MFLib

Release 1.0.7

Fabric UKY Team

Nov 15, 2025

CONTENTS

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.

CHAPTER

ONE

DOCUMENTATION RESOURCES

For more information about FABRIC vist 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 serveral 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.

CHAPTER

TWO

MFLIB INSTALLATION

2.1 Instaling 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 .
```

CHAPTER

THREE

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 distributition.

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
```

Note that if the package has already been published to testpypi, it cannot be published again until the version is updated. There is not an obvious error for this. Instead you will just get the following error:

```
requests.exceptions.HTTPError: 400 Client Error: Bad Request for url: https://test.pypi.
↪org/legacy/
```

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.

CHAPTER

FOUR

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 higer-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 instumentized by MFLib

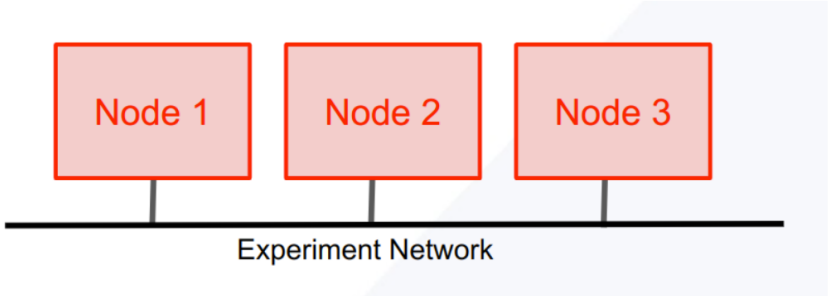
4.1 MFLib Methods

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.

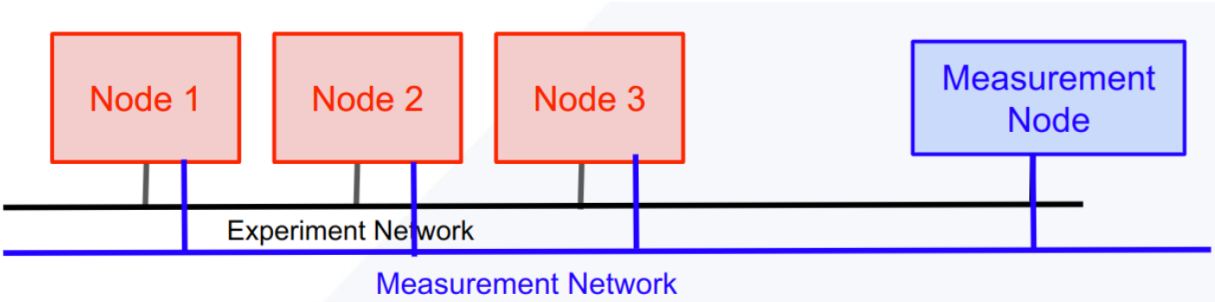
4.1.1 Creating a Slice

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 ndoes and an experimental network.



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



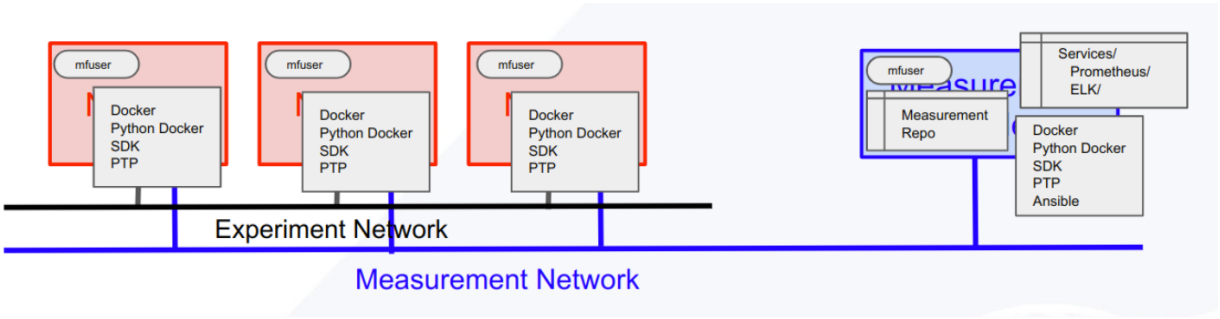
ment network.

4.1.2 Init & Instrumentize

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

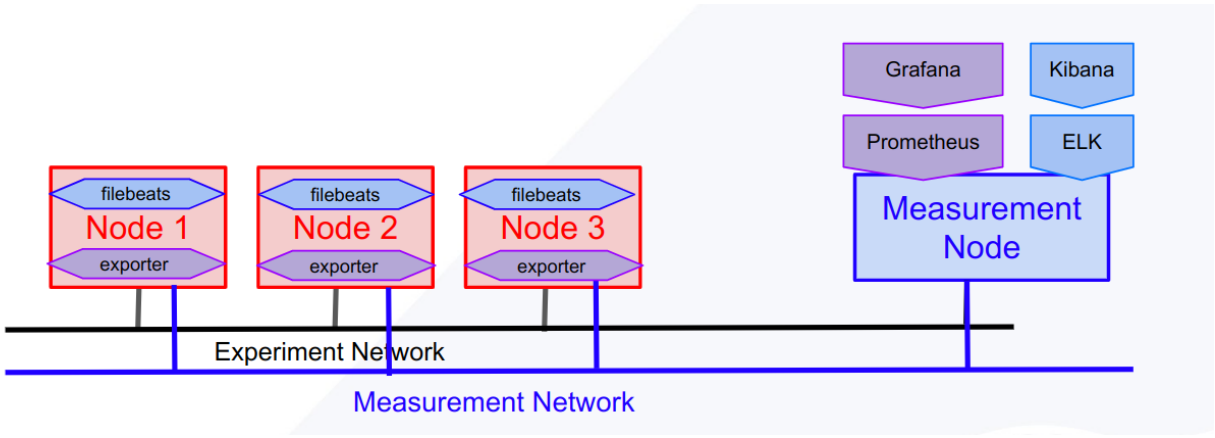
MFLib(slice_name) `mf = MFLib(slice_name, local_storage_directory="/tmp/mflib")` First you must create the MFLib object by passing the slice name to MFLib() . You can optionally pass a string for where you would like the local working files for the slice to be stored. These files include keys, Ansible 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. mv.instrumentize(["prometheus"]) This is called the instrumentizing the slice. This process:

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



4.1.3 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.

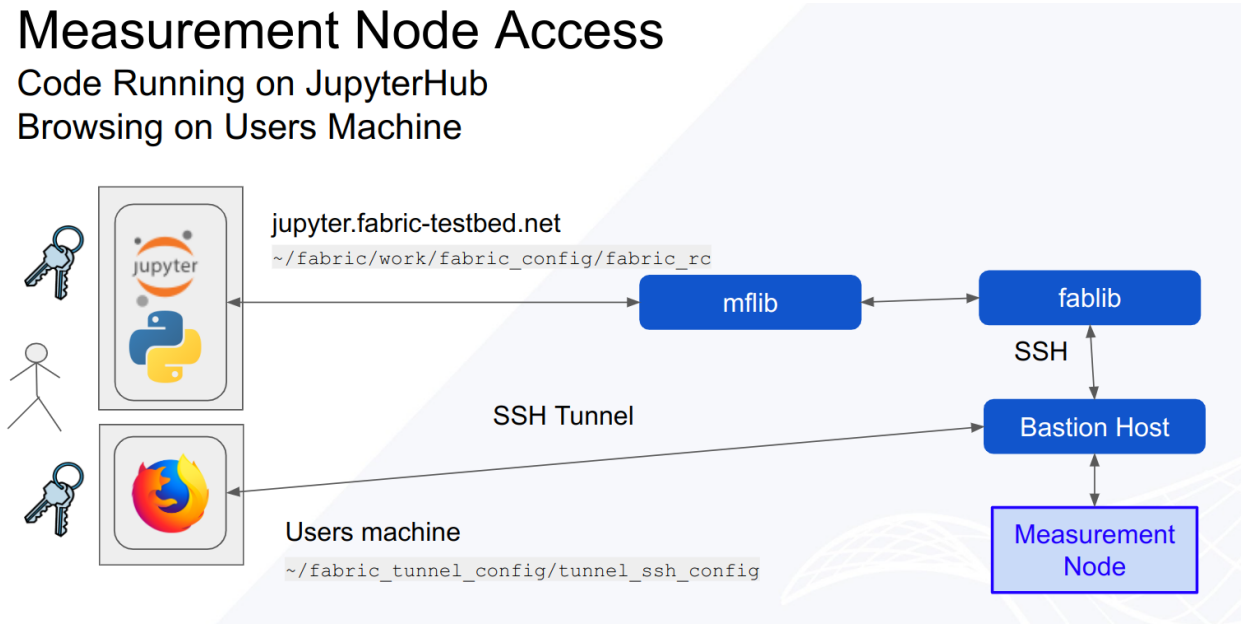
mflib.download_common_hosts `mf.download_common_hosts()` retrieves an ansible hosts.ini file for the slice. The hosts file will contain 2 groups: Experiment_nodes and Measurement_node

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.



CHAPTER

FIVE

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 experiementers slice.

static addMeasNode(*slice*, *cores=4*, *ram=16*, *disk=500*, *site='EDC'*, *image='docker_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 instrumetizing.

Return type
dict

`mflib_class_version = '1.0.40'`

`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

`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.

CHAPTER

SIX

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 tp the local copy of the bootsrap 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 bootstraping has occured, 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

`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: ?

CHAPTER

SEVEN

MFLIB MF_TIMESTAMP

MFLib's Timestamp functions are defined in this class.

class mflib.mf_timestamp.**mf_timestamp**(*slice_name*, *container_name*)

Creates precision timestamps.

deploy_influxdb_dashboard(*dashboard_file*, *influxdb_node_name*, *bind_mount_volume*)

Uploads a dashboard template file from Jupyterhub to influxdb and apply the template :param dashboard_file: path of the dashboard file on Jupyterhub :type dashboard_file: str :param influxdb_node_name: which fabric node is influxdb running on :type influxdb_node_name: str :param bind_mount_volume: bind mount volume of the influxdb docker container :type bind_mount_volume: str

download_file_from_influxdb(*data_node*, *data_type*, *influxdb_node_name*, *local_file*)

Downlaods the .csv data file from the influxdb container to juputerhub :param data_node: where the data comes from :type data_node: str :param data_type: packet_timestamp or event timestamp :type data_type: str :param influxdb_node_name: which fabric node is influxdb running on :type influxdb_node_name: str :param local_file: path on Jupyterhub for the downlaod .csv file :type local_file: str

download_timestamp_file(*node*, *data_type*, *local_file*, *bind_mount_volume*)

Downloads the collected timestamp file to Jupyterhub Use fablib node.download_file() to download the timestamp data file that can be accessed from the bind mount volume :param node: fabric node on which the timestamp docker container is running :type node: fablib.node :param data_type: packet_timestamp or event timestamp :type data_type: str :param local_file: path on Jupyterhub for the download file :type local_file: str :param bind_mount_volume: bind mount volume of the running timestamp docker container :type bind_mount_volume: str

download_timestamp_from_influxdb(*node*, *data_type*, *bucket*, *org*, *token*, *name*, *influxdb_ip=None*)

Downloads the timestamp data from influxdb :param node: fabric node on which the timestamp docker container is running :type node: fablib.node :param data_type: packet_timestamp or event timestamp :type data_type: str :param bucket: name of the influxdb bucket to dump data to :type bucket: str :param org: org of the bucket :type org: str :param token: token of the bucket :type token: str :param name: name of the timestamp experiment :type name: str :param influxdb_ip: the IP address of the node where the influxdb container is running :type influxdb_ip: str, optional

get_event_timestamp(*node*, *name*, *verbose=False*)

Prints the collected event timestamp by calling timestamptool.py in the timestamp docker container running on node It reads the local event timestamp file and prints the result :param node: fabric node on which the timestamp docker container is running :type node: fablib.node :param name: name of the event timestamp experiment :type name: str

get_packet_timestamp(*node*, *name*, *verbose=False*)

Prints the collected packet timestamp by calling timestamptool.py in the timestamp docker container running on the node It reads the local packet timestamp file and prints the result :param node: fabric node on

which the timestamp docker container is running :type node: fablib.node :param name: name of the packet timestamp experiment :type name: str

get_query_for_csv(*data_node*, *name*, *data_type*, *bucket*, *org*, *token*)

Generates a .csv file in the influxdb container for the query data :param data_node: where the data comes from :type data_node: str :param name: name of the timestamp experiment :type name: str :param data_type: packet_timestamp or event timestamp :type data_type: str :param bucket: name of the influxdb bucket to dump data to :type bucket: str :param org: org of the bucket :type org: str :param token: token of the bucket :type token: str :param influxdb_node_name: which fabric node is influxdb running on :type influxdb_node_name: str

plot_event_timestamp(*json_obj*)

Plots the count of events :param json_obj: list of json objects with timestamp info :type json_obj: list

plot_packet_timestamp(*json_obj*)

Plots the count of packets captured :param json_obj: list of json objects with timestamp info :type json_obj: list

read_from_local_file(*file*)

Reads and processes the downloaded timestamp data file :param file: file path on Jupyterhub for the final timestamp result :type file: str

record_event_timestamp(*node*, *name*, *event*, *description=None*, *verbose=False*)

Records event timestamp by calling timestamptool.py in the timestamp docker container running on the node :param node: fabric node on which the timestamp docker container is running :type node: fablib.node :param name: name of the event timestamp experiment :type name: str :param event: name of the event :type event: str :param description: description of the event :type description: str, optional

record_packet_timestamp(*node*, *name*, *interface*, *ipversion*, *protocol*, *duration*, *host=None*, *port=None*, *verbose=False*)

Records packet timestamp by calling timestamptool.py in the timestamp docker container running on the node :param node: fabric node on which the timestamp docker container is running :type node: fablib.node :param name: name of the packet timestamp experiment :type name: str :param interface: which interface tcpdump captures the packets :type interface: str :param ipversion: IPv4 or IPv6 :type ipversion: str :param protocol: tcp or udp :type protocol: str :param duration: seconds to run tcpdump :type duration: str :param host: host for tcpdump command :type host: str, optional :param port: port for tcpdump command :type port: str, optional

upload_timestamp_to_influxdb(*node*, *data_type*, *bucket*, *org*, *token*, *influxdb_ip=None*)

Uploads the timestamp data to influxdb :param node: fabric node on which the timestamp docker container is running :type node: fablib.node :param data_type: packet_timestamp or event timestamp :type data_type: str :param bucket: name of the influxdb bucket to dump data to :type bucket: str :param org: org of the bucket :type org: str :param token: token of the bucket :type token: str :param influxdb_ip: IP of the node where influxdb container is running :type influxdb_ip: str, optional

CHAPTER

EIGHT

OWL

mflib.owl.**check_owl_all**(*slice*)

Prints the list of all running containers on all nodes in the slice.

Parameters
slice (*fablib.Slice*)

mflib.owl.**check_owl_prerequisites**(*slice*)

Checks whether remote nodes have PTP and Docker required for running OWL.

Parameters
slice (*fablib.Slice*)

mflib.owl.**download_output**(*node*, *local_out_dir*)

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

Parameters

- **node** (*fablib.Node*) – remote node
- **local_out_dir** (*str*) – /path/to/local/dir/under/which/pcaps/will/be/saved

mflib.owl.**get_node_ip_addr**(*slice*, *node_name*)

Get the node (named 'node_name') experiment IP address. This IP is useful because the pcap file to send to InfluxDB is stored in the format of the sender's IP address ("${node_ip}.pcap").

Parameters

- **slice** (*fablib.Slice*)
- **node_name** (*str*)

Returns
IP addresses

Return type
str

mflib.owl.**nodes_ip_addrs**(*slice*)

List the node experiment IP address for all nodes. This is particularly useful when using a (Measurement Framework) meas-node + meas_net since the returned dictionary will NOT include the meas_net addresses. It assumes each node has only one experiment interface.

Parameters
slice (*fablib.Slice*)

Returns
list of IP addresses

Return type
List[str]

mflib.owl.**pull_owl_docker_image**(*node*, *image_name*)

Pull Docker OWL image to remote node.

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

mflib.owl.**send_to_influxdb**(*node*, *pcapfile*, *img_name*, *influxdb_token=None*, *influxdb_org=None*, *influxdb_url=None*, *influxdb_bucket=None*, *desttype='local'*)

Send OWL pcap data to InfluxDB in a remote server. Invokes OWL container to call parse_and_send() in MF's sock_ops/send_data.py.

Parameters

- **node** (*fablib.Node*) – fablib Node object where pcap data will be sent to InfluxDB.
- **pcapfile** (*str*) – Packet Capture file name.
- **img_name** (*str*) – OWL Docker image name.
- **influxdb_token** (*str*) – InfluxDB token string.
- **influxdb_org** (*str*) – InfluxDB org name.
- **influxdb_url** (*str*) – InfluxDB URL (if IP address, omit http and port; just have the IP address).
- **desttype** – Destination type, or where InfluxDB server lives; "cloud" or "meas_node".
- **influxdb_bucket** (*str*) – InfluxDB bucket ID.

mflib.owl.**start_owl**(*slice*, *src_node*, *dst_node*, *img_name*, *probe_freq=1*, *no_ptp=False*, *outfile=None*, *duration=None*, *delete_previous_output=True*, *src_addr=None*, *dst_addr=None*)

Start OWL on a given link defined by source and destination nodes.

Parameters

- **slice** (*fablib.Slice*)
- **src_node** (*fablib.Node*) – source (sender) node
- **dst_node** (*fablib.Node*) – destination (capturer) node
- **img_name** (*str*) – Docker image name
- **prob_freq** (*int*) – (default=1) interval (sec) at which probe packets are sent
- **duration** (*int*) – (default=None) how long (sec) to run OWL
- **no_ptp** (*bool*) – (default=False) Set this to True only when testing the functionalities on non-PTP node
- **outfile** (*str*) – /path/on/remote/node/to/ouput.pcap (if None, */home/rocky/owl-output/{dst_ip}.pcap*)
- **delete_previous** (*bool*) – (default=True) whether to delete all existing pcap files from previous runs

Src_addr
(default=None) Specify source IP address only if source node has more than 1 experiment interface

Dst_addr
(default=None) Specify destination IP address only if dest node has more than 1 experiment interface

mflib.owl.**start_owl_all**(*slice*, *img_name*, *probe_freq=1*, *outfile=None*, *duration=None*, *no_ptp=False*, *delete_previous=True*)

Start OWL on all possible combination of nodes in the slice. It asssumes there is only 1 experimenter's network interface on each node (exclu. meas-net interface)

Parameters

- **slice** (*fablib.Slice*)
- **src_node** (*fablib.Node*) – source (sender) node
- **dst_node** (*fablib.Node*) – destination (capturer) node
- **img_name** (*str*) – Docker image name
- **prob_freq** (*int*) – (default=1) interval (sec) at which probe packets are sent
- **outfile** (*str*) – /path/on/remote/node/to/ouput.pcap (if None, */home/rocky/owl-output/{dst_ip}.pcap*)
- **duration** (*int*) – (default=None) how long (sec) to run OWL
- **no_ptp** (*bool*) – (default=False) Set this to True only when testing the functionalities on non-PTP node
- **delete_previous** (*bool*) – (default=True) whether to delete all existing pcap files from previous runs

mflib.owl.**start_owl_capturer**(*slice*, *dst_node*, *img_name*, *outfile=None*, *duration=None*, *delete_previous=True*, *dst_addr=None*, *verbose=True*)

Start OWL capturer inside a Docker container on a remote node by running owl_capturer.py. One container instance per node (sometimes serving multiple source nodes). Docker container name will be in the form of "owl-capturer_10.0.0.1"

Parameters

- **slice** (*fablib.Slice*)
- **dst_node** – destination (capturer) node
- **img_name** (*str*) – Docker image name
- **outfile** (*str*) – /path/on/remote/node/to/ouput.pcap (if None, */home/rocky/owl-output/{dst_ip}.pcap*)
- **duration** (*int*) – (default=None) how long (sec) to run OWL
- **delete_previous** (*bool*) – (default=True) whether to delete all existing pcap files from previous runs

Dst_addr
(default=None) Specify destination IP address only if dest node has more than 1 experiment interface

Verbose
(default=False) if True, prints docker run command

mflib.owl.**start_owl_sender**(*slice*, *src_node*, *dst_node*, *img_name*, *probe_freq=1*, *duration=None*, *no_ptp=False*, *src_addr=None*, *dst_addr=None*, *verbose=True*)

Start OWL sender inside a Docker container on a remote node by running owl_sender.py. Docker container name will be in the form of "owl-sender_10.0.0.1-10.0.1.1"

Parameters

- **slice** (*fablib.Slice*)
- **src_node** (*fablib.Node*) – source (sender) node
- **dst_node** (*fablib.Node*) – destination (capturer) node
- **img_name** (*str*) – Docker image name
- **prob_freq** (*int*) – (default=1) interval (sec) at which probe packets are sent
- **duration** (*int*) – (default=None) how long (sec) to run OWL
- **no_ptp** (*bool*) – (default=False) Set this to True only when testing the functionalities on non-PTP node

Src_addr
(default=None) Specify source IP address only if source node has more than 1 experiment interface

Dst_addr
(default=None) Specify destination IP address only if dest node has more than 1 experiment interface

Verbose
(default=True) if True, prints docker run command

mflib.owl.**stop_influxdb_sender**(*dst_node*)

Stops OWL InfluxDB container if there is one running on a remote node.

Parameters
dst_node (*fablib.Node*) – destination (capturer) node

mflib.owl.**stop_owl_all**(*slice*)

Stop ALL running instances of OWL containers on all nodes in the slice

Parameters
slice (*fablib.Slice*)

mflib.owl.**stop_owl_capturer**(*slice*, *dst_node*, *dst_addr=None*)

Stops OWL capturer container if there is one running on a remote node.

Parameters

- **slice** (*fablib.Slice*)
- **dst_node** (*fablib.Node*) – destination (capturer) node

Dst_addr
(default=None) Specify destination IP address only if dest node has more than 1 experiment interface

mflib.owl.**stop_owl_sender**(*slice*, *src_node*, *dst_node*, *src_addr=None*, *dst_addr=None*)

Stops OWL sender container if there is one or more running on a remote node.

Parameters

- **slice** (*fablib.Slice*)
- **src_node** (*fablib.Node*) – source (sender) node

- **dst_node** (*fablib.Node*) – destination (capturer) node

Src_addr
(default=None) Specify source IP address only if source node has more than 1 experiment interface

Dst_addr
(default=None) Specify destination IP address only if dest node has more than 1 experiment interface

CHAPTER

NINE

OWL DATA

mflib.owl_data.**convert_pcap_to_csv**(*pcap_files*, *outfile='out.csv'*, *append_csv=False*, *verbose=False*)

Extract data from the list of pcap files and write to one csv file.

Parameters

- **pcap_files** (*[posix.Path]*) – list of pcap file paths
- **outfile** (*str*) – name of csv file
- **append_csv** (*bool*) – whether to append data to an existing csv file of that name
- **verbose** (*bool*) – if True, prints each line as it is appended to csv

mflib.owl_data.**convert_to_df**(*owl_csv*)

Convert CSV output from the method above to Panda Dataframe

Parameters

owl_csv (*str*) – path/to/csv/file

mflib.owl_data.**filter_data**(*df*, *src_ip*, *dst_ip*)

Filter data by source and destination IPs

Parameters

- **df** (*Panda Dataframe*) – latency data
- **src[dst]_ip** (*str*) – Source and destination IPv4 addresses

mflib.owl_data.**get_summary**(*df*, *src_node*, *dst_node*, *src_ip=None*, *dst_ip=None*)

Print summary of latency data collected between source and destination nodes

Parameters

- **df** (*Panda Dataframe*) – latency data
- **src[dst]_node** – source/destination nodes

:type src[dst]_node:fablib.Node :param src[dst]_ip: needed only if there are multiple experimenter IP interfaces :type src[dst]_ip: str

mflib.owl_data.**graph_latency_data**(*df*, *src_node*, *dst_node*, *src_ip=None*, *dst_ip=None*)

Graph latency data collected between source and destination nodes

Parameters

- **df** (*Panda Dataframe*) – latency data
- **src[dst]_node** – source/destination nodes

:type src[dst]_node:fablib.Node :param src[dst]_ip: needed only if there are multiple experimenter IP interfaces
:type src[dst]_ip: str

mflib.owl_data.**list_experiment_ip_addrs**(*node*)

Get experimenter IPv4 addresses for each node.

Parameters
node (*fablib.Node*) – Node on which IPv4 address is queried.

Returns
a list of of IPv4 addresses assigned to node interfaces

Return type
[ipaddress.IPv4Address,]

mflib.owl_data.**list_pcap_files**(*root_dir*)

Search recursively for pcap files under root_dir

Parameters
root_dir (*str*) – Directory that will be treated as root for this search

Return files_list
absolute paths for all the *.pcap files under the root_dir

Return type
[posix.Path]

PYTHON MODULE INDEX

m

INDEX