where:

Property	Description
`<id>`	Internal identifier for the imported dataset

Request URI - Example:

```
/v3/importedDatasets/2
```

Request Body:

Empty.

Response

Response Status Code - Success: `204 - No Content`

Response Body Example:

Empty.

Reference

For more information on the properties of an imported dataset, see *API ImportedDatasets Get v3*.

API ImportedDatasets Get List v3

Contents:

Get the list of accessible imported datasets for the authenticated user.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/importedDatasets
```

Endpoint with embedded reference:

```
/v3/importedDatasets/?embed=connection
```

For more information, see *API ImportedDatasets Get v3*.

Endpoint with paged retrieval:

By default, this endpoint returns results in sets of 25.

You can apply query parameters to change the size of the default set and to page through result sets. The following example queries for results 100 at a time. In this case, the query asks for results 201-300:

```
/v3/importedDatasets?embed=connection&limit=100%offset=2
```

If the count of retrieved results is less than the limit, you have reached the end of the results.

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

In the following example, you can see the data for three separate connections:

- JDBC with custom SQL query
- JDBC
- File-based (uploaded)

```
{
  "data": [
    {
      "id": 15,
      "size": "-1",
      "path": null,
      "isSharedWithAll": false,
      "type": "jdbc",
      "bucket": null,
      "isSchematized": true,
      "createdAt": "2017-02-03T07:34:40.727Z",
      "updatedAt": "2017-02-03T07:34:43.910Z",
      "createdBy": 1,
      "updatedBy": 1,
      "name": "all_objects",
      "description": null,
      "connection": {
        "id": 2
      },
      "parsingRecipe": {
        "id": 37
      },
      "relationalSource": {
        "relationalPath": [
          "adam_test",
          "sys"
        ],
        "columns": [
          "name",
          "object_id",
          "principal_id",
          "schema_id",
          "parent_object_id",
          "type",
          "type_desc",
          "create_date",
          "modify_date",
          "is_ms_shipped",
          "is_published",
          "is_schema_published"
        ],
        "filter": null,
        "raw": null,
        "id": 1,
        "tableName": "all_objects",
        "createdAt": "2017-02-03T07:34:40.760Z",
        "updatedAt": "2017-02-03T07:34:40.760Z",
        "datasourceId": 15
      }
    },
    {
      "id": 14,
      "size": "2049672",
      "path":
"/trifacta/uploads/1/1a0d9144-a0bc-44a2-ae93-fc3f53613930/base_type_map_ar
ray_record_large.avro",
      "isSharedWithAll": false,
      "type": "hdfs",
      "bucket": null,
      "isSchematized": true,
      "createdAt": "2017-02-03T05:16:49.934Z",
      "updatedAt": "2017-02-03T05:16:51.200Z",
      "createdBy": 1,
      "updatedBy": 1,
      "name": "base_type_map_array_record_large.avro",
      "description": null,
      "parsingRecipe": {
        "id": 35
      }
    },
    {
      "id": 13,
      "size": "711558",
      "path":
"/trifacta/queryResults/administrator/hello_world/original_2_1_random_rows
.json",
      "isSharedWithAll": false,
      "type": "s3",
      "bucket": "3fac-jlong-test",
      "isSchematized": false,
      "createdAt": "2017-02-03T05:15:07.398Z",
      "updatedAt": "2017-02-03T05:15:11.724Z",
      "createdBy": 1,
      "updatedBy": 1,
      "name": "original_2_1_random_rows.json",
      "description": null,
      "parsingRecipe": {
        "id": 33
      }
```

```
        }
    ]
}
```

Reference

For more information on the properties of an imported dataset, see *API ImportedDatasets Get v3*.

API ImportedDatasets Get v3

Contents:

Get the specified imported dataset.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/importedDatasets/<id>
```

where:

Parameter	Description
`<id>`	Internal identifier for the imported dataset

Endpoint with embedded reference:

Use the following embedded reference to embed in the response data about the connection used to acquire the source dataset if it was created from a Hive or relational connection.

```
/v3/importedDatasets/<id>?embed=connection
```

Request URI - Example:

```
/v3/importedDatasets/196
```

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
  "id": 196,
  "size": "-1",
  "path": null,
  "isSharedWithAll": false,
  "type": "jdbc",
  "bucket": null,
  "isSchematized": true,
  "createdAt": "2017-02-17T19:07:12.757Z",
  "updatedAt": "2017-02-17T19:09:10.117Z",
  "createdBy": 1,
  "updatedBy": 1,
  "name": "SQL Dataset 1 - 4",
  "description": null,
  "connection": {
    "id": 7
  },
  "parsingRecipe": {
    "id": 378
  },
  "relationalSource": {
    "relationalPath": null,
    "columns": null,
    "filter": null,
    "raw": [
      "SELECT INST#,BUCKET#,INST_LOB# FROM
\"AUDSYS\".\"CLI_SWP$7395268a$1$1\""
    ],
    "id": 109,
    "tableName": null,
    "createdAt": "2017-02-17T19:07:12.767Z",
    "updatedAt": "2017-02-17T19:07:12.767Z",
    "datasourceId": 196
  }
}
```

Response Body Example with embedded reference:

The following response includes embedded information on the connection used to import the data.

```
{
  "id": 313,
  "size": "35651584",
  "path": null,
  "isSharedWithAll": false,
  "type": "jdbc",
  "bucket": null,
    "isSchematized": true,
    "createdAt": "2017-02-22T23:33:54.400Z",
    "updatedAt": "2017-02-22T23:34:18.148Z",
    "createdBy": 1,
    "updatedBy": 1,
    "name": "TestOracleDS",
    "description": null,
    "connection": {
      "id": 7,
      "name": "Oracle",
      "description": "",
      "type": "jdbc",
      "createdBy": 1,
      "isGlobal": true,
      "credentialType": "basic",
      "createdAt": "2017-01-11T01:21:54.950Z",
      "updatedAt": "2017-01-11T01:22:20.107Z",
      "updatedBy": 1
    },
    "parsingRecipe": {
      "id": 645
    },
    "relationalSource": {
      "relationalPath": [
        "AUDSYS"
      ],
      "columns": [
        "INST#",
        "BUCKET#",
        "INST_LOB#",
        "MAX_SEQ#",
        "FLUSH_SCN",
        "FLUSH_TIME",
        "MIN_SCN",
        "MAX_SCN",
        "MIN_TIME",
        "MAX_TIME",
        "SID#",
        "SERIAL#",
        "STATUS",
        "LOG_PIECE"
      ],
      "filter": null,
      "raw": null,
      "id": 121,
      "tableName": "CLI_SWP$7395268a$1$1",
      "createdAt": "2017-02-22T23:33:54.406Z",
      "updatedAt": "2017-02-22T23:33:54.406Z",
      "datasourceId": 313
    }
  }
```

Reference

Common Properties:

The following properties are common to file-based and JDBC datasets.

Property	Description
id	Internal identifier of the imported dataset
size	Size of the source file in bytes (if applicable)
path	For HDFS and S3 file sources, this value defines the path to the source. For JDBC sources, this value is not specified.
isSharedWithAll	If `true`, the source is shared among all users of the platform.
type	Identifies where the type of storage where the source is located. Values: • `hdfs` • `s3` • `jdbc`
bucket	(If `type=s3`) Bucket on S3 where source is stored.
isSchematized	(If source file is avro, or `type=jdbc`) If `true`, schema information is available for the source.
createdAt	Timestamp for when the dataset was imported
UpdatedAt	Timestamp for when the dataset was last updated
createdBy	Internal identifier of the user who created the imported dataset
updatedBy	Internal identifier of the user who last updated the imported dataset
name	Internal name of the imported dataset
description	User-friendly description for the imported dataset

connection	Internal identifier of the connection to the server hosting the dataset.
	If this value is `null`, the file was uploaded from a local file system.
	To acquire the entire connection for this dataset, you can use either of the following endpoints:
	`/v3/importedDatasets?embed=connection`
	`/v3/importedDatasets/:id?embed=connection`
	For more information, see *API Connections Get v3*.

parsingRecipe	Internal identifier of the recipe that is used to parse the imported dataset for wrangling.

Hive or Relational Source:

If the source data is from Hive or a relational system (`type=jdbc`), the following properties contain information on the source table, the imported columns, and any custom SQL filters applied to the table:

Property	Description

relationalPath	Name of the database from which the source was queried. If a custom SQL query has been applied, this value is `null`.
columns	List of columns imported from the source, pre-filtered. If a custom SQL query has been applied, this value is `null`.
filter	This value is empty.
raw	If custom SQL has been applied to the data source to filter the data before it is imported, all SQL statements are listed. For more information, see *Enable Custom SQL Query*.
id	Internal identifier for the relational source
tableName	Name of the table from which the data is extracted. If a custom SQL query has been applied, this value is `null`.
createdAt	Timestamp for when the source was imported
updatedAt	Timestamp for when the source was last updated
datasourceId	Internal identifier for the datasource.

File:

File-based datasets support the common properties only.

API ImportedDatasets Post AddToFlow v3

Contents:

Add the specified imported dataset to a flow based on its internal identifier.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `POST`

Endpoint:

```
/v3/importedDatasets/<id>/addToFlow
```

where:

Parameter	Description
`<id>`	Internal identifier for the imported dataset

Request URI - Example:

```
/v3/importedDatasets/4/addToFlow
```

Request Body:

```
{
    "flow": {
        "id": 4
    }
}
```

Response

Response Status Code - Success: `201 - Created`

Response Body:

```
{
  "id": 14,
  "createdBy": 1,
  "updatedBy": 1,
  "scriptId": 7,
  "flowId": 4,
  "wrangled": false,
  "updatedAt": "2017-06-28T19:38:29.275Z",
  "createdAt": "2017-06-28T19:38:29.016Z",
  "flowNodeId": null,
  "deleted_at": null,
  "activesampleId": 15
}
```

Reference

Property	Description
id	Internal identifier for the new wrangled dataset.
createdBy	Internal identifier of the user who created the flow.
updatedBy	Internal identifier of the user who performed the update.
scriptId	Internal identifier for the recipe for the dataset. If `null`, the dataset has not been wrangled in the Transformer page.
flowId	Internal identifier of the flow that contains this dataset
wrangled	If `true`, this dataset is a wrangled dataset.
updatedAt	Timestamp for when the dataset was updated.
createdAt	Timestamp for when the dataset was created.
flowNodeId	Internal identifier of the dataset within the flow.
deleted_at	Timestamp for when the dataset was deleted. If `null`, the dataset has not been deleted.
activesampleId	Internal identifier for the currently active sample for the dataset. If `null`, there is no currently active sample for the dataset.

For more information on the other properties, see *API ImportedDatasets Get v3*.

API JobGroups Create v3

Contents:

Create a jobGroup, which launches the specified job as the authenticated user.

The request specification depends on one of the following conditions:

- Dataset has already had a job run against it and just needs to be re-run.
- Dataset has not had a job run, or the job definition needs to be re-specified.

> **NOTE:** In this release, you cannot execute jobs sourced from datasets in Redshift or SQL DW or publish to these locations via the API. This known issue will be fixed in a future release.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `POST`

Endpoint:

```
/v3/jobGroups
```

Request Body - Re-run job:

If you are re-running a job that has already executed and do not need to modify any job settings, you can use the following simplified body to launch it:

```
{
  "wrangledDataset": {
    "id": 7
  }
}
```

Request Body - Specify job:

If you are specifying a new job or must re-run a job with new settings, you must include a version of the following request body. Required parameters are listed below:

```
{
  "wrangledDataset": {
    "id": 1
  },
  "overrides": {
    "execution": "photon",
    "profiler": false,
    "writesettings": [
      {
        "path":
"hdfs://hadoop:50070/trifacta/queryResults/admin@trifacta.local/cdr_txt.cs
v",
        "action": "create",
        "format": "csv",
        "compression": "none",
        "header": false,
        "asSingleFile": false
      }
    ]
  },
  "ranfrom": null
}
```

Response

Response Status Code - Success: `201 - Created`

Response Body Example:

```
{
  "jobgroupId": 3,
  "jobIds": [
    5,
    6
  ],
  "reason": "JobStarted",
  "sessionId": "9c2c6220-ef2d-11e6-b644-6dbff703bdfc"
}
```

Reference

Request Reference:

Property	Description
`wrangledDataset`	(required) Internal identifier for the object whose results you wish to generate. The recipes of all preceding datasets on which this dataset depends are executed as part of the job.
`overrides.execution`	(required, if first time running the job) Indicates the running environment on which the job is executed. Accepted values: • `photon` • spark For more information, see *Running Environment Options*.
`overrides.profiler`	(required, if first time running the job) When set to `true`, a visual profile of the job is generated as specified by the profiling options for the platform. See *Profiling Options*.
`overrides.writesettings`	(required, if first time running the job) These settings define the publishing options for the job. See below.
`ranfrom`	(optional) If this value is set to `null`, then the job does not show up in the Job Results page. If set to `cli`, the job appears as a CLI job. See *Job Results Page*.

writesettings Reference:

The `writesettings` values allow you to specify aspects of the publication of results to the specified `path` location.

NOTE: `writesettings` values are required if you are running this specified job for the dataset for the first time.

NOTE: To specify multiple outputs, you can include additional `writesettings` objects in the request. For example, if you want to generate output to `csv` and `json`, you can duplicate the `writesettings` object for `csv` and change the `format` value in the second one to `json`.

These settings correspond to values that you can apply through the UI or through the command line interface.

- For UI information, see *Run Job Page*.
- For CLI information, see *CLI for Jobs*.

Property	Description
`path`	(required) The fully qualified path to the output location where to write the results
`action`	(required) If the output file or directory exists, you can specify one of the following actions: • `create` - Create a new, parallel location, preserving the old results. • `append` - Add the new results to the old results. • `overwrite` - Replace the old results with the new results.

`format`	(required) Output format for the results. Specify one of the following values: • `csv` • `json` • `avro` • `pqt` **NOTE:** Parquet format requires execution on a Hadoop running environment (`overrides.execution` must be `spark`). **NOTE:** To specify multiple output formats, create additional `writesettings` object for each output format.
`compression`	(optional) For `csv` and `json` results, you can optionally compress them using `bzip2` or `gzip` compression. Default is `none`.
`header`	(optional) For `csv` results with `action` set to `create` or `append`, this value determines if a header row with column names is inserted at the top of the results. Default is `false`.
`asSingleFile`	(optional) For `csv` and `json` results, this value determines if the results are concatenated into a single file or stored as multiple files. Default is `false`.

API JobGroups Delete v3

Contents:

Delete the specified jobGroup.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `DELETE`

Endpoint:

```
/v3/jobGroups/<id>
```

where:

Parameter	Description
`<id>`	Internal identifier for the job group

Request URI - Example:

```
/v3/jobGroups/2
```

Request Body:

Empty.

Response

Response Status Code - Success: `204 - No Content`

Response Body Example:

Empty.

Reference

None.

API JobGroups Get Jobs v3

Contents:

Get list of jobs for the specified jobGroup. For more information on jobGroups, see *API JobGroups Get v3*.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/jobGroups/<id>/jobs
```

where:

Parameter	Description
`<id>`	Internal identifier for the job group

Request URI - Example:

```
/v3/jobGroups/2/jobs
```

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
  "data": [
    {
      "id": 5,
      "writeSetting": {
        "id": 3
      },
      "scriptResult": {
        "id": 4
      },
      "createdAt": "2017-05-05T20:38:15.883Z",
      "updatedAt": "2017-05-05T20:38:19.411Z",
      "status": "Complete",
      "jobType": "filewriter",
      "sampleSize": 100,
      "percentComplete": 100,
      "cpJob": null,
      "createdBy": 1,
      "errorMessage": null,
      "jobGroup": {
        "id": 3
      }
    },
    {
      "id": 6,
      "writeSetting": {
        "id": 4
      },
      "scriptResult": {
        "id": 5
      },
      "createdAt": "2017-05-05T20:38:15.888Z",
      "updatedAt": "2017-05-05T20:38:19.007Z",
      "status": "Complete",
      "jobType": "filewriter",
      "sampleSize": 100,
      "percentComplete": 100,
      "cpJob": null,
      "createdBy": 1,
      "errorMessage": null,
      "jobGroup": {
        "id": 3
      }
    },
    {
      "id": 4,
      "executionLanguage": "photon",
      "createdAt": "2017-05-05T20:38:15.808Z",
      "updatedAt": "2017-05-05T20:38:18.433Z",
      "status": "Complete",
      "jobType": "wrangle",
      "sampleSize": 100,
      "percentComplete": 100,
      "cpJob": null,
      "createdBy": 1,
      "errorMessage": null,
      "jobGroup": {
        "id": 3
      }
```

```
      }
    ]
  }
```

Reference

Property	Description
`writeSetting`	User-visible output settings. Contents may vary depending on the type of output.
`scriptResult`	Internal identifier for job execution. Used by other dependent jobs to identify where to write results to or to collect results from.
`executionLanguage`	Indicator for the running environment where the job was executed. Values: • `photon` - Photon running environment internal to the Trifacta platform. • `spark` - Spark running environment on Hadoop cluster. For more information on running environments, see *Running Environment Options*.
`status`	Current status of the job. See *API JobGroups Get v3*.
`jobType`	Type of job. Values: • `filewriter` - output results to a specified file • `ingest` - internal job for transferring data into HDFS. • `profile` - job to compute statistical information about set of results. • `publish` - job to publish previously wrangled results to a new destination. • `wrangle` - execute the specified set of Wrangle steps on a dataset.
`sampleSize`	Size of sample as a percentage of the entire dataset. `100` means that the entire dataset is used as the sample.
`percentComplete`	Percentage of completion of the job at the time of the request. `100` means that the job has finished or failed.
`createdBy`	Internal identifier of the user who launched the job.
`errorMessage`	If the job failed, any error message is displayed here.

For more information on the other properties, see *API JobGroups Get v3*.

API JobGroups Get List v3

Contents:

Get list of all jobGroups accessible to the authenticated user.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/jobGroups
```

Endpoint with embedded reference:

```
/v3/jobGroups/?embed=jobs,wrangledDataset
```

For more information, *API JobGroups Get v3*.

Endpoint with paged retrieval:

By default, this endpoint returns results in sets of 25.

You can apply query parameters to change the size of the default set and to page through result sets. The following example queries for results 100 at a time. In this case, the query asks for results 201-300:

```
/v3/jobGroups/?embed=jobs,wrangledDataset&limit=100%offset=2
```

If the count of retrieved results is less than the limit, you have reached the end of the results.

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
  "data": [
    {
      "id": 9,
      "name": null,
      "description": null,
      "ranfrom": "ui",
      "status": "InProgress",
      "profilingEnabled": true,
      "createdAt": "2017-01-31T20:01:57.513Z",
      "updatedAt": "2017-01-31T20:01:58.984Z",
      "createdBy": 2,
      "updatedBy": 2,
      "wrangledDataset": {
        "id": 92
      },
      "snapshot": {
        "id": 54
      },
      "wrangleScript": {
        "id": 62
      },
      "jobs": {
        "data": null
      }
    },
    {
      "id": 8,
      "name": null,
      "description": null,
      "ranfrom": "ui",
      "status": "Complete",
      "profilingEnabled": true,
      "createdAt": "2017-01-31T19:59:23.804Z",
      "updatedAt": "2017-01-31T20:00:28.278Z",
      "createdBy": 2,
      "updatedBy": 2,
      "wrangledDataset": {
        "id": 92
      },
      "snapshot": {
        "id": 53
      },
      "wrangleScript": {
        "id": 60
      },
      "jobs": {
        "data": null
      }
    }
  ]
}
```

Reference

For more information on the properties of a jobGroup, see *API JobGroups Get v3*.

API JobGroups Get Status v3

Contents:

Get current status of the specified jobGroup. For more information on jobGroups, see *API JobGroups Get List v3*.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/jobGroups/<id>/status
```

where:

Parameter	Description
`<id>`	Internal identifier for the job group

Request URI - Example:

```
/v3/jobGroups/2/status
```

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
"Complete"
```

Reference

For more information on the available status messages, see *API JobGroups Get List v3*.

API JobGroups Get v3

Contents:

Get information on the specified job group. A **job group** is a job that is executed from a specific node in a flow. The job group may contain:

- Wrangling job on the dataset associated with the node
- Jobs on all datasets on which the selected job may depend
- A profiling job for the job group

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/jobGroups/<id>
```

where:

Parameter	Description
`<id>`	Internal identifier for the job group

Endpoint with embedded references:

Use the following endpoint to embed additional information:

Embed Parameter	Description
jobs	Embed information about the child jobs within the job group. Array includes information on transformation, profiling, and publishing jobs that are part of the job group.
wrangledDataset	This field contains the internal identifier for the dataset on which the job was run.

```
/v3/jobGroups/<id>?embed=jobs,wrangledDataset
```

Request URI - Example:

```
/v3/jobGroups/8
```

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
  "id": 8,
  "name": null,
  "description": null,
  "ranfrom": "ui",
  "status": "Complete",
  "profilingEnabled": true,
  "createdAt": "2017-01-31T19:59:23.804Z",
  "updatedAt": "2017-01-31T20:00:28.278Z",
  "createdBy": 2,
  "updatedBy": 2,
  "wrangledDataset": {
    "id": 92
  },
  "snapshot": {
    "id": 53
  },
  "wrangleScript": {
    "id": 60
  },
  "jobs": {
    "data": null
  }
}
```

Response Body Example with embedded reference:

The following example response includes embedded information on jobs and the recipe associated with it.

```
{
  "id": 11,
  "name": null,
  "description": null,
  "ranfrom": "ui",
  "status": "Complete",
  "profilingEnabled": true,
  "createdAt": "2017-02-22T23:24:46.247Z",
  "updatedAt": "2017-02-22T23:24:48.627Z",
  "createdBy": 1,
  "updatedBy": 1,
  "wrangledDataset": {
    "id": 118,
    "wrangled": true,
    "createdAt": "2017-02-22T22:54:13.782Z",
    "updatedAt": "2017-02-22T22:54:22.107Z",
    "createdBy": 1,
    "updatedBy": 1,
      "activeSample": {
        "id": 56
      },
      "flow": {
        "id": 29
      },
      "script": {
        "id": 107
      }
    },
    "snapshot": {
      "id": 65
    },
    "wrangleScript": {
      "id": 79
    },
    "jobs": {
      "data": [
        {
          "id": 23,
          "jobId": 23,
          "writesettingId": 12,
          "scriptresultId": 41,
          "createdAt": "2017-02-22T23:24:46.789Z",
          "updatedAt": "2017-02-22T23:24:48.618Z",
          "status": "Complete",
          "jobType": "filewriter",
          "sampleSize": 100,
          "percentComplete": 100,
          "cpJobId": null,
          "createdBy": 1,
          "errorMessage": null,
          "jobGroup": {
            "id": 11
          }
        },
        {
          "id": 22,
          "executionLanguage": "photon",
          "jobId": 22,
          "createdAt": "2017-02-22T23:24:46.740Z",
          "updatedAt": "2017-02-22T23:24:48.426Z",
          "wranglescriptId": 80,
          "status": "Complete",
          "jobType": "wrangle",
          "sampleSize": 100,
          "percentComplete": 100,
          "cpJobId": null,
          "createdBy": 1,
          "errorMessage": null,
          "jobGroup": {
            "id": 11
          }
        }
        ]
      }
    }
```

Reference

Property	Description
id	Internal identifier for the job group
name	Internal name of the job group
description	User-friendly description for the job group
ranfrom	Where the job group was executed from: `ui` - Trifacta Application `cli` - command line interface
status	Current status of the job group: • `Pending` - job group is queued for execution. • `Running` - job group is currently running. • `Completed` - job group has completed successfully. • `Failed` - job group has failed.
profilingEnabled	When `true`, a profiling job was executed as part of this job group.
createdAt	Timestamp for when the job group was launched
updatedAt	Timestamp for when the job group was last updated
createdBy	Internal identifier for the user who created the job group
updatedBy	Internal identifier for the user who last updated the job group
wrangledDataset	Internal identifier of the object from where the job group was executed.
snapshot	Internal identifier of the data snapshot for the job group
wrangleScript	Internal identifier of the Wrangle script to execute for the job group
jobs	A list of all jobs that were launched based on this job group

API JobGroups Put Publish v3

Contents:

For a specified jobGroup, this endpoint performs an ad-hoc publish of the results to the designated target.

- Target information is based on the specified connection.
- Job results to published are based on the specified jobGroup.

You can specify:

- Database and table to which to publish
- Type of action to be applied to the target table. Details are below.

Supported targets:

- Hive

- Redshift

For more information on jobGroups, see *API JobGroups Get v3*.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `PUT`

Endpoint:

```
/v3/jobGroups/<id>/publish
```

where:

Parameter	Description
`<id>`	Internal identifier for the job group

Request URI - Example:

```
/v3/jobGroups/2/publish
```

Request Body:

```
{
  "connection": {
    "id": 1
  },
  "database": "default",
  "table": "test_table",
  "action": "create",
  "inputFormat": "avro"
}
```

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
  "jobgroupId": 2,
  "jobIds": [
    11
  ],
  "reason": "JobStarted",
  "sessionId": "d9f13aa0-3b35-11e7-9bff-c764dff4fad8"
}
```

Reference

Request Reference:

Property	Description
`connection`	Internal identifier of the connection to use to write the results.
`database`	Name of database to which to write the results.
`table`	Name of table in the database to which to write the results.
`action`	Type of writing action to perform with the results. Supported actions: • `create` - Create a new table with each publication. This table is empty except for the schema, which is taken from the results. A new table receives a timestamp extension to its name. • `load` - Append a pre-existing table with the results of the data. The schema of the results and the table must match. • `createAndLoad` - Create a new table with each publication and load it with the results data. A new table receives a timestamp extension to its name. • `truncateAndLoad` - Truncate a pre-existing table and load it with fresh data from the results. • `dropAndLoad` - Drop the target table and load a new table with the schema and data from the results.
`inputFormat`	Source format of the results. Supported values: **Hive:** • `avro` • `pqt` **Redshift:** **NOTE:** For results to be written to Redshift, the source must be stored in S3 and accessed through an S3 connection. **NOTE:** By default, data is published to Redshift using the `public` schema. To publish using a different schema, preface the `table` value with the name of the schema to use: `MySchema.MyTable`. • `csv` • `json` • `avro`

For more information on the available status messages, see *API JobGroups Get v3*.

API People Create v3

Contents:

Create a new user.

Version: `v3`

Required Permissions

The authenticated user must be an admin.

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `POST`

Endpoint:

```
/v3/people/
```

Request Body:

```
{
  "accept": "accept",
  "password": "foo",
  "password2": "foo",
  "email": "abc2234@trifacta.com",
  "name": "abc2"
}
```

Response

Response Status Code - Success: `201 - Created`

Response Body Example:

```
{
  "outputHomeDir": "/trifacta/queryResults/abc2234@trifacta.com",
  "isAdmin": false,
  "isDisabled": false,
  "id": 2,
  "email": "abc2234@trifacta.com",
  "name": "abc2",
  "updatedAt": "2017-02-08T19:07:08.985Z",
  "createdAt": "2017-02-08T19:07:08.985Z",
  "ssoPrincipal": null,
  "hadoopPrincipal": null,
  "cpPrincipal": null,
  "lastLoginTime": null,
  "awsConfig": null
}
```

Reference

Request properties:

Property	Description
accept	This property must be set to `accept` to create the user.
password2	This value confirms the value for `password`. These two property values must be identical.

For more information on the properties of a user, see *API People Get v3*.

API People Delete v3

Contents:

Delete the specified user.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `DELETE`

Endpoint:

```
/v3/people/<id>
```

where:

Parameter	Description
`<id>`	Internal identifier for the user

Request URI - Example:

```
/v3/people/2
```

Request Body:

Empty.

Response

Response Status Code - Success: `204 - No Content`

Response Body Example:

Empty.

Reference

For more information on the properties of a user, see *API People Get v3*.

API People Get List v3

Contents:

Returns a list of all users of the platform.

Version: `v3`

Required Permissions

- If you are not logged in or are logged as a non-admin user, you can retrieve a very limited set of properties for each user.
- If you are logged in as an admin, you can retrieve the full property set for each user.

Request

Request Type: `GET`

Endpoint:

```
/v3/people
```

Endpoint with paged retrieval:

By default, this endpoint returns results in sets of 25.

You can apply query parameters to change the size of the default set and to page through result sets. The following example queries for results 100 at a time. In this case, the query asks for results 201-300:

```
/v3/people?limit=100%offset=2
```

If the count of retrieved results is less than the limit, you have reached the end of the results.

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Examples:

When not logged in, or logged in as a non-admin user:

```
{
  "data": [
    {
      "name": "ExampleUser",
      "email": "joe@example.com",
      "id": 955
    },
    {
      "name": "Example User 2",
      "email": "jim@example.com",
      "id": 888
    }
  ]
}
```

When logged in as an admin:

```
{
  "data": [
    {
      "outputHomeDir": "/trifacta/queryResults/joe@example.com",
      "id": 955,
      "email": "joe@example.com",
      "name": "ExampleUser",
      "ssoPrincipal": null,
      "hadoopPrincipal": "ExampleUser",
      "cpPrincipal": null,
      "isAdmin": false,
      "isDisabled": false,
      "lastLoginTime": null,
      "createdAt": "2017-01-25T23:22:36.707Z",
      "updatedAt": "2017-01-25T23:22:36.707Z",
      "awsConfig": null
    },
    {
      "outputHomeDir": "/trifacta/queryResults/jim@example.com",
      "id": 888,
      "email": "jim@example.com",
      "name": "Example User 2",
      "ssoPrincipal": null,
      "hadoopPrincipal": null,
      "cpPrincipal": null,
      "isAdmin": false,
      "isDisabled": false,
      "lastLoginTime": null,
      "createdAt": "2017-01-25T21:38:59.537Z",
      "updatedAt": "2017-01-25T21:38:59.537Z",
      "awsConfig": null
    }
  ]
}
```

Reference

For more information on the properties of a user, see *API People Get v3*.

API People Get v3

Contents:

Retrieve the platform account information for a user specified by userId.

Version: `v3`

Required Permissions

The authenticated user must be an admin.

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/people/<id>
```

where:

Parameter	Description
`<id>`	Internal identifier of the user to retrieve.

Request URI - Example:

```
/v3/people/2
```

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
  "outputHomeDir": "/trifacta/queryResults/foo@trifacta.com",
  "id": 2,
  "email": "foo@trifacta.com",
  "name": "foobar379216291",
  "ssoPrincipal": null,
  "hadoopPrincipal": "foo",
  "cpPrincipal": null,
  "isAdmin": false,
  "isDisabled": false,
  "lastLoginTime": "2017-01-27T23:24:54.930Z",
  "createdAt": "2017-01-12T00:37:03.138Z",
  "updatedAt": "2017-01-27T23:24:54.936Z",
  "awsConfig": null
}
```

Reference

Property	Description
outputHomeDir	Home directory where the user's generated results are written
id	Internal user identifier
email	Email address (and loginId) for the user
name	Display name for the user
ssoPrincipal	(If SSO is enabled) Principal value of the user for single-sign on
hadoopPrincipal	(If secure impersonation is enabled) Hadoop principal value for the user, which determines permissions on the Hadoop cluster
cpPrincipal	(If enabled) Principal value used to integrate with cloud platform
isAdmin	If `true`, the user account is an administrator account.
isDisabled	If `true`, the account is disabled.
lastLoginTime	Timestamp for the last time that the user logged in
createdAt	Timestamp for when the user account was created
updatedAt	Timestamp for when the user account was last modified
awsConfig	(If AWS integration is enabled) Value contains the S3 credentials, default bucket, and any extra buckets to which the user has access

API People Patch v3

Contents:

Updates the platform account information for a user specified by userId.

Version: `v3`

Required Permissions

The authenticated user must be an admin.

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `PATCH`

Endpoint:

```
/v3/people/<id>
```

where:

Parameter	Description
`<id>`	Internal identifier of the user to update.

Request URI - Example:

```
/v3/people/2
```

Request Body:

```
{
  "outputHomeDir": "/trifacta/queryResults/joe@example.com",
  "email": "joe@example.com",
  "name": "Joe Example",
  "ssoPrincipal": null,
  "hadoopPrincipal": null,
  "cpPrincipal": null,
  "isAdmin": false,
  "isDisabled": false,
  "awsConfig": null
}
```

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
  "id": 2,
  "updatedAt": "2017-05-18T19:46:46.839Z"
}
```

Reference

For more information on these properties, see *API People Get v3*.

API Releases Create DryRun v3

Contents:

Perform a dry-run of creating a release for the specified deployment, which performs a check of all permissions required to import the package, as well as any specified import rules.

- For more information on import rules, see *Define Import Mapping Rules*.

If they occur, errors are reported in the response.

After you have successfully completed a dry-run, you can formally create the release via API. See *API Releases Create v3*.

NOTE: Releases pertain to Production instances of the Trifacta® platform. For more information, see *Overview of Deployment Management*.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `POST`

Endpoint:

```
/v3/deployments/<id>/releases/dryRun
```

where:

Parameter	Description
`<id>`	Internal identifier for the deployment

Request URI - Example:

```
/v3/deployments/2/releases/dryRun
```

Request Body:

The request body must include the following key and value combination submitted as form data:

key	value
data	"@path-to-file"

Example request - curl:

```
curl -X POST \
  http://example.com:3005/v3/deployments/1/releases/dryRun \
  -H 'authorization: Basic c29sc29uQHRyaWZhY3RhLmNvbTphZG1pbg==' \
  -H 'cache-control: no-cache' \
  -H 'content-type: multipart/form-data' \
  -F data=@response.zip
```

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
    "importRuleChanges": {
        "object": [],
        "value": []
    },
    "flowName": "POS-r01 Flow"
}
```

Reference

For more information on import rule changes, see *Define Import Mapping Rules*.

API Releases Create v3

Contents:

Create a release for the specified deployment.

Release is created from a local ZIP containing the package of the flow exported from the source system.

> **NOTE:** Releases pertain to Production instances of the Trifacta® platform. For more information, see *Overview of Deployment Management*.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `POST`

Endpoint:

```
/v3/deployments/<id>/releases
```

where:

Parameter	Description
`<id>`	Internal identifier for the deployment

Request URI - Example:

```
/v3/deployments/2/releases
```

Request Body:

The request body must include the following key and value combination submitted as form data:

key	value
data	"@path-to-file"

Example request - curl:

```
curl -X POST \
  http://example.com:3005/v3/deployments/1/releases \
  -H 'authorization: Basic c29sc29uQHRyaWZhY3RhLmNvbTphZG1pbg==' \
  -H 'cache-control: no-cache' \
  -H 'content-type: multipart/form-data' \
  -F data=@response.zip
```

Response

Response Status Code - Success: `201 - Created`

Response Body Example:

```
{
    "importRuleChanges": {
        "object": [],
        "value": []
    },
    "flowName": "POS-r01 Flow"
}
```

Reference

For more information on import rule changes, see *Define Import Mapping Rules*.

API Releases Delete v3

Contents:

Delete the specified release.

> **NOTE:** Releases pertain to Production instances of the Trifacta® platform. For more information, see *Overview of Deployment Management*.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `DELETE`

Endpoint:

```
/v3/releases/<id>
```

where:

Parameter	Description
`<id>`	Internal identifier for the release

Request URI - Example:

```
/v3/releases/2
```

Request Body:

Empty.

Response

Response Status Code - Success: `204 - No Content`

Response Body Example:

Empty.

Reference

None.

API Releases Get v3

Contents:

Get the specified release.

> **NOTE:** Releases pertain to Production instances of the Trifacta® platform. For more information, see *Overview of Deployment Management*.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/releases/<id>
```

where:

Parameter	Description
`<id>`	Internal identifier for the release

Request URI - Example:

```
/v3/releases/2
```

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
    "id": 2,
    "notes": "My second release",
    "packageUuid": "b6b76bc0-a1c6-11e7-8c9d-f53cb0bb7b0a",
    "active": null,
    "createdAt": "2017-08-01T07:00:00.000Z",
    "updatedAt": "2017-10-05T12:26:36.326Z",
    "deploymentId": 1,
    "createdBy": 1,
    "updatedBy": 2
}
```

Reference

Property	Description
id	Internal identifier for the release
notes	Display value for notes that you can add to describe the release
packageUuid	Internal identifier for the package
active	If `true`, the release is the active one for the deployment. If set to `null`, the release is not active.
createdAt	Timestamp for when the release was created.
updatedAt	Timestamp for when the release was last updated.
createdBy	Internal identifier for the user who created the release.
updatedBy	Internal identifier for the user who last updated the release.

API Releases Patch v3

Contents:

Update the specified release.

> **NOTE:** Releases pertain to Production instances of the Trifacta® platform. For more information, see *Overview of Deployment Management*.

Version: `v3`

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `PATCH`

Endpoint:

```
/v3/releases/<id>
```

where:

Parameter	Description
`<id>`	Internal identifier for the release

Request URI - Example:

```
/v3/releases/2
```

Request Body: Example - Activate the release

You can use the following example to make the current release the active one for the deployment.

> **NOTE:** You can have only one active release per deployment. If this release is made active as part of this execution, the currently active release is made inactive.

> **Tip:** You can use this endpoint to deactivate a release, which prevents its jobs from being run. If there is no active release for the deployment, no jobs are run via the deployment job run endpoint. See *API Deployments Run v3*.

```
{
    "active": true
}
```

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
    "id": 2,
    "updatedBy": 1,
    "updatedAt": "2017-10-13T00:06:12.147Z"
}
```

Reference

For more information on the properties of a release, see *API Releases Get v3*.

API WrangledDatasets Create v3

Contents:

In Release 4.2, the wrangled dataset object was removed from the application interface. This API endpoint remains, as the internal platform objects have not been modified. For more information, see *Changes to the Object Model*.

Create a new wrangled dataset from the specified imported dataset or wrangled dataset. Wrangled dataset is owned by the authenticated user.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `POST`

Endpoint:

```
/v3/wrangledDatasets/
```

Request Body - Imported Dataset:

```
{
  "name": "Copy of Imported Dataset 2",
  "importedDataset": {
    "id": 2
  },
  "flow": {
    "id": 2
  }
}
```

Request Body - Wrangled Dataset:

```
{
  "name": "Copy of Wrangled Dataset 18",
  "wrangledDataset": {
    "id": 18
  },
  "flow": {
    "id": 1
  }
}
```

Response

Response Status Code - Success: `201 - Created`

Response Body Example - Imported Dataset:

```
{
  "id": 23,
  "flowId": 2,
  "scriptId": 24,
  "wrangled": true,
  "createdBy": 1,
  "updatedBy": 1,
  "updatedAt": "2017-02-08T20:28:06.067Z",
  "createdAt": "2017-02-08T20:28:06.067Z",
  "flowNodeId": null,
  "deleted_at": null,
  "activesampleId": null,
  "name": "Copy of Imported Dataset 2",
  "active": true
}
```

Response Body Example - Wrangled Dataset:

```
{
  "id": 20,
  "flowId": 2,
  "scriptId": 21,
  "wrangled": true,
  "createdBy": 1,
  "updatedBy": 1,
  "updatedAt": "2017-02-08T20:26:09.446Z",
  "createdAt": "2017-02-08T20:26:09.446Z",
  "flowNodeId": null,
  "deleted_at": null,
  "activesampleId": null,
  "name": "Copy of Wrangled Dataset 18",
  "active": true
}
```

Reference

For more information on the properties of a wrangled dataset, see *API WrangledDatasets Get v3*.

API WrangledDatasets Delete v3

Contents:

In Release 4.2, the wrangled dataset object was removed from the application interface. This API endpoint remains, as the internal platform objects have not been modified. For more information, see *Changes to the Object Model*.

Delete the specified wrangled dataset.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `DELETE`

Endpoint:

```
/v3/wrangledDatasets/<id>
```

where:

Parameter	Description

`<id>`	Internal identifier for the imported dataset

Request URI - Example:

```
/v3/wrangledDatasets/2
```

Request Body:

Empty.

Response

Response Status Code - Success: `204 - No Content`

Response Body Example:

Empty.

Reference

For more information on the properties of a wrangled dataset, see *API ImportedDatasets Get v3*.

API WrangledDatasets Get List v3

Contents:

In Release 4.2, the wrangled dataset object was removed from the application interface. This API endpoint remains, as the internal platform objects have not been modified. For more information, see *Changes to the Object Model*.

Get the list of accessible wrangled datasets for the authenticated user.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/wrangledDatasets
```

Endpoint with embedded flow:

```
/v3/wrangledDatasets?embed=flow
```

For more information, see *API WrangledDatasets Get v3*.

Endpoint with paged retrieval:

By default, this endpoint returns results in sets of 25.

You can apply query parameters to change the size of the default set and to page through result sets. The following example queries for results 100 at a time. In this case, the query asks for results 201-300:

```
/v3/wrangledDatasets?embed=flow&limit=100%offset=2
```

If the count of retrieved results is less than the limit, you have reached the end of the results.

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
  "data": [
    {
      "id": 35,
      "wrangled": true,
      "createdAt": "2017-02-03T05:16:55.844Z",
      "updatedAt": "2017-02-03T05:16:56.998Z",
      "createdBy": 1,
      "updatedBy": 1,
      "name": "base_type_map_array_record_large",
      "description": null,
      "activeSample": {
        "id": 12
      },
      "flow": {
        "id": 12
      },
      "script": {
        "id": 36
      }
    },
    {
      "id": 33,
      "wrangled": true,
      "createdAt": "2017-02-03T05:15:16.145Z",
      "updatedAt": "2017-02-03T05:15:18.859Z",
      "createdBy": 1,
      "updatedBy": 1,
      "name": "original_2_1_random_rows",
      "description": null,
      "activeSample": {
        "id": 10
      },
      "flow": {
        "id": 11
    },
    "script": {
      "id": 34
    }
  },
  {
    "id": 31,
    "wrangled": true,
    "createdAt": "2017-02-03T01:53:41.284Z",
    "updatedAt": "2017-02-03T01:53:41.284Z",
    "createdBy": 1,
    "updatedBy": 1,
    "name": "cdr.txt",
    "description": null,
    "flow": {
      "id": 10
    },
    "script": {
      "id": 32
    }
  },
  {
    "id": 30,
    "wrangled": true,
    "createdAt": "2017-02-03T01:53:40.930Z",
    "updatedAt": "2017-02-03T01:53:40.930Z",
    "createdBy": 1,
    "updatedBy": 1,
    "name": "customer_above_512k",
    "description": null,
    "flow": {
      "id": 10
    },
    "script": {
      "id": 31
    }
```

```
        }
      ]
    }
```

Reference

For more information on the properties of a wrangled dataset, see *API ImportedDatasets Get v3*.

For more information on the embedded flow properties, see *API Flows Get v3*.

API WrangledDatasets Get PrimaryInputDataset v3

Contents:

In Release 4.2, the wrangled dataset object was removed from the application interface. This API endpoint remains, as the internal platform objects have not been modified. For more information, see *Changes to the Object Model*.

Get the primary input dataset for the specified wrangled dataset. For a wrangled dataset, its **primary input dataset** is the original dataset from which the wrangled dataset was created.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/wrangledDatasets/<id>/primaryInputDataset
```

where:

Parameter	Description
`<id>`	Internal identifier for the wrangled dataset

Request URI - Example:

```
/v3/wrangledDatasets/3/primaryInputDataset
```

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example - imported dataset:

```
{
  "importedDataset": {
    "id": 47,
    "size": "292817",
    "path":
"/trifacta/uploads/1/2a677cbe-ca19-4d47-b038-65cda938588d/POS-r01.txt",
    "isSharedWithAll": false,
    "type": "hdfs",
    "cpProject": null,
    "bucket": null,
    "connectionId": null,
    "isSchematized": false,
    "createdAt": "2017-02-21T17:54:56.621Z",
    "updatedAt": "2017-02-21T17:54:56.840Z",
    "createdBy": 1,
    "updatedBy": 1,
    "parsingScriptId": 92
  }
}
```

Response Body Example - wrangled dataset:

```
{
  "wrangledDataset": {
    "id": 50,
    "scriptId": 49,
    "flowId": 10,
    "flowNodeId": null,
    "deleted_at": null,
    "wrangled": true,
    "createdAt": "2017-02-21T16:53:15.619Z",
    "updatedAt": "2017-02-21T17:04:03.257Z",
    "activesampleId": 33,
    "createdBy": 1,
    "updatedBy": 1
  }
}
```

Reference

Imported Dataset:

For more information on these properties, see *API ImportedDatasets Get v3*.

Wrangled Dataset:

Property	Description

scriptId	Internal identifier of the recipe associated with this dataset
flowId	Internal identifier of the flow that contains this dataset
flowNodeId	Internal identifier for the node of the flow to which the dataset is attached
deletedAt	Timestamp for when the dataset was deleted. If `null`, the dataset has not been deleted.
wrangled	If `true`, this dataset is a wrangled dataset

For more information on the other properties, see *API WrangledDatasets Get v3*.

API WrangledDatasets Get v3

Contents:

In Release 4.2, the wrangled dataset object was removed from the application interface. This API endpoint remains, as the internal platform objects have not been modified. For more information, see *Changes to the Object Model*.

Get the specified wrangled dataset.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v3/wrangledDatasets/<id>
```

Endpoint with embedded reference:

Use the following endpoint to embed information about the flow that contains the dataset in the response body.

```
/v3/wrangledDatasets/<id>?embed=flow
```

where:

Parameter	Description
`<id>`	Internal identifier for the wrangled dataset

Request URI - Example:

```
/v3/wrangledDatasets/35/
```

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
  "id": 35,
  "wrangled": true,
  "createdAt": "2017-02-03T05:16:55.844Z",
  "updatedAt": "2017-02-03T05:16:56.998Z",
  "createdBy": 1,
  "updatedBy": 1,
  "name": "base_type_map_array_record_large",
  "description": null,
  "activeSample": {
    "id": 12
  },
  "flow": {
    "id": 12
  },
  "script": {
    "id": 36
  }
}
```

Response Body Example with embedded reference:

```
{
  "id": 35,
  "wrangled": true,
  "createdAt": "2017-02-03T05:16:55.844Z",
  "updatedAt": "2017-02-03T05:16:56.998Z",
  "createdBy": 1,
  "updatedBy": 1,
  "name": "base_type_map_array_record_large",
  "description": null,
  "activeSample": {
    "id": 12
  },
  "flow": {
    "id": 12,
    "name": "base_type_map_array_record_large Flow",
    "description": null,
    "createdAt": "2017-02-03T05:16:55.684Z",
    "updatedAt": "2017-02-03T05:16:55.684Z",
    "createdBy": 1,
    "updatedBy": 1
  },
  "script": {
    "id": 36
  }
}
```

Reference

Wrangled Dataset:

These properties apply to the source of the wrangled dataset.

Property	Description
id	Internal identifier of the wrangled dataset
wrangled	If `true`, this dataset is a wrangled dataset
createdAt	Timestamp for when the dataset was imported
UpdatedAt	Timestamp for when the dataset was last updated
createdBy	Internal identifier of the user who created the wrangled dataset
updatedBy	Internal identifier of the user who last updated the wrangled dataset
name	Internal name of the wrangled dataset
description	User-friendly description for the wrangled dataset
activeSample	Internal identifier of the currently active sample for this dataset
flow	Internal identifier of the flow that contains this dataset
script	Internal identifier of the recipe associated with this dataset

Embedded Flow:

For more information on the embedded flow properties, see *API Flows Get v3*.

API WrangledDatasets Put PrimaryInputDataset v3

Contents:

In Release 4.2, the wrangled dataset object was removed from the application interface. This API endpoint remains, as the internal platform objects have not been modified. For more information, see *Changes to the Object Model*.

Updated the primary input dataset for the specified wrangled dataset. Each wrangled dataset must have one and only one primary input dataset, which can be an imported or wrangled dataset.

This action performs a dataset swap for the source of a wrangled dataset, which can be done through the UI. See *Flow View Page*.

Tip: After you have created a job via API, you can use this API to swap out the source data for the job's dataset. In this manner, you can rapidly re-execute a pre-existing job using fresh data. See *API JobGroups Create v3*.

Version: `v3`

Required Permissions

NOTE: Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `PUT`

Endpoint:

```
/v3/wrangledDatasets/<id>/primaryInputDataset
```

where:

Parameter	Description
`<id>`	Internal identifier for the wrangled dataset

Request URI - Example:

```
/v3/wrangledDatasets/3/primaryInputDataset
```

Request Body - imported dataset:

```
{
  "importedDataset": {
    "id": <id>
  }
}
```

Request Body - wrangled dataset:

```
{
  "wrangledDataset": {
    "id": <id>
  }
}
```

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
  "id": 50,
  "scriptId": 49,
  "flowId": 10,
  "flowNodeId": null,
  "deleted_at": null,
  "wrangled": true,
  "createdAt": "2017-02-21T16:53:15.619Z",
  "updatedAt": "2017-02-21T17:04:03.257Z",
  "activesampleId": 33,
  "createdBy": 1,
  "updatedBy": 1
}
```

Reference

For more information on these properties, see *API WrangledDatasets Get PrimaryInputDataset v3*.

API Session Get

Contents:

Get the specified session information.

Version: all

Required Permissions

> **NOTE:** Each request to the Trifacta® platform must include authentication credentials. See *API Authentication*.

Request

Request Type: `GET`

Endpoint:

```
/v2/session
```

Request Body:

Empty.

Response

Response Status Code - Success: `200 - OK`

Response Body Example:

```
{
    "outputHomeDir": "/trifacta/queryResults/admin@trifacta.local",
    "id": 1,
    "email": "me@example.com",
    "name": "My Account",
    "ssoPrincipal": null,
    "hadoopPrincipal": null,
    "cpPrincipal": null,
    "isAdmin": true,
    "isDisabled": false,
    "forcePasswordChange": false,
    "lastLoginTime": "2018-01-24T21:03:54.813Z",
    "deleted_at": null,
    "createdAt": "2018-01-24T08:29:11.248Z",
    "updatedAt": "2018-01-24T21:03:54.813Z",
    "awsconfigId": null,
    "roles": [
        {
            "id": 1,
            "role": "dataAdmin",
            "createdAt": "2018-01-24T08:28:34.369Z",
            "updatedAt": "2018-01-24T08:28:34.369Z",
            "peopleworkspaces": {
                "workspaceId": 1,
                "personId": 1,
                "roleId": 1,
                "createdAt": "2018-01-24T08:29:11.360Z",
                "updatedAt": "2018-01-24T08:29:11.360Z"
            }
        },
        {
            "id": 2,
            "role": "wrangler",
            "createdAt": "2018-01-24T08:28:34.369Z",
            "updatedAt": "2018-01-24T08:28:34.369Z",
            "peopleworkspaces": {
                "workspaceId": 1,
                "personId": 1,
                "roleId": 2,
                "createdAt": "2018-01-24T08:29:11.362Z",
                "updatedAt": "2018-01-24T08:29:11.362Z"
            }
        },
        {
            "id": 4,
            "role": "admin",
            "createdAt": "2018-01-24T08:28:39.321Z",
            "updatedAt": "2018-01-24T08:28:39.321Z",
            "peopleworkspaces": {
                "workspaceId": 1,
                "personId": 1,
                "roleId": 4,
                "createdAt": "2018-01-24T08:29:11.362Z",
                "updatedAt": "2018-01-24T08:29:11.362Z"
            }
            }
        ]
    }
```

Reference

User information:

For more information on user properties, see *API People Get v3*.

Roles:

Property	Description
id	Internal identifier for the role
role	User-friendly name of the role. Values: • `dataAdmin` • `Wrangler` **NOTE:** All valid user accounts must have the `Wrangler` role.
createdAt	Timestamp for when the role was added to the user account.
updatedAt	Timestamp for when the role was last updated to the user account.
peopleworkspaces.workspaceId	Internal identifier of the workspace to which the role applies.
peopleworkspaces.personId	Internal identifier for the user within the workspace.
peopleworkspaces.roleId	Internal identifier for this role within the workspace.
peopleworkspaces.createdAt	Timestamp for when the workspace was created.
peopleworkspaces..updatedAt	Timestamp for when the workspace was last updated.

API Version Support Matrix

Versions of the REST API are supported according to the following dates.

API Version	Status	Active Support Start Date	Active Support End/ Maintenance Start Date	End of Life Date
v4	Active	2018-04-06 (Release 5.0)		
v3	Active	2017-02-27 (Release 4.0)		

API - UI Integrations

You can automate execution of tasks against the Trifacta platform by referencing URL destinations through your Chrome browser.

Pre-requisites:

- If the user has not authenticated with the Trifacta platform, the user is redirected to the Login page, where s/he can login before completing the UI integration.
- The integration must be executed through Google Chrome.

How to:

To execute a UI integration, login to the platform and execute the following:

```
[http|https]://<host>:<port>/<UI endpoint>
```

where:

- `http` or`https` is the appropriate protocol identifier, depending on whether you have enabled SSL connections
- `<host>` is the Trifacta platform.
- `<port>` is the port number to connect to the Trifacta platform. Default value is`3005`.
- `<UI endpoint>` is the URI to use, as specified for the UI endpoint.

UI Integrations:

UI Integration - Create Dataset

Contents:

Using the following URL endpoint, you can create a dataset from another application through the Trifacta Application.

NOTE: This integration is not supported in the Wrangler Enterprise desktop application.

Pre-requisites

- If you are calling from a source application, you must be logged into that application first. See *Authentication* below.
- You must authenticate with the Trifacta platform before you are redirected to the target destination. See*API - UI Integrations*.
- This URL integration is supported on HDFS and S3 datastores.
- It is assumed that there are no conflicting datasets with the names that are used to create the dataset in this set of steps. No name validation is performed as part of this action.

Authentication

NOTE: Before using any UI integration, you must first login to the application. If you are not logged in, you are redirected to the login page, where you can input your credentials before reaching your target URL.

In addition to authentication with the Trifacta platform, the authenticated user must also have the appropriate permissions to access the assets on the datastore. This includes:

- Permissions to access the folder or directory
- Appropriate impersonated user configured for the account, if secure impersonation is enabled.
- If this dataset is going to be executed later via command line interface, you must create the dataset with the same user that will execute the job.

For more information:

Topic	Section
HDFS: permissions and security	See *Configure Hadoop Authentication*.
HDFS: usage	See *Using HDFS*. See *HDFS Browser*.
S3: permissions and security	See *Enable S3 Access*.

S3: usage	See *Using S3*. See *S3 Browser*.

Sources of Data

You can use this integration to create datasets from single files or a single directory. Below are some example URLs for sources from Hadoop HDFS or S3:

Datastore	Source Type	Example URL	Results
HDFS	Directory	hdfs:///user/warehouse/campaign_data/	User can choose the file through the UI to use for the dataset.
	File	hdfs:///user/warehouse/campaign_data/d000001_01.csv	User can complete the steps through the UI to create the dataset.
S3	Directory	s3:///3fad-demo/data/biosci/source/	User can choose the file through the UI to use for the dataset.
	File	s3:///3fad-demo/data/biosci/source/1-DRUG15Q1.txt	User can complete the steps through the UI to create the dataset.

NOTE: The above results assume that the user has the appropriate permissions to access the file or directory. If the user lacks permissions, an HTTP 404 error is displayed.

Step-by-Step Guide

Steps:

1. Acquire the target URL for the datastore through the Trifacta® Application or through the datastore itself. Examples URLs:
   a. HDFS (file):

      ```
      hdfs:///user/warehouse/campaign_data/d000001_01.csv
      ```

   b. S3 (directory):

      ```
      s3:///3fad-demo/data/biosci/source/
      ```

2. Navigate the browser to the appropriate URL in the Trifacta platform. The following example applies to the HDFS file example from above. It must be preceded by the base URL for the platform. For more information, see *API - UI Integrations*.

   ```
   <base_url>/import/data?uri=hdfs:///user/warehouse/campaign_data/d0000
   01_01.csv
   ```

3. For file-based URLs, the file is selected automatically.
4. For directory-based URLs, the user can select which ones to include through the browser. Click the Add Datasets to a Flow. Add the dataset to an existing flow or create a new one for it.
5. After the datasets have been imported, open the flow in which your import is located. For the datasets that you wish to execute, you should do the following in the Flow View page:
   a. Click the icon for the dataset.
   b. From the URL, retrieve the identifiers for the flow and the dataset. These values are needed for later execution through the command line interface.
   c. Example:

Dataset URL	flowId	datasetId
`http://latest-dev.trifacta.net:3005/flows/31#dataset=186`	`31`	`186`

      The flowId is consistent across all datasets that you imported through the above steps.
6. You can open the datasets and wrangle them as needed.

7. Complete any required actions from within your source application.

You can run jobs on the dataset through the following interfaces:

- **UI:** See *Run Job Page*.
- **API:** See *API JobGroups Create v3*.
- **CLI:** See *CLI for Jobs*.

API Workflows

In this section, you can review examples of how to execute workflows using one or more of the available API endpoints.

Topics:

API Workflow - Develop a Flow

Contents:

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 workflow applies to a Development instance of the Trifacta® platform, which is the default platform instance type. For more information on Development and Production instance, see *Overview of Deployment Management*.

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 workflow.

Example Files:

The following files are stored on your HDFS deployment:

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 *API ImportedDatasets Create v3*.
- See *API WrangledDatasets Create v3*.

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 *API Flows Get List v3*.

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/v3/flows`
Authentication	Required
Method	`POST`

Request Body	`{ "name": "Point of Sale - 2013", "description": "Point of Sale data for state" }`

```
{
  "name": "Point of Sale - 2013",
  "description": "Point of Sale data for state"
}
```

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

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

3. Retain the flow identifier (`10`) for later use.

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:

1. Create an imported dataset for each file.
2. 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:

1. 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.
   a. path
   b. type
   c. name
   d. description
   e. bucket (if a file stored on S3)
2. Construct the following request:

Endpoint	`http://www.example.com:3005/v3/importedDataset`
Authentication	Required
Method	`POST`

Request Body	`{ "path": "/user/pos/POS-r01.txt", "type": "hdfs", "bucket": null, "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",
  "path": "/user/pos/POS-r01.txt",
  "isSharedWithAll": false,
  "type": "hdfs",
  "bucket": null,
  "isSchematized": false,
  "createdBy": 1,
  "updatedBy": 1,
  "updatedAt": "2017-02-08T18:38:56.640Z",
  "createdAt": "2017-02-08T18:38:56.560Z",
  "connectionId": null,
  "parsingScriptId": 14,
  "cpProject": null
}
```

4. You must retain the `id` value so you can reference it when you create the recipe.
5. See *API ImportedDatasets Create v3*.
6. Next, you create the recipe. Construct the following request:

Endpoint	`http://www.example.com:3005/v3/wrangledDataset`
Authentication	Required
Method	`POST`
Request Body	`{ "name":"POS-r01", "importedDataset":{"id":8}, "flow":{"id":10} }`

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

```
{
  "id": 23,
  "flowId": 10,
  "scriptId": 24,
  "wrangled": true,
  "createdBy": 1,
  "updatedBy": 1,
  "updatedAt": "2017-02-08T20:28:06.067Z",
  "createdAt": "2017-02-08T20:28:06.067Z",
  "flowNodeId": null,
  "deleted_at": null,
  "activesampleId": null,
  "name": "POS-r01",
  "active": true
}
```

8. From the recipe, you must retain the value for the `id`. For more information, see *API WrangledDatasets Create v3*.

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 page, 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` for the transform. The Join page opens.
3. Select the `RED_PROD` dataset. Click **Preview Selected Dataset**.
4. Click the Join Keys tab. Review the two keys to verify that they are the proper columns on which to structure the join.
5. Click the All link for the fields to add.
6. 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 - 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`.

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	`http://www.example.com:3005/v3/jobGroups`
Authentication	Required
Method	`POST`

Request Body	
	```
	{
	  "wrangledDataset": {
	    "id": 23
	  },
	  "overrides": {
	    "execution": "photon",
	    "profiler": true,
	    "writesettings": [
	      {
	        "path":
	"hdfs://hadoop:50070/trifacta/queryResults/admin@trifacta.local/
	        "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 Photon running environment with profiling enabled.
   a. Output format is CSV to the designated path. For more information on these properties, see *API JobGroups Create v3*.
   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:

```
{
  "jobgroupId": 3,
  "jobIds": [
    5,
    6
  ],
  "reason": "JobStarted",
  "sessionId": "9c2c6220-ef2d-11e6-b644-6dbff703bdfc"
}
```

5. Retain the `jobgroupId` value for monitoring.

Step - Monitoring Your Job

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

Endpoint	`http://www.example.com:3005/v3/jobgroup/<id>/status`
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 *API JobGroups Get Status v3*.

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 *API WrangledDatasets Put PrimaryInputDataset v3*.

Endpoint	`http://www.example.com:3005/v3/jobGroups`
Authentication	Required
Method	`POST`
Request Body	`{ "wrangledDataset": { "id": 23 } }`

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.

API Workflow - Deploy a Flow

Contents:

Overview

In this workflow, 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 Management*.

Pre-requisites

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.

  > **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.

Workflow

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/v3/flows`
Authentication	Required
Method	`GET`
Request Body	None.

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

```
[
    {
        "id": 21,
        "name": "Wrangle Orders",
        "description": null,
        "deleted_at": null,
        "cpProject": null,
        "createdAt": "2017-11-27T18:19:12.763Z",
        "updatedAt": "2017-11-27T18:19:12.763Z",
        "createdBy": 2,
        "updatedBy": 2,
        "associatedPeople": [
            {
                "outputHomeDir":
"/trifacta-hdp26/queryResults/user1@example.com",
                "name": "User 1",
                "email": "user1@example.com",
                "id": 2,
                "flowpermission": {
                    "flowId": 21,
                    "personId": 2,
                    "role": "owner"
                }
            }
        ]
    },
    {
        "id": 19,
        "name": "example Flow",
        "description": null,
        "deleted_at": null,
        "cpProject": null,
        "createdAt": "2017-11-15T23:00:24.263Z",
        "updatedAt": "2017-11-15T23:00:24.263Z",
        "createdBy": 2,
        "updatedBy": 2,
        "associatedPeople": [
            {
                "outputHomeDir":
"/trifacta-hdp26/queryResults/user1@example.com",
                "name": "User 1",
                "email": "user1@example.com",
                "id": 2,
                "flowpermission": {
                    "flowId": 19,
                    "personId": 2,
                    "role": "owner"
                }
            }
        ]
    }
]
```

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

> **Checkpoint:** You have identified the flow to export.

For more information, see *API Flows Get v3*.

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/v3/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 *API Flows Package Get v3*.

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/v3/deployments`
Authentication	Required **NOTE:** Username and password credentials must be submitted for the `Admin2` account.
Method	`POST`
Request Body	`{ "name": "Production Orders" }`

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

```
{   "id": 3,
    "name": "Production Orders",
    "createdBy": 1,
    "updatedBy": 1,
    "updatedAt": "2017-11-27T23:48:54.340Z",
    "createdAt": "2017-11-27T23:48:54.340Z"
}
```

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

For more information, see *API Deployments Create v3*.

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/v3/connections`
Authentication	Required **NOTE:** Username and password credentials must be submitted for the `User1` account.
Method	`GET`
Request Body	None.

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

```
{
    "data": [
        {
            "connectParams": {
                "vendor": "redshift",
                "host": "dev-redshift.example.com",
                "port": "5439",
                "defaultDatabase": "devWrangleDB",
                "extraLoadParams": "BLANKSASNULL EMPTYASNULL
TRIMBLANKS TRUNCATECOLUMNS"
            },
            "id": 9,
            "host": "dev-redshift.example.com",
            "port": 5439,
            "vendor": "redshift",
            "params": {
                "connectStrOpts": "",
                "defaultDatabase": "devWrangleDB",
                "extraLoadParams": "BLANKSASNULL EMPTYASNULL
TRIMBLANKS TRUNCATECOLUMNS"
            },
            "ssl": false,
            "name": "Dev Redshift Conn",
            "description": "",
            "type": "jdbc",
            "createdBy": 1,
            "isGlobal": true,
            "credentialType": "custom",
            "credentialsShared": true,
            "uuid": "b8014610-ce56-11e7-9739-27deec2c3249",
            "createdAt": "2017-11-21T00:55:50.770Z",
            "updatedAt": "2017-11-21T00:55:50.770Z",
            "updatedBy": 2,
            "credentials": [
                {
                    "user": "devDBuser"
                }
            ]
        }
    ],
    "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/v3/connections`

Authentication	Required **NOTE:** Username and password credentials must be submitted for the `Admin2` account.
Method	`POST`
Request Body	see below

```
{
      "name": "Redshift Conn Prod",
      "description": "",
      "isGlobal": true,
      "type": "jdbc",
      "host": "prod-redshift.example.com",
      "port": 1433,
      "vendor": "redshift",
      "params": {
        "connectStrOpts": "",
        "defaultDatabase": "prodWrangleDB",
        "extraLoadParams": "BLANKSASNULL EMPTYASNULL
TRIMBLANKS TRUNCATECOLUMNS"
      },
      "ssl": false,
      "credentialType": "custom",
      "credentials": [
          {
            "username": "prodDBUser",
            "password": "<password>"
          }
      ]
}
```

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

```
{
  "host": "prod-redshift.example.com",
  "port": 5439,
  "vendor": "redshift",
     "params": {
       "connectStrOpts": "",
       "defaultDatabase": "prodWrangleDB",
       "extraLoadParams": "BLANKSASNULL EMPTYASNULL TRIMBLANKS
TRUNCATECOLUMNS"
     },
  "ssl": false,
  "name": "Redshift Conn Prod",
  "description": "",
  "type": "jdbc",
  "isGlobal": true,
  "credentialType": "custom",
  "credentialsShared": true,
  "credentials": [
          "username": "prodDBUser"
  ]
}
```

6. When you hit the `/v3/connections` endpoint again, you can retrieve the connectionId for this connection. In this case, let's assume that the connectionId value is `12`.

See *API Connections Create v3*.

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 *Define Import Mapping Rules*.

Steps:

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

Endpoint	`http://www.wrangle-dev.example.com:3005/v3/connections`
Authentication	Required **NOTE:** Username and password credentials must be submitted for the `User1` account.
Method	`GET`
Request Body	None.

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

```
{
    "data": [
        {
            "connectParams": {
                "vendor": "redshift",
                "host": "dev-redshift.example.com",
                "port": "5439",
                "defaultDatabase": "devWrangleDB",
                "extraLoadParams": "BLANKSASNULL EMPTYASNULL
TRIMBLANKS TRUNCATECOLUMNS"
            },
            "id": 9,
            "host": "dev-redshift.example.com",
            "port": 5439,
            "vendor": "redshift",
            "params": {
                "connectStrOpts": "",
                "defaultDatabase": "devWrangleDB",
                "extraLoadParams": "BLANKSASNULL EMPTYASNULL
TRIMBLANKS TRUNCATECOLUMNS"
            },
            "ssl": false,
            "name": "Dev Redshift Conn",
            "description": "",
            "type": "jdbc",
            "createdBy": 1,
            "isGlobal": true,
            "credentialType": "custom",
            "credentialsShared": true,
            "uuid": "b8014610-ce56-11e7-9739-27deec2c3249",
            "createdAt": "2017-11-21T00:55:50.770Z",
            "updatedAt": "2017-11-21T00:55:50.770Z",
            "updatedBy": 2,
            "credentials": [
                {
                    "user": "devDBuser"
                }
            ]
        }
    ],
    "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/v3/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 *API Deployments Object Import Rules Patch v3*.

See *API Deployments Value Import Rules Patch v3*.

Step - Import Release

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

Steps:

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

Endpoint	`http://www.wrangle-prod.example.com:3005/v3/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 *API Releases Create v3*.

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/v3/releases/5`
Authentication	Required
Method	`PATCH`
Request Body	`{ "active": true }`

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

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

See *API Releases Patch v3*.

Step - Run Deployment

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

Steps:

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

Endpoint	`http://www.wrangle-prod.example.com:3005/v3/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,
            "jobs": {
                "data": [
                    {
                        "id": 68
                    },
                    {
                        "id": 69
                    },
                    {
                        "id": 70
                    }
                ]
            }
        }
    ]
}
```

See *API Deployments Run v3*.

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 Workflow - Run Job on Dataset with Parameters

Contents:

Overview

This example workflow describes how to run jobs on datasets with parameters. 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.

> **NOTE:** This API workflow only works with version 4 (v4) or later of the APIs.

For more information on datasets with parameters, see *Overview of Parameterization*.

Basic Workflow

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 workflow follow the same steps for the standard workflow. Where the steps overlap links have been provided to the non-parameterized workflow. For more information, see *API Workflow - 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: `pacific`, `mountain`, `central`, and `eastern`. In this case, you can create a parameter called `r egion`, 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.

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 Identifer.

For more information, see*API Workflow - 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 recipeId values from the URL.

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

- flowId: 35
- recipeId: 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 se 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: `Regular expression`
- Matching regular expression: `[0-9][0-9]`
- Click **Save**.

In this case, the regular expression 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.

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

In the following example, the Datetime parameter has been overridden with the value `2018-04-03` as part of the job creation.

NOTE: You cannot apply overrides to these types of datasets with parameters.

1.

Endpoint	`http://www.example.com:3005/v4/jobGroups`
Authentication	Required
Method	`POST`

Request Body	`{ "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": {} } } }`

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

2. In the above example, the job has been launched for recipe `127` to execute on the Photon running environment with profiling enabled.
   a. Output format is CSV to the designated path. For more information on these properties, see *API JobGroups Create v4*.
   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:

```
{
    "reason": "JobStarted",
    "sessionId": "5b883530-3920-11e8-a37a-db6dae3c6e43",
    "id": 29,
    "jobs": {
        "data": [
            {
                "id": 62
            },
            {
                "id": 63
            }
        ]
    }
}
```

4. Retain the `jobgroupId=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	`http://www.example.com:3005/v4/jobGroups`

Authentication	Required
Method	`POST`

Request Body

```
{
  "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"
      }
    ]}
  }
}
```

2. In the above example, the job has been launched for recipe `123` to execute on the Photon running environment with profiling enabled.
    a. Output format is CSV to the designated path. For more information on these properties, see*API JobGroups Create v4*.
    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:

```
{
    "reason": "JobStarted",
    "sessionId": "aa0f9f00-391f-11e8-a37a-db6dae3c6e43",
    "id": 27,
    "jobs": {
        "data": [
            {
                "id": 58
            },
            {
                "id": 59
            }
        ]
    }
}
```

4. Retain the `jobgroupId=27` value for monitoring.

Example 3 - Dataset with pattern parameter

In the following example, the value `02` has been inserted into the pattern to execute the job on `POS-r02.csv`:

NOTE: You cannot apply overrides to these types of datasets with parameters.

1.

Endpoint	`http://www.example.com:3005/v4/jobGroups`
Authentication	Required
Method	`POST`

Request Body	
	see code below

```
{
  "wrangledDataset": {
    "id": 121
  },
  "overrides": {
    "execution": "photon",
    "profiler": false,
    "writesettings": [
      {
        "path":
"hdfs://hadoop:50070/trifacta/queryResults/admin@trifacta.local/
        "action": "create",
        "format": "csv",
        "compression": "none",
        "header": false,
        "asSingleFile": false
      }
    ],
    "runParameters": {}
  }
}
```

2. In the above example, the job has been launched for recipe `121` to execute on the Photon running environment with profiling enabled.
   a. Output format is CSV to the designated path. For more information on these properties, see*API JobGroups Create v4*.
   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:

```
{
    "reason": "JobStarted",
    "sessionId": "16424a60-3920-11e8-a37a-db6dae3c6e43",
    "id": 28,
    "jobs": {
        "data": [
            {
                "id": 60
            },
            {
                "id": 61
            }
        ]
    }
}
```

4. Retain the `jobgroupId=28` value for monitoring.

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 *API JobGroups Get Status v3*.

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

```
{
  "wrangledDataset": {
    "id": 123
  },
  "runParameters": {
    "overrides": {
      "data": [{
        "key": "region",
        "value": "central"
      }
    ]}
  }
}
```

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

TRIFACTA