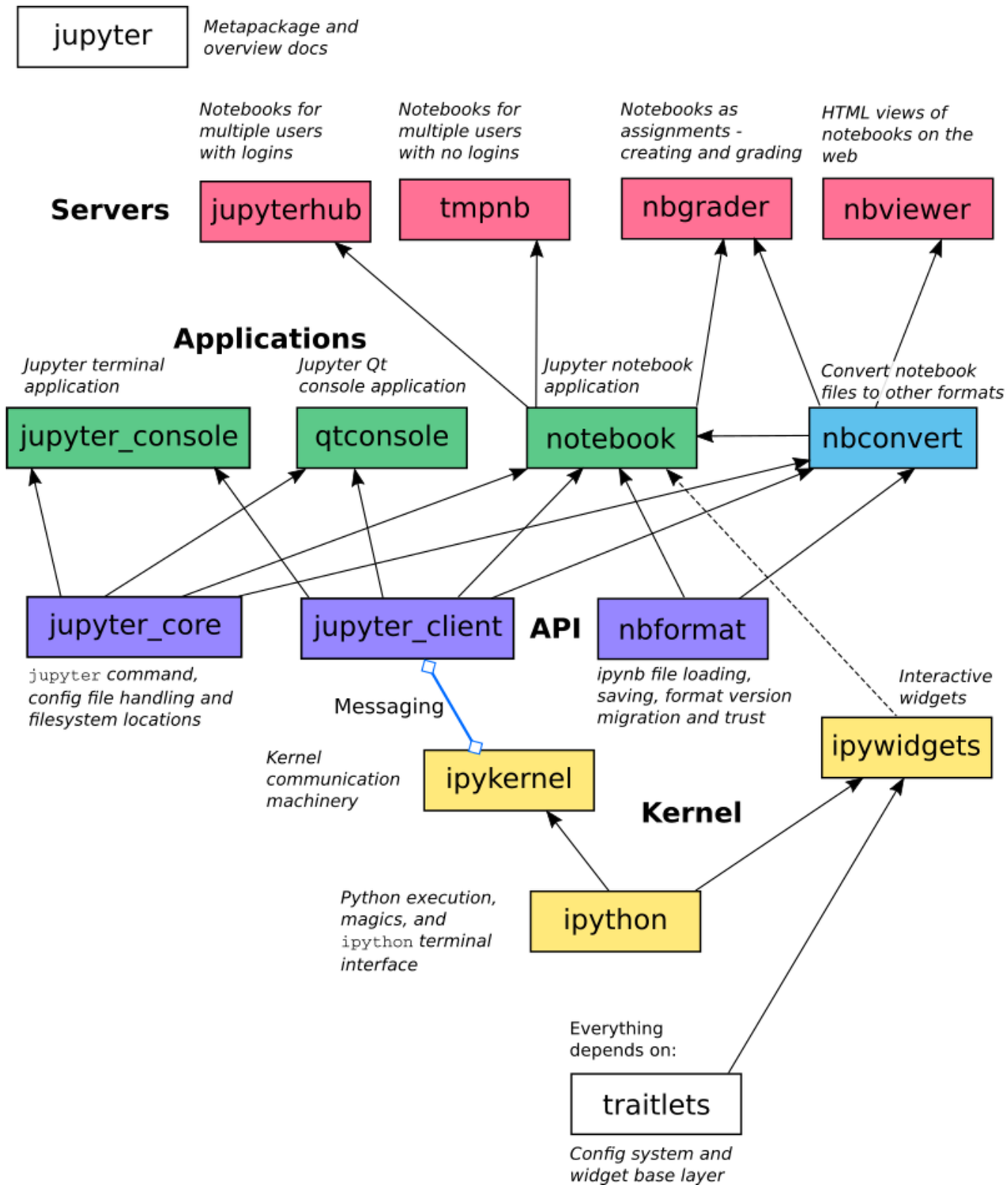
Jupyter Ecosystem Overview

Jupyter is an overloaded word

- Jupyter notebook format
- Jupyter core command interpreter
- Jupyter subcommands and extensions
  - notebook
  - lab
  - nbconvert
  - …
- Jupyter subsystem components
- Jupyter protocol
- Systems which use Jupyter protocol and components

Subset of the Jupyter ecosystem



How to get Jupyter

- conda install jupyter

- http://tripython.org/Members/cbc/conda.pdf

Jupyter notebook format (nbformat)

- Formerly known as iPython notebooks
- A JSON file format containing
    - Text
    - Source code
    - Rich media output
    - Metadata
- Extensible
- https://nbformat.readthedocs.io/

Jupyter notebook format (nbformat)

- metadata dictionary
- nbformat major version integer (currently 4)
- nbformat minor version integer (currently 0 to 4)
- list of cells

Jupyter notebook format (nbformat)

```
{
  "metadata" : {
    "kernel_info": {
        # if kernel_info is defined, its name field is required.
        "name" : "the name of the kernel"
    },
    "language_info": {
        # if language_info is defined, its name field is required.
        "name" : "the programming language of the kernel",
        "version": "the version of the language",
        "codemirror_mode": "The name of the codemirror mode to use [optional]"
    }
  },
  "nbformat": 4,
  "nbformat_minor": 0,
  "cells" : [
      # list of cell dictionaries
  ],
}
```

Cell dictionary

```
{
  "cell_type" : "type",
  "metadata" : {},
  "source" : "single string or [list, of, strings]",
}
```

Cell type: markdown

```
{
  "cell_type" : "markdown",
  "metadata" : {},
  "source" : "[multi-line *markdown*]",
}
```

Cell type: code

```
{
  "cell_type" : "code",
  "execution_count": 1, # integer or null
  "metadata" : {
      "collapsed" : True, # whether the output of the cell is collapsed
      "scrolled": False, # any of true, false or "auto"
  },
  "source" : "[some multi-line code]",
  "outputs": [{
      # list of output dicts
      "output_type": "stream",
      ...
  }],
}
```

Code cell output types

- stream
- data_display (rich media mime bundles)
- execute_result
- error

Cell type: raw

- For use by extensions like nbconvert

```
{
  "cell_type" : "raw",
  "metadata" : {
    # the mime-type of the target nbconvert format.
    # nbconvert to formats other than this will exclude this cell.
    "format" : "mime/type"
  },
  "source" : "[some nbformat output text]"
}
```

Notebook metadata

Key	Value	Interpretation
kernelspec	dict	A kernel specification*
authors	list of dicts	A list of dictionaries of authors of the document

- https://jupyter-client.readthedocs.io/en/stable/kernels.html#kernelspecs
- `jupyter kernelspec` subcommand for managing kernels
- `name` is the only required key for an author dictionary
- `nb.metadata` is a shared namespace for notebook metadata keys
- Custom notebook metadata should define its own namespace for keys

Cell metadata

Key	Value	Interpretation
collapsed	bool	Whether the cell's output container should be collapsed
scrolled	bool or 'auto'	Whether the cell's output is scrolled, unscrolled, or autoscrolled
deletable	bool	If False, prevent deletion of the cell
format	mime/type	The mime-type of a Raw NBConvert Cell
name	str	A name for the cell. Should be unique across the notebook. Uniqueness must be verified outside of the json schema.
tags	list of str	A list of string tags on the cell. Commas are not allowed in a tag

- `metadata.jupyter` is the namespace for cell metadata keys

Python API for Jupyter notebook files

- `import nbformat`
- functions for reading and writing notebook files
- class with which to create instances of `nbformat.NotebookNode`, a dictionary-like object
- functions to create NotebookNodes of particular types (notebook, code cells, markdown cells, raw cells, outputs, etc.)
- classes to manage notebook signatures

Jupyter_core

- contains the `jupyter` script to run the jupyter subcommand interpreter

```
>>> import jupyter_core.command as jcc

>>> jcc.list_subcommands()

['bundlerextension', 'console', 'dashboards-server', 'kernel', 'kernelspec', 'lab',
'labextension', 'labhub', 'migrate', 'nbconvert', 'nbextension', 'notebook',
'qtconsole', 'run', 'serverextension', 'troubleshoot', ‘trust']
```

- jupyter extensions are managed with the `nbextension` and `serverextension` subcommands
- installed an extension may add more jupyter subcommands
- extensions may extend the notebook format
- well-formed conda packages will install and enable their own extensions

Jupyter_core

- various conda packages will install scripts in your `PATH` of the form `jupyter-something`

- note: the conda package will most likely named something similar but not the same

- this will allow you to execute `jupyter something`

```
pylantic:~ cbc$ jupyter-
jupyter-bundlerextension  jupyter-kernelspec   jupyter-migrate     jupyter-qtconsole jupyter-trust
jupyter-console           jupyter-lab          jupyter-nbconvert   jupyter-run
jupyter-dashboards-server jupyter-labextension jupyter-nbextension jupyter-serverextension
jupyter-kernel            jupyter-labhub       jupyter-notebook    jupyter-troubleshoot
pylantic:~ cbc$ jupyter-
```

Jupyter_core

- some extensions have neither subcommands nor scripts:

```
pylantic:~ cbc$ jupyter nbextension list
Known nbextensions:
  config dir: /Users/cbc/.jupyter/nbconfig
    notebook section
      jupyter-matplotlib/extension  enabled
      - Validating: OK
      jupyter-vega/index  enabled
      - Validating: OK
  config dir: /Users/cbc/anaconda3/etc/jupyter/nbconfig
    notebook section
      nbpresent/js/nbpresent.min  enabled
      - Validating: OK
      nb_conda/main  enabled
      - Validating: OK
    tree section
      nb_conda/tree  enabled
      - Validating: OK
pylantic:~ cbc$
```

Jupyter_core

- some packages install both nb and server extensions:

```
pylantic:~ cbc$ jupyter serverextension list
config dir: /Users/cbc/anaconda3/etc/jupyter
    nb_conda  enabled
    - Validating...
      nb_conda  OK
    nbpresent  enabled
    - Validating...
      nbpresent  OK
    nb_anacondacloud  enabled
    - Validating...
      nb_anacondacloud  OK
    jupyterlab  enabled
    - Validating...
      jupyterlab  OK
pylantic:~ cbc$
```

Jupyter_core subcommands

- the most common Jupyter subcommands are:
    - `notebook` (start local server with classic UI)
    - `lab` (start local server with contemporary UI)
- lab subcommand installed by the `jupyterlab` conda pacakge

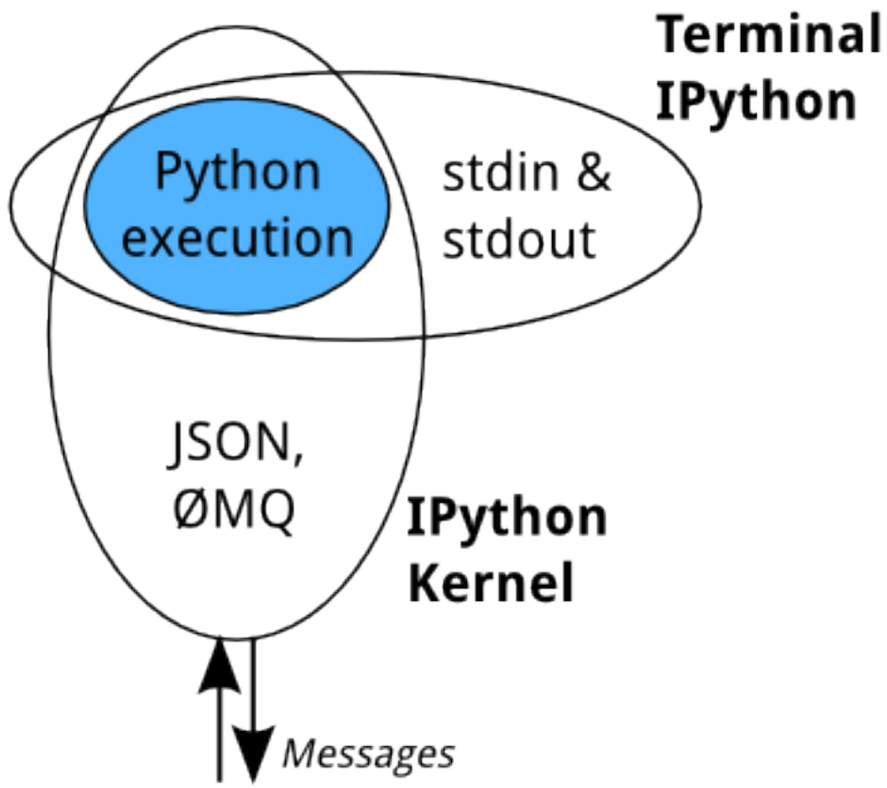
Useful Jupyter extension packages

- nbpresent: turn notebooks into reveal.js presentations
- nbconvert: turn notebooks into PDFs, HTML, etc.
- nb_conda: make all your conda environments available as kernels to Jupyter
- ipywidgets: code-defined notebook-embedded interactive widgets

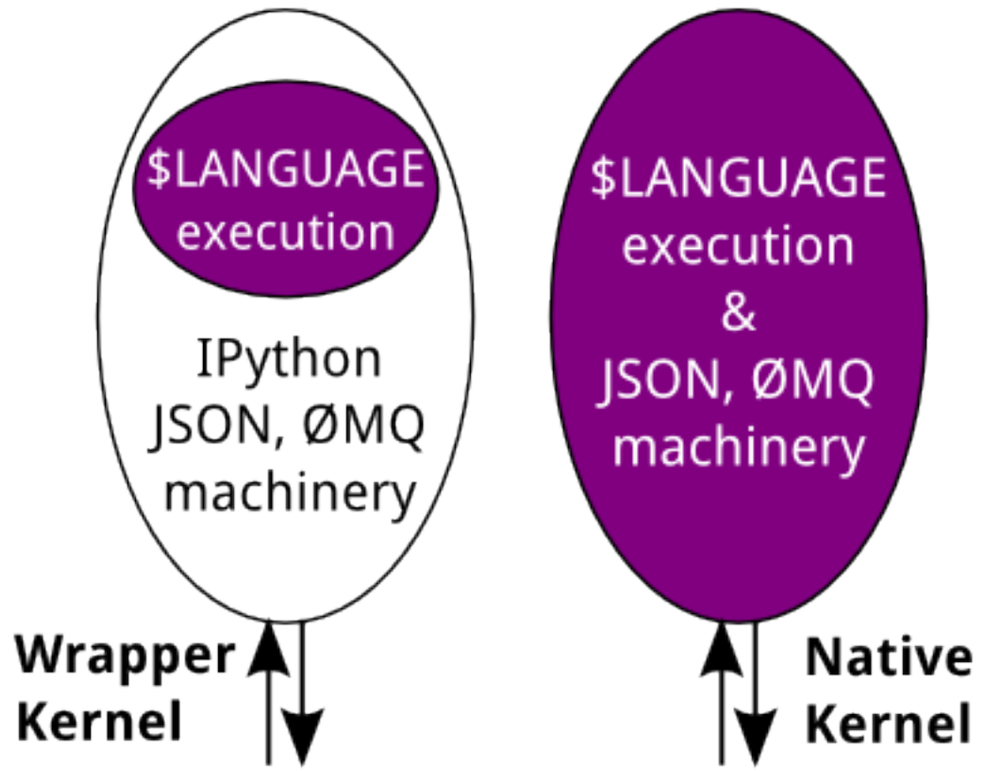
Jupyter subsystem components

- iPython kernel (`ipykernel`)
  - other kernels and wrappers (`ipyparallel`, `ipywidgets`, etc.)
- notebook server
  - other user interfaces (`lab`, `console`, `qtconsole`, etc.)
- kernel gateway
- messaging protocol (`jupyter_client`)

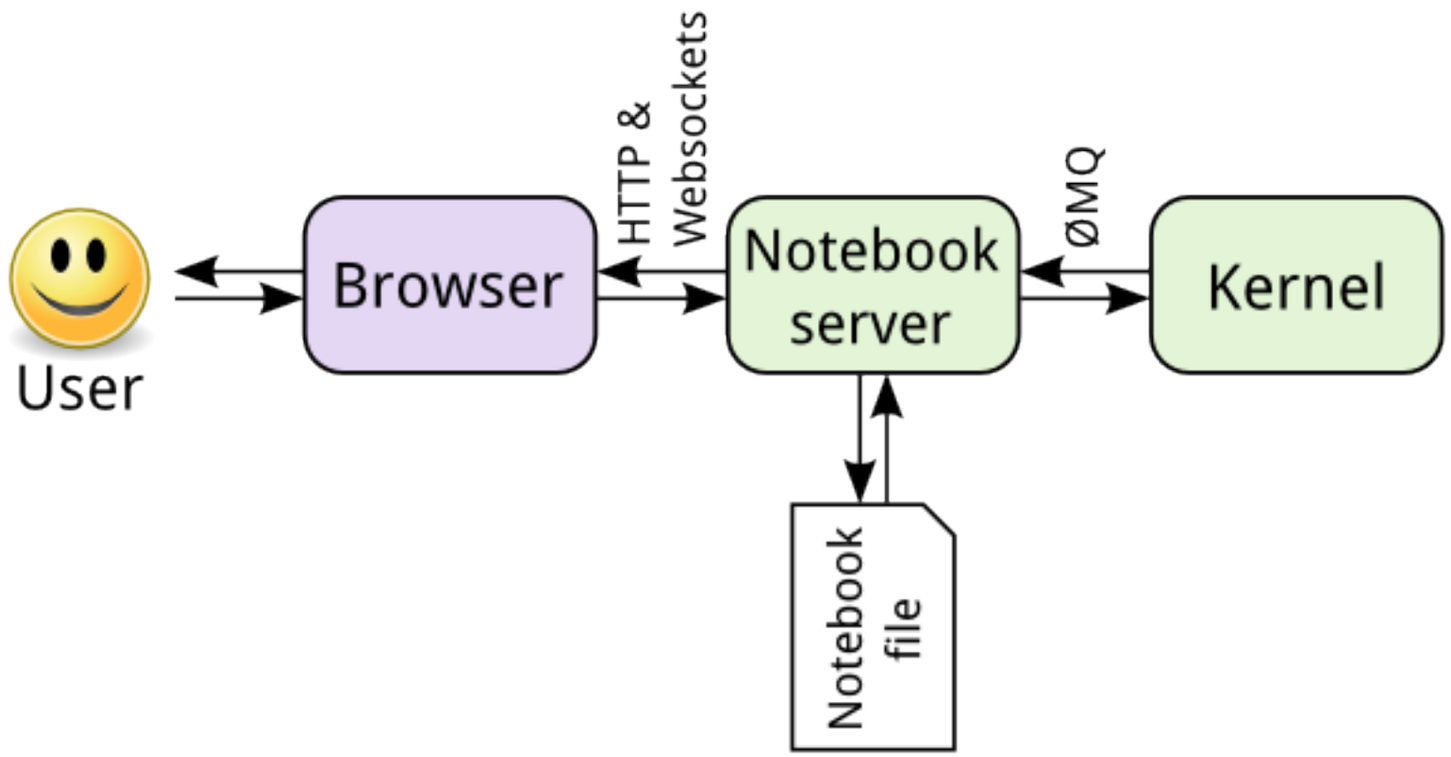
iPython kernel



iPython kernel

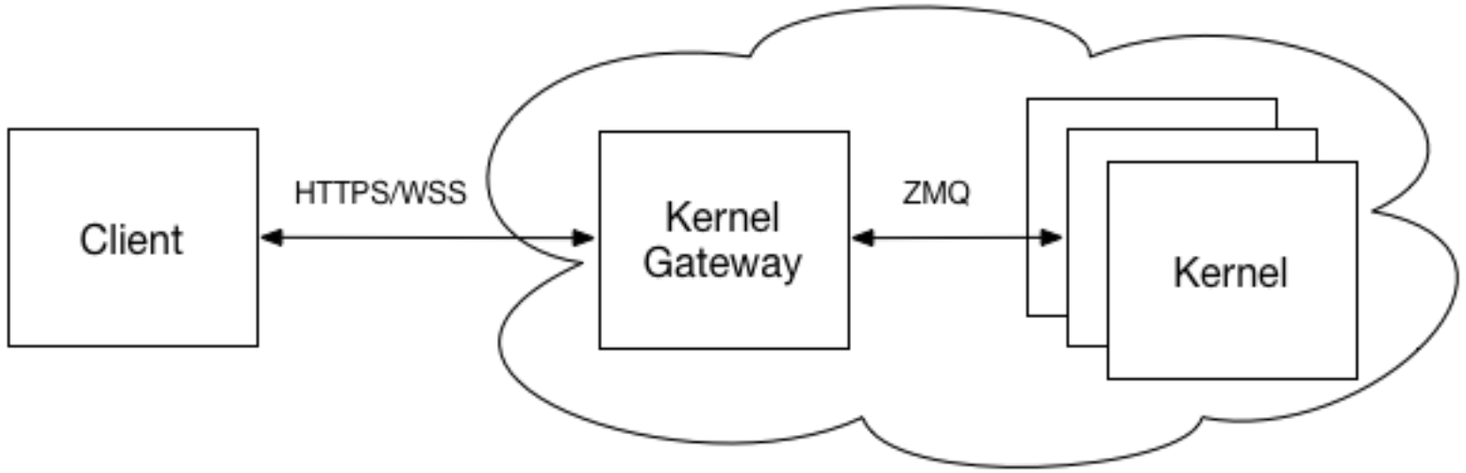


Notebook server



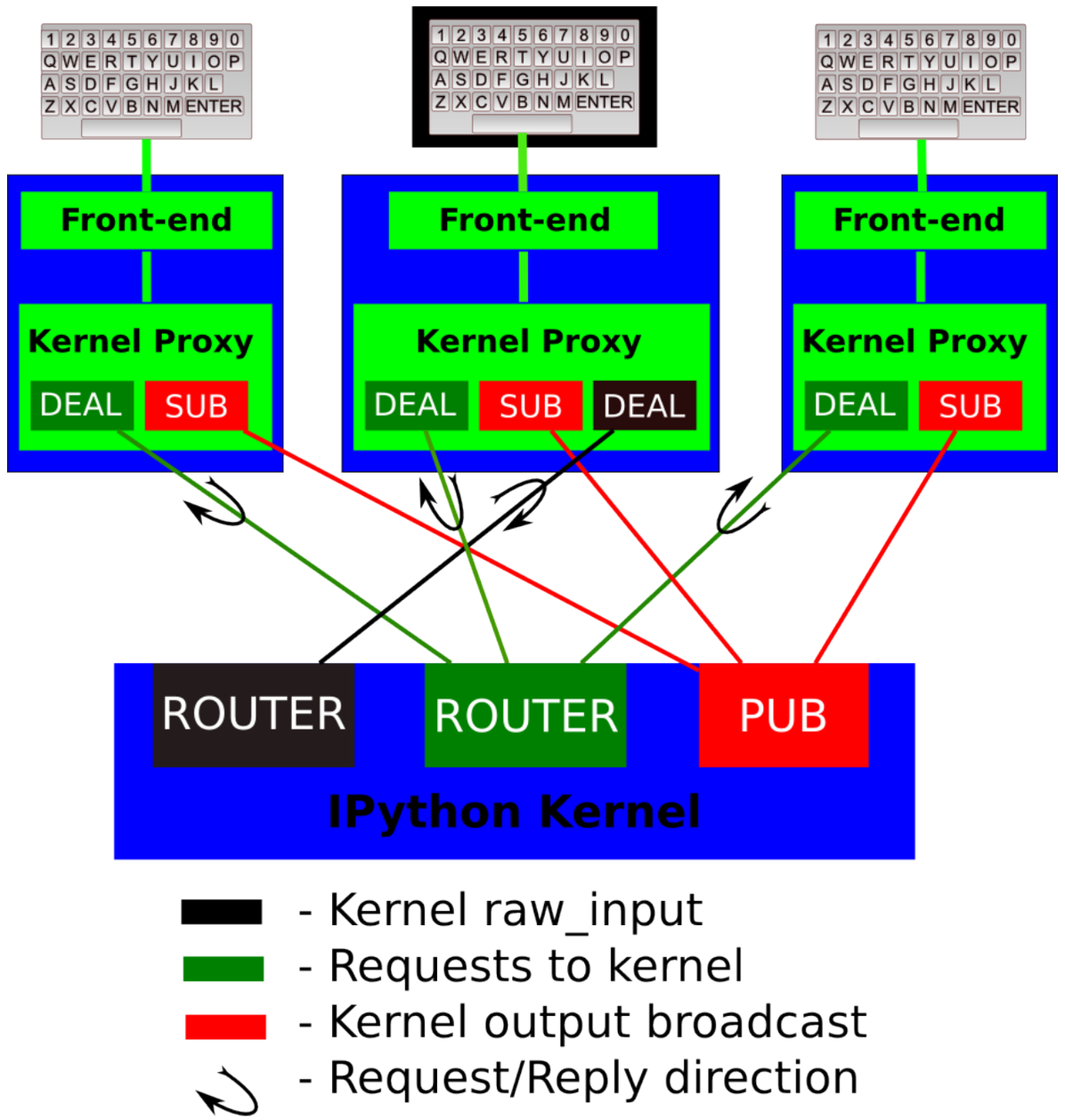
Jupyter kernel gateway

- factorization of reusable notebook server services



Jupyter protocol (jupyter_client)

- another factorization of reusable notebook server services connecting multiple front-ends with one or more kernels



jupyter_client sockets

- a kernel has dedicated sockets for messages with the client:
  - shell - code to execute
  - IOPub - stdout, stderr, etc.
  - stdin - to receive `raw_input`
  - control - identical to shell but for priority requests (shutdown, etc.)
  - heartbeat

jupyter_client high level message format

- `content` and `metadata` are dependent on the msg_type
- `buffers` are used by extensions to the protocol

```
{
 'header' : {
                'msg_id' : str, # typically UUID
                'username' : str,
                'session' : str, # typically UUID
                'date': str, # ISO 8601 timestamp
                'msg_type' : str,
                'version' : '5.0',
    },
 'parent_header' : dict,
 'metadata' : dict,
 'content' : dict,
 'buffers': list,
}
```

jupyter_client wire protocol

- the message format is serialized:

```
[
  b'u-u-i-d',          # zmq identity(ies)
  b'<IDS|MSG>',        # delimiter
  b'baddad42',         # HMAC signature
  b'{header}',         # serialized header dict
  b'{parent_header}', # serialized parent header dict
  b'{metadata}',       # serialized metadata dict
  b'{content}',        # serialized content dict
  b'\xf0\x9f\x90\xb1' # extra raw data buffer(s)
  ...
]
```

shell channel message types

- these messages follow a router/dealer request-reply pattern:
    - execute
    - execution count
    - introspection
    - completion
    - history
    - code completeness
    - connect
    - comm info (custom message for things such as updating widget state)
    - kernel info
    - kernel shutdown
    - kernel interrupt

IOPub channel message types

- display data
- update display data
- code input
- execution results
- execution errors
- kernel status (busy, idle, starting)
- clear output

jupyter_client API

- kernelspec - discovering kernels
    - stored in `kernel.json` files
        - args - to start kernel
        - display name
        - language
        - interrupt mode
        - environment variables
        - metadata
- kernelmanager - start, stop, signal kernels
- client - communicate with kernels

jupyter_client API

- kernel manager uses connection files for each kernel


```
{
  "control_port": 50160,
  "shell_port": 57503,
  "transport": "tcp",
  "signature_scheme": "hmac-sha256",
  "stdin_port": 52597,
  "hb_port": 42540,
  "ip": "127.0.0.1",
  "iopub_port": 40885,
  "key": "a0436f6c-1916-498b-8eb9-e81ab9368e84"
}
```

JupyterHub

- pluggable auth
- centralized deployment
- container friendly
- code next to data
- ReST API and services
- https://jupyterhub.readthedocs.io/

JupyterHub

- multi-user hub (tornado)
- configurable http proxy (node.js)
- multiple single user Jupyter notebook servers

JupyterHub-supplied authenticators

- LocalAuthenticator (PAM)

- OAuthenticator
  - Auth0
  - Bitbucket
  - CILogon
  - GitHub
  - GitLab
  - Globus
  - Google
  - MediaWiki
  - Okpy
  - OpenShift

- LDAPAuthenticator

- jhub_shibboleth_auth

- jhub_remote_user_authenticator

JupyterHub spawners

- SudoSpawner - single non-root local user notebook server
- DockerSpawner
  - `dockerspawner.DockerSpawner` - identical Docker containers for each users
  - `dockerspawner.SystemUserSpawner` - Docker containers with an environment and home directory for each user
  - both may user Docker Swarm to spawn on remote machines
- ImageSpawner - allow users to choose which Docker image to spawn
- BatchSpawner
  - TorqueSpawner
  - MoabSpawner
  - SlurmSpawner
  - GridengineSpawner
  - CondorSpawner
  - LsfSpawner
- KubeSpawner - https://zero-to-jupyterhub.readthedocs.io/
- Many others including AWS ECS, Marathon, UCS, Yarn, etc.

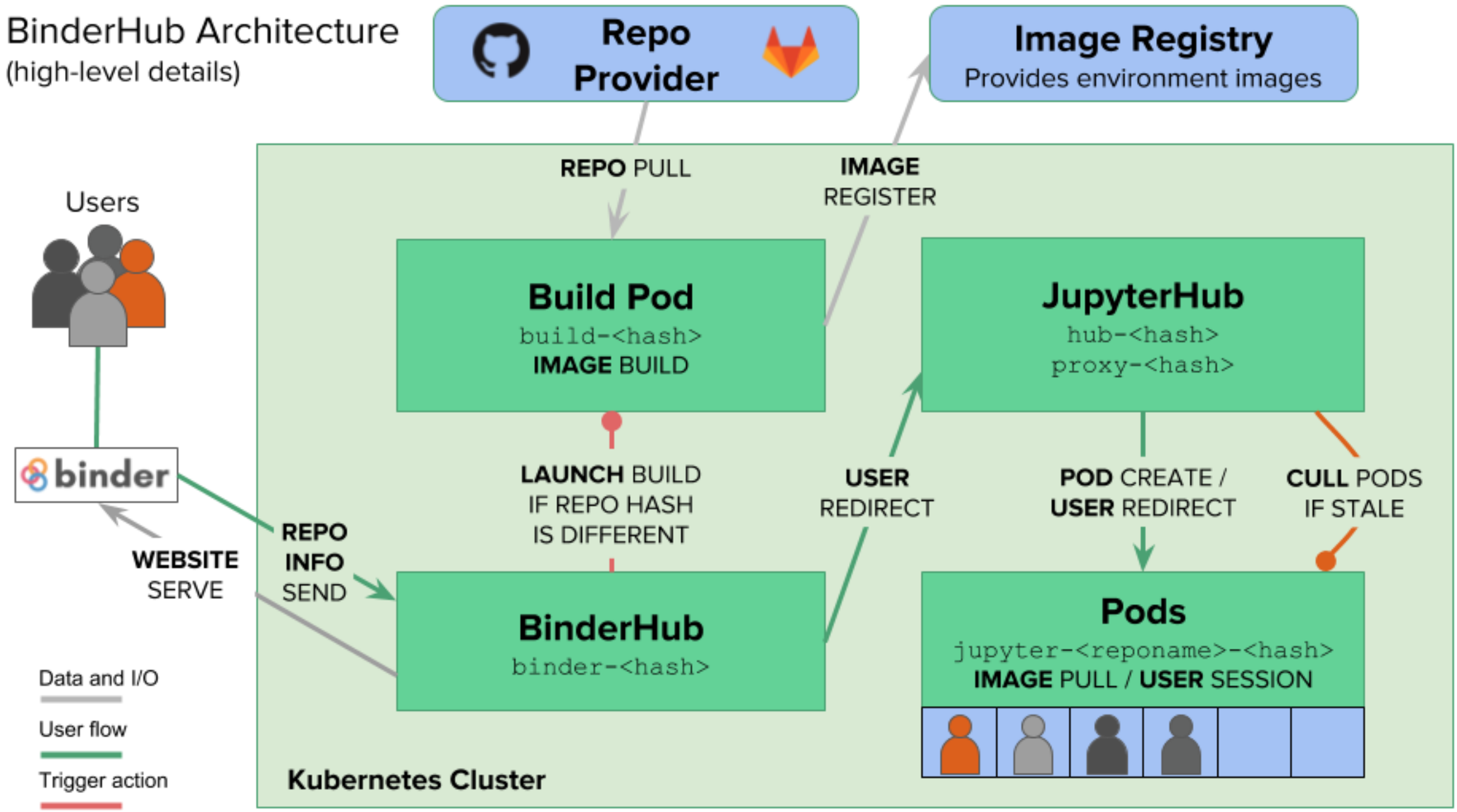
Binder

- custom computing environments (binders) that can be shared and used by many remote users
- powered by BinderHub
- https://mybinder.org/
- hosted by Project Jupyter
- bleeding edge

Binder

- A **cloud provider** such Google Cloud, Microsoft Azure, Amazon EC2, and others
- **Kubernetes** to manage resources on the cloud
- **Helm** to configure and control Kubernetes
- **Docker** to use containers that standardize computing environments
- A **BinderHub UI** that users can access to specify Git repos they want built
- **BinderHub** to generate Docker images using the URL of a Git repository
- A **Docker registry** (such as gcr.io) that hosts container images
- **JupyterHub** to deploy temporary containers for users

Binder


BinderHub Architecture
(high-level details)
Repo Provider
Image Registry
Provides environment images
REPO PULL
IMAGE REGISTER
Users
Build Pod
build-<hash>
IMAGE BUILD
JupyterHub
hub-<hash>
proxy-<hash>
binder
LAUNCH BUILD IF REPO HASH IS DIFFERENT
USER REDIRECT
POD CREATE / USER REDIRECT
CULL PODS IF STALE
WEBSITE SERVE
REPO INFO SEND
BinderHub
binder-<hash>
Pods
jupyter-<reponame>-<hash>
IMAGE PULL / USER SESSION
Data and I/O
User flow
Trigger action
Kubernetes Cluster

nbviewer

- predecessor to Binder
- Static rendering of notebooks stored on GitHub
- based on nbconvert
- https://nbviewer.jupyter.org/
- Hosted by Rackspace

Jupyter dashboards

- Jupyter Dashboard
    - add drag and drop layout editor to classic notebook server
    - dashboard bundler - export to node.js bundle
    - dashboard server - server dashboard bundle
        - heavy use of kernelgateway
        - example: http://dashboards.renci.org:3000/
        - DEPRECATED!
- voila - https://github.com/QuantStack/voila
    - sprint in Paris June 3-6: https://blog.jupyter.org/jupyter-community-workshop-dashboarding-with-project-jupyter-b0e421bdf164
- app mode - https://github.com/oschuett/appmode