
MFLib

Release 0.1.0b1

Fabric UKY Team

Apr 17, 2023

CONTENTS

1	Documentation Resources	2
1.1	Example Jupyter Notebooks	2
1.2	FABRIC Learn Site	2
1.3	MFLib Python Package Documentation	2
2	MFLib Installation	3
2.1	Instaling via PIP	3
2.2	Installing via Source Code	3
3	Building & Deploying	4
3.1	Spinx Documentation	4
3.2	Distribution Package	5
4	MFLib Methods	6
4.1	Creating a Slice	6
4.2	Init & Instrumentize	7
4.3	Service Methods	8
4.4	Accessing Experiment Nodes via Bastion Host	9
5	MFLib	10
6	MFLib Core	12
7	MF mf_timestamp	17
8	OWL	18
9	OWL Data	21
	Python Module Index	22
	Index	23

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

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 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

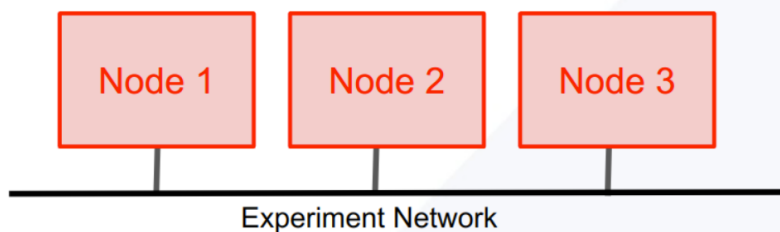
MFLIB METHODS

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.

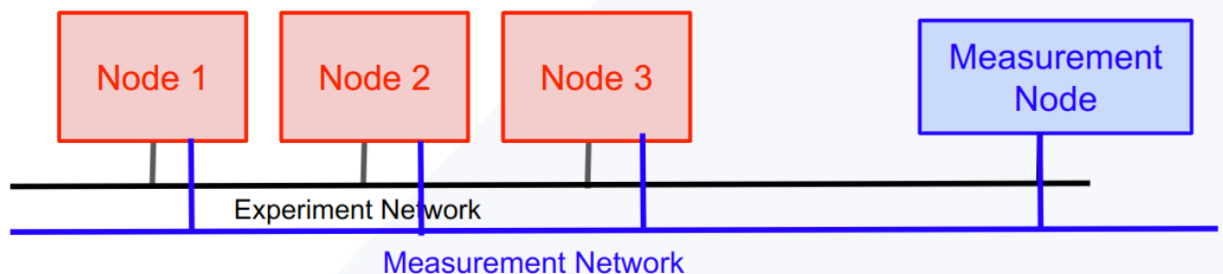
4.1 Creating a Slice

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.

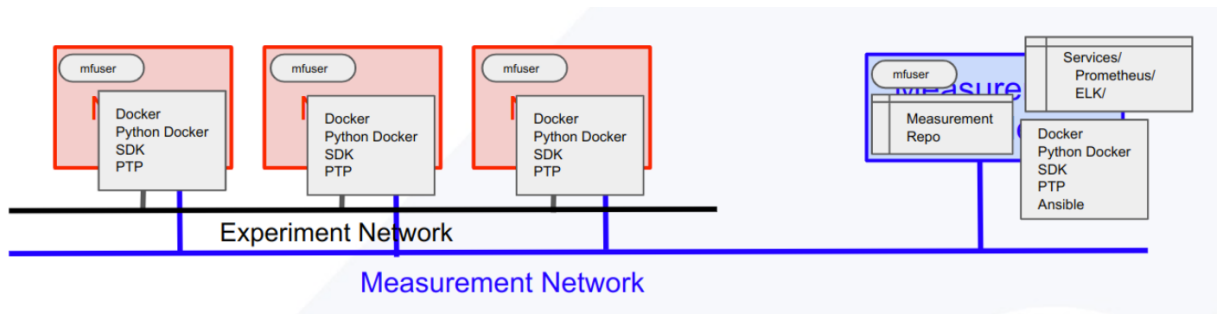
4.2 Init & Instrumentize

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.

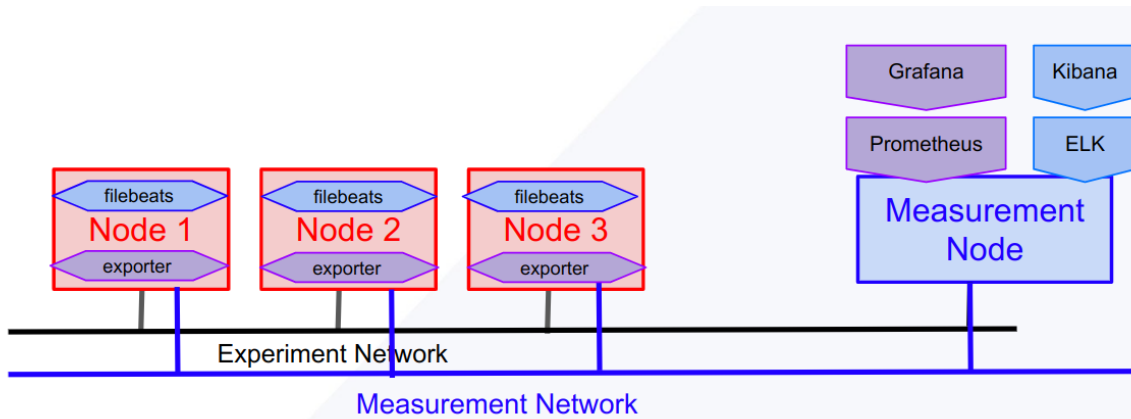
This is called initializing the slice. This process:

- 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. `mv.instrumentize(["prometheus"])` This is called the instrumentizing the slice. This process:

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



4.3 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.

4.4 Accessing Experiment Nodes via Bastion Host

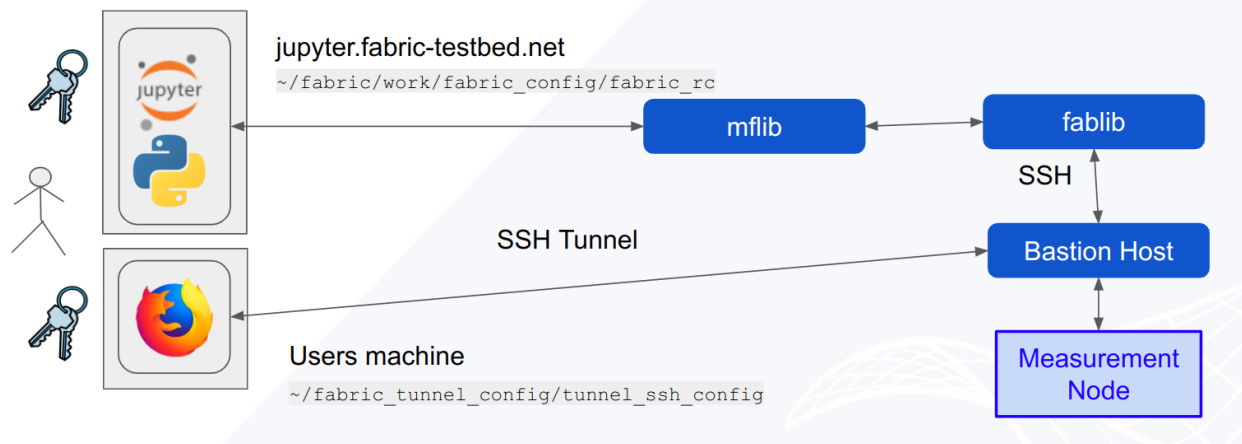
Many of the services set up by MFLib run web accessible user interfaces on the Measurement Node. Since experimental resources are secured by the FABRIC Bastion host, a tunnel must be created to access the web pages.

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 = '1.0.37'

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

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 inormation 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 createing an SSH tunnel for accesing 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

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`, [17](#)

`mflib.mflib`, [10](#)

`mflib.owl`, [18](#)

INDEX

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

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*), 14
`meas_node_ip` (*mflib.core.Core property*), 15
`MFLib` (*class in mflib.mflib*), 10
`mflib.core`
 module, 12
`mflib.mf_timestamp`
 module, 17
`mflib.mflib`
 module, 10
`mflib.owl`
 module, 18
`mflib_class_version` (*mflib.mflib.MFLib attribute*), 11
`module`
 mflib.core, 12

mflib.mf_timestamp, 17
 mflib.mflib, 10
 mflib.owl, 18

O

Owl (*class in mflib.owl*), 18
 OwlDock (*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.OwlDock method*), 19

R

remove() (*mflib.core.Core method*), 15
 remove_mflib_log_handler() (*mflib.mflib.MFLib method*), 11
 remove_owl_docker_image() (*mflib.owl.OwlDock 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.OwlDock method*), 20
 start_owl_sender() (*mflib.owl.OwlDock method*), 20
 stop() (*mflib.core.Core method*), 15
 stop_owl_docker() (*mflib.owl.OwlDock method*), 20

T

tunnel_host (*mflib.core.Core property*), 16

U

update() (*mflib.core.Core method*), 16