

---

# **MFLib**

***Release 0.3.1***

**Fabric UKY Team**

**Apr 22, 2023**

# CONTENTS

<b>1</b>	<b>Documentation Resources</b>	<b>2</b>
1.1	Example Jupyter Notebooks . . . . .	2
1.2	FABRIC Learn Site . . . . .	2
1.3	MFLib Python Package Documentation . . . . .	2
<b>2</b>	<b>MFLib Installation</b>	<b>3</b>
2.1	Instaling via PIP . . . . .	3
2.2	Installing via Source Code . . . . .	3
<b>3</b>	<b>Building &amp; Deploying</b>	<b>4</b>
3.1	Spinx Documentation . . . . .	4
3.2	Distribution Package . . . . .	5
<b>4</b>	<b>MFLib Overview</b>	<b>6</b>
4.1	MFLib Methods . . . . .	6
<b>5</b>	<b>MFLib</b>	<b>10</b>
<b>6</b>	<b>MFLib Core</b>	<b>12</b>
<b>7</b>	<b>MF mf_timestamp</b>	<b>17</b>
<b>8</b>	<b>OWL</b>	<b>18</b>
<b>9</b>	<b>OWL Data</b>	<b>21</b>
	<b>Python Module Index</b>	<b>22</b>
	<b>Index</b>	<b>23</b>

docs passing

Welcome to the FABRIC Measurement Framework Library. MFLib makes it easy to install monitoring systems to a FABRIC experimenter's slice. The monitoring system makes extensive use of industry standards such as Prometheus, Grafana, Elastic Search and Kibana while adding customized monitoring tools and dashboards for quick setup and visualization.

## DOCUMENTATION RESOURCES

For more information about FABRIC visit [fabric-testbed.net](http://fabric-testbed.net)

### 1.1 Example Jupyter Notebooks

[FABRIC Jupyter Examples](#) GitHub repository contains many examples for using FABRIC from simple slice setup to advanced networking setups. Look for the MFLib section. These notebooks are designed to be easily used on the [FABRIC JupyterHub](#)

### 1.2 FABRIC Learn Site

[FABRIC Knowledge Base](#)

### 1.3 MFLib Python Package Documentation

Documentation for the package is presented in several different forms (and maybe include later in this document):

- [ReadTheDocs](#)
- [MFLib.pdf](#) in the source code/GitHub.
- Or you may build the documentation from the source code. See Sphinx Documentation later in this document.

## MFLIB INSTALLATION

### 2.1 Instalng via PIP

MFLib may be installed using PIP and PyPI [fabrictestbed-mflib](#)

```
pip install --user fabrictestbed-mflib
```

### 2.2 Installing via Source Code

If you need a development version, clone the git repo, then use pip to install.

```
git clone https://github.com/fabric-testbed/mflib.git
cd mflib
pip install --user .
```

## **BUILDING & DEPLOYING**

### **3.1 Spinx Documentation**

This package is documented using sphinx. The source directories are already created and populated with reStructuredText ( .rst ) files. The build directories are deleted and/or are not included in the repository,

API documentation can also be found at <https://fabrictestbed-mflib.readthedocs.io/>.

#### **3.1.1 Build HTML Documents**

Install the extra packages required to build API docs: (sphinx, furo theme, and myst-parser for parsing markdown files):

```
pip install -r docs/requirements.txt
```

Build the documentation by running the following command from the root directory of the repo.

```
./create_html_doc.sh
```

The completed documentation may be accessed by clicking on /docs/build/html/index.html. Note that the HTML docs are not saved to the repository.

#### **3.1.2 Build PDF Document**

Latex must be installed. For Debian use:

```
sudo apt install texlive-latex-extra  
sudo apt install latexmk
```

Run the bash script to create the MFLIB.pdf documentation. MFLIB.pdf will be placed in the root directory of the repository.

```
./create_pdf_doc.sh
```

## 3.2 Distribution Package

MFLib package is created using [Flit](#). Be sure to create and commit the PDF documentation to GitHub before building and publishing to PyPi. The MFLib.pdf is included in the distribution.

To build python package for PyPi run

```
./create_release.sh
```

### 3.2.1 Uploading to PyPi

First test the package by uploading to test.pypi.org then test the install.

```
flit publish --repository testpypi
```

Once install is good, upload to PiPy

```
flit publish
```

Note that Flit places a .pypirc file in your home directory if you do not already have one. Flit may also store your password in the keyring which may break if the password is changed. see [Flit Controlling package uploads](#). The password can also be added to the .pypirc file. If password contains % signs it will break the .pypirc file.

## MFLIB OVERVIEW

MFLib consists of several classes.

Core – Core makes up the base class that defines methods needed to interact with the nodes in a slice, most notably with the special Measurement Node. This class is used by higher-level classes so the average user will not need to use this class directly. MFLib – MFLib is the main class that a user will use to instrumentize a slice and interact with the monitoring systems. MFVis – MFVis makes it easy to show and download Grafana graphs directly from python code. MFVis requires that the slice has been previously instrumentized by MFLib

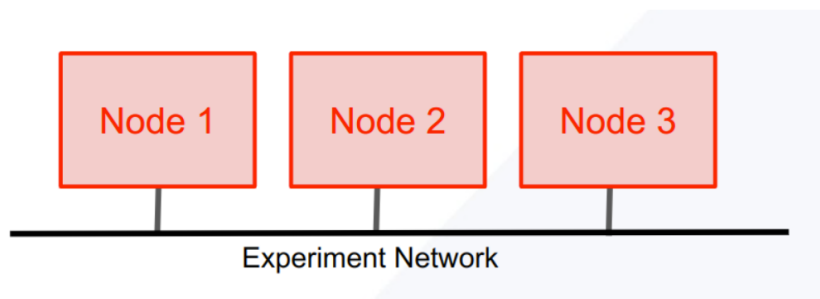
### 4.1 MFLib Methods

MFLib is a python library that enables automatic monitoring of a FABRIC experiment using Prometheus, Grafana and ELK. It can be installed using `pip install fabrictestbed-mflib`. You can also install the latest code by cloning the `fabrictestbed/mflib` source code and following the install instructions. Here are the most common methods you will need for interacting with MFLib. For more details see the class documentation.

#### 4.1.1 Creating a Slice

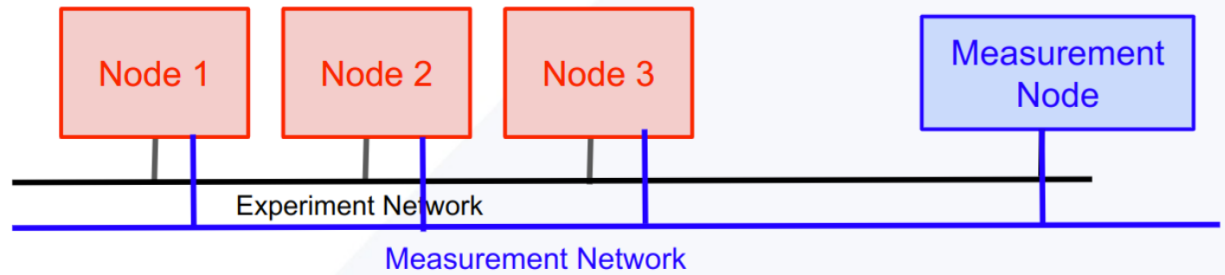
Slice creation is done as you would normally create a slice, but requires an extra step before you submit the slice.

**MFLib().addMeasNode(slice)** where slice is a fabric slice that has been specified but not yet submitted. This example has 3 nodes and an experimental network.



The **MFLib().addMeasNode(slice)** adds an extra node called the Measurement Node (`meas_node`) and an measure-





ment network.

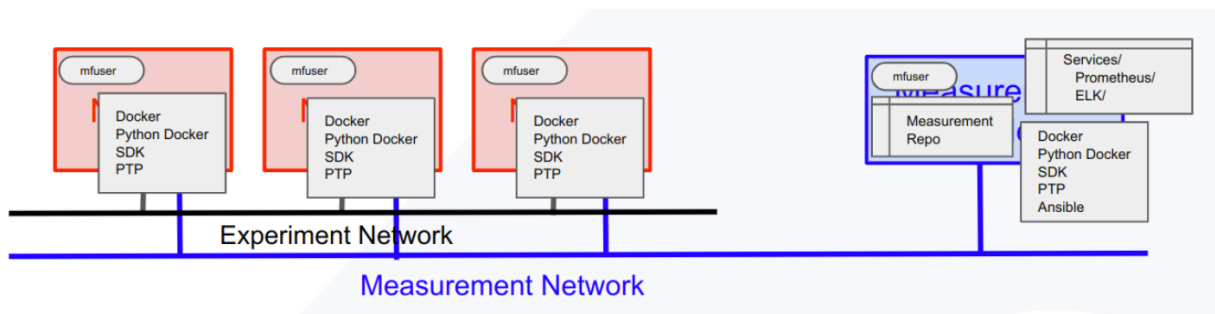
### 4.1.2 Init & Instrumentize

MFLib works on an existing slice to which MFLib must first add some software and services.

**MFLib(slice\_name) mf = MFLib(slice\_name, local\_storage\_directory="/tmp/mflib")** First you must create the MFLib object by passing the slice name to MFLib(). You can optionally pass a string for where you would like the local working files for the slice to be stored. These files include keys, Ansible hosts file, progress and log files and any downloaded files. The default location is in the tmp directory. If you plan on revisiting the slice or the log files later, you should change the directory to a persistent directory of your choosing.

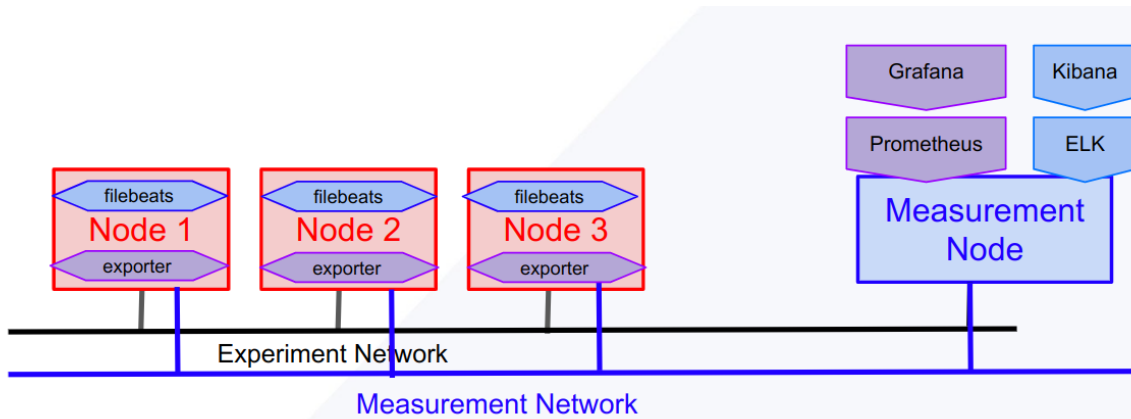
This is called initializing the slice. This process:

- Add mfuser to all the nodes
- Clones MeasurementFramework repository to mfuser account on the Measurement Node
- Creates ansible.ini file for the slice and uploads it to the Measurement Node
- Runs a BASH script on the Measurement Node
- Runs an Ansible script on the Measurement Node
- Sets up Measurement Services that can later be installed
- Ensures Docker & PTP services are running on the experiment's nodes



**MFLib.instrumentize() mf.instrumentize()** MFLib needs to be “instrumentized” to start the monitoring collection. This process will add the systems needed to collect Prometheus metrics and Elastic logs along with Grafana and Kibana for visualizing the collect data. If you only need one of these, or if you want to add other services, include a list of strings naming the services you would like to install. `mf.instrumentize(["prometheus"])` This is called the instrumentizing the slice. This process:

- By default, installs and starts monitoring with:
  - Prometheus & Grafana
  - ELK with Kibana



### 4.1.3 Service Methods

Measurement Framework has “services” which are installed on the measurement node. The `MFLib.instrumentize()` method installs Prometheus, Grafana, and ELK.. MFLib is built on top of `MFLib.Core()`. MFLib uses Core methods to interact with services using several basic methods: `create`, `info`, `update`, `start`, `stop` and `remove`. Each of these methods require the service name. Most can also take an optional dictionary and an optional list of files to be uploaded. All will return a json object with at least a “success” and “msg” values.

**MFLib.create** `mf.create(service, data=None, files=[])` is used to add a service to the slice. By default, Prometheus, ELK, Grafana and Overview services are added during instrumentation.

**MFLib.info** `mf.info(service, data=None)` is used to get information about the service. This will be the most commonly used method. For example Prometheus adds Grafana to the experiment to easily access and visualize the data collected by Prometheus. In order to access Grafana as an admin user, you will need a password. The password can be retrieved using

```
data = {}
data["get"] = ["grafana_admin_password"]
info_results = mf.info("prometheus", data)
print(info_results["grafana_admin_password"])
Info calls should not alter the service in anyway.
```

**MFLib.update** `mf.info(service, data=None, files=[])` is used to update the service configurations or make other changes to the service’s behavior. For example you can add custom dashboards to Grafana using the `grafana_manager`.

```
data = {"dashboard": "add"}
files = ["path_to_dashboard_config.json"]
mf.update("grafana_manager", data, files)
```

**MFLib.stop** `mf.stop(service)` is used to stop a service from using resources. The service is not removed and can be restarted.

**MFLib.start** `mf.start(service)` is used to restart a stopped service.

**mflib.remove** `mf.remove(service)` is used to remove a service. This will stop and remove any artifacts that were installed on the experiment’s nodes.

**mflib.download\_common\_hosts** `mf.download_common_hosts()` retrieves an ansible hosts.ini file for the slice. The hosts file will contain 2 groups: `Experiment_nodes` and `Measurement_node`

**mflib.download\_log\_file** `mf.download_log_file(service, method)` retrieves the log files for runs of the given service's method: create, info, update...etc. This can be useful for debugging.

#### 4.1.4 Accessing Experiment Nodes via Bastion Host

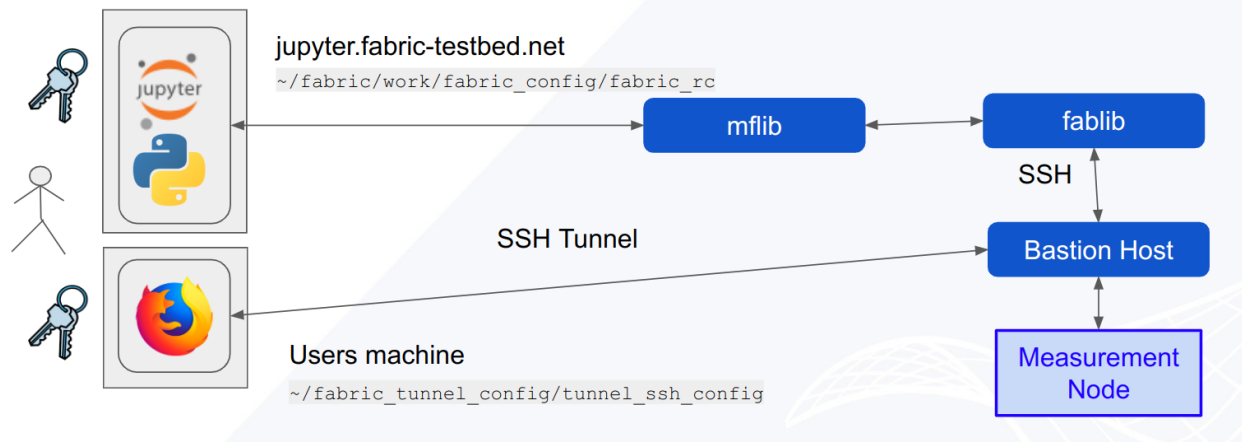
Many of the services set up by MFLib run web accessible user interfaces on the Measurement Node. Since experimental resources are secured by the FABRIC Bastion host, a tunnel must be created to access the web pages.

MFLib has properties, **MFLIB.grafana\_tunnel** & **MFLib.kibana\_tunnel** that will return the needed commands to create tunnels to access the relative service.

## Measurement Node Access

Code Running on JupyterHub

Browsing on Users Machine



## MFLIB

MFLib is the main class that is used to interact with the Measurement Framework set up in a user's FABRIC Experiment.

```
class mflib.mflib.MFLib(slice_name="", local_storage_directory='/tmp/mflib', mf_repo_branch='main',  
                        optimize_repos=False)
```

Bases: [Core](#)

MFLib allows for adding and controlling the MeasurementFramework in a Fabric experimenters slice.

```
static addMeasNode(slice, cores=4, ram=16, disk=500, network_type='FABNetv4', site='NCSA',  
                  image='default_ubuntu_20')
```

Adds Measurement node and measurement network to an unsubmitted slice object.

### Parameters

- **slice** (*fablib.slice*) – Slice object already set with experiment topology.
- **cores** (*int, optional*) – Cores for measurement node. Defaults to 4 cores.
- **ram** (*int, optional*) – *\_description\_*. Defaults to 16 GB ram.
- **disk** (*int, optional*) – *\_description\_*. Defaults to 500 GB disk.
- **network\_type** (*string, optional*) – *\_description\_*. Defaults to FABNetv4.
- **site** (*string, optional*) – *\_description\_*. Defaults to NCSA.

```
add_mflib_log_handler(log_handler)
```

Adds the given log handler to the mflib\_logger. Note log handler needs to be created with set\_mflib\_logger first.

### Parameters

**log\_handler** (*logging handler*) – Log handler to add to the mflib\_logger.

```
download_common_hosts()
```

Downloads hosts.ini file and returns file text. Downloaded hosts.ini file will be stored locally for future reference.

```
init(slice_name, optimize_repos)
```

Sets up the slice to ensure it can be monitored. Sets up basic software on Measurement Node and experiment nodes. Slice must already have a Measurement Node. See log file for details of init output.

### Parameters

**slice\_name** (*str*) – The name of the slice to be monitored.

### Returns

False if no Measure Node found or a init process fails. True otherwise.

**Return type**

Bool

**instrumentize**(*services*=['prometheus', 'elk'])

Instrumentize the slice. This is a convenience method that sets up & starts the monitoring of the slice. Sets up Prometheus, ELK & Grafana.

**Parameters**

**services** (*List of Strings*) – Just add the listed components. Options are elk or prometheus.

**Returns**

The output from each phase of instrumentizing.

**Return type**

dict

**mflib\_class\_version** = '1.0.37'**remove\_mflib\_log\_handler**(*log\_handler*)

Removes the given log handler from the mflib\_logger.

**Parameters**

**log\_handler** (*logging handler*) – Log handler to remove from mflib\_logger

**restore\_DNS**(*node*)**restore\_DNS\_all\_nodes**()

Restores the DNS to default if previously set. See set\_DNS\_all\_nodes.

**Returns**

“restored” if restored, “not needed” if not needed

**Return type**

string

**set\_DNS**(*node*)

Sets the DNS on IPv6 only nodes to enable access to IPv4 sites.

**set\_DNS\_all\_nodes**()

Sets DNS for nodes to allow them to access ipv4 networks.

**Returns**

“set” if DNS set, “not needed” otherwise.

**Return type**

string

**set\_mflib\_logger**()

Sets up the mflib logging file. The filename is created from the self.logging\_filename. Note that the self.logging\_filename will be set with the slice when the slice name is set.

This method uses the logging filename inherited from Core.

## MFLIB CORE

MFLib's Core functions are defined in this class. This class is the base class for all MFLib classes and is not meant to be used directly.

**class** `mflib.core.Core`(*local\_storage\_directory*='/tmp/mflib', *mf\_repo\_branch*='main', *logging\_level*=10)

MFLib core contains the core methods needed to create and interact with the Measurement Framework installed in a slice. It is not intended to be used by itself, but rather, it is the base object for creating Measurement Framework Library objects.

**property** `bootstrap_status_file`

The full path to the local copy of the bootstrap status file.

**Returns**

The full path to the local copy of the bootstrap status file.

**Return type**

String

**property** `common_hosts_file`

The full path to a local copy of the hosts.ini file.

**Returns**

The full path to a local copy of the hosts.ini file.

**Return type**

String

**core\_class\_version** = '1.0.38'

An updatable version for debugging purposes to make sure the correct version of this file is being used. Anyone can update this value as they see fit. Should always be increasing.

**Returns**

Version.sub-version.build

**Return type**

String

**create**(*service*, *data*=None, *files*=[])

Creates a new service for the slice.

**Parameters**

- **service** (*String*) – The name of the service.
- **data** (*JSON serializable object*) –
- **files** (*List of Strings*) – List of filepaths to be uploaded.

**Returns**

Dictionary of creation results.

**Return type**

dict

**download\_log\_file**(*service, method*)

Download the log file for the given service and method. Downloaded file will be stored locally for future reference. :param service: The name of the service. :type service: String :param method: The method name such as create, update, info, start, stop, remove. :type method: String :return: Writes file to local storage and returns text of the log file. :rtype: String

**download\_service\_file**(*service, filename, local\_file\_path=""*)

Downloads service files from the meas node and places them in the local storage directory. Denies the user from downloading files anywhere outside the service directory :param service: Service name :type service: String :param filename: The filename to download from the meas node. :param local\_file\_path: Optional filename for local saved file. :type local\_file\_path: String

**get\_bootstrap\_status**(*force=True*)

Returns the bootstrap status for the slice. Default setting of force will always download the most recent file from the meas node. The downloaded file will be stored locally for future reference at self.bootstrap\_status\_file.

**Parameters**

**force** (*Boolean*) – If downloaded file already exists locally, it will not be downloaded unless force is True. .

**Returns**

Bootstrap dict if any type of bootstrapping has occurred, Empty dict otherwise.

**Return type**

Dictionary

**get\_mfuser\_private\_key**(*force=True*)

Downloads the mfuser private key. Default setting of force will always download the most recent file from the meas node. The downloaded file will be stored locally for future reference at self.local\_mfuser\_private\_key\_filename.

**Parameters**

**force** (*Boolean*) – If downloaded file already exists locally, it will not be downloaded unless force is True.

**Returns**

True if file is found, false otherwise.

**Return type**

Boolean

**property grafana\_tunnel**

Returns the command for createing an SSH tunnel for accesing Grafana.

**Returns**

ssh command

**Return type**

String

**property grafana\_tunnel\_local\_port**

If a tunnel is used for grafana, this value must be set for the port. :returns: port number :rtype: String

**info**(*service*, *data=None*)

Gets information from an existing service. Strictly gets information, does not change how the service is running.

**Parameters**

- **service** (*String*) – The name of the service.
- **data** (*JSON Serializable Object*) – Data to be passed to a JSON file place in the service's meas node directory.

**Returns**

Dictionary of info results.

**Return type**

dict

**property kibana\_tunnel**

Returns the command for creating an SSH tunnel for accessing Kibana.

**Returns**

ssh command

**Return type**

String

**property kibana\_tunnel\_local\_port**

If a tunnel is used for Kibana, this value must be set for the port

**property local\_mfuser\_private\_key\_filename**

The local copy of the private ssh key for the mfuser account.

**Returns**

The local copy of the private ssh key for the mfuser account.

**Return type**

String

**property local\_mfuser\_public\_key\_filename**

The local copy of the public ssh key for the mfuser account.

**Returns**

The local copy of the public ssh key for the mfuser account.

**Return type**

String

**property local\_slice\_directory**

The directory where local files associated with the slice are stored.

**Returns**

The directory where local files associated files are stored.

**Return type**

str

**property log\_directory**

The full path for the log directory.

**Returns**

The full path to the log directory.



**Return type**

String

**property meas\_node**

The fablib node object for the Measurement Node in the slice.

**Returns**

The fablib node object for the Measurement Node in the slice.

**Return type**

fablib.node

**property meas\_node\_ip**

The management ip address for the Measurement Node

**Returns**

ip address

**Return type**

String

**remove(*services=[]*)**

Stops a service running and removes anything setup on the experiment's nodes. Service will then need to be re-created using the create command before service can be started again.

**Parameters**

**services** (*List of Strings*) – The names of the services to be removed.

**Returns**

List of remove result dictionaries.

**Return type**

List

**set\_core\_logger()**

Sets up the core logging file. Note that the self.logging\_filename will be set with the slice name when the slice is set. Args: filename (*\_type\_*, optional): *\_description\_*. Defaults to None.

**property slice\_name**

Returns the name of the slice associated with this object.

**Returns**

The name of the slice.

**Return type**

String

**property slice\_username**

The default username for the Measurement Node for the slice.

**Returns**

username

**Return type**

String

**start(*services=[]*)**

Restarts a stopped service using existing configs on meas node.

**Parameters**

**services** (*List of Strings*) – The name of the services to be restarted.

**Returns**

List of start result dictionaries.

**Return type**

List

**stop**(*services=[]*)

Stops a service, does not remove the service, just stops it from using resources.

**Parameters**

**services** (*List of Strings*) – The names of the services to be stopped.

**Returns**

List of stop result dictionaries.

**Return type**

List

**property tunnel\_host**

If a tunnel is used, this value must be set for the localhost, Otherwise it is set to empty string.

**Returns**

tunnel hostname

**Return type**

String

**update**(*service, data=None, files=[]*)

Updates an existing service for the slice.

**Parameters**

- **service** (*String*) – The name of the service.
- **data** (*JSON Serializable Object*) – Data to be passed to a JSON file place in the service's meas node directory.
- **files** (*List of Strings*) – List of filepaths to be uploaded.

**Returns**

Dictionary of update results.

**Return type**

dict

**upload\_service\_directory**(*service, local\_directory\_path, force=False*)

Uploads the given local directory to the given service's directory on the meas node. :param service: Service name for which the files are being upload to the meas node. :type service: String :param local\_directory\_path: Directory path on local machine. :type local\_directory\_path: String :param force: Whether to overwrite existing directory, if it exists. :type force: Bool :raises: Exception: for misc failures.... :return: ? :rtype: ?

**upload\_service\_files**(*service, files*)

Uploads the given local files to the given service's directory on the meas node. Denies the user from uploading files anywhere outside the service directory :param service: Service name for which the files are being upload to the meas node. :type service: String :param files: List of file paths on local machine. :type files: List :raises: Exception: for misc failures.... :return: ? :rtype: ?

MF MF\_TIMESTAMP

## OWL

**class** `mflib.owl.Owl`(*local\_owl\_dir*, *remote\_out\_dir*, *remote\_conf\_dir=None*)

Parent class

**create\_local\_service\_dirs**()

Create local directories where OWL config and output files will be kept

**create\_remote\_conf\_dir**(*node*)

Creates owl conf directory on a remote node.

**Parameters**

**node** (*fablib.Node*) – remote node

**create\_remote\_out\_dir**(*node*)

Creates owl output directory on a remote node.

**Parameters**

**node** (*fablib.Node*) – remote node

**delete\_remote\_output**(*node*)

Deletes the contents of owl output directory on a remote node.

**Parameters**

**node** (*fablib.Node*) – remote node

**download\_output**(*node*)

Download the contents of owl output directory on a remote node to the local owl output directory.

**Parameters**

**node** (*fablib.Node*) – remote node

**generate\_local\_config**(*send\_int=0.5*, *port=5005*, *cap\_mode='save'*, *pcap\_int=120*)

Generate local copy of owl.conf file (to serve as a master copy)

**Parameters**

- **send\_int** (*float*) – interval at which probe packets will be sent (in seconds)
- **port** (*int*) – port to be used by OWL
- **cap\_mode** (*str*) – capture mode for OWL. Currently only “save” is supported.
- **pcap\_int** (*int*) – time interval (sec) at which tcpdump will start a new pcap file.

**generate\_local\_links\_file**(*links*)

Create a local copy of links.json file.

**Parameters**

**links** (*[ (str, str) ]*) – list of endpoint pairs [(‘src\_ip’, ‘dst\_ip’)]

**get\_local\_service\_file\_paths()**

Get paths for owl.conf and links.json files.

**Returns**

owl.conf path, links.json path

**Return type**

[str, str]

**static list\_experiment\_ip\_addrs(*node*)**

Get a list of IPs for a given remote node, excluding the interfaces used for management and MF networks

**Parameters**

**node** (*fablib.Node*) – remote node

**Returns**

list of IP addresses

**Return type**

List[str]

**list\_remote\_output(*node*)**

List the contents of owl output directory on a remote node.

**Parameters**

**node** (*fablib.Node*) – remote node

**print\_local\_service\_files()**

Prints local copies of owl.conf and links.json.

```
class mflib.owl.OwlDocker(local_owl_dir, remote_out_dir, image_name='fabrictestbed/owl:0.1.3',
                        remote_conf_dir=None)
```

To be used when running OWL (on a non-MF slice) through directly interacting with Docker OWL image/containers.

**static check\_node\_environment(*node*)**

Checks whether remote node has PTP and Docker required for running OWL.

**Parameters**

**node** (*fablib.Node*) – remote node

**get\_owl\_log(*node*, *name*='fabric-owl')**

Print the contents of container log while the container is running.

**Parameters**

- **node** (*fablib.Node*) – remote node
- **name** (*str*) – container name

**pull\_owl\_docker\_image(*node*)**

Pull Docker OWL image to remote node.

**Parameters**

**node** (*fablib.Node*) – remote node

**remove\_owl\_docker\_image(*node*, *name*='fabric-owl')**

Remove container with the name given and OWL Docker image saved on the node.

**Parameters**

- **node** (*fablib.Node*) – remote node

- **name** (*str*) – container name

**start\_owl\_receiver**(*dst\_node, receiving\_ip, name='fabric-owl', pcap\_time\_limit=360, duration=180*)

Start Docker container to run OWL receiver.

#### Parameters

- **dst\_node** (*fablib.Node*) – OWL destination node
- **receiving\_ip** (*str*) – IPv4 address of dst node where OWL packets are expected.
- **name** (*str*) – container name
- **pcap\_time\_limit** (*int*) – time interval (sec) at which tcpdump starts a new pcap file.
- **duration** (*int*) – duration (sec) OWL should run

**start\_owl\_sender**(*src\_node, dst\_ip, name='fabric-owl', frequency=0.5, seq\_n=1234, duration=180, python\_time=False*)

Start Docker container to run OWL sender.

#### Parameters

- **src\_node** (*fablib.Node*) – OWL source node
- **dst\_ip** (*str*) – destination IPv4 address
- **name** (*str*) – container name
- **frequency** (*float*) – interval (sec) at which probe packets should be sent
- **duration** (*int*) – duration (sec) OWL should run
- **python\_time** (*bool*) – True = timestamp obtained through Python `time.time_ns()`; False = timestamp obtained through OWL ptp timestamp script

**stop\_owl\_docker**(*node, name='fabric-owl'*)

Stop container.

#### Parameters

- **node** (*fablib.Node*) – remote node
- **name** (*str*) – container name

**class** `mflib.owl.OwlMf`(*local\_owl\_dir*)

To be used when interacting with OWL on an MF slice with a measurement node. Initializes with the default MF paths for remote conf and out directories.

**OWL DATA**

## PYTHON MODULE INDEX

### m

- `mflib.core`, [12](#)
- `mflib.mf_timestamp.mf_timestamp`, [17](#)
- `mflib.mflib`, [10](#)
- `mflib.owl`, [18](#)
- `mflib.owl_data.OwlDataAnalyzer`, [21](#)



## A

`add_mflib_log_handler()` (*mflib.mflib.MFLib method*), 10  
`addMeasNode()` (*mflib.mflib.MFLib static method*), 10

## B

`bootstrap_status_file` (*mflib.core.Core property*), 12

## C

`check_node_environment()` (*mflib.owl.OwlDocker static method*), 19  
`common_hosts_file` (*mflib.core.Core property*), 12  
`Core` (*class in mflib.core*), 12  
`core_class_version` (*mflib.core.Core attribute*), 12  
`create()` (*mflib.core.Core method*), 12  
`create_local_service_dirs()` (*mflib.owl.Owl method*), 18  
`create_remote_conf_dir()` (*mflib.owl.Owl method*), 18  
`create_remote_out_dir()` (*mflib.owl.Owl method*), 18

## D

`delete_remote_output()` (*mflib.owl.Owl method*), 18  
`download_common_hosts()` (*mflib.mflib.MFLib method*), 10  
`download_log_file()` (*mflib.core.Core method*), 13  
`download_output()` (*mflib.owl.Owl method*), 18  
`download_service_file()` (*mflib.core.Core method*), 13

## G

`generate_local_config()` (*mflib.owl.Owl method*), 18  
`generate_local_links_file()` (*mflib.owl.Owl method*), 18  
`get_bootstrap_status()` (*mflib.core.Core method*), 13  
`get_local_service_file_paths()` (*mflib.owl.Owl method*), 19

`get_mfuser_private_key()` (*mflib.core.Core method*), 13  
`get_owl_log()` (*mflib.owl.OwlDocker method*), 19  
`grafana_tunnel` (*mflib.core.Core property*), 13  
`grafana_tunnel_local_port` (*mflib.core.Core property*), 13

## I

`info()` (*mflib.core.Core method*), 13  
`init()` (*mflib.mflib.MFLib method*), 10  
`instrumentize()` (*mflib.mflib.MFLib method*), 11

## K

`kibana_tunnel` (*mflib.core.Core property*), 14  
`kibana_tunnel_local_port` (*mflib.core.Core property*), 14

## L

`list_experiment_ip_addrs()` (*mflib.owl.Owl static method*), 19  
`list_remote_output()` (*mflib.owl.Owl method*), 19  
`local_mfuser_private_key_filename` (*mflib.core.Core property*), 14  
`local_mfuser_public_key_filename` (*mflib.core.Core property*), 14  
`local_slice_directory` (*mflib.core.Core property*), 14  
`log_directory` (*mflib.core.Core property*), 14

## M

`meas_node` (*mflib.core.Core property*), 15  
`meas_node_ip` (*mflib.core.Core property*), 15  
`MFLib` (*class in mflib.mflib*), 10  
`mflib.core` module, 12  
`mflib.mf_timestamp.mf_timestamp` module, 17  
`mflib.mflib` module, 10  
`mflib.owl` module, 18  
`mflib.owl_data.OwlDataAnalyzer`

module, 21  
 mflib\_class\_version (mflib.mflib.MFLib attribute), 11  
 module  
   mflib.core, 12  
   mflib.mf\_timestamp.mf\_timestamp, 17  
   mflib.mflib, 10  
   mflib.owl, 18  
   mflib.owl\_data.OwlDataAnalyzer, 21  
 upload\_service\_files() (mflib.core.Core method), 16

## O

Owl (class in mflib.owl), 18  
 OwlDocker (class in mflib.owl), 19  
 OwlMf (class in mflib.owl), 20

## P

print\_local\_service\_files() (mflib.owl.Owl method), 19  
 pull\_owl\_docker\_image() (mflib.owl.OwlDocker method), 19

## R

remove() (mflib.core.Core method), 15  
 remove\_mflib\_log\_handler() (mflib.mflib.MFLib method), 11  
 remove\_owl\_docker\_image() (mflib.owl.OwlDocker method), 19  
 restore\_DNS() (mflib.mflib.MFLib method), 11  
 restore\_DNS\_all\_nodes() (mflib.mflib.MFLib method), 11

## S

set\_core\_logger() (mflib.core.Core method), 15  
 set\_DNS() (mflib.mflib.MFLib method), 11  
 set\_DNS\_all\_nodes() (mflib.mflib.MFLib method), 11  
 set\_mflib\_logger() (mflib.mflib.MFLib method), 11  
 slice\_name (mflib.core.Core property), 15  
 slice\_username (mflib.core.Core property), 15  
 start() (mflib.core.Core method), 15  
 start\_owl\_receiver() (mflib.owl.OwlDocker method), 20  
 start\_owl\_sender() (mflib.owl.OwlDocker method), 20  
 stop() (mflib.core.Core method), 16  
 stop\_owl\_docker() (mflib.owl.OwlDocker method), 20

## T

tunnel\_host (mflib.core.Core property), 16

## U

update() (mflib.core.Core method), 16  
 upload\_service\_directory() (mflib.core.Core method), 16