## **Data workflow automation using Tableau Hyper API – Part 1 (Create, Insert)**

## **What is Hyper API?**

# The Hyper API contains a set of functions you can use to automate your interactions with Tableau extract (.hyper) files. You can use the API to create new extract files, or to open existing files, and then insert, delete, update, or read data from those files.

# Using the Hyper API developers and administrators can perform:

* Create extract files for data sources not currently supported by Tableau.
* Automate custom extract, transform and load (ETL) processes (for example, implement rolling window updates or custom incremental updates).
* Retrieve data from an extract file.

Use Case:

Scenario:

1. 20M records in data base having 134 columns.
2. Data Base- SQL Server 2019 Ent.
3. Total time (avg.) taken to refresh is 6300 seconds + as on date which will keep on increasing as long as data is also increasing.
4. Data Base Server and Tableau Server are in different data centres

Thought Applied:

1. Setup required stuff on data base server
2. Created extract in local server where data base resides using Python
3. Called Rest API to Publish the same
4. Used CURD operation to setup incremental refresh. Tableau server can refresh extract only on date fields only, so used this.

Outcome:

1. Total refresh time reduced to 300 seconds as incremental is being performed here
2. Since extract creation is getting refreshed on locally hence saved backgrounder usage
3. Most important part now we can release dashboard to end users 1.6 hours early which really made sales force happy, as they can have meeting with their team before leaving to market on data points.

The Hyper API gives you the tools for interacting with local .hyper extract files. For information about how to programmatically publish the extracts to Tableau Server.

The Hyper API, allows developers and admins to create .hyper files and insert, delete, update, and read data from those files.

New features that come with the Hyper API:

1. Full CRUD: Read, update, delete, and insert data in .hyper files.
2. Full speed: Leverage the full speed of Hyper for creating .hyper files.
3. Direct CSV loading: Directly load from .CSV files instead of writing code to do so.
4. SQL-based API: Unleash the power of SQL to interact with .hyper files.
5. Multi-table: Create multi-table extracts that match your data model.

## **Supported languages:**

The Hyper API only supports 64-bit platforms. The Hyper API libraries are available for the following programming languages:

* Python (3.6 or newer)
* C++ (C++11 or newer)
* Java (Java 8 or newer)
* C#/.NET (.NET Standard 2.0)

While it is expected that the Hyper API will work on newer versions of these languages, it may not be fully tested.

## **Supported platforms:**

* Microsoft Windows Server 2016, 2012, 2012 R2, 2008 R2, 2019
* Amazon Linux 2, Red Hat Enterprise Linux (RHEL) 7.3+, CentOS 7.3+, Oracle Linux 7.3+, Ubuntu 16.04 LTS and 18.04 LTS
* Microsoft Windows 7 or newer (64-bit)
* macOS 10.11 or newer (macOS 10.15 not yet supported)

## **Hardware requirements:**

The Hyper API has the following minimum hardware requirements.

* Intel Nehalem or AMD Bulldozer processor or newer
* 2 GB memory
* 1.5 GB minimum free disk space

Reference of above content taken from [**Tableau Hyper API Document**](https://help.tableau.com/current/api/hyper_api/en-us/index.html)

# Creating and publishing Tableau Hyper extracts with Python

## **This document will help you understand step by step in/out about Hyper API**

**Let’s Set the Stage:**

Out of supported languages by Hyper API will follow Python as these days Python is one of the hottest and favourite language of developers, data scientists. ☺

We need following stuff on laptop /desktop / server

1. Python (Jupyter Notebook)
2. Some sample data in csv or excel form
3. Important Python libraries
4. tableau-api-lib
5. tableauhyperapi
6. pandas (this will get installed by default during the installation of Jupyter)
7. datetime (this will get installed by default during the installation of Jupyter)

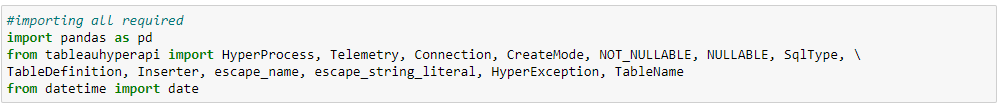
To get above API please use below commands to install for python

Pip install --upgrade tableau-api-lib

Pip install --upgrade tableauhyperapi

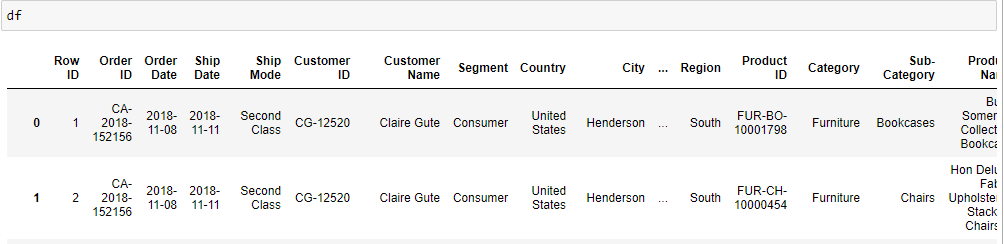
Pip install --upgrade pandas

Let’s start writing our script





Sample of my input excel file



### **1. Start the HyperProcess**

Starting Hyper Api and allowing tableau to take status of API performance as this will help them understand the usage so that they can improve it more better way.

Note: The HyperProcess can be instructed to send telemetry on Hyper API usage to Tableau. To send usage data, set telemetry to Telemetry.SEND\_USAGE\_DATA\_TO\_TABLEAU when you start the process. To opt out, set telemetry to Telemetry.DO\_NOT\_SEND\_USAGE\_DATA\_TO\_TABLEAU.

**with HyperProcess(telemetry=Telemetry.SEND\_USAGE\_DATA\_TO\_TABLEAU) as hyper:**

This starts up a local Hyper database server (hyperd). You should only start one instance of Hyper at a time. And as starting up and shutting down the server takes time, you should keep the process running and only close or shutdown the HyperProcess when your application is finished. If you call the HyperProcess in a with statement (Python), using statement (C#), scope (C++), or try-with-resources statement (Java), the hyperd process will safely shutdown.

While the HyperProcess is running, however, you can create and connect to as many .hyper files as you want.

### **2. Open a connection to the .hyper file**

**with Connection(hyper.endpoint, 'SampleSuperStore.hyper', CreateMode.CREATE\_AND\_REPLACE) as connection:**

### **3. Define the table(s)**



### **4. Create the table(s)**

**connection.catalog.create\_table(orders\_table)**

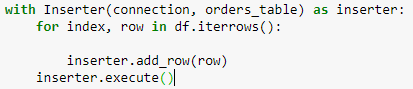
### **5. Add data to the table(s)**

Populate the table using the Inserter or use SQL commands to copy or add data.

Note: We will take data into df (data frame). Now I am using df.iterrows() function here

**What is iterrows()?**

The **iterrows()** function is used to iterate over DataFrame rows as (index, Series) pairs. Iterates over the DataFrame columns, returning a tuple with the column name and the content as a Series. The index of the row



You don’t need to manually buffer the data you are adding with the Inserter, as this handled for you.

### **6. Close the connection and shutdown the HyperProcess**

When your application is finished populating the extract file with data, you first close the connection you opened to the database (the .hyper file) and shutdown the HyperProcess.

As discussed in Step 2, if you use the Python with construct to start the process and open the connection, you don’t need to explicitly shutdown the server (hyperd) or close the connection

**Let’s put complete code together:**

# 

# Now let’s see data into extract inserted by above step.

# 

# Output:

# 

Note: If you are getting below error to see extract data.

IOPub data rate exceeded.  
The notebook server will temporarily stop sending output  
to the client in order to avoid crashing it.  
To change this limit, set the config variable  
--NotebookApp.iopub\_data\_rate\_limit.

No need to worry, please follow below steps to solve the above problem.

Open your anaconda prompt and execute below commands

jupyter notebook --generate-config

If you have already generated your Jupyter notebook file in past then no need to do it again. Just follow below steps.

jupyter notebook --generate-config

After running above command anaconda prompt show you the path of the file where it got saved, visit to the path and follow below

NotebookApp.iopub\_data\_rate\_limit

NotebookApp.rate\_limit\_window

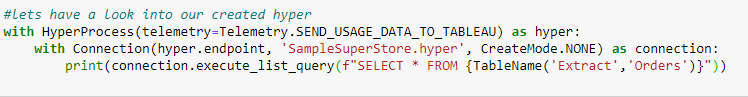
Un-comment above row in config file like below and increase the limit

c.NotebookApp.iopub\_data\_rate\_limit = 1000000000

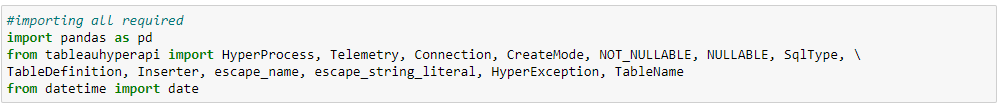
c.NotebookApp.rate\_limit\_window = 30

Save config file and restart the Jupyter notebook

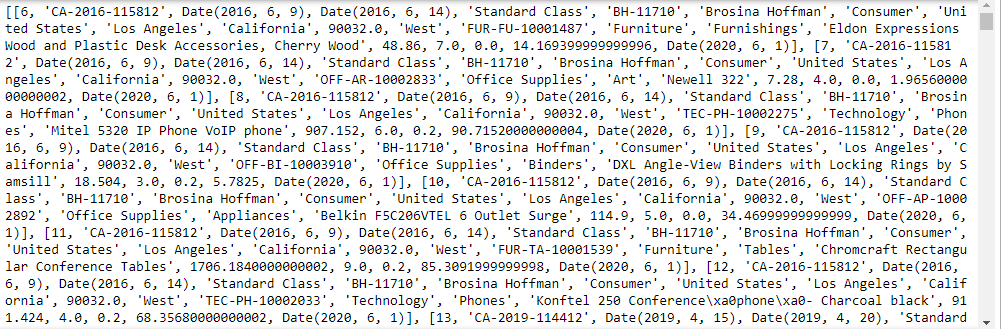
And again try this



Note: Since you have restarted the Jupyter, please import all required dependencies then try above step



You will get below result



Now you can consume this hyper extract into your Tableau desktop or can also publish the same to server.

Now let’s try to publish the extract to server



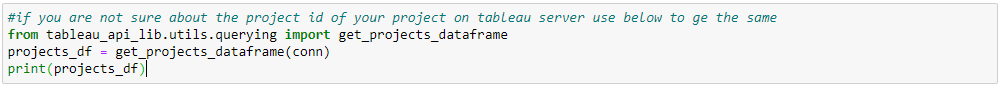
If you are not sure about the API version please visit below table.

|  |  |  |
| --- | --- | --- |
| **Tableau Server version** | **REST API version** | **Schema version** |
| 8.3, 9.0 | 2 | 2 |
| 9.0.1 and later versions of 9.0 | 2 | 2.0.1 |
| 9.1 | 2 | 2.0.1 |
| 9.2 | 2.1 | 2.1 |
| 9.3 | 2.2 | 2.2 |
| 10 | 2.3 | 2.3 |
| 10.1 | 2.4 | 2.4 |
| 10.2 | 2.5 | 2.5 |
| 10.3 | 2.6 | 2.6 |
| 10.4 | 2.7 | 2.7 |
| 10.5 | 2.8 | 2.8 |
| 2018.1 | 3 | 3 |
| 2018.2 | 3.1 | 3.1 |
| 2018.3 | 3.2 | 3.2 |
| 2019.1 | 3.3 | 3.3 |
| 2019.2 | 3.4 | 3.4 |
| 2019.3 | 3.5 | 3.5 |
| 2019.4 | 3.6 | 3.6 |
| 2020.1 | 3.7 | 3.7 |
| 2020.2 | 3.8 | 3.8 |

Click here to know more about Rest [**API version**](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_concepts_versions.htm)



If you are not sure about the Project ID please use below to get the same



**Next episode will be on Delete, Update & Insert (only incremental records)!!**