Kernel Gateway Documentation

Release 2.2.0.dev

Project Jupyter team

Sep 07, 2018

User Documentation

Jupyter Kernel Gateway is a web server that provides headless access to Jupyter kernels. Your application communicates with the kernels remotely, through REST calls and Websockets rather than ZeroMQ messages. There are no provisions for editing notebooks through the Kernel Gateway. The following operation modes, called personalities, are supported out of the box:

- Send code snippets for execution using the Jupyter kernel protocol over Websockets. Start and stop kernels through REST calls. This HTTP API is compatible with the respective API sections of the Jupyter Notebook server.

- Serve HTTP requests from annotated notebook cells. The code snippets are cells of a static notebook configured in the Kernel Gateway. Annotations define which HTTP verbs and resources it supports. Incoming requests are served by executing one of the cells in a kernel.

Jupyter Kernel Gateway uses the same code as Jupyter Notebook to launch kernels in its local process/filesystem space. It can be containerized and scaled out using common technologies like tmpnb, Cloud Foundry, and Kubernetes.
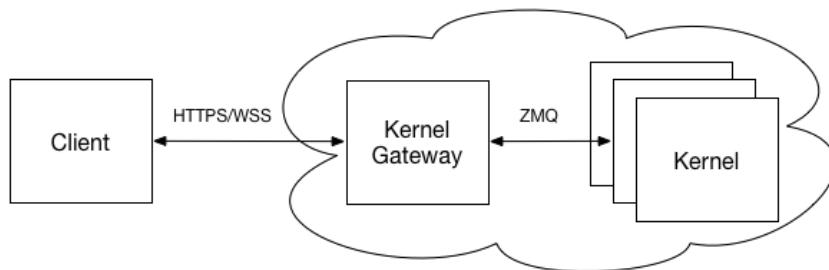
Getting started

This document describes some of the basics of installing and running the Jupyter Kernel Gateway.

1.1 Using pip

We make stable releases of the kernel gateway to PyPI. You can use `pip` to install the latest version along with its dependencies.

```
# install from pypi
pip install jupyter_kernel_gateway
```

Once installed, you can use the `jupyter` CLI to run the server.

```
# run it with default options
jupyter kernelgateway
```

1.2 Using conda

You can install the kernel gateway using conda as well.

```
conda install -c conda-forge jupyter_kernel_gateway
```

Once installed, you can use the `jupyter` CLI to run the server as shown above.

1.3 Using a docker-stacks image

You can add the kernel gateway to any docker-stacks image by writing a Dockerfile patterned after the following example:

```
# start from the jupyter image with R, Python, and Scala (Apache Toree) kernels pre-
→installed
FROM jupyter/all-spark-notebook

# install the kernel gateway
RUN pip install jupyter_kernel_gateway

# run kernel gateway on container start, not notebook server
EXPOSE 8888
CMD ["jupyter", "kernelgateway", "--KernelGatewayApp.ip=0.0.0.0", "--KernelGatewayApp.
→port=8888"]
```

You can then build and run it.

```
docker build -t my/kernel-gateway .
docker run -it --rm -p 8888:8888 my/kernel-gateway
```

Use Cases

The Jupyter Kernel Gateway makes possible the following novel uses of kernels:

- Attach a local Jupyter Notebook server to a compute cluster in the cloud running near big data (e.g., interactive gateway to Spark)

- Enable a new breed of non-notebook web clients to provision and use kernels (e.g., dashboards using jupyter-js-services)

- Scale kernels independently from clients (e.g., via tmpnb, Binder, or your favorite cluster manager)

- Create microservices from notebooks via `notebook-http` mode

The following diagram shows how you might use `tmpnb` to deploy a pool of kernel gateway instances in Docker containers to support on-demand interactive compute:
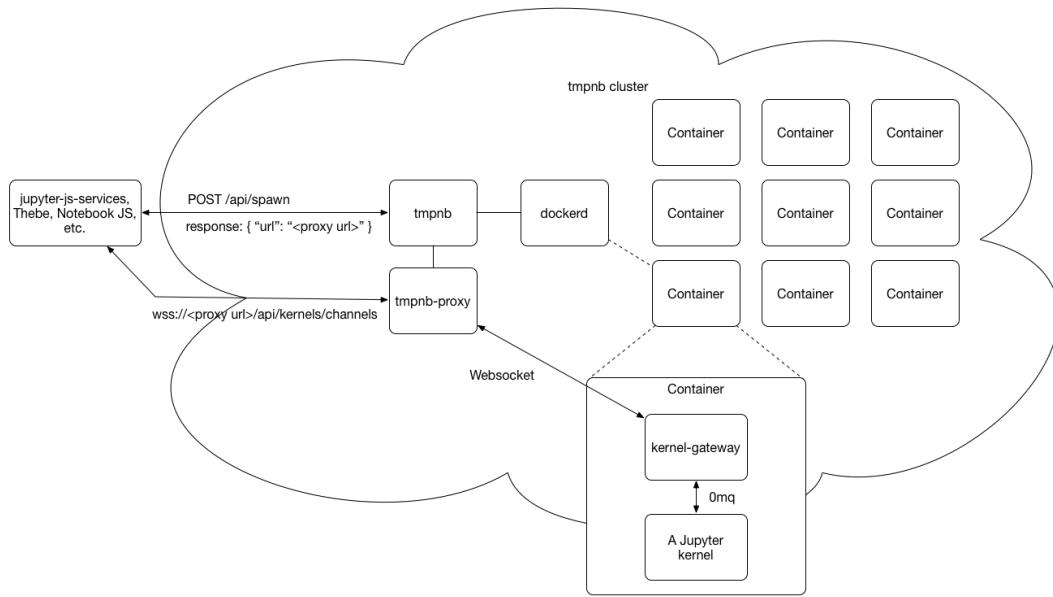
For more inspiration, see the jupyter/kernel_gateway_demos.

CHAPTER 3

Features

The Jupyter Kernel Gateway has the following features:

- `jupyter-websocket` mode which provides a Jupyter Notebook server-compatible API for requesting kernels and communicating with them using Websockets

- `notebook-http` mode which maps HTTP requests to cells in annotated notebooks

- Option to enable other kernel communication mechanisms by plugging in third party personalities

- Option to set a shared authentication token and require it from clients

- Options to set CORS headers for servicing browser-based clients

- Option to set a custom base URL (e.g., for running under tmpnb)

- Option to limit the number kernel instances a gateway server will launch (e.g., to force scaling at the container level)

- Option to pre-spawn a set number of kernel instances

- Option to set a default kernel language to use when one is not specified in the request

- Option to pre-populate kernel memory from a notebook

- Option to serve annotated notebooks as HTTP endpoints, see notebook-http

- Option to allow downloading of the notebook source when running in `notebook-http` mode

- Generation of Swagger specs for notebook-defined API in `notebook-http` mode

- A CLI for launching the kernel gateway: `jupyter kernelgateway OPTIONS`

- A Python 2.7 and 3.3+ compatible implementation

`jupyter-websocket` Mode

The `KernelGatewayApp.api` command line argument defaults to `kernel_gateway.jupyter_websocket`. This mode, or *personality*, has the kernel gateway expose:

1. a superset of the HTTP API provided by the Jupyter Notebook server, and

2. the equivalent Websocket API implemented by the Jupyter Notebook server.

4.1 HTTP Resources

The HTTP API consists of kernel, session, monitoring, and metadata resources. All of these are documented in a swagger.yaml file. You can use the Swagger UI to interact with a running instance of the kernel gateway by pointing the tool to the `/api/swagger.json` resource.

4.2 Websocket Resources

The `/api/kernels/{kernel_id}/channels` resource multiplexes the Jupyter kernel messaging protocol over a single Websocket connection.

See the NodeJS and Python client demos for two simple examples of using these resources to send code to kernels for interactive evaluation.

notebook-http Mode

The `KernelGatewayApp.api` command line argument can be set to `kernel_gateway.notebook_http`. This mode, or *personality*, has the kernel gateway expose annotated cells in the `KernelGatewayApp.seed_uri` notebook as HTTP resources.

To turn a notebook cell into a HTTP handler, you must prefix it with a single line comment. The comment describes the HTTP method and resource, as in the following Python example:

```python
# GET /hello/world
print("hello world")
```

The annotation above declares the cell contents as the code to execute when the kernel gateway receives a HTTP GET request for the path `/hello/world`. For other languages, the comment prefix may change, but the rest of the annotation remains the same.

Multiple cells may share the same annotation. Their content is concatenated to form a single code segment at runtime. This facilitates typical, iterative development in notebooks with lots of short, separate cells: The notebook author does not need to merge all of the cells into one, or refactor to use functions.

```python
# GET /hello/world
print("I'm cell #1")

# GET /hello/world
print("I'm cell #2")
```

5.1 Getting the Request Data

Before the gateway invokes an annotated cell, it sets the value of a global notebook variable named `REQUEST` to a JSON string containing information about the request. You may parse this string to access the request properties.

For example, in Python:

```
# GET /hello/world
req = json.loads(REQUEST)
# do something with req
```

You may specify path parameters when registering an endpoint by prepending a `:` to a path segment. For example, a path with parameters `firstName` and `lastName` would be defined as the following in a Python comment:

```
# GET /hello/:firstName/:lastName
```

The `REQUEST` object currently contains the following properties:

- `body` - The value of the body, see the *Body And Content Type* section below

- `args` - An object with keys representing query parameter names and their associated values. A query parameter name may be specified multiple times in a valid URL, and so each value is a sequence (e.g., list, array) of strings from the original URL.

- `path` - An object of key-value pairs representing path parameters and their values.

- `headers` - An object of key-value pairs where a key is a HTTP header name and a value is the HTTP header value. If there are multiple values are specified for a header, the value will be an array.

5.1.1 Request Content-Type and Request Body Processing

If the HTTP request to the kernel gateway has a `Content-Type` header the value of `REQUEST.body` may change. Below is the list of outcomes for various mime-types:

- `application/json` - The `REQUEST.body` will be an object of key-value pairs representing the request body

- `multipart/form-data` and `application/x-www-form-urlencoded` - The `REQUEST.body` will be an object of key-value pairs representing the parameters and their values. Files are currently not supported for `multipart/form-data`

- `text/plain` - The `REQUEST.body` will be the string value of the body

- All other types will be sent as strings

5.2 Setting the Response Body

The response from an annotated cell may be set in one of two ways:

1. Writing to stdout in a notebook cell

2. Emitting output in a notebook cell

The first method is preferred because it is explicit: a cell writes to stdout using the appropriate language statement or function (e.g. Python `print`, Scala `println`, R `print`, etc.). The kernel gateway collects all bytes from kernel stdout and returns the entire byte string verbatim as the response body.

The second approach is used if nothing appears on stdout. This method is dependent upon language semantics, kernel implementation, and library usage. The response body will be the `content.data` structure in the Jupyter `execute_result` message.

In both cases, the response defaults to status `200 OK` and `Content-Type: text/plain` if cell execution completes without error. If an error occurs, the status is `500 Internal Server Error`. If the HTTP request method is not one supported at the given path, the status is `405 Not Supported`. If you wish to return custom status or headers, see the next section.

See the api_intro.ipynb notebook for basic request and response examples.

5.3 Setting the Response Status and Headers

Annotated cells may have an optional metadata companion cell that sets the HTTP response status and headers. Consider this Python cell that creates a person entry in a database table and returns the new row ID in a JSON object:

```
# POST /person
req = json.loads(REQUEST)
row_id = person_table.insert(req['body'])
res = {'id' : row_id}
print(json.dumps(res))
```

Now consider this companion cell which runs after the cell above and sets a custom response header and status:

```
# ResponseInfo POST /person
print(json.dumps({
    "headers" : {
        "Content-Type" : "application/json"
    },
    "status" : 201
}))
```

Currently, `headers` and `status` are the only fields supported. `headers` should be an object of key-value pairs mapping header names to header values. `status` should be an integer value. Both should be printed to stdout as JSON.

Given the two cells above, a `POST` request to `/person` produces a HTTP response like the following from the kernel gateway, assuming no errors occur:

```
HTTP/1.1 200 OK
Content-Type: application/json

{"id": 123}
```

See the setting_response_metadata.ipynb notebook for examples of setting response metadata.

5.4 Swagger Spec

The resource `/_api/spec/swagger.json` is automatically generated from the notebook used to define the HTTP API. The response is a simple Swagger spec which can be used with the Swagger editor, a Swagger ui, or with any other Swagger-aware tool.

Currently, every response is listed as having a status of `200 OK`.

5.5 Running

The minimum number of arguments needed to run in HTTP mode are `--KernelGatewayApp.api=kernel_gateway.notebook_http` and `--KernelGatewayApp.seed_uri=some/notebook/file.ipynb`.

The notebook-http mode will honor the `prespawn_count` command line argument. This will start the specified number of kernels and execute the `seed_uri` notebook on each one. Requests will be distributed across the pool of prespawned kernels, providing a minimal layer of scalability. An example which starts a pool of 5 kernels follows:

```
jupyter kernelgateway \
    --KernelGatewayApp.api='kernel_gateway.notebook_http' \
    --KernelGatewayApp.seed_uri='/srv/kernel_gateway/etc/api_examples/api_intro.ipynb
↪' \
    --KernelGatewayApp.prespawn_count=5
```

Refer to the scotch recommendation API demo for more detail.

If you have a development setup, you can run the kernel gateway in `notebook-http` mode using the Makefile in this repository:

```
make dev ARGS="--KernelGatewayApp.api='kernel_gateway.notebook_http' \
--KernelGatewayApp.seed_uri=/srv/kernel_gateway/etc/api_examples/api_intro.ipynb"
```

With the above Make command, all of the notebooks in `etc/api_examples` are mounted into `/srv/kernel_gateway/etc/api_examples/` and can be run in HTTP mode.

CHAPTER 6

Developing New Modes

The `KernelGatewayApp.api` can be set to the name of any module in the Python path supplying a personality. This allows for alternate kernel communications mechanisms.

The module must contain a `create_personality` function whose `parent` argument will be the kernel gateway application, and which must return a *personality* object. That object will take part in the kernel gateway's lifecycle and act as a delegate for certain responsibilities. An example module, subclassing `LoggingConfigurable` as recommended, is shown here:

```python
from traitlets.config.configurable import LoggingConfigurable

class TemplatePersonality(LoggingConfigurable):
    def init_configurables(self):
        """This function will be called when the kernel gateway has completed its own
        `init_configurables`, typically after its traitlets have been evaluated."""
        pass

    def shutdown(self):
        """During a proper shutdown of the kernel gateway, this will be called so that
        any held resources may be properly released."""
        pass

    def create_request_handlers(self):
        """Returns a list of zero or more tuples of handler path, Tornado handler
        class
        name, and handler arguments, that should be registered in the kernel gateway
        's
        web application. Paths are used as given and should respect the kernel gateway
        's
        `base_url` traitlet value."""
        pass

    def should_seed_cell(self, code):
        """Determines whether the kernel gateway will include the given notebook code
        cell when seeding a new kernel. Will only be called if a seed notebook has
```

(continues on next page)

15

```python
        been specified."""
        pass


def create_personality(self, parent):
    """Put docstring here."""
    return TemplatePersonality(parent=parent)
```

Provided personalities include kernel_gateway.jupyter_websocket and kernel_gateway.notebook_http.

Configuration options

The kernel gateway adheres to the Jupyter common configuration approach. You can configure an instance of the kernel gateway using:

1. A configuration file

2. Command line parameters

3. Environment variables

To generate a template configuration file, run the following:

```
jupyter kernelgateway --generate-config
```

To see the same configuration options at the command line, run the following:

```
jupyter kernelgateway --help-all
```

A snapshot of this help appears below for ease of reference on the web.

```
Jupyter Kernel Gateway

Provisions Jupyter kernels and proxies HTTP/Websocket traffic to them.

Options
-------

Arguments that take values are actually convenience aliases to full
Configurables, whose aliases are listed on the help line. For more information
on full configurables, see '--help-all'.

--debug
    set log level to logging.DEBUG (maximize logging output)
--generate-config
    generate default config file
-y
    Answer yes to any questions instead of prompting.
```

```
--certfile=<Unicode> (KernelGatewayApp.certfile)
    Default: None
    The full path to an SSL/TLS certificate file. (KG_CERTFILE env var)
--config=<Unicode> (JupyterApp.config_file)
    Default: ''
    Full path of a config file.
--port_retries=<Int> (KernelGatewayApp.port_retries)
    Default: 50
    Number of ports to try if the specified port is not available
    (KG_PORT_RETRIES env var)
--ip=<Unicode> (KernelGatewayApp.ip)
    Default: '127.0.0.1'
    IP address on which to listen (KG_IP env var)
--keyfile=<Unicode> (KernelGatewayApp.keyfile)
    Default: None
    The full path to a private key file for usage with SSL/TLS. (KG_KEYFILE env
    var)
--seed_uri=<Unicode> (KernelGatewayApp.seed_uri)
    Default: None
    Runs the notebook (.ipynb) at the given URI on every kernel launched. No
    seed by default. (KG_SEED_URI env var)
--client-ca=<Unicode> (KernelGatewayApp.client_ca)
    Default: None
    The full path to a certificate authority certificate for SSL/TLS client
    authentication. (KG_CLIENT_CA env var)
--port=<Int> (KernelGatewayApp.port)
    Default: 8888
    Port on which to listen (KG_PORT env var)
--api=<Unicode> (KernelGatewayApp.api)
    Default: 'kernel_gateway.jupyter_websocket'
    Controls which API to expose, that of a Jupyter notebook server, the seed
    notebook's, or one provided by another module, respectively using values
    'kernel_gateway.jupyter_websocket', 'kernel_gateway.notebook_http', or
    another fully qualified module name (KG_API env var)
--log-level=<Enum> (Application.log_level)
    Default: 30
    Choices: (0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITICAL')
    Set the log level by value or name.

Class parameters
----------------

Parameters are set from command-line arguments of the form:
`--Class.trait=value`. This line is evaluated in Python, so simple expressions
are allowed, e.g.:: `--C.a='range(3)'` For setting C.a=[0,1,2].

KernelGatewayApp options
----------------------
--KernelGatewayApp.allow_credentials=<Unicode>
    Default: ''
    Sets the Access-Control-Allow-Credentials header. (KG_ALLOW_CREDENTIALS env
    var)
--KernelGatewayApp.allow_headers=<Unicode>
    Default: ''
    Sets the Access-Control-Allow-Headers header. (KG_ALLOW_HEADERS env var)
--KernelGatewayApp.allow_methods=<Unicode>
    Default: ''
```

```
         Sets the Access-Control-Allow-Methods header. (KG_ALLOW_METHODS env var)
--KernelGatewayApp.allow_origin=<Unicode>
     Default: ''
     Sets the Access-Control-Allow-Origin header. (KG_ALLOW_ORIGIN env var)
--KernelGatewayApp.answer_yes=<Bool>
     Default: False
     Answer yes to any prompts.
--KernelGatewayApp.api=<Unicode>
     Default: 'kernel_gateway.jupyter_websocket'
     Controls which API to expose, that of a Jupyter notebook server, the seed
     notebook's, or one provided by another module, respectively using values
     'kernel_gateway.jupyter_websocket', 'kernel_gateway.notebook_http', or
     another fully qualified module name (KG_API env var)
--KernelGatewayApp.auth_token=<Unicode>
     Default: ''
     Authorization token required for all requests (KG_AUTH_TOKEN env var)
--KernelGatewayApp.base_url=<Unicode>
     Default: '/'
     The base path for mounting all API resources (KG_BASE_URL env var)
--KernelGatewayApp.certfile=<Unicode>
     Default: None
     The full path to an SSL/TLS certificate file. (KG_CERTFILE env var)
--KernelGatewayApp.client_ca=<Unicode>
     Default: None
     The full path to a certificate authority certificate for SSL/TLS client
     authentication. (KG_CLIENT_CA env var)
--KernelGatewayApp.config_file=<Unicode>
     Default: ''
     Full path of a config file.
--KernelGatewayApp.config_file_name=<Unicode>
     Default: ''
     Specify a config file to load.
--KernelGatewayApp.default_kernel_name=<Unicode>
     Default: ''
     Default kernel name when spawning a kernel (KG_DEFAULT_KERNEL_NAME env var)
--KernelGatewayApp.expose_headers=<Unicode>
     Default: ''
     Sets the Access-Control-Expose-Headers header. (KG_EXPOSE_HEADERS env var)
--KernelGatewayApp.force_kernel_name=<Unicode>
     Default: ''
     Override any kernel name specified in a notebook or request
     (KG_FORCE_KERNEL_NAME env var)
--KernelGatewayApp.generate_config=<Bool>
     Default: False
     Generate default config file.
--KernelGatewayApp.ip=<Unicode>
     Default: '127.0.0.1'
     IP address on which to listen (KG_IP env var)
--KernelGatewayApp.keyfile=<Unicode>
     Default: None
     The full path to a private key file for usage with SSL/TLS. (KG_KEYFILE env
     var)
--KernelGatewayApp.log_datefmt=<Unicode>
     Default: '%Y-%m-%d %H:%M:%S'
     The date format used by logging formatters for %(asctime)s
--KernelGatewayApp.log_format=<Unicode>
     Default: '[%(name)s]%(highlevel)s %(message)s'
```

```
    The Logging format template
--KernelGatewayApp.log_level=<Enum>
    Default: 30
    Choices: (0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITICAL')
    Set the log level by value or name.
--KernelGatewayApp.max_age=<Unicode>
    Default: ''
    Sets the Access-Control-Max-Age header. (KG_MAX_AGE env var)
--KernelGatewayApp.max_kernels=<Int>
    Default: None
    Limits the number of kernel instances allowed to run by this gateway.
    Unbounded by default. (KG_MAX_KERNELS env var)
--KernelGatewayApp.port=<Int>
    Default: 8888
    Port on which to listen (KG_PORT env var)
--KernelGatewayApp.port_retries=<Int>
    Default: 50
    Number of ports to try if the specified port is not available
    (KG_PORT_RETRIES env var)
--KernelGatewayApp.prespawn_count=<Int>
    Default: None
    Number of kernels to prespawn using the default language. No prespawn by
    default. (KG_PRESPAWN_COUNT env var)
--KernelGatewayApp.seed_uri=<Unicode>
    Default: None
    Runs the notebook (.ipynb) at the given URI on every kernel launched. No
    seed by default. (KG_SEED_URI env var)
--KernelGatewayApp.env_process_whitelist=<List>
    Default: []
    Environment variables allowed to be inherited from current process by a
    new kernel.

NotebookHTTPPersonality options
-------------------------------
--NotebookHTTPPersonality.allow_notebook_download=<Bool>
    Default: False
    Optional API to download the notebook source code in notebook-http mode,
    defaults to not allow
--NotebookHTTPPersonality.cell_parser=<Unicode>
    Default: 'kernel_gateway.notebook_http.cell.parser'
    Determines which module is used to parse the notebook for endpoints and
    documentation. Valid module names include
    'kernel_gateway.notebook_http.cell.parser' and
    'kernel_gateway.notebook_http.swagger.parser'. (KG_CELL_PARSER env var)
--NotebookHTTPPersonality.comment_prefix=<Dict>
    Default: {None: '#', 'scala': '//'}
    Maps kernel language to code comment syntax
--NotebookHTTPPersonality.static_path=<Unicode>
    Default: None
    Serve static files on disk in the given path as /public, defaults to not
    serve

JupyterWebsocketPersonality options
-----------------------------------
--JupyterWebsocketPersonality.env_whitelist=<List>
    Default: []
    Environment variables allowed to be set when a client requests a new kernel
```

```
--JupyterWebsocketPersonality.list_kernels=<Bool>
    Default: False
    Permits listing of the running kernels using API endpoints /api/kernels and
    /api/sessions (KG_LIST_KERNELS env var). Note: Jupyter Notebook allows this
    by default but kernel gateway does not.
```

Troubleshooting

8.1 I can't access kernel gateway in my Docker container.

The kernel gateway server listens on port 8888 by default. Make sure this internal port is exposed on an external port when starting the container. For example, if you run:

```
docker run -it --rm -p 9000:8888 jupyter/minimal-kernel
```

you can access your kernel gateway on the IP address of your Docker host an port 9000.

8.2 Kernel gateway raises an error when I use `notebook-http` mode.

The `notebook-http` mode publishes a web API defined by annotations and code in a notebook. Make sure you are specifying a path or URL of a notebook (`*.ipynb`) file when you launch the kernel gateway in this mode. Set the `--KernelGatewayApp.seed_uri` command line parameter or `KG_SEED_URI` environment variable to do so.

Development Workflow

This document includes instructions for setting up a development environment for the Jupyter Kernel Gateway. It also includes common steps in the developer workflow such as running tests, building docs, updating specs, etc.

9.1 Prerequisites

Install miniconda and GNU make on your system.

9.2 Clone the repo

Clone this repository in a local directory.

```
# make a directory under ~ to put source
mkdir -p ~/projects
cd !$

# clone this repo
git clone https://github.com/jupyter/kernel_gateway.git
```

9.3 Build a conda environment

Build a Python 3 conda environment containing the necessary dependencies for running the kernel gateway, running tests, and building documentation.

```
make env
```

9.4 Run the tests

Run the tests suite.

```
make test
```

9.5 Run the gateway server

Run an instance of the kernel gateway server in `jupyter-websocket` mode.

```
make dev
```

Then access the running server at the URL printed in the console.

Run an instance of the kernel gateway server in `notebook-http` mode using the `api_intro.ipynb` notebook in the repository.

```
make dev-http
```

Then access the running server at the URL printed in the console.

9.6 Build the docs

Run Sphinx to build the HTML documentation.

```
make docs
```

9.7 Update the Swagger API spec

After modifying any of the APIs in `jupyter-websocket` mode, you must update the project's Swagger API specification.

1. Load the current swagger.yaml file into the Swagger editor.
2. Make your changes.
3. Export both the `swagger.json` and `swagger.yaml` files.
4. Place the files in `kernel_gateway/jupyter_websocket`.
5. Add, commit, and PR the changes.

Summary of changes

See `git log` for a more detailed summary of changes.

10.1 2.1

10.1.1 2.1.0 (2018-08-13)

- PR-299: adds x_header configuration option for use behind proxies
- PR-294: Allow access from remote hosts (Notebook 5.6)
- PR-292: Update dependencies of Jupyter components
- PR-290: Include LICENSE file in wheels
- PR-285: Update Kernel Gateway test base class to be compatible with Tornado 5.0
- PR-284: Add reason argument to set_status() so that custom messages flow back to client
- PR-280: Add whitelist of environment variables to be inherited from gateway process by kernel
- PR-275: Fix broken links to notebook-http mode page in docs
- PR-272: Fix bug when getting kernel language in notebook-http mode
- PR-271: Fix IPerl notebooks running in notebook-http mode

10.2 2.0

10.2.1 2.0.2 (2017-11-10)

- PR-266: Make KernelManager and KernelSpecManager configurable
- PR-263: Correct JSONErrorsMixin for compatibility with notebook 5.2.0

10.2.2 2.0.1 (2017-09-09)

- PR-258: Remove auth token check for OPTIONS requests (CORS)

10.2.3 2.0.0 (2017-05-30)

- Update compatibility to notebook>=5.0
- Remove kernel activity API in favor of the one in the notebook package
- Update project overview in the documentation
- Inherit the server `PATH` when launching a new kernel via POST request with custom environment variables
- Fix kernel cleanup upon SIGTERM
- Fix security requirements in the swagger spec
- Fix configured headers for OPTIONS requests

10.3 1.2

10.3.1 1.2.2 (2017-05-30)

- Inherit the server `PATH` when launching a new kernel via POST request with custom environment variables
- Fix kernel cleanup upon SIGTERM

10.3.2 1.2.1 (2017-04-01)

- Add support for auth token as a query parameter

10.3.3 1.2.0 (2017-02-12)

- Add command line option to whitelist environment variables for `POST /api/kernels`
- Add support for HTTPS key and certificate files
- Improve the flow and explanations in the `api_intro` notebook
- Fix incorrect use of `metadata.kernelspec.name` as a language name instead of `metadata.language.info`
- Fix lingering kernel regression after Ctrl-C interrupt
- Switch to a conda-based dev setup from docker

10.4 1.1

10.4.1 1.1.1 (2016-09-10)

- Add LICENSE file to package distributions

10.4.2 1.1.0 (2016-09-08)

- Add an option to force a specific kernel spec for all requests and seed notebooks
- Add support for specifying notebook-http APIs using full Swagger specs
- Add option to serve static web assets from Tornado in notebook-http mode
- Add command line aliases for common options (e.g., `--ip`)
- Fix Tornado 4.4 compatbility: sending an empty body string with a 204 response

10.5 1.0

10.5.1 1.0.0 (2016-07-15)

- Introduce an [API for developing mode plug-ins](#)
- Separate `jupyter-websocket` and `notebook-http` modes into plug-in packages
- Move mode specific command line options into their respective packages (see `--help-all`)
- Report times with respect to UTC in `/_api/activity` responses

10.6 0.6

10.6.1 0.6.0 (2016-06-17)

- Switch HTTP status from 402 for 403 when server reaches the max kernel limit
- Explicitly shutdown kernels when the server shuts down
- Remove `KG_AUTH_TOKEN` from the environment of kernels
- Fix missing swagger document in release
- Add `--KernelGateway.port_retries` option like in Jupyter Notebook
- Fix compatibility with Notebook 4.2 session handler `create_session` call

10.7 0.5

10.7.1 0.5.1 (2016-04-20)

- Backport `--KernelGateway.port_retries` option like in Jupyter Notebook
- Fix compatibility with Notebook 4.2 session handler `create_session` call

10.7.2 0.5.0 (2016-04-04)

- Support multiple cells per path in `notebook-http` mode
- Add a Swagger specification of the `jupyter-websocket` API
- Add `KERNEL_GATEWAY=1` to all kernel environments

- Support environment variables in `POST /api/kernels`
- numpydoc format docstrings on everything
- Convert README to Sphinx/ReadTheDocs site
- Convert `ActivityManager` to a traitlets `LoggingConfigurable`
- Fix `base_url` handling for all paths
- Fix unbounded growth of ignored kernels in `ActivityManager`
- Fix caching of Swagger spec in `notebook-http` mode
- Fix failure to install due to whitespace in `setup.py` version numbers
- Fix call to kernel manager base class when starting a kernel
- Fix test fixture hangs

10.8 0.4

10.8.1 0.4.1 (2016-04-20)

- Backport `--KernelGateway.port_retries` option like in Jupyter Notebook
- Fix compatibility with Notebook 4.2 session handler `create_session` call

10.8.2 0.4.0 (2016-02-17)

- Enable `/_api/activity` resource with stats about kernels in `jupyter-websocket` mode
- Enable `/api/sessions` resource with in-memory name-to-kernel mapping for non-notebook clients that want to look-up kernels by associated session name
- Fix prespawn kernel logic regression for `jupyter-websocket` mode
- Fix all handlers so that they return application/json responses on error
- Fix missing output from cells that emit display data in `notebook-http` mode

10.9 0.3

10.9.1 0.3.1 (2016-01-25)

- Fix CORS and auth token headers for `/_api/spec/swagger.json` resource
- Fix `allow_origin` handling for non-browser clients
- Ensure base path is prefixed with a forward slash
- Filter stderr from all responses in `notebook-http` mode
- Set Tornado logging level and Jupyter logging level together with `--log-level`

10.9.2 0.3.0 (2016-01-15)

- Support setting of status and headers in `notebook-http` mode
- Support automatic, minimal Swagger doc generation in `notebook-http` mode
- Support download of a notebook in `notebook-http` mode
- Support CORS and token auth in `notebook-http` mode
- Expose HTTP request headers in `notebook-http` mode
- Support multipart form encoding in `notebook-http` mode
- Fix request value JSON encoding when passing requests to kernels
- Fix kernel name handling when pre-spawning
- Fix lack of access logs in `notebook-http` mode

10.10 0.2

10.10.1 0.2.0 (2015-12-15)

- Support notebook-defined HTTP APIs on a pool of kernels
- Disable kernel instance list by default

10.11 0.1

10.11.1 0.1.0 (2015-11-18)

- Support Jupyter Notebook kernel CRUD APIs and Jupyter kernel protocol over Websockets
- Support shared token auth
- Support CORS headers
- Support base URL
- Support seeding kernels code from a notebook at a file path or URL
- Support default kernel, kernel pre-spawning, and kernel count limit
- First PyPI release

CHAPTER 11

Indices and tables

- genindex
- modindex
- search