pylivy

Andrew Crozier

Jun 20, 2020

CONTENTS

Livy is an open source REST interface for interacting with Spark. `pylivy` is a Python client for Livy, enabling easy remote code execution on a Spark cluster.

CHAPTER

ONE

INSTALLATION

```
$ pip install -U livy
```

Note that `pylivy` requires Python 3.6 or later.

CHAPTER

TWO

BASIC USAGE

The *LivySession* class is the main interface provided by `pylivy`:

```
from livy import LivySession

LIVY_URL = "http://spark.example.com:8998"

with LivySession.create(LIVY_URL) as session:
    # Run some code on the remote cluster
    session.run("filtered = df.filter(df.name == 'Bob')")
    # Retrieve the result
    local_df = session.read("filtered")
```

Similarly, batch sessions in Livy can be created and managed with the *LivyBatch* class:

```
from livy import LivyBatch

LIVY_URL = "http://spark.example.com:8998"

batch = LivyBatch.create(
    LIVY_URL,
    file=(
        "https://repo.typesafe.com/typesafe/maven-releases/org/"
        "apache/spark/spark-examples_2.11/1.6.0-typesafe-001/"
        "spark-examples_2.11-1.6.0-typesafe-001.jar"
    ),
    class_name="org.apache.spark.examples.SparkPi",
)
batch.wait()
```

See *LivySession.create* or *LivyBatch.create* for the full range of options that can be specified when creating sessions or batches.

CHAPTER

THREE

AUTHENTICATION

Authenticate requests sent to Livy by passing any requests Auth object to the *LivySession*. For example, to perform HTTP basic auth do:

```
from livy import LivySession
from requests.auth import HTTPBasicAuth

auth = HTTPBasicAuth("username", "password")

with LivySession.create(LIVY_URL, auth) as session:
    session.run("filtered = df.filter(df.name == 'Bob')")
    local_df = session.read("filtered")
```

CHAPTER

FOUR

CUSTOM REQUESTS SESSION

`pylivy` uses requests to make HTTP requests to your Livy server. You can specify your own requests session in order to customise how requests are made to the server.

For example, to add a custom header to all requests make to Livy:

```
from livy import LivySession
import requests

LIVY_URL = "http://spark.example.com:8998"

requests_session = requests.Session()
requests_session.headers.update(
    {"X-Auth-Token": "MY-SECURITY-TOKEN"}
)

with LivySession.create(
    LIVY_URL,
    requests_session=requests_session
) as session:
    session.run("filtered = df.filter(df.name == 'Bob')")
```

CHAPTER

FIVE

API DOCUMENTATION

5.1 livy.session

class livy.session.**LivySession**(*url*, *session_id*, *auth=None*, *verify=True*, *requests_session=None*, *kind=<SessionKind.PYSPARK: 'pyspark'>*, *echo=True*, *check=True*)

Manages a remote Livy session and high-level interactions with it.

Parameters

- **url** (str) – The URL of the Livy server.
- **session_id** (int) – The ID of the Livy session.
- **auth** (Union[AuthBase, Tuple[str, str], None]) – A requests-compatible auth object to use when making requests.
- **verify** (Union[bool, str]) – Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use. Defaults to True.
- **requests_session** (Optional[Session]) – A specific requests.Session to use, allowing advanced customisation. The caller is responsible for closing the session.
- **kind** (SessionKind) – The kind of session to create.
- **echo** (bool) – Whether to echo output printed in the remote session. Defaults to True.
- **check** (bool) – Whether to raise an exception when a statement in the remote session fails. Defaults to True.

classmethod create(*url*, *auth=None*, *verify=True*, *requests_session=None*, *kind=<SessionKind.PYSPARK: 'pyspark'>*, *proxy_user=None*, *jars=None*, *py_files=None*, *files=None*, *driver_memory=None*, *driver_cores=None*, *executor_memory=None*, *executor_cores=None*, *num_executors=None*, *archives=None*, *queue=None*, *name=None*, *spark_conf=None*, *heartbeat_timeout=None*, *echo=True*, *check=True*)

Create a new Livy session.

The py_files, files, jars and archives arguments are lists of URLs, e.g. ["s3://bucket/object", "hdfs://path/to/file", ...] and must be reachable by the Spark driver process. If the provided URL has no scheme, it's considered to be relative to the default file system configured in the Livy server.

URLs in the py_files argument are copied to a temporary staging area and inserted into Python's sys.path ahead of the standard library paths. This allows you to import .py, .zip and .egg files in Python.

URLs for jars, py_files, files and archives arguments are all copied to the same working directory on the Spark cluster.

The driver_memory and executor_memory arguments have the same format as JVM memory strings with a size unit suffix ("k", "m", "g" or "t") (e.g. 512m, 2g).

See https://spark.apache.org/docs/latest/configuration.html for more information on Spark configuration properties.

Parameters

- **url** (`str`) – The URL of the Livy server.
- **auth** (`Union[AuthBase, Tuple[str, str], None]`) – A requests-compatible auth object to use when making requests.
- **verify** (`Union[bool, str]`) – Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use. Defaults to `True`.
- **requests_session** (`Optional[Session]`) – A specific `requests.Session` to use, allowing advanced customisation. The caller is responsible for closing the session.
- **kind** (`SessionKind`) – The kind of session to create.
- **proxy_user** (`Optional[str]`) – User to impersonate when starting the session.
- **jars** (`Optional[List[str]]`) – URLs of jars to be used in this session.
- **py_files** (`Optional[List[str]]`) – URLs of Python files to be used in this session.
- **files** (`Optional[List[str]]`) – URLs of files to be used in this session.
- **driver_memory** (`Optional[str]`) – Amount of memory to use for the driver process (e.g. '512m').
- **driver_cores** (`Optional[int]`) – Number of cores to use for the driver process.
- **executor_memory** (`Optional[str]`) – Amount of memory to use per executor process (e.g. '512m').
- **executor_cores** (`Optional[int]`) – Number of cores to use for each executor.
- **num_executors** (`Optional[int]`) – Number of executors to launch for this session.
- **archives** (`Optional[List[str]]`) – URLs of archives to be used in this session.
- **queue** (`Optional[str]`) – The name of the YARN queue to which submitted.
- **name** (`Optional[str]`) – The name of this session.
- **spark_conf** (`Optional[Dict[str, Any]]`) – Spark configuration properties.
- **heartbeat_timeout** (`Optional[int]`) – Optional Timeout in seconds to which session be automatically orphaned if no heartbeat is received.
- **echo** (`bool`) – Whether to echo output printed in the remote session. Defaults to `True`.
- **check** (`bool`) – Whether to raise an exception when a statement in the remote session fails. Defaults to `True`.

Return type *LivySession*

wait()

Wait for the session to be ready.

Return type `None`

property state

The state of the managed Spark session.

Return type SessionState

close()
Kill the managed Spark session.

Return type None

run(*code*)
Run some code in the managed Spark session.

Parameters **code** (str) – The code to run.

Return type Output

read(*dataframe_name*)
Evaluate and retrieve a Spark dataframe in the managed session.

Parameters **dataframe_name** (str) – The name of the Spark dataframe to read.

Return type DataFrame

read_sql(*code*)
Evaluate a Spark SQL satatement and retrieve the result.

Parameters **code** (str) – The Spark SQL statement to evaluate.

Return type DataFrame

5.2 livy.batch

class livy.batch.**LivyBatch**(*url*, *batch_id*, *auth=None*, *verify=True*, *requests_session=None*)
Manages a remote Livy batch and high-level interactions with it.

Parameters

- **url** (str) – The URL of the Livy server.
- **batch_id** (int) – The ID of the Livy batch.
- **auth** (Union[AuthBase, Tuple[str, str], None]) – A requests-compatible auth object to use when making requests.
- **verify** (Union[bool, str]) – Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use. Defaults to True.
- **requests_session** (Optional[Session]) – A specific requests.Session to use, allowing advanced customisation. The caller is responsible for closing the session.

classmethod create(*url*, *file*, *auth=None*, *verify=True*, *requests_session=None*, *class_name=None*, *args=None*, *proxy_user=None*, *jars=None*, *py_files=None*, *files=None*, *driver_memory=None*, *driver_cores=None*, *executor_memory=None*, *executor_cores=None*, *num_executors=None*, *archives=None*, *queue=None*, *name=None*, *spark_conf=None*)
Create a new Livy batch session.

The py_files, files, jars and archives arguments are lists of URLs, e.g. ["s3://bucket/object", "hdfs://path/to/file", ...] and must be reachable by the Spark driver process. If the provided URL has no scheme, it's considered to be relative to the default file system configured in the Livy server.

URLs in the py_files argument are copied to a temporary staging area and inserted into Python's sys.path ahead of the standard library paths. This allows you to import .py, .zip and .egg files in Python.

URLs for jars, py_files, files and archives arguments are all copied to the same working directory on the Spark cluster.

The driver_memory and executor_memory arguments have the same format as JVM memory strings with a size unit suffix ("k", "m", "g" or "t") (e.g. 512m, 2g).

See https://spark.apache.org/docs/latest/configuration.html for more information on Spark configuration properties.

Parameters

- **url** (`str`) – The URL of the Livy server.
- **file** (`str`) – File containing the application to execute.
- **auth** (`Union[AuthBase, Tuple[str, str], None]`) – A requests-compatible auth object to use when making requests.
- **verify** (`Union[bool, str]`) – Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use. Defaults to `True`.
- **requests_session** (`Optional[Session]`) – A specific `requests.Session` to use, allowing advanced customisation. The caller is responsible for closing the session.
- **class_name** (`Optional[str]`) – Application Java/Spark main class.
- **proxy_user** (`Optional[str]`) – User to impersonate when starting the session.
- **jars** (`Optional[List[str]]`) – URLs of jars to be used in this session.
- **py_files** (`Optional[List[str]]`) – URLs of Python files to be used in this session.
- **files** (`Optional[List[str]]`) – URLs of files to be used in this session.
- **driver_memory** (`Optional[str]`) – Amount of memory to use for the driver process (e.g. '512m').
- **driver_cores** (`Optional[int]`) – Number of cores to use for the driver process.
- **executor_memory** (`Optional[str]`) – Amount of memory to use per executor process (e.g. '512m').
- **executor_cores** (`Optional[int]`) – Number of cores to use for each executor.
- **num_executors** (`Optional[int]`) – Number of executors to launch for this session.
- **archives** (`Optional[List[str]]`) – URLs of archives to be used in this session.
- **queue** (`Optional[str]`) – The name of the YARN queue to which submitted.
- **name** (`Optional[str]`) – The name of this session.
- **spark_conf** (`Optional[Dict[str, Any]]`) – Spark configuration properties.

Return type *LivyBatch*

wait()

Wait for the batch session to finish.

Return type `SessionState`

property state

The state of the managed Spark batch.

Return type `SessionState`

log(*from_=None*, *size=None*)
Get logs for this Spark batch.

Parameters

- **from** – The line number to start getting logs from.
- **size** (Optional[int]) – The number of lines of logs to get.

Return type List[str]

kill()
Kill the managed Spark batch session.

Return type None

5.3 livy.client

class livy.client.**LivyClient**(*url*, *auth=None*, *verify=True*, *requests_session=None*)
A client for sending requests to a Livy server.

Parameters

- **url** (str) – The URL of the Livy server.
- **auth** (Union[AuthBase, Tuple[str, str], None]) – A requests-compatible auth object to use when making requests.
- **verify** (Union[bool, str]) – Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use. Defaults to True.
- **requests_session** (Optional[Session]) – A specific requests.Session to use, allowing advanced customisation. The caller is responsible for closing the session.

close()
Close the underlying requests session, if managed by this class.

Return type None

server_version()
Get the version of Livy running on the server.

Return type Version

legacy_server()
Determine if the server is running a legacy version.

Legacy versions support different session kinds than newer versions of Livy.

Return type bool

list_sessions()
List all the active sessions in Livy.

Return type List[Session]

create_session(*kind*, *proxy_user=None*, *jars=None*, *py_files=None*, *files=None*, *driver_memory=None*, *driver_cores=None*, *executor_memory=None*, *executor_cores=None*, *num_executors=None*, *archives=None*, *queue=None*, *name=None*, *spark_conf=None*, *heartbeat_timeout=None*)
Create a new session in Livy.

The py_files, files, jars and archives arguments are lists of URLs, e.g. ["s3://bucket/object", "hdfs://path/to/file", ...] and must be reachable by the Spark driver process. If the provided URL has no scheme, it's considered to be relative to the default file system configured in the Livy server.

URLs in the py_files argument are copied to a temporary staging area and inserted into Python's sys.path ahead of the standard library paths. This allows you to import .py, .zip and .egg files in Python.

URLs for jars, py_files, files and archives arguments are all copied to the same working directory on the Spark cluster.

The driver_memory and executor_memory arguments have the same format as JVM memory strings with a size unit suffix ("k", "m", "g" or "t") (e.g. 512m, 2g).

See https://spark.apache.org/docs/latest/configuration.html for more information on Spark configuration properties.

Parameters

- **kind** (`SessionKind`) – The kind of session to create.
- **proxy_user** (`Optional[str]`) – User to impersonate when starting the session.
- **jars** (`Optional[List[str]]`) – URLs of jars to be used in this session.
- **py_files** (`Optional[List[str]]`) – URLs of Python files to be used in this session.
- **files** (`Optional[List[str]]`) – URLs of files to be used in this session.
- **driver_memory** (`Optional[str]`) – Amount of memory to use for the driver process (e.g. '512m').
- **driver_cores** (`Optional[int]`) – Number of cores to use for the driver process.
- **executor_memory** (`Optional[str]`) – Amount of memory to use per executor process (e.g. '512m').
- **executor_cores** (`Optional[int]`) – Number of cores to use for each executor.
- **num_executors** (`Optional[int]`) – Number of executors to launch for this session.
- **archives** (`Optional[List[str]]`) – URLs of archives to be used in this session.
- **queue** (`Optional[str]`) – The name of the YARN queue to which submitted.
- **name** (`Optional[str]`) – The name of this session.
- **spark_conf** (`Optional[Dict[str, Any]]`) – Spark configuration properties.
- **heartbeat_timeout** (`Optional[int]`) – Optional Timeout in seconds to which session be automatically orphaned if no heartbeat is received.

Return type `Session`

get_session(*session_id*)

Get information about a session.

Parameters **session_id** (`int`) – The ID of the session.

Return type `Optional[Session]`

delete_session(*session_id*)

Kill a session.

Parameters **session_id** (`int`) – The ID of the session.

Return type `None`

list_statements(*session_id*)
Get all the statements in a session.

Parameters **session_id** (int) – The ID of the session.

Return type List[Statement]

create_statement(*session_id*, *code*, *kind=None*)
Run a statement in a session.

Parameters

- **session_id** (int) – The ID of the session.
- **code** (str) – The code to execute.
- **kind** (Optional[StatementKind]) – The kind of code to execute.

Return type Statement

get_statement(*session_id*, *statement_id*)
Get information about a statement in a session.

Parameters

- **session_id** (int) – The ID of the session.
- **statement_id** (int) – The ID of the statement.

Return type Statement

create_batch(*file*, *class_name=None*, *args=None*, *proxy_user=None*, *jars=None*, *py_files=None*, *files=None*, *driver_memory=None*, *driver_cores=None*, *executor_memory=None*, *executor_cores=None*, *num_executors=None*, *archives=None*, *queue=None*, *name=None*, *spark_conf=None*)
Create a new batch in Livy.

The py_files, files, jars and archives arguments are lists of URLs, e.g. ["s3://bucket/object", "hdfs://path/to/file", ...] and must be reachable by the Spark driver process. If the provided URL has no scheme, it's considered to be relative to the default file system configured in the Livy server.

URLs in the py_files argument are copied to a temporary staging area and inserted into Python's sys.path ahead of the standard library paths. This allows you to import .py, .zip and .egg files in Python.

URLs for jars, py_files, files and archives arguments are all copied to the same working directory on the Spark cluster.

The driver_memory and executor_memory arguments have the same format as JVM memory strings with a size unit suffix ("k", "m", "g" or "t") (e.g. 512m, 2g).

See https://spark.apache.org/docs/latest/configuration.html for more information on Spark configuration properties.

Parameters

- **file** (str) – File containing the application to execute.
- **class_name** (Optional[str]) – Application Java/Spark main class.
- **args** (Optional[List[str]]) – An array of strings to be passed to the Spark app.
- **proxy_user** (Optional[str]) – User to impersonate when starting the session.
- **jars** (Optional[List[str]]) – URLs of jars to be used in this session.
- **py_files** (Optional[List[str]]) – URLs of Python files to be used in this session.
- **files** (Optional[List[str]]) – URLs of files to be used in this session.

- **driver_memory** (Optional[str]) – Amount of memory to use for the driver process (e.g. '512m').
- **driver_cores** (Optional[int]) – Number of cores to use for the driver process.
- **executor_memory** (Optional[str]) – Amount of memory to use per executor process (e.g. '512m').
- **executor_cores** (Optional[int]) – Number of cores to use for each executor.
- **num_executors** (Optional[int]) – Number of executors to launch for this session.
- **archives** (Optional[List[str]]) – URLs of archives to be used in this session.
- **queue** (Optional[str]) – The name of the YARN queue to which submitted.
- **name** (Optional[str]) – The name of this session.
- **spark_conf** (Optional[Dict[str, Any]]) – Spark configuration properties.

Return type Batch

delete_batch(*batch_id*)
Kill a batch session.

Parameters **batch_id** (int) – The ID of the session.

Return type None

get_batch(*batch_id*)
Get information about a batch.

Parameters **batch_id** (int) – The ID of the batch.

Return type Optional[Batch]

get_batch_log(*batch_id*, *from_=None*, *size=None*)
Get logs for a batch.

Parameters

- **batch_id** (int) – The ID of the batch.
- **from** – The line number to start getting logs from.
- **size** (Optional[int]) – The number of lines of logs to get.

Return type Optional[BatchLog]

list_batches()
List all the active batches in Livy.

Return type List[Batch]

CHAPTER

SIX

CONTRIBUTING

6.1 Contributing to pylivy

Thanks for considering contributing to `pylivy`!

6.1.1 Asking questions and reporting issues

If you have any questions on using `pylivy` or would like to make a suggestion on improving `pylivy`, please open an issue on GitHub:

https://github.com/acroz/pylivy/issues

6.1.2 Submitting code changes

Before opening a PR, have a look at the information below on code formatting and tests. Tests will be run automatically on Travis and must pass before a PR can be merged.

Code formatting

Code must be formatted with Black (with a line length of 79, as configured in `pyproject.toml`), plus pass Flake8 linting and mypy static type checks.

It's recommend that you configure your editor to autoformat your code with Black and to highlight any Flake8 or mypy errors. This will help you catch them early and avoid disappointment when the tests are run later!

Running tests

`pylivy` includes two types of code tests; unit tests and integration tests. The unit tests test individual classes of the code base, while the integration tests verify the behaviour of the library against an actual running Livy server.

To run the unit tests, which run quickly and do not require a Livy server to be running, first install `tox` (a Python testing tool) if you do not already have it:

```
pip install tox
```

then run:

```
tox -e py37
```

`tox` will build the project into a package, prepare a Python virtual environment with additional test dependencies, and execute the tests. You can also run tests against Python 3.6 by replacing `py37` with `py36` in the above command.

To run integration tests, you need to first start a Livy server to test against. For this purpose, I've prepared a Docker image that runs a basic Livy setup. To run it:

```
docker run --publish 8998:8998 acroz/livy
```

Then, in a separate shell, run the integration tests:

```
tox -e py37-integration
```

Again, you can replace `py37` with `py36` to change the Python version used.

Adding tests

Any new contributions to the library should include appropriate tests, possibly including unit tests, integration tests, or both. Please get in touch by *opening an issue* if you'd like to discuss what makes sense.

Both unit tests and integration tests are written with the pytest testing framework. If you're not familiar with it, I suggest having a look at their extensive documentation and examples first.

PYTHON MODULE INDEX

l

INDEX