Getting Started with JupyterHub Tutorial Documentation

Release 1.0

Project Jupyter

Feb 01, 2018

Contents

This tutorial is based on Min Ragan Kelley's PyData London talk.

CHAPTER 1

JupyterHub References

1.1 JupyterHub Cheatsheet

1.1.1 JupyterHub on GitHub

JupyterHub organization on GitHub https://github.com/jupyterhub

Project

JupyterHub

- GitHub
- Documentation online
- PDF download

Tutorial

JupyterHub Tutorial - "Tutorial materials for deploying JupyterHub"

- GitHub
- Documentation online

Authenticators

LDAPauthenticator - "Simple LDAP Authenticator Plugin for JupyterHub" https://github.com/jupyterhub/ldapauthenticator

OAuthenticator - "GitHub OAuth + JupyterHub Authenticator = OAuthenticator" https://github.com/jupyterhub/oauthenticator

Spawners

Batchspawner - "Custom Spawner for Jupyterhub to start servers in batch scheduled systems" https://github.com/jupyterhub/batchspawner

Dockerspawner - "Enables JupyterHub to spawn user servers in docker containers." https://github.com/jupyterhub/dockerspawner

Sudospawner - "enables JupyterHub to spawn single-user servers without being root, by spawning an intermediate process via sudo, which takes actions on behalf of the user" https://github.com/jupyterhub/sudospawner

Proxy

Configurable HTTP Proxy - "node-http-proxy plus a REST API"

Deployment examples

JupyterHub deployment using Ansible

JupyterHub deployment with Docker

JupyterHub using Rackspace Carina

1.2 Timeline of tutorial video

PyData London 2016 YouTube Video

1.2.1 Introduction to Jupyter notebooks and JupyterHub

0:00:00 Welcome and Intro

0:01:00 GitHub repo that accompanies the talk

0:01:44 What is the Notebook?

0:03:00 What is a Notebook Server?

1.2.2 Overview of JupyterHub

0:04:17 JupyterHub

0:05:41 Login

0:05:55 Spawner

0:06:27 Proxy

0:07:11 Redirect user

0:07:17 Browser to ask hub for auth

[Additional reading for overview] (http://jupyterhub.readthedocs.io/en/latest/getting-started.html#overview)

1.2.3 Installation of JupyterHub

0:07:56 Installation (as admin)

0:10:44 Installation (this repo)

0:11:18 Installation: Caveats

0:12:18 conda-forge

- community maintained conda packages
- Add conda-forge to default conda sources

0:13:38 Installation docker (covered later on)

Installation instructions can be found under Prerequisites and Installation

1.2.4 Configuring JupyterHub

0:14:18 JupyterHub Defaults

- Default behavior
- Auth: PAM
- Spawning: Local users
- Hub run as root (alternative: sudospawner is fraught with peril)

0:15:12 Type jupyterhub in terminal

- message returned that the hub will not start since there is no SSL provisioned
- If you want to run without SSL, do so at your own risk.

0:16:26 SSL

- Use a self signed cert
- Let's Encrypt 0:17:30

0:18:12 Configure jupyterhub

- create file
- edit config file 0:19:12

0:20:33 Connect to hub publicly

- login, spawn server, redirect
- control panel 0:21:20

1.2.5 Authenticators

0:21:52 Installing kernels for all users

0:24:15 Using GitHub OAuth

- We have simple PAM; tell server to use GitHub OAuth
- Authorization callback URL
- Client ID

- Client Secret
- ./env -> export the variables

0:30:44 Tell Jupyter to use oauthenticator

0:32:48 Sign in with GitHub

0:34:20 Specifying users

- PAM ok
- GitHub probably not ok
- user whitelist - put in a python set in config file
- admin users - put in a python set in config file

0:36:26 Jupyterhub Custom Authenticators

- PAM - form based fairly simple
- Secure Authenticator
- jupyterhub hashing salted functions

1.2.6 Spawning Processes

0:42:14 Using DockerSpawner

- netifaces - python convenience library
- local GH to general GH users
- GH - DockerSpawner and whitelist

0:44:00 Initially missing piece Hub API if cookie is valid

- docker0 ip address netifaces

0:48:00 Lots you can do with DockerSpawner

0:51:19 Customizing JupyterHub Spawners

- Start my server goes to a form

1:07:00 JupyterHub with supervisor

1:09:00 Reference deployments

1:11:00 Q & A

1:13:00 Simula deployment with persistence in Hub

CHAPTER 2

Tutorial notebooks

2.1 Getting Started with JupyterHub Tutorial

2.1.1 Resources

- JupyterHub Documentation
- the PDF of PyData London 2016 slidedeck
- the video on YouTube of PyData London 2016 tutorial
- *Timeline of video*

Introduction

0:00:00 Welcome and Intro

0:01:00 GitHub repo that accompanies the talk

Review of Notebook and Notebook Server

0:01:44 What is the Notebook?

0:03:00 What is a Notebook Server?

Overview of JupyterHub

0:04:17 JupyterHub

0:05:41 Login

0:05:55 Spawner

0:06:27 Proxy

0:07:11 Redirect user

0:07:17 Browser to ask hub for auth

Installation

0:07:56 Installation (as admin)

0:10:44 Installation (this repo)

0:11:18 Installation: Caveats

0:12:18 conda-forge

- community maintained conda packages
- Add conda-forge to default conda sources

0:13:38 Installation docker (covered later on)

JupyterHub Defaults

0:14:18 JupyterHub Defaults

- Default behavior
- Auth: PAM
- Spawning: Local users
- Hub run as root (alternative: sudospawner is fraught with peril)

Security and SSL

0:15:12 Type jupyterhub in terminal

- message returned that the hub will not start since there is no SSL provisioned
- If you want to run without SSL, do so at your own risk.

0:16:26 SSL

- Use a self signed cert
- Let's Encrypt 0:17:30

Configuration of Hub

0:18:12 Configure jupyterhub

- create file
- edit config file 0:19:12

0:20:33 Connect to hub publicly

- login, spawn server, redirect
- control panel 0:21:20

Kernels and Programming Languages

0:21:52 Installing kernels for all users

Authentication

0:24:15 Using GitHub OAuth

- We have simple PAM; tell server to use GitHub OAuth
- Authorization callback URL
- Client ID
- Client Secret
- ./env -> export the variables

0:30:44 Tell Jupyter to use oauthenticator

0:32:48 Sign in with GitHub

0:34:20 Specifying users

- PAM ok
- GitHub probably not ok
- user whitelist - put in a python set in config file
- admin users - put in a python set in config file

Custom Authenticators

0:36:26 JupyterHub Custom Authenticators

- PAM - form based fairly simple
- Secure Authenticator
- jupyterhub hashing salted functions

Spawners - DockerSpawner

0:42:14 Using DockerSpawner

- netifaces - python convenience library
- local GH to general GH users
- GH - DockerSpawner and whitelist

0:44:00 Initially missing piece Hub API if cookie is valid

- docker0 ip address netifaces

0:48:00 Lots you can do with DockerSpawner

Custom Spawners

0:51:19 Customizing JupyterHub Spawners

- Start my server goes to a form

Deployment

1:07:00 JupyterHub with supervisor

1:09:00 Reference deployments

Wrap Up

1:11:00 Q & A

1:13:00 Simula deployment with persistence in Hub

2.2 Custom Authenticators

Let's peek at the Authenticator classes:

```
In [1]: from jupyterhub.auth import Authenticator, PAMAuthenticator
```

```
In [2]: Authenticator.authenticate?
```

```
[0;31mSignature:[0m Authenticator.authenticate(self, handler, data)
[0;31mDocstring:[0m
Authenticate a user with login form data

This must be a tornado gen.coroutine.
It must return the username on successful authentication,
and return None on failed authentication.

Checking the whitelist is handled separately by the caller.

Args:
    handler (tornado.web.RequestHandler): the current request handler
    data (dict): The formdata of the login form.
                 The default form has 'username' and 'password' fields.
Returns:
    username (str or None): The username of the authenticated user,
    or None if Authentication failed
[0;31mFile:[0m      ~/dev/jpy/jupyterhub/jupyterhub/auth.py
[0;31mType:[0m      function
```

PAM calls out to a library with the given username and password:

```
In [3]: PAMAuthenticator.authenticate??
```

```
[0;31mSignature:[0m PAMAuthenticator.authenticate(self, handler, data)
[0;31mSource:[0m
    @gen.coroutine
    def authenticate(self, handler, data):
        """Authenticate with PAM, and return the username if login is successful.

        Return None otherwise.
```

```
        """
        username = data['username']
        try:
            pamela.authenticate(username, data['password'], service=self.service)
        except pamela.PAMError as e:
            if handler is not None:
                self.log.warning("PAM Authentication failed (%s@%s): %s", username, handler.request.r
            else:
                self.log.warning("PAM Authentication failed: %s", e)
        else:
            return username
[0;31mFile:[0m      ~/dev/jpy/jupyterhub/jupyterhub/auth.py
[0;31mType:[0m      function
```

Here's a super advanced Authenticator that does very secure password verification:

```
In [4]: class SuperSecureAuthenticator(Authenticator):
            def authenticate(self, handler, data):
                username = data['username']
                # check password:
                if data['username'] == data['password']:
                    return username
```

2.2.1 Exercise:

Write a custom username+password Authenticator where:

1. passwords are loaded from a dict
2. hashed+salted passwords are stored somewhere, but not cleartext passwords

```
In [5]: # possibly useful:

        from jupyterhub.utils import hash_token, compare_token
        hash_token('mypassword')
```

```
Out[5]: 'sha512:16384:98400e241da5a64d:6e2c468dea2e6ec6f185936f6ac1a96e6046c4cdf6c0156aecb03fcd2e9963
```

```
In [6]: compare_token(_, 'mypassword')
```

```
Out[6]: True
```

2.3 JupyterHub Spawners

Let's peek at the base classes:

```
In [1]: from jupyterhub.spawner import Spawner, LocalProcessSpawner
```

```
In [2]: Spawner??
```

```
Init signature: Spawner(self, **kwargs)
Source:
class Spawner(LoggingConfigurable):
    """Base class for spawning single-user notebook servers.

    Subclass this, and override the following methods:

    - load_state
    - get_state
```

```
- start
- stop
- poll
"""

db = Any()
user = Any()
hub = Any()
authenticator = Any()
api_token = Unicode()
ip = Unicode('127.0.0.1',
    help="The IP address (or hostname) the single-user server should listen on"
).tag(config=True)
start_timeout = Integer(60,
    help="""Timeout (in seconds) before giving up on the spawner.

    This is the timeout for start to return, not the timeout for the server to respond.
    Callers of spawner.start will assume that startup has failed if it takes longer than this.
    start should return when the server process is started and its location is known.
    """
).tag(config=True)

http_timeout = Integer(30,
    help="""Timeout (in seconds) before giving up on a spawned HTTP server

    Once a server has successfully been spawned, this is the amount of time
    we wait before assuming that the server is unable to accept
    connections.
    """
).tag(config=True)

poll_interval = Integer(30,
    help="""Interval (in seconds) on which to poll the spawner."""
).tag(config=True)
_callbacks = List()
_poll_callback = Any()

debug = Bool(False,
    help="Enable debug-logging of the single-user server"
).tag(config=True)

options_form = Unicode("", help="""
    An HTML form for options a user can specify on launching their server.
    The surrounding `<form>` element and the submit button are already provided.

    For example:

        Set your key:
        <input name="key" val="default_key"></input>
        <br>
        Choose a letter:
        <select name="letter" multiple="true">
          <option value="A">The letter A</option>
          <option value="B">The letter B</option>
        </select>
""").tag(config=True)

def options_from_form(self, form_data):
    """Interpret HTTP form data
```

```
        Form data will always arrive as a dict of lists of strings.
        Override this function to understand single-values, numbers, etc.

        This should coerce form data into the structure expected by self.user_options,
        which must be a dict.

        Instances will receive this data on self.user_options, after passing through this function,
        prior to `Spawner.start`.
        """
        return form_data

    user_options = Dict(help="This is where form-specified options ultimately end up.")

    env_keep = List([
        'PATH',
        'PYTHONPATH',
        'CONDA_ROOT',
        'CONDA_DEFAULT_ENV',
        'VIRTUAL_ENV',
        'LANG',
        'LC_ALL',
    ],
        help="Whitelist of environment variables for the subprocess to inherit"
    ).tag(config=True)
    env = Dict(help="""Deprecated: use Spawner.get_env or Spawner.environment

    - extend Spawner.get_env for adding required env in Spawner subclasses
    - Spawner.environment for config-specified env
    """)

    environment = Dict(
        help="""Environment variables to load for the Spawner.

        Value could be a string or a callable. If it is a callable, it will
        be called with one parameter, which will be the instance of the spawner
        in use. It should quickly (without doing much blocking operations) return
        a string that will be used as the value for the environment variable.
        """
    ).tag(config=True)

    cmd = Command(['jupyterhub-singleuser'],
        help="""The command used for starting notebooks."""
    ).tag(config=True)
    args = List(Unicode(),
        help="""Extra arguments to be passed to the single-user server"""
    ).tag(config=True)

    notebook_dir = Unicode('',
        help="""The notebook directory for the single-user server

        `~` will be expanded to the user's home directory
        `%U` will be expanded to the user's username
        """
    ).tag(config=True)

    default_url = Unicode('',
        help="""The default URL for the single-user server.
```

```
        Can be used in conjunction with --notebook-dir=/ to enable
        full filesystem traversal, while preserving user's homedir as
        landing page for notebook

        `%U` will be expanded to the user's username
        """
    ).tag(config=True)

    disable_user_config = Bool(False,
        help="""Disable per-user configuration of single-user servers.

        This prevents any config in users' $HOME directories
        from having an effect on their server.
        """
    ).tag(config=True)

    def __init__(self, **kwargs):
        super(Spawner, self).__init__(**kwargs)
        if self.user.state:
            self.load_state(self.user.state)

    def load_state(self, state):
        """load state from the database

        This is the extensible part of state

        Override in a subclass if there is state to load.
        Should call `super`.

        See Also
        --------

        get_state, clear_state
        """
        pass

    def get_state(self):
        """store the state necessary for load_state

        A black box of extra state for custom spawners.
        Subclasses should call `super`.

        Returns
        -------

        state: dict
             a JSONable dict of state
        """
        state = {}
        return state

    def clear_state(self):
        """clear any state that should be cleared when the process stops

        State that should be preserved across server instances should not be cleared.

        Subclasses should call super, to ensure that state is properly cleared.
        """
        self.api_token = ''
```

```
    def get_env(self):
        """Return the environment dict to use for the Spawner.

        This applies things like `env_keep`, anything defined in `Spawner.environment`,
        and adds the API token to the env.

        Use this to access the env in Spawner.start to allow extension in subclasses.
        """
        env = {}
        if self.env:
            warnings.warn("Spawner.env is deprecated, found %s" % self.env, DeprecationWarning)
            env.update(self.env)

        for key in self.env_keep:
            if key in os.environ:
                env[key] = os.environ[key]

        # config overrides. If the value is a callable, it will be called with
        # one parameter - the current spawner instance - and the return value
        # will be assigned to the environment variable. This will be called at
        # spawn time.
        for key, value in self.environment.items():
            if callable(value):
                env[key] = value(self)
            else:
                env[key] = value

        env['JPY_API_TOKEN'] = self.api_token
        return env

    def get_args(self):
        """Return the arguments to be passed after self.cmd"""
        args = [
            '--user=%s' % self.user.name,
            '--port=%i' % self.user.server.port,
            '--cookie-name=%s' % self.user.server.cookie_name,
            '--base-url=%s' % self.user.server.base_url,
            '--hub-host=%s' % self.hub.host,
            '--hub-prefix=%s' % self.hub.server.base_url,
            '--hub-api-url=%s' % self.hub.api_url,
            ]
        if self.ip:
            args.append('--ip=%s' % self.ip)
        if self.notebook_dir:
            self.notebook_dir = self.notebook_dir.replace("%U",self.user.name)
            args.append('--notebook-dir=%s' % self.notebook_dir)
        if self.default_url:
            self.default_url = self.default_url.replace("%U",self.user.name)
            args.append('--NotebookApp.default_url=%s' % self.default_url)

        if self.debug:
            args.append('--debug')
        if self.disable_user_config:
            args.append('--disable-user-config')
        args.extend(self.args)
        return args

    @gen.coroutine
```

```
    def start(self):
        """Start the single-user process"""
        raise NotImplementedError("Override in subclass. Must be a Tornado gen.coroutine.")

    @gen.coroutine
    def stop(self, now=False):
        """Stop the single-user process"""
        raise NotImplementedError("Override in subclass. Must be a Tornado gen.coroutine.")

    @gen.coroutine
    def poll(self):
        """Check if the single-user process is running

        return None if it is, an exit status (0 if unknown) if it is not.
        """
        raise NotImplementedError("Override in subclass. Must be a Tornado gen.coroutine.")

    def add_poll_callback(self, callback, *args, **kwargs):
        """add a callback to fire when the subprocess stops

        as noticed by periodic poll_and_notify()
        """
        if args or kwargs:
            cb = callback
            callback = lambda : cb(*args, **kwargs)
        self._callbacks.append(callback)

    def stop_polling(self):
        """stop the periodic poll"""
        if self._poll_callback:
            self._poll_callback.stop()
            self._poll_callback = None

    def start_polling(self):
        """Start polling periodically

        callbacks registered via `add_poll_callback` will fire
        if/when the process stops.

        Explicit termination via the stop method will not trigger the callbacks.
        """
        if self.poll_interval <= 0:
            self.log.debug("Not polling subprocess")
            return
        else:
            self.log.debug("Polling subprocess every %is", self.poll_interval)

        self.stop_polling()

        self._poll_callback = PeriodicCallback(
            self.poll_and_notify,
            1e3 * self.poll_interval
        )
        self._poll_callback.start()

    @gen.coroutine
    def poll_and_notify(self):
        """Used as a callback to periodically poll the process,
        and notify any watchers
```

```
        """
        status = yield self.poll()
        if status is None:
            # still running, nothing to do here
            return

        self.stop_polling()

        for callback in self._callbacks:
            try:
                yield gen.maybe_future(callback())
            except Exception:
                self.log.exception("Unhandled error in poll callback for %s", self)
        return status

    death_interval = Float(0.1)
    @gen.coroutine
    def wait_for_death(self, timeout=10):
        """wait for the process to die, up to timeout seconds"""
        for i in range(int(timeout / self.death_interval)):
            status = yield self.poll()
            if status is not None:
                break
            else:
                yield gen.sleep(self.death_interval)

File:           ~/conda/envs/jupyterhub-tutorial/lib/python3.5/site-packages/jupyterhub/spawner.py
Type:           MetaHasTraits
```

Start is the key method in a Spawner. It's how we decide how to start the process that will become the single-user server:

```
In [3]: LocalProcessSpawner.start??
```

```
Signature: LocalProcessSpawner.start(self)
Source:
    @gen.coroutine
    def start(self):
        """Start the process"""
        if self.ip:
            self.user.server.ip = self.ip
        self.user.server.port = random_port()
        cmd = []
        env = self.get_env()

        cmd.extend(self.cmd)
        cmd.extend(self.get_args())

        self.log.info("Spawning %s", ' '.join(pipes.quote(s) for s in cmd))
        self.proc = Popen(cmd, env=env,
            preexec_fn=self.make_preexec_fn(self.user.name),
            start_new_session=True, # don't forward signals
        )
        self.pid = self.proc.pid

File:      ~/conda/envs/jupyterhub-tutorial/lib/python3.5/site-packages/jupyterhub/spawner.py
Type:      function
```

Here is an example of a spawner that allows specifying extra *arguments* to pass to a user's notebook server, via `.options_form`. It results in a form like this:

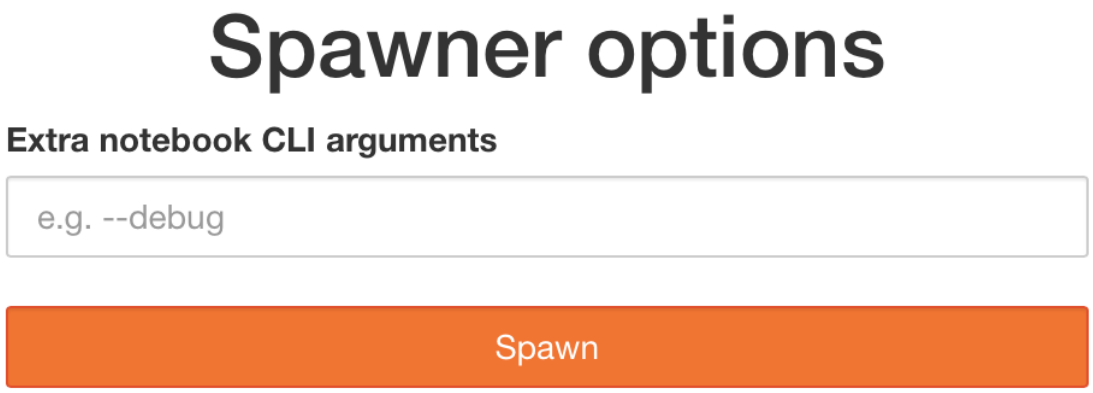

Fig. 2.1: form

```
In [4]: from traitlets import default

        class DemoFormSpawner(LocalProcessSpawner):
            @default('options_form')
            def _options_form(self):
                default_env = "YOURNAME=%s\n" % self.user.name
                return """
                <label for="args">Extra notebook CLI arguments</label>
                <input name="args" placeholder="e.g. --debug"></input>
                """.format(env=default_env)

            def options_from_form(self, formdata):
                """Turn html formdata (always lists of strings) into the dict we want."""
                options = {}
                arg_s = formdata.get('args', [''])[0].strip()
                if arg_s:
                    options['argv'] = shlex.split(arg_s)
                return options

            def get_args(self):
                """Return arguments to pass to the notebook server"""
                argv = super().get_args()
                if self.user_options.get('argv'):
                    argv.extend(self.user_options['argv'])
                return argv

            def get_env(self):
                """Return environment variable dict"""
                env = super().get_env()
                return env
```

2.3.1 Exercise:

Write a custom Spawner that allows users to specify *environment variables* to load into their server.

```
In [5]: from dockerspawner import DockerSpawner

In [6]: DockerSpawner.start??

Signature: DockerSpawner.start(self, image=None, extra_create_kwargs=None, extra_start_kwargs=None, e
Source:
    @gen.coroutine
    def start(self, image=None, extra_create_kwargs=None,
        extra_start_kwargs=None, extra_host_config=None):
        """Start the single-user server in a docker container. You can override
        the default parameters passed to `create_container` through the
        `extra_create_kwargs` dictionary and passed to `start` through the
        `extra_start_kwargs` dictionary.  You can also override the
        'host_config' parameter passed to `create_container` through the
        `extra_host_config` dictionary.

        Per-instance `extra_create_kwargs`, `extra_start_kwargs`, and
        `extra_host_config` take precedence over their global counterparts.

        """
        container = yield self.get_container()
        if container is None:
            image = image or self.container_image

            # build the dictionary of keyword arguments for create_container
            create_kwargs = dict(
                image=image,
                environment=self.get_env(),
                volumes=self.volume_mount_points,
                name=self.container_name)
            create_kwargs.update(self.extra_create_kwargs)
            if extra_create_kwargs:
                create_kwargs.update(extra_create_kwargs)

            # build the dictionary of keyword arguments for host_config
            host_config = dict(binds=self.volume_binds, links=self.links)

            if not self.use_internal_ip:
                host_config['port_bindings'] = {8888: (self.container_ip,)}

            host_config.update(self.extra_host_config)

            if extra_host_config:
                host_config.update(extra_host_config)

            self.log.debug("Starting host with config: %s", host_config)

            host_config = self.client.create_host_config(**host_config)
            create_kwargs.setdefault('host_config', {}).update(host_config)

            # create the container
            resp = yield self.docker('create_container', **create_kwargs)
            self.container_id = resp['Id']
            self.log.info(
                "Created container '%s' (id: %s) from image %s",
                self.container_name, self.container_id[:7], image)

        else:
            self.log.info(
                "Found existing container '%s' (id: %s)",
```

```
                self.container_name, self.container_id[:7])

        # TODO: handle unpause
        self.log.info(
            "Starting container '%s' (id: %s)",
            self.container_name, self.container_id[:7])

        # build the dictionary of keyword arguments for start
        start_kwargs = {}
        start_kwargs.update(self.extra_start_kwargs)
        if extra_start_kwargs:
            start_kwargs.update(extra_start_kwargs)

        # start the container
        yield self.docker('start', self.container_id, **start_kwargs)

        ip, port = yield from self.get_ip_and_port()
        self.user.server.ip = ip
        self.user.server.port = port

File:       ~/conda/envs/jupyterhub-tutorial/lib/python3.5/site-packages/dockerspawner/dockerspawner.p
Type:       function
```

2.3.2 Exercise:

Subclass DockerSpawner so that users can specify via `options_form` what docker image to use.

Candidates from the Jupyter docker-stacks repo include:

- jupyter/minimal-singleuser
- jupyter/scipy-singleuser
- jupyter/r-singleuser
- jupyter/datascience-singleuser
- jupyter/pyspark-singleuser

Or, build your own images with

```
FROM jupyterhub/singleuser
```

The easiest version will assume that the images are fetched already.

2.3.3 Extra credit:

Subclass DockerSpawner so that users can specify via `options_form` a GitHub repository to clone and install, a la binder.

2.4 JupyterHub's API

JupyterHub has a REST API:

```
In [1]: import requests
        hub_api = 'http://127.0.0.1:8081/hub/api/'
```

```
        token = "result of token=$(jupyterhub token yourname)"
```

In [2]:
```
        import json
        from requests import HTTPError

        def api_request(path, method='get', data=None):
            if data:
                data = json.dumps(data)

            r = requests.request(method, hub_api + path,
                headers={'Authorization': 'token %s' % token},
                data=data,
            )
            try:
                r.raise_for_status()
            except Exception as e:
                try:
                    info = r.json()
                except Exception:
                    raise e
                if 'message' in info:
                    # raise nice json error if there was one
                    raise HTTPError("%s: %s" % (r.status_code, info['message'])) from None
                else:
                    # raise original
                    raise e
            if r.text:
                return r.json()
            else:
                return None
```

We can list users and their status:

In [3]:
```
api_request('users')
```

Out[3]:
```
[{'admin': True,
  'last_activity': '2016-05-06T11:53:51.627000',
  'name': 'minrk',
  'pending': None,
  'server': '/user/minrk'},
 {'admin': False,
  'last_activity': '2016-05-06T11:57:13.329254',
  'name': 'takluyver',
  'pending': None,
  'server': None}]
```

We can also start user servers:

In [4]:
```
api_request('users/takluyver/server', method='post')
api_request('users')
```

Out[4]:
```
[{'admin': True,
  'last_activity': '2016-05-06T11:53:51.627000',
  'name': 'minrk',
  'pending': None,
  'server': '/user/minrk'},
 {'admin': False,
  'last_activity': '2016-05-06T11:57:29.285044',
  'name': 'takluyver',
  'pending': None,
  'server': '/user/takluyver'}]
```

And stop them:

```
In [5]: api_request('users/takluyver/server', method='delete')
        api_request('users')
```

```
Out[5]: [{'admin': True,
          'last_activity': '2016-05-06T11:53:51.627000',
          'name': 'minrk',
          'pending': None,
          'server': '/user/minrk'},
         {'admin': False,
          'last_activity': '2016-05-06T11:57:30.145986',
          'name': 'takluyver',
          'pending': None,
          'server': None}]
```

We can also see the proxy routing table:

```
In [6]: api_request('proxy')
```

```
Out[6]: {'/': {'last_activity': '2016-05-06T11:56:29.543Z',
          'target': 'http://127.0.0.1:8081'},
         '/user/minrk': {'last_activity': '2016-05-06T11:57:29.159Z',
          'target': 'http://127.0.0.1:45683',
          'user': 'minrk'}}
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search