**Table of Contents**

[**1.** **Prerequisites** 2](#_Toc4081658)

[**2.** **Start, monitor and stop daemon process in one DAG** 2](#_Toc4081659)

[**3.** **FileSensor using CustomOperator** 3](#_Toc4081660)

[**4.** **Airflow API endpoints** 5](#_Toc4081661)

[**5.** **Airflow API endpoints through Postman** 7](#_Toc4081662)

# **Prerequisites**

* Python 3.6.7
* Airflow 1.10.2
* CentOS 7
* PostgreSQL 11

# **Airflow Installation**

This file demonstrates installation of Python 3.6, Apache airflow on CentOS 7 and configuring PostgreSQL database as the meta database for airflow instead of the default SQLite database.

## **Perquisites**

Root access on CentOS 7 machine

## **System update commands**

[root@localhost ~]# yum update

[root@localhost ~]# yum install yum-utils

[root@localhost ~]# yum groupinstall development

## **Installing Python 3.6**

[root@localhost ~]# yum install https://centos7.iuscommunity.org/ius-release.rpm

[root@localhost ~]# yum install python36u

[root@localhost ~]# yum install python36u-pip

[root@localhost ~]# yum install python36u-devel

Testing you Python 3.6 installation

[root@localhost ~]# python3.6 –V

## **Installing PostgreSQL 11**

yum update # Only if necessary

rpm -Uvh https://download.postgresql.org/pub/repos/yum/11/redhat/rhel-7-x86\_64/pgdg-centos11-11-2.noarch.rpm

yum install postgresql11-server postgresql11-contrib

/usr/pgsql-11/bin/postgresql-11-setup initdb

systemctl start postgresql-11.service

To check if PostgreSQL is running

ps -eaf | grep postgres

#### To access the PostgreSQL console to type commands

[root@localhost ~]# sudo su - postgres

-bash-4.2$ psql

Creating database for airflow and adding user in PostgreSQL database

postgres=# CREATE USER <your\_username\_here> PASSWORD '<your\_password\_here>';

postgres=# CREATE DATABASE your\_database\_name;

postgres=# GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO <your\_username\_here>;

postgres=# ALTER USER <your\_username-here> SUPERUSER;

postgres=# ALTER USER <your\_username-here> CREATEDB;

If it is PostgreSQL role instead of user then use commands below

postgres=# ALTER ROLE <your\_username-here> SUPERUSER;

postgres=# ALTER ROLE <your\_username-here> CREATEDB;

postgres=# ALTER ROLE <your\_username-here> WITH LOGIN;

Restart of PostgreSQL is required. To restart the PostgreSQL service

[root@localhost ~]# systemctl restart postgresql-11.service

Changing the necessary PostgreSQL configuration files to help airflow connect to PostgreSQL database

[root@localhost ~]# sudo su - postgres

-bash-4.2$ cd <postgresql\_version\_directory>/data

Once in this directory we need to edit two file

* pg\_hba.conf

Under IPv4 Local Connections

TYPE DATABASE USER ADDRESS METHOD

\# IPv4 Local Connections:

host all all 127.0.0.1/32 trust

* postgresql.conf

Under Connection Settings

# — Connection Settings -

#listen\_addresses = ‘localhost’ # what IP address(es) to listen on;

listen\_addresses = ‘\*’ # Add this line for Apache-Airflow connection

Restart of PostgreSQL is required after changing the config files. To restart the PostgreSQL service

[root@localhost ~]# systemctl restart postgresql-11.service

## **Creating virtual environment to set up Apache Airflow**

[root@localhost ~]# python3.6 -m venv <your\_environment\_name>

[root@localhost ~]# . <your\_environment\_name>/bin/activate

## **Installing Apache Airflow**

[root@localhost ~]# AIRFLOW\_GPL\_UNIDECODE=yes pip install apache-airflow[postgres]

Change directory to airflow home to edit the airflow.cfg file. Change the SQLalchemy connection string to point to PostgreSQL instead of SQLite

sql\_alchemy\_conn = postgresql+psycopg2://<postgresql\_username>@127.0.0.1:5432/<database\_name>

Example:

sql\_alchemy\_conn = postgresql+psycopg2://nmathew@127.0.0.1:5432/airflow

Save the changes in config and initialize the airflow database using below:

[root@localhost ~]# airflow initdb

Open new terminal and start the airflow scheduler

[root@localhost ~]# airflow scheduler

Open another new terminal and start the airflow webserver

[root@localhost ~]# airflow webserver

Open your web browser and navigate to <http://localhost:8080> to view the airflow UI

# **Start, monitor and stop daemon process in one DAG**

The main aim was to build a DAG which would help to start, stop and monitor daemons through the Apache Airflow UI. The code for the DAG is given below.

Code:



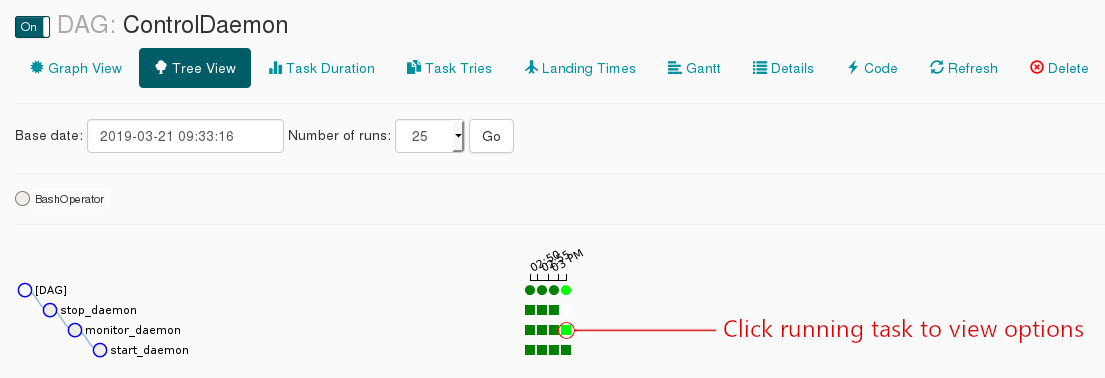
In this Airflow DAG, we perform three tasks

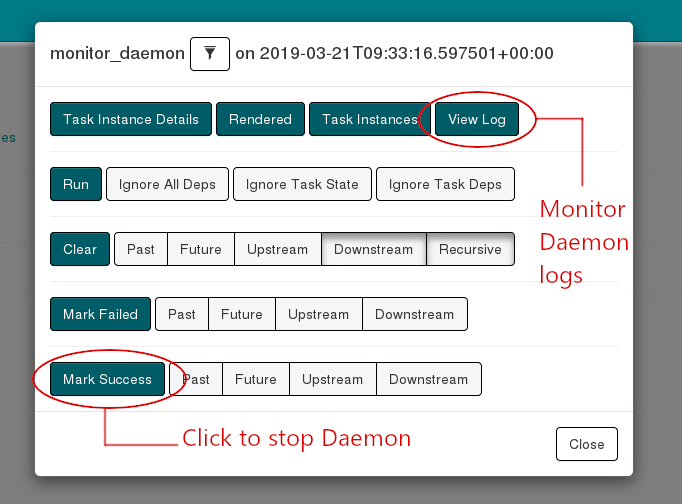
* start\_daemon service

In this task we start the daemon service using the Airflow BashOperator and passing the bash command systemctl start <service-name>.service

* keep\_watch task

In this task we monitor the daemon started in the previous task using the Airflow BashOperator and starting a forever running while loop with systemctl status <service-name>.service as the bash command. This ensures that the task keeps running forever to help monitor the daemon. In order to stop the daemon, this task state has to be marked success to proceed.





* stop\_daemon service

In this task we stop the daemon service using the Airflow BashOperator and passing the bash command systemctl stop <service-name>.service

# **FileSensor using CustomOperator**

This task senses files of a particular pattern in a folder. Here, it senses files with “nous” in their names. The code for DAG and the custom operator to sense pattern in file names is given below

Code:



filepath and filepattern are set as variables. sense\_path has the path of the destination folder inside which the files are to be sensed. sense\_pattern has regex to sense the particular pattern. In this case, files with “nous” in the names must be sensed. In addition, the sensor as mentioned pokes the folder every 3 seconds. i.e., poke\_interval = 3.

Custom Operator:



* This is a custom operator extended using BaseSensorOperator. Create a Plugins folder in the path where airflow.cfg is present. Inside plugins folder, all the custom operators file must be created.
* Inside poke method, we have used regular expressions re.match method to match the file pattern. It keeps sensing until the folder gets a file with “nous” in its name. Once, the file is present, it returns true and the task gets over.
* Plugin has to be created to add the custom operator to airflow.operators.

# **Airflow API endpoints**

Every task is written using BashOperator. And the common URL is http://172.16.0.4:8080/api/experimental/

Code:





* task\_check\_server\_ok: It checks if the REST API server works correctly and returns status as ‘OK’
* task\_trigger\_dag: It triggers a particular dag mentioned in the bash command.

*Note: We are able to trigger the dag but tasks are not running. Need to work on this.*

* task\_list\_task\_info: It returns the detailed information about the task.
* task\_latest\_runs: It returns the latest dagrun of each dag formatted for the UI.
* task\_simple\_info: It returns a JSON object with brief information of the task.
* task\_view\_pools: It returns the details of all the pools. We can even get a particular pool by specifying the pool name.
* task\_create\_pools: It creates a pool. We can specify the name, slots and description in the command.
* task\_delete\_pool: It deletes a particular pool mentioned in the code.

# **Airflow API endpoints through Postman**

**Syntax:**

**POST /api/experimental/dags/<DAG\_ID>/dag\_runs**

Example:

POST <http://172.16.0.4:8080/api/experimental/dags/hello_world_dag/dag_runs>

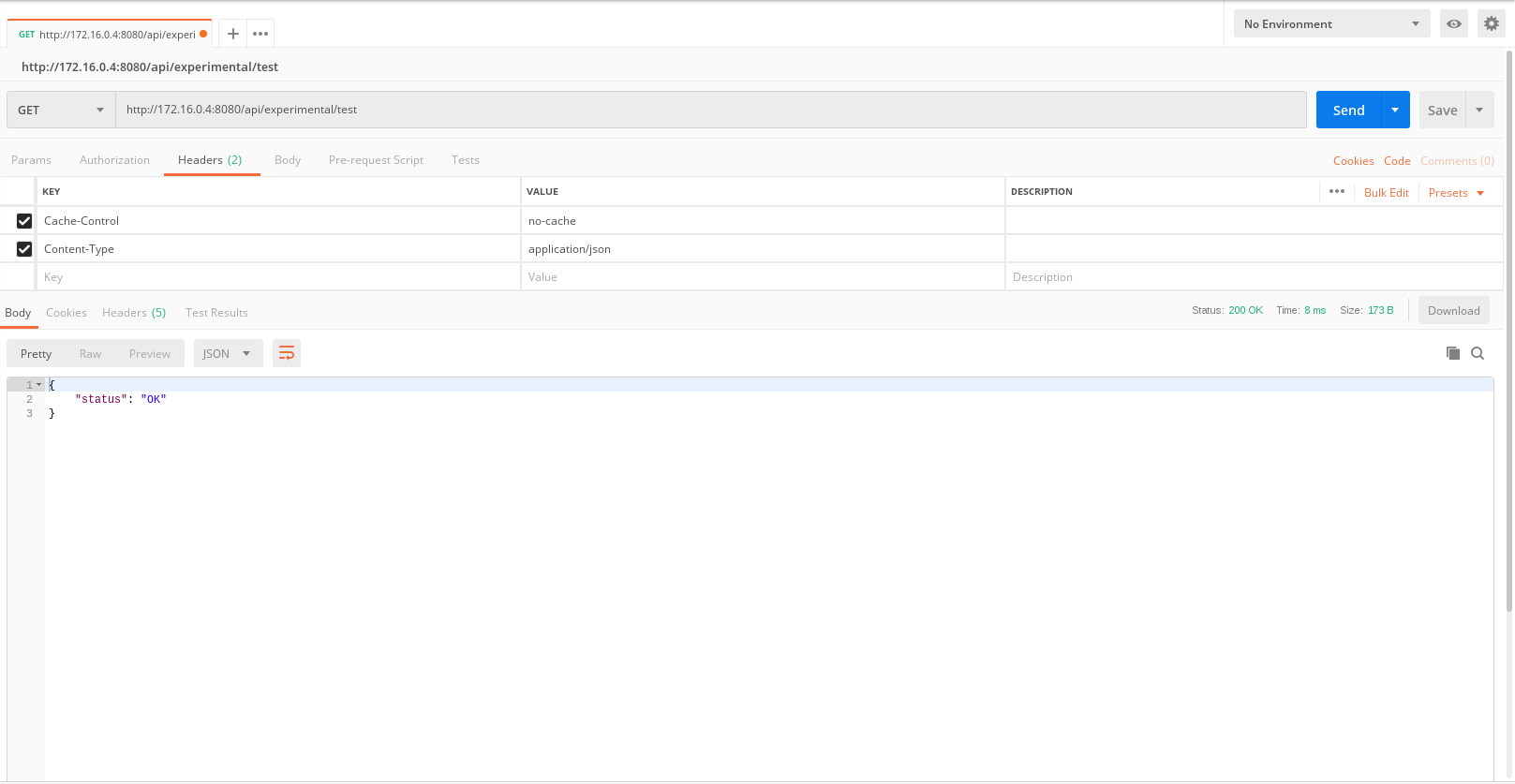
****

**Syntax:**

**GET /api/experimental/test**

Example:

GET <http://172.16.0.4:8080/api/experimental/test>

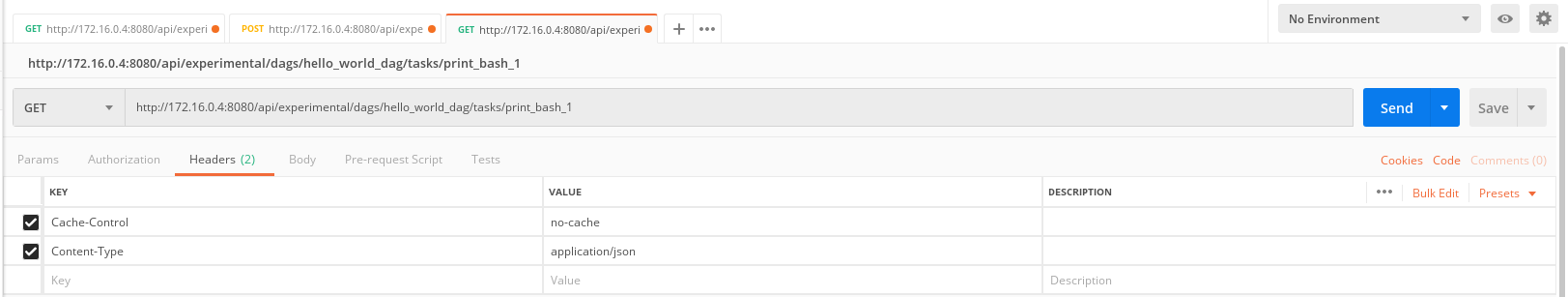
****

**Syntax:**

**GET /api/experimental/dags/<DAG\_ID>/tasks/<TASK\_ID>**

Example:

GET <http://172.16.0.4:8080/api/experimental/dags/hello_world_dag/tasks/print_bash_1>

****

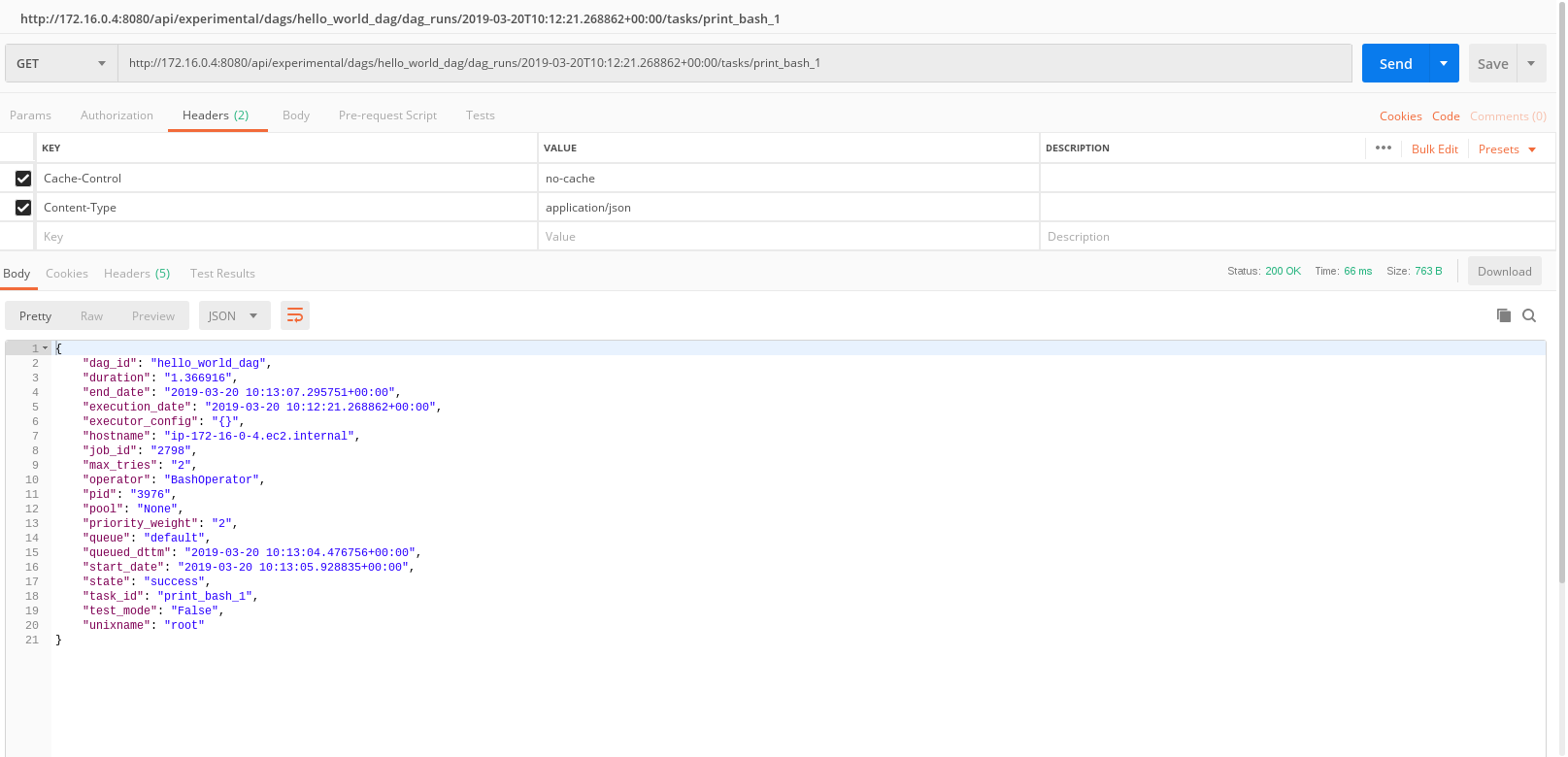
****

**Syntax:**

**GET /api/experimental/dags/<DAG\_ID>/dag\_runs/<string:execution\_date>/tasks/<TASK\_ID>**

Example:

GET <http://172.16.0.4:8080/api/experimental/dags/hello_world_dag/dag_runs/2019-03-20T10:12:21.268862+00:00/tasks/print_bash_1>

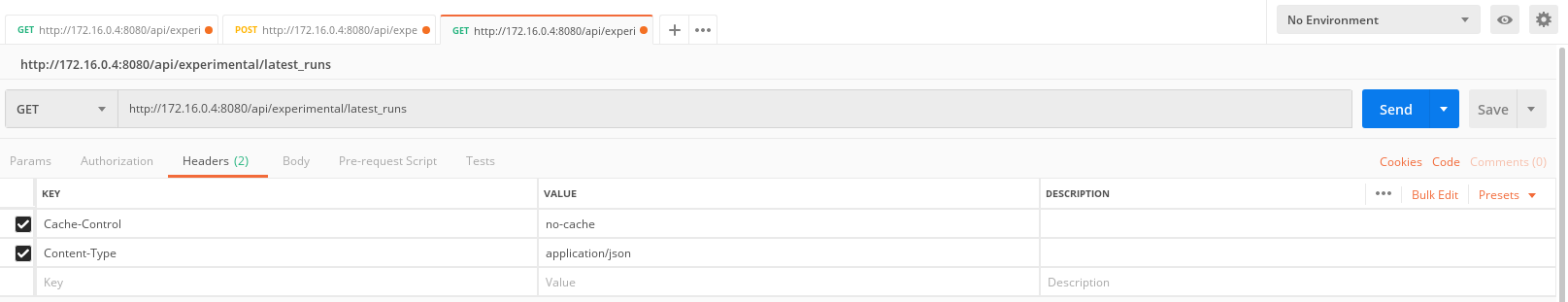
****

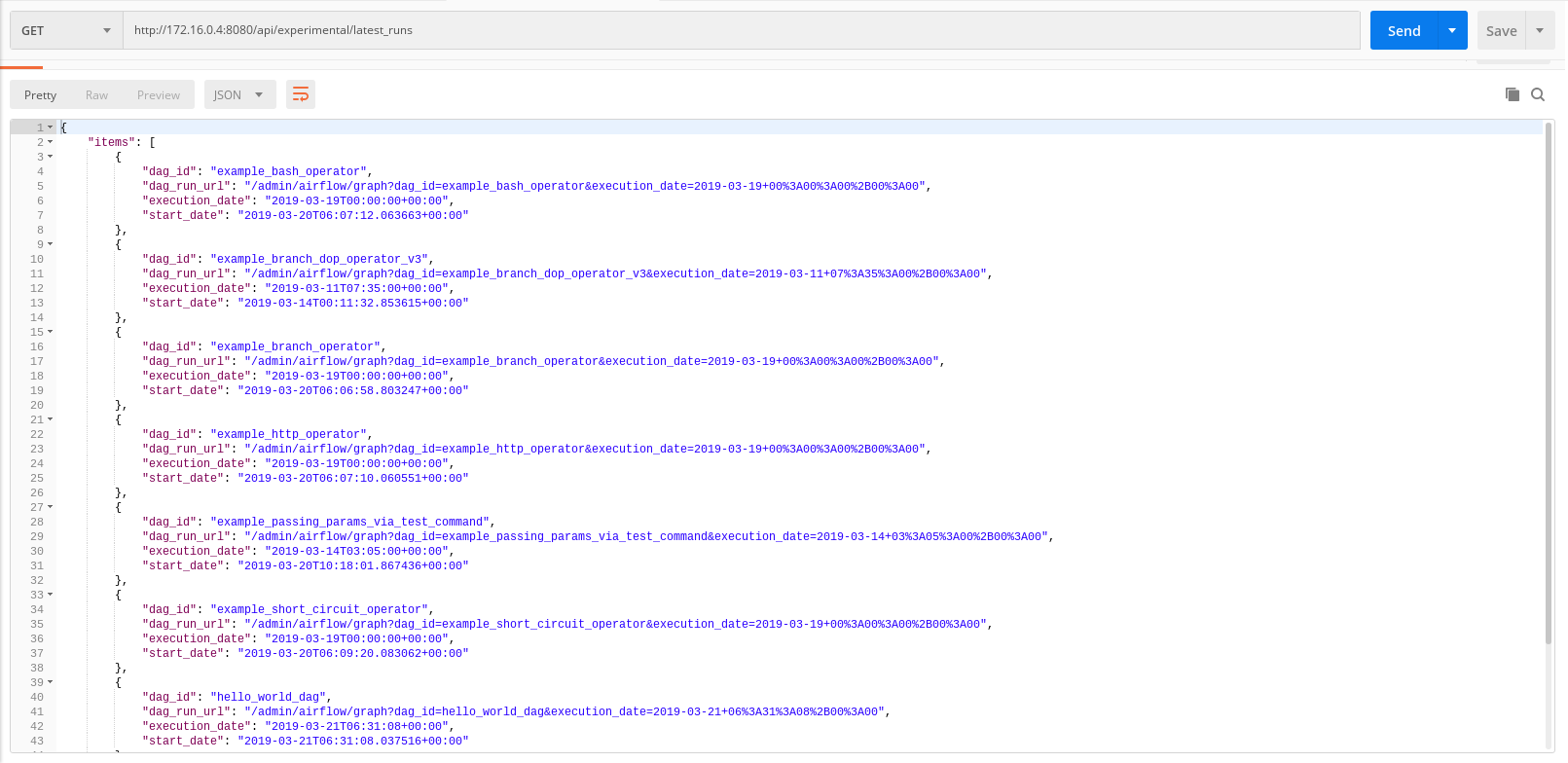
**Syntax:**

**GET /api/experimental/latest\_runs**

Example:

GET <http://172.16.0.4:8080/api/experimental/latest_runs>

****

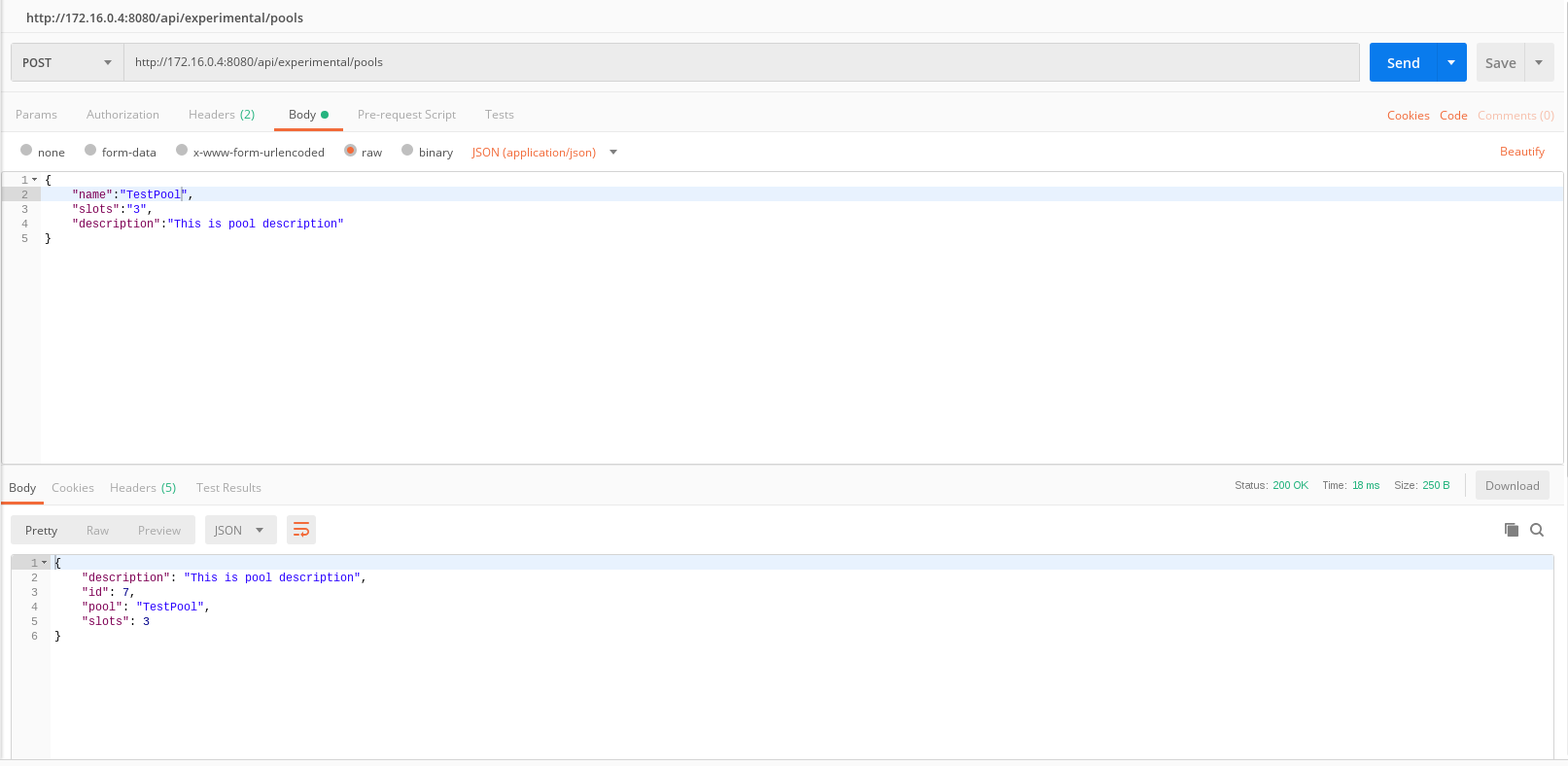
****

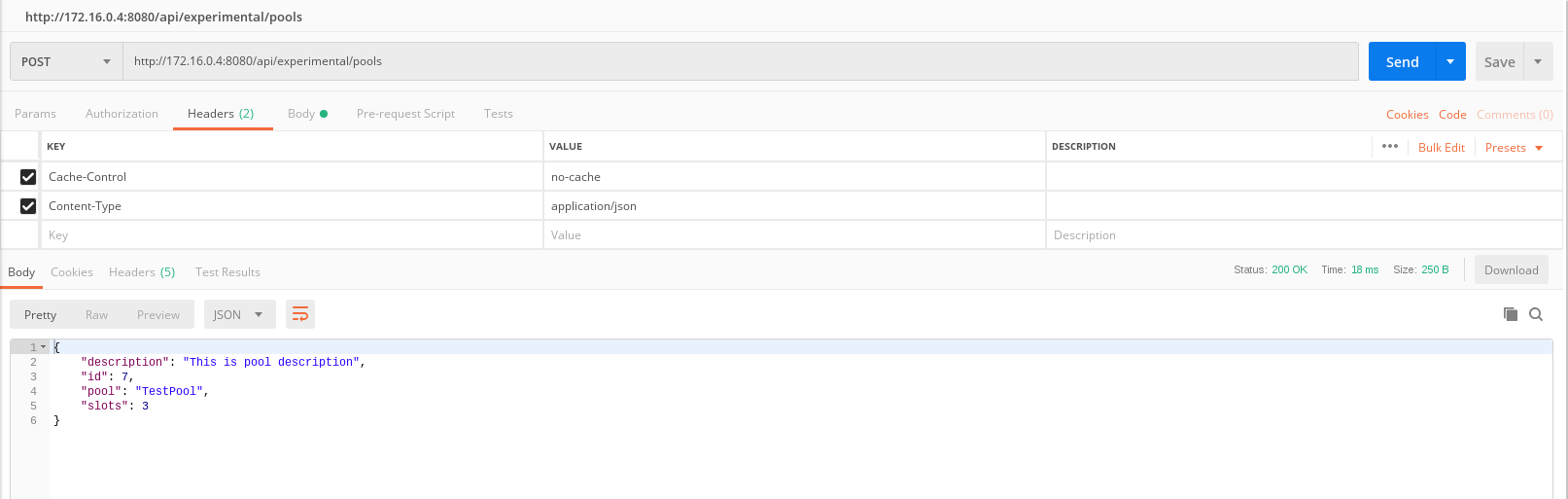
**Syntax:**

**POST /api/experimental/pools**

Example:

POST http://172.16.0.4:8080/api/experimental/pools -H 'Cache-Control: no-cache' -H 'Content-Type: application/json' -d '{\"name\":\"TestPool6\", \"slots\":\"3\", \"description\":\"This is pool 4\"}

****

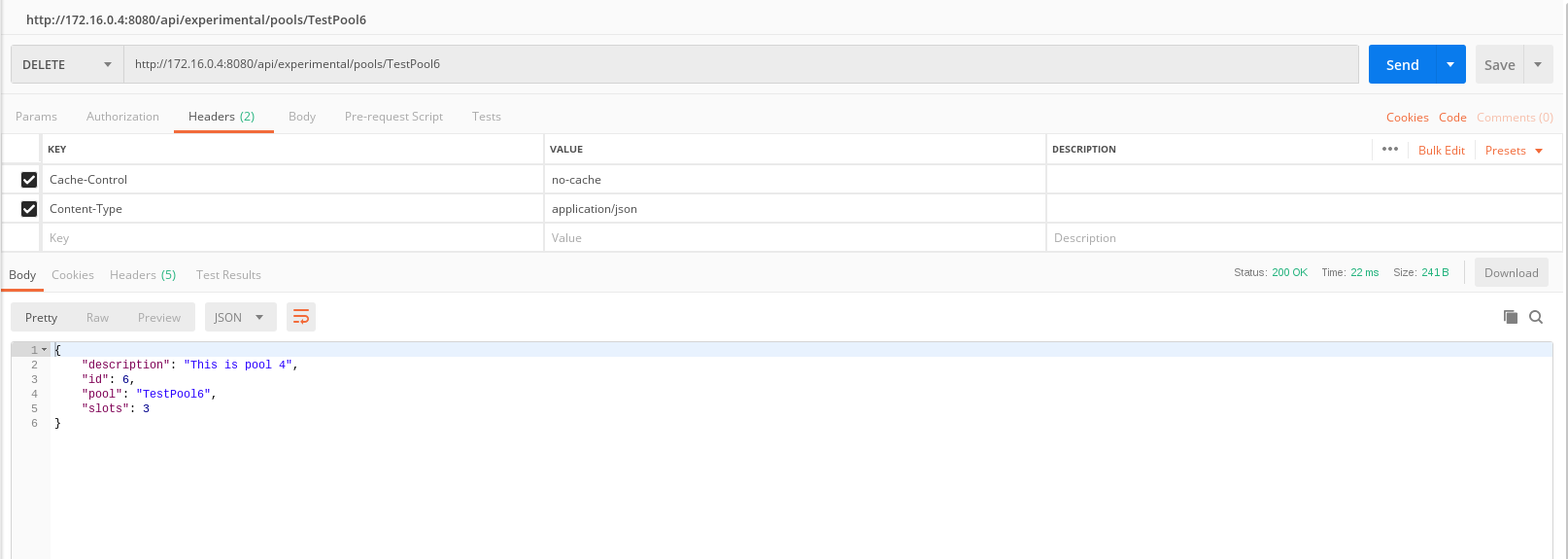
****

**Syntax:**

**DELETE /api/experimental/pools/<string:name>**

Example:

DELETE <http://172.16.0.4:8080/api/experimental/pools/TestPool5>

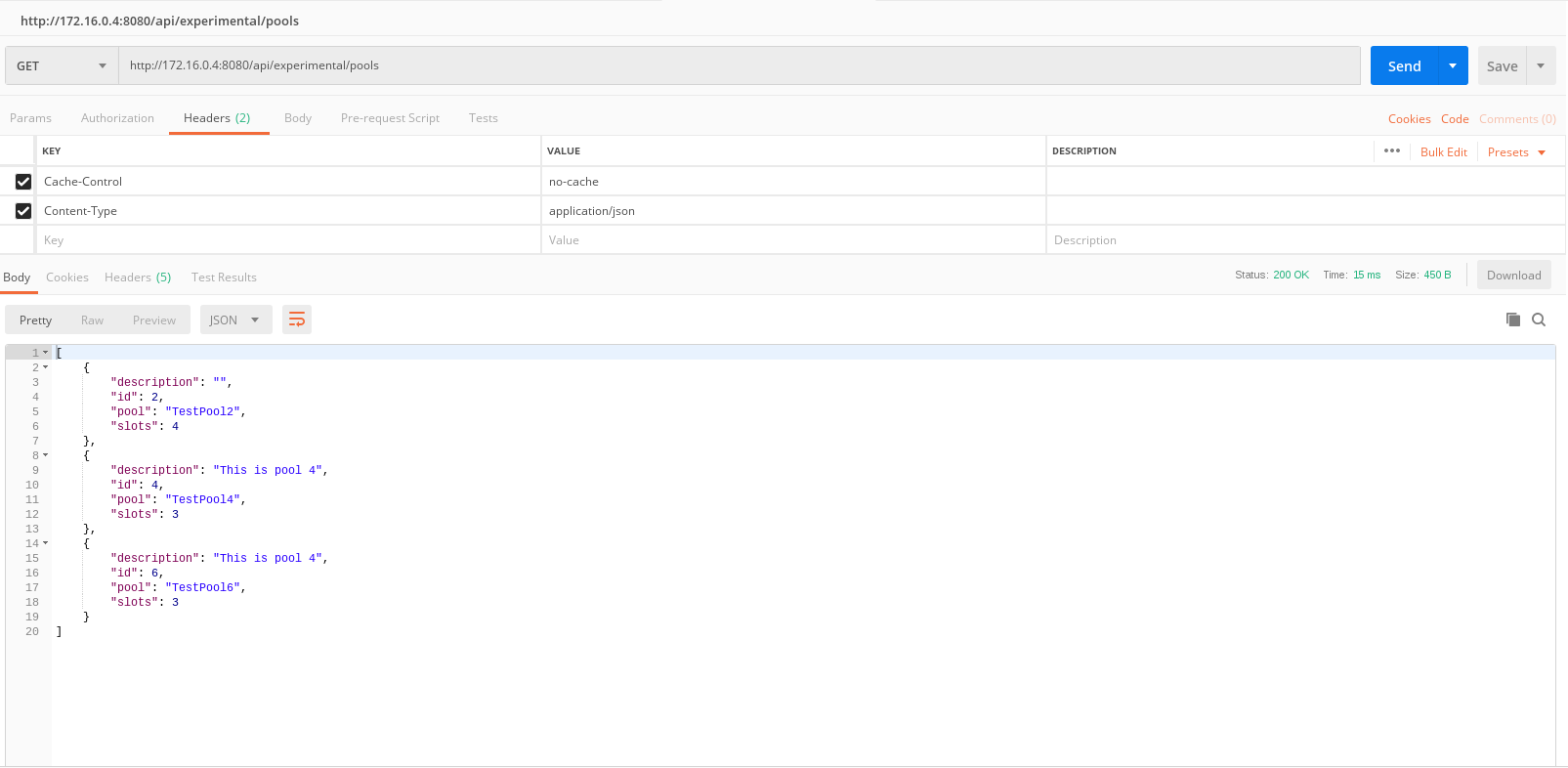
****

**Syntax:**

**GET /api/experimental/pools**

Example:

GET <http://172.16.0.4:8080/api/experimental/pools>

****

**Syntax:**

**GET /api/experimental/dags/<DAG\_ID>/dag\_runs**

***Note: This end point did not work***

Example:

GET <http://172.16.0.4:8080/api/experimental/dags/hello_world_dag/dag_runs>

**Syntax:**

**GET /api/experimental/dags/<DAG\_ID>/paused/<string:paused>**

***Note: This end point did not work***

Example:

GET <http://172.16.0.4:8080/api/experimental/dags/hello_world_dag/paused/true>

**Syntax:**

**GET /api/experimental/dags/<string:dag\_id>/dag\_runs/<string:execution\_date>**

***Note: This end point did not work***

Example:

GET <http://172.16.0.4:8080/api/experimental/dags/hello_world_dag/dag_runs/2019-03-20T00:00:00>