HTCondor's Python API – The Python Bindings

Jason Patton
Center for High Throughput Computing

HTCondor Clients in 2013

Command Line Clients

Fully Featured!

Requires fork/exec and process handling

Outputs in multiple formats

Something Missing In The Middle

SOAP Clients

Features! (Some)

Language agnostic (everyone hates XML equally?)

Caveats with respect to scalability, security.

Command Line Clients

Fully Featured!

Requires fork/exec and process handling

Outputs in multiple formats

Something

Middle

SOAP Clients

Language (everyone

ha (?)

Ca with t to

scalability, security.

Command Line Clients

Fully Featured!

Requires fork/exec and process handling

Outputs in multiple formats

RESTful clients, workflow managers, monitoring tools

Features! (Some)

Language agnostic (everyone feels better w/ JSON?)

Caveats with respect to scalability, security.

Design Philosophy

ClassAds: Everything based on ClassAds; make these the “core” of the bindings.

Pythonic: Semantics and APIs should feel natural to a Python programmer.

- Use iterators, exceptions, guards.
- ClassAds behave as much like a dict as reasonable.

Backward compatible: APIs are here to stay for as long as possible.

- When we absolutely must, use standard Python `DeprecationWarning` techniques.

- Yes, this means that we keep even design warts for far longer than we’d like!

Native code: Call same HTCondor library code as CLI; identical in performance.

Complete: If you can do it with the command line tools, you should be able to do it with Python.

Pythonic!

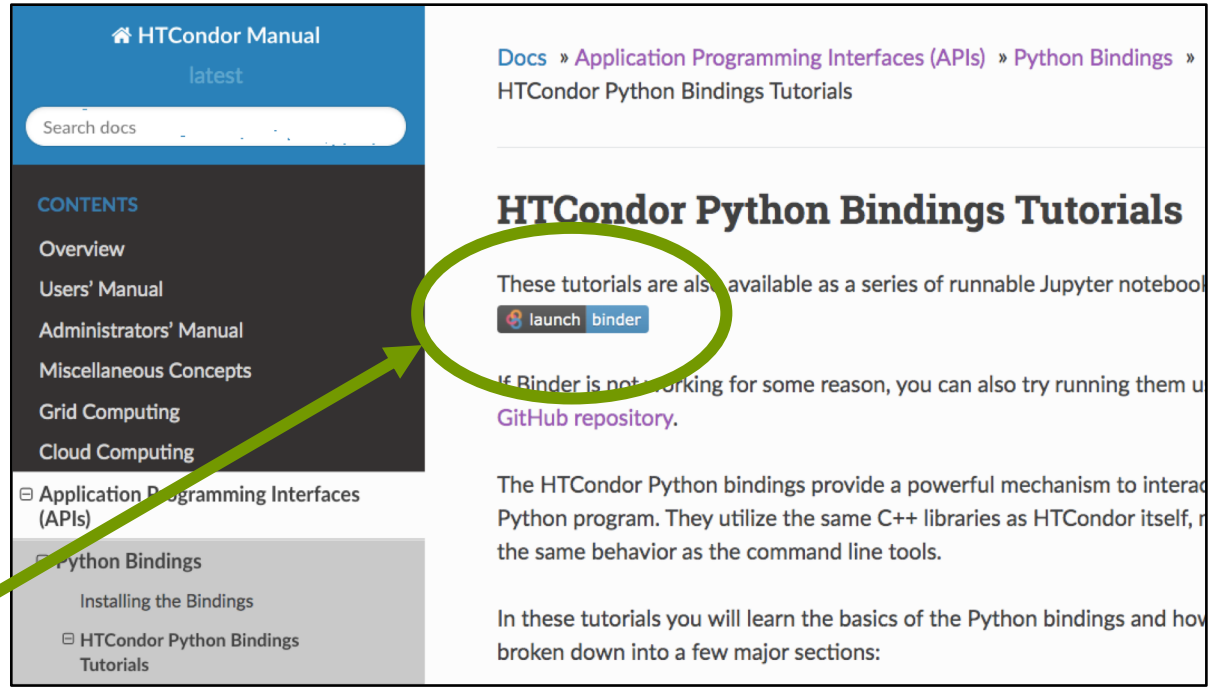
Since *pythonic* is in our design philosophy, the education tools should use the tools favored by the Python community:

- Sphinx-based documentation. Hosted on ReadTheDocs; looks / feels / smells like Python documentation.

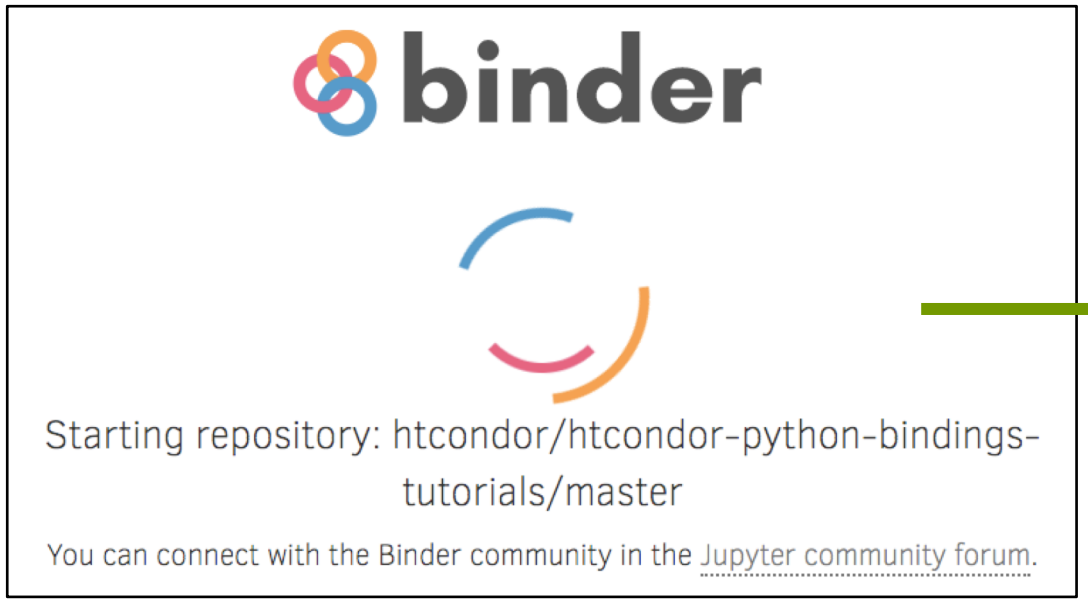
- Jupyter-based tutorials. Use Binder.org service to spawn a Docker container with a private HTCondor instance (or use Docker locally). Interact via your browser.

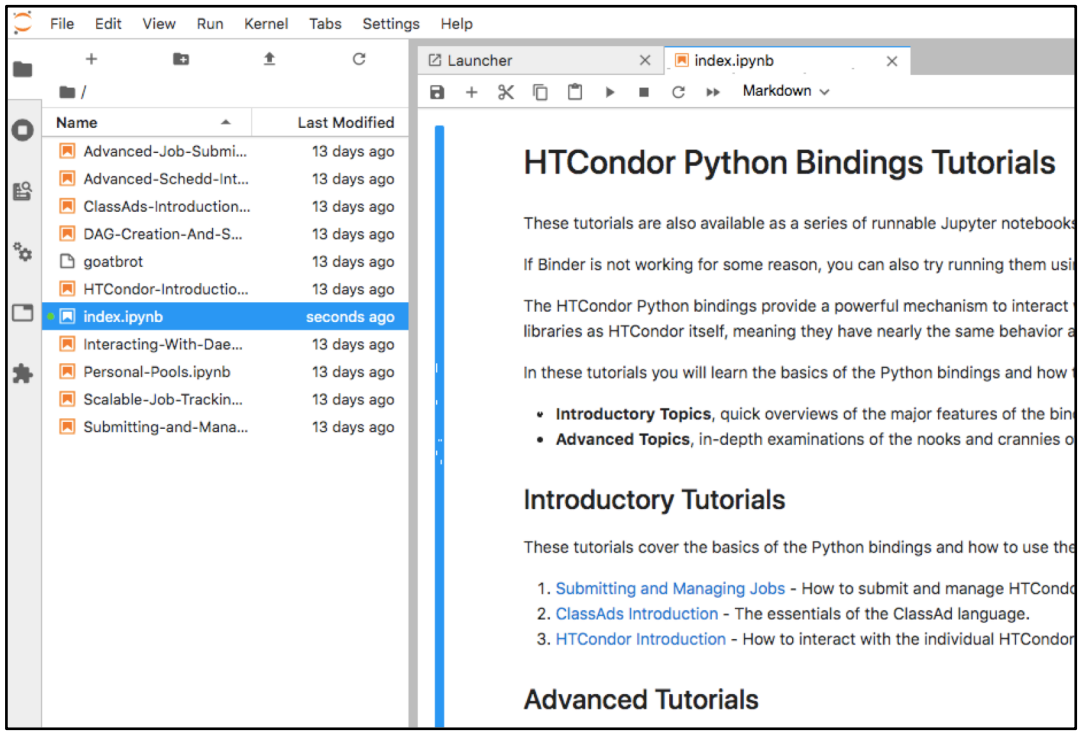
Tutorial

- Head over to the HTCondor documentation https://htcondor.readthedocs.io
- Application Programming Interfaces
  > Python Bindings
  > HTCondor Python Bindings Tutorials
  > Launch Binder



Tutorial


binder
Starting repository: htcondor/htcondor-python-bindings-tutorials/master
You can connect with the Binder community in the Jupyter community forum.


File Edit View Run Kernel Tabs Settings Help
Launcher
index.ipynb
Markdown
Name
Last Modified
Advanced-Job-Submi... 13 days ago
Advanced-Schedd-Int... 13 days ago
ClassAds-Introduction... 13 days ago
DAG-Creation-And-S... 13 days ago
goatbrot 13 days ago
HTCondor-Introductio... 13 days ago
index.ipynb seconds ago
Interacting-With-Dae... 13 days ago
Personal-Pools.ipynb 13 days ago
Scalable-Job-Trackin... 13 days ago
Submitting-and-Mana... 13 days ago
HTCondor Python Bindings Tutorials
These tutorials are also available as a series of runnable Jupyter notebooks
If Binder is not working for some reason, you can also try running them usi
The HTCondor Python bindings provide a powerful mechanism to interact
libraries as HTCondor itself, meaning they have nearly the same behavior a
In these tutorials you will learn the basics of the Python bindings and how
Introductory Topics, quick overviews of the major features of the bin
Advanced Topics, in-depth examinations of the nooks and crannies o
Introductory Tutorials
These tutorials cover the basics of the Python bindings and how to use the
1. Submitting and Managing Jobs - How to submit and manage HTCondo
2. ClassAds Introduction - The essentials of the ClassAd language.
3. HTCondor Introduction - How to interact with the individual HTCondor
Advanced Tutorials

You can help!

The contents of the tutorials and documentation are kept on GitHub:

- https://github.com/htcondor/htcondor-python-bindings-tutorials
- JupyterLab & Binder integration developed by Josh Karpel.
- Find a bug? Spot some missing content?
  - Send a pull request; Travis-CI will test and update the static content once merged.

Installing Python Bindings

- On Linux
  - Included in RPM and DEB packages for the system Python(s)
  - Available in PyPI on Linux:
    - `pip install htcondor`
    - For a specific version: `pip install htcondor==8.8.10`
  - Available in Anaconda (via conda-forge) on Linux:
    - `conda install -c conda-forge python-htcondor`
- On Windows
  - Windows MSI installs bindings for both Python 2.7 and Python 3.6
- On Mac
  - Included in the HTCondor tarball for the system Python

Key Concept #1: ClassAds
The *lingua franca* of HTCondor

Simple Example

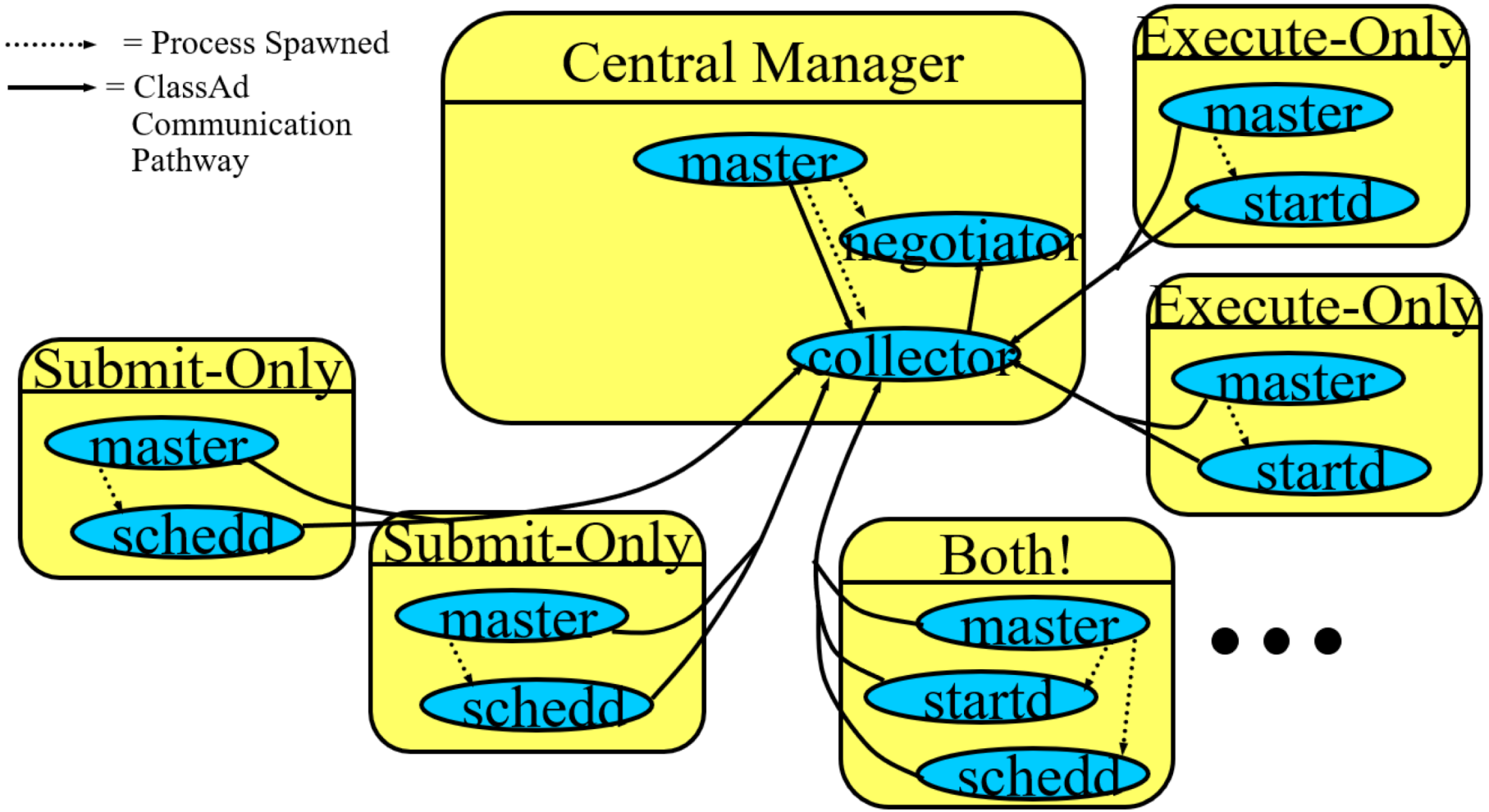
Job Ad

```
Type   = "Job"
Requirements =
    HasMatlabLicense
      == True &&
    Memory >= 1024
Rank = kflops + 1000000
  * Memory
Cmd= "/bin/sleep"
Args = "3600"
Owner = "gthain"
NumJobStarts = 8
KindOfJob = "simulation"
Department = "Math"
```

Machine Ad

```
Type = "Machine"
Cpus   = 40
Memory = 2048
Requirements =
  (Owner == “gthain”)   ||
  (KindOfJob ==
  “simulation”)
Rank = Department == "Math"
HasMatlabLicense = true
MaxTries = 4
kflops = 41403
```

Key Concept #2: HTCondor Daemons!



Two key Concepts, two modules

- `import classad`
  - Provides two classes: `ClassAd` and `ExprTree`
  - The ClassAd Python class behaves a lot like a Python dictionary (key-value pairs, though the values may be `ExprTrees` that refer to other keys!)
- `import htcondor`
  - Provides explicit classes for most daemon types (e.g. Schedd, Collector, Startd) and methods to perform operations against them
  - Provides a Submit class for condor_submit type functionality
  - Lots of other stuff (job event reader, config reader)

Daemon Classes

- `import htcondor`
  - `htcondor.Collector()`
    - `query, locate, advertise : condor_status, advertise`
  - `htcondor.Schedd()`
    - `query, xquery, history : condor_q, history`
    - `act, edit : condor_rm, hold, qedit`
    - `transaction, spool, retrieve : job submission (*)`

* Explicit user-controlled transactions to be deprecated in favor of `htcondor.Schedd.submit()` in the near future.

Submit Class

- `htcondor.Submit()`
  - Takes a submit file-like dict or string and hands back a Submit object containing queuing methods
  - Submit objects can be queued using a transaction via:
    - `submit_obj.queue(txn, n)`
    - `submit_obj.queue_with_itemdata(txn, n, itemdata)`
      - `itemdata` is an iterable containing dicts
      - Keys in the dicts represent variables in the submit object
      - `itemdata = iter([{“foo”: “bar”}, {“foo”: “baz”}])`
        `queue_with_itemdata(2, itemdata) # == “queue 2 foo in bar, baz"`

TIME FOR TUTORIAL DEMO?

Python API – Big Changes

- New features
  - Security Management (i.e. `htcondor.Credd` bindings)
  - `htcondor.Submit.from_dag()`
    - Creates a Submit object from a DAG file (must still be `queued`)
  - `htcondor.dags`
    - Creates DAG files using child-parent objects
  - `htcondor.htchirp`
    - condor_chirp functionality without having to leave Python
    - Also installable separately via `pip` or `conda-forge` (`htchirp` package)
  - (See more in the release notes)

Python API – Big Changes

- Deprecations
  - All the previous exceptions; new exceptions inherit from `htcondor.HTCondorException` or `classad.ClassAdException`
  - Non-standard argument names for many functions (e.g. `requirements`, `attr_list`)
  - Many classes and functions related to reading events/logs that are not `htcondor.JobEventLog`
  - User-controllable Schedd transactions not deprecated yet but will be soon, check the release notes and docs closely!
  - (See more in the release notes)

Questions?

- Python bindings API developers and users alike are active on the HTCondor Users mailing list
- (I will not be responsive to emails soon and for the next few weeks after… 👶)