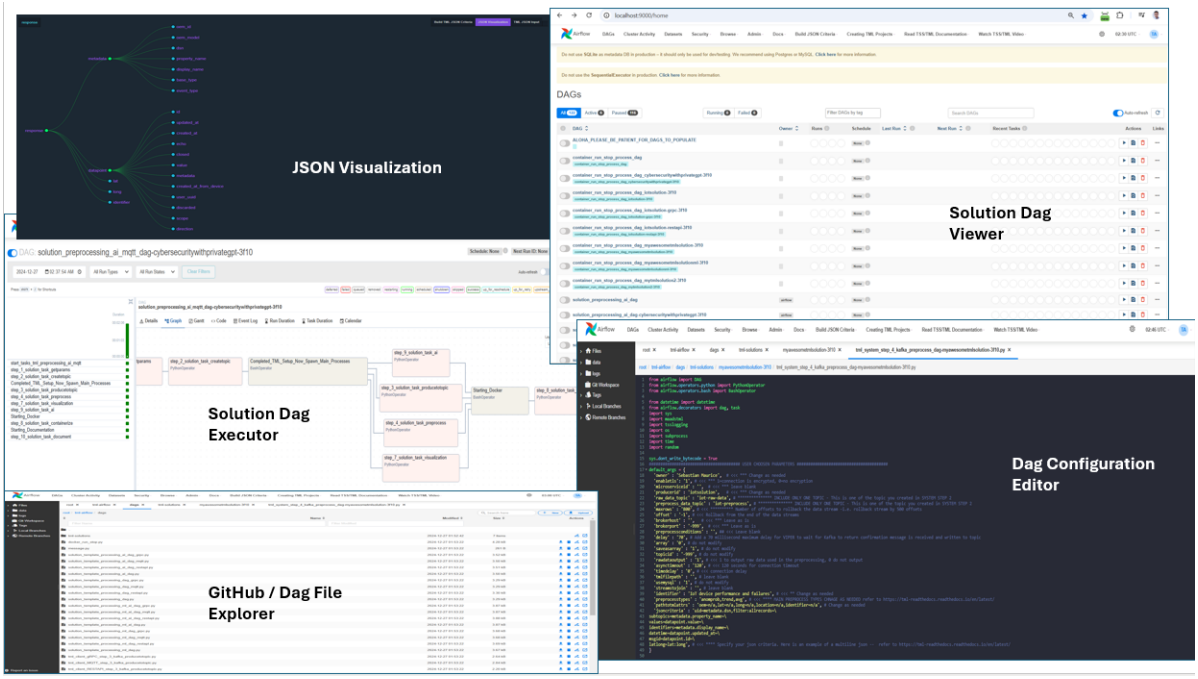
Welcome to Transactional Machine Learning (TML) and TML Solution Studio (TSS) Documentation!

Building Real-Time Solutions to Process the World's Real-Time data with TML and Multi-Agentic AI

TSS is a revolutionary platform for building, enterprise-ready, process-driven, advanced, scalable, intelligent real-time TML solutions **with NO CODE** using :ref:`Pre-Written 10 Apache Airflow DAGs To Speed Up TML Solution Builds` - within minutes - what can take organizations up to 6 months to build.

IDC estimates that 30% of global data will be real-time data by 2025 - so learning how to process real-time data with TML is becoming critical.

Here are the TSS Screens:



TML solutions can process any real-time data from devices and systems. The real-time data are ingested using RESTful, gRPC, MQTT APIs or localfile on the file system.

> Important
>
> **All TML solutions are containerized with Docker (scale with Kubernetes), come with automated documentation about the solution, automated commits to Github for real-time logging, and automated real-time dashboard for visualization using websockets.**

TML solutions can be deployed on the cloud (**with Internet**) or on-premise (**without Internet**) to process unlimited real-time data globally. TML solutions can run on any cloud infrastructure - anywhere.

Watch The TSS Youtube Videos

Video Name	YouTube URL
TSS Video:	Youtube Video

Managing TML Projects (Creating, deleting, copying and stopping):	Youtube Video See details here :ref:`Lets Start Building a TML Solution`
Scaling TML Solutions with Kubernetes	Youtube Video See details here :ref:`Scaling TML Solutions with Kubernetes`
Storing Secure Passwords in Kubernetes	Youtube Video Describes how to store secure passwords in Kubernetes to make your TML solutions more secure and in compliance with IT security standards.
Running TML Projects in TSS:	Youtube Video Describes how to run a TML project. See details here :ref:`Lets Start Building a TML Solution`
Ingesting Real-Time Data into TML Solutions:	Youtube Video Describes how to ingest real-time data using: MQTT, RESTful API, gRPC API or Local file. See here :ref:`STEP 3: Produce to Kafka Topics`
How To Create Amazing TML Real-Time Dashboards:	Youtube Video Describes how to create TML real-time dashboards using websockets, and no third-party tools. See here :ref:`TML Real-Time Dashboards` and, :ref:`STEP 7: Real-Time Visualization: tml-system-step-7-kafka-visualization-dag`
TML Examples Video:	YouTube Video Describes examples shown here: :ref:`TML Solution Examples`
TML JSON Criteria Creator:	YouTube Video Describes Json Criteria creation for Json Processing shown here: :ref:`JSON PROCESSING`
Zoom student meeting at Seneca Polytechnic:	YouTube Video Zoom meeting to students to show how to run Examples 1 nand 3 with data streaming to a MQTT broker in HiveMQ. See here :ref:`TML Solution Examples`
TML Step 1 Dag Configurations:	YouTube Video Describes key parameter configurations in: :ref:`STEP 1: Get TML Core Params: tml_system_step_1_getparams_dag`
TML Step 2 Dag Configurations:	YouTube Video Describes key parameter configurations in: :ref:`STEP 2: Create Kafka Topics: tml_system_step_2_kafka_createtopic_dag`
TML Step 3 Dag Configurations:	Refer to **Ingesting Real-Time Data into TML Solutions:** :ref:`STEP 3: Produce to Kafka Topics`
TML Step 4 Dag Configurations:	YouTube Video Describes key parameter configurations in: :ref:`STEP 4: Preprocesing Data: tml-system-step-4-kafka-preprocess-dag`

TML Step 4a and 4c for RTMS Dag Configurations:	YouTube Video Describes key parameter configurations for the RTMS solution: :ref:`STEP 4a: Preprocesing Data: tml-system-step-4a-kafka-preprocess-dag` :ref:`STEP 4c: Preprocesing 3 Data: tml-system-step-4c-kafka-preprocess-dag`
TML Step 4b Dag Configurations:	YouTube Video Describes key parameter configurations in: :ref:`STEP 4b: Preprocesing 2 Data: tml-system-step-4b-kafka-preprocess-dag`
TML Step 5 Dag Configurations:	YouTube Video Describes key parameter configurations in: :ref:`STEP 5: Entity Based Machine Learning : tml-system-step-5-kafka-machine-learning-dag`
TML Step 6 Dag Configurations:	YouTube Video Describes key parameter configurations in: :ref:`STEP 6: Entity Based Predictions: tml-system-step-6-kafka-predictions-dag`
TML Step 7 Dag Configurations:	Refer to **How To Create Amazing TML Real-Time Dashboards:** :ref:`STEP 7: Real-Time Visualization: tml-system-step-7-kafka-visualization-dag`
TML Step 8 Dag:	Step 8 Dag for docker container - Does not require any configurations. See :ref:`STEP 8: Deploy TML Solution to Docker : tml-system-step-8-deploy-solution-to-docker-dag`
TML Step 9 Dag Configurations:	YouTube Video Describes key parameter configurations in: :ref:`STEP 9: PrivateGPT and Qdrant Integration: tml-system-step-9-privategpt_qdrant-dag`
TML Step 10 Dag Configurations:	YouTube Video Describes key parameter configurations in: :ref:`STEP 10: Create TML Solution Documentation: tml-system-step-10-documentation-dag`

Important

TML Solution Studio (TSS)

You will learn how to use the :ref:`TML Solution Studio (TSS) Container` to build advanced, scalable, end-end real-time TML solutions easily with little to no- code. **The TSS redefines how complex real-time solutions can be built by anyone**, using pre-written Apache Airflow TML DAGs, that are integrated Github, AI (PrivateGPT), Docker, and Kubernetes, with automated online documentation of your entire solution.

TSS enables TML solutions to be developed WITHOUT WRITING ANY CODE - ONLY CONFIGURATIONS TO DAGS - dramatically accelerating real-time solution builds.

Transactional Machine Learning (TML) Overview

> **Important**
>
> **Transactional Machine Learning (TML): The Machine Learning and AI Platform for Real-Time Data Streams**
>
> TML Is Based On the Belief that **"Fast data requires fast machine learning and AI for fast decision-making"** that provides **a faster way to build advanced, scalable, cost- effective and secure real-time solutions that can be built by anyone.**
>
> TML gives rise in the industy to a **Data Stream Scientist** versus a **Data Scientist** in conventional machine learning (CML).

Transactional Machine Learning (TML) using Data Streams and AutoML is a platform for building and streaming cloud native (or on-prem) solutions using Apache Kafka or Redpanda as the data backbone, with Kubernetes and Docker as core infrastucture components, running on Confluent, AWS, GCP, AZURE, for advanced machine learning solutions using transactional data to learn from, and provide insights, quickly and continuously to any number of devices and humans in any format!

TML Book Details Found Here: Springer (Publisher) site

TML Video: Watch the Youtube Video

Apply data preprocessing and auto machine learning to data streams and create transactional machine learning (TML) solutions that are:

1. frictionless: require minimal to no human intervention

2. elastic: machine learning solutions that can scale up or down using Kubernetes to control or enhance the number of data streams, algorithms (or machine learning models) and predictions instantly and continuously.

> **Note**
>
> TML is ideal when data are highly erratic (nonlinear) and you want the machine to learn from the latest dataset by creating sliding windows of training datasets and auto creating micro-machine learning models quickly, that can be easily scaled, managed and the insights used immediately from any device! There are many TML use cases such as:
>
> 1. IoT: Capture real-time, fast, data, and build custom micro-machine learning models for every IoT device specific to the environment that the device operates in and predict failures, optimize device settings, and more.
> 2. IoT Edge: TML is ideal for edge devices with No Internet connections. Simply use the On-Prem version of TML software, with On-Prem Kafka and create large and powerful, real-time, edge solutions.
> 3. HealthCare: TML is ideal for health data processing for patients, payers, and providers. Open access to health data has been mandated by CMS, which opens up enormous opportunities for TML.
> 4. Banking/Finance Fraud Detect: Detect fraud using unsupervised learning on data streams and process millions of transactions for fraud - see the LinkedIn blog
> 5. Financial Trading: Use TML to analyse stock prices and predict sub-second stock prices!
> 6. Pricing: Use TML to build optimal pricing of products at scale.

7. Oil/Gas: Use TML to optimize oil drilling operations sub-second and drill oil wells faster and cheaper with minimal downtime
8. SO MUCH MORE...

The above usecases are not possible with conventional machine learning methods that require frequent human interventions that create lots of friction, and not very elastic.

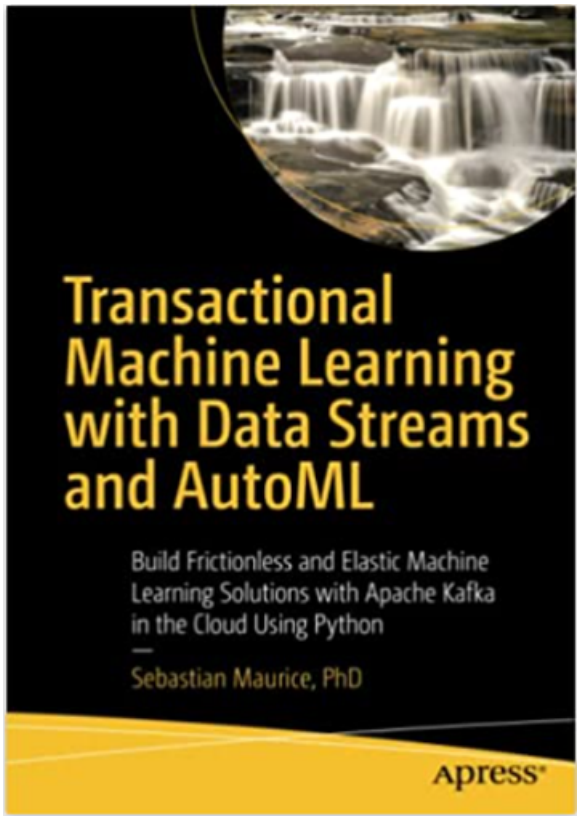
By using Apache Kafka On-Premise many advanced, and large, TML usecases are 80-90% cheaper than cloud-native usecases, mainly because storage, compute, Egress/Ingress and Kafka partitions are localized. Given Compute and Storage are extremely low-cost On-Premise solutions with TML are on the rise. TML On-Prem is ideal for small companies or startups that do not want to incur large cloud costs but still want to provide TML solutions to their customers.

Strengthen your knowledge of the inner workings of TML solutions using data streams with auto machine learning integrated with Apache Kafka. You will be at the forefront of an exciting area of machine learning that is focused on speed of data and algorithm creation, scale, and automation.

Contents

TML Book

The TML book can be found here on Amazon



QUICK START: Run TWO TML Solutions Right Now!

> **Important**
>
> The power of TML is not only in how it can process and perform machine learning at the entity level, in-memory, but the amazing real-time visualizations that users can create with the TML output for faster, deeper, insights from real-time data streams.

QUICK START: TML Solution with Real-Time Entity Based Processing

For users who want to quickly see a running solution now, just do the following.

> **Note**
>
> You must have docker installed.

Run this docker command

```
docker run -d -p 9005:9005 maadsdocker/seneca-iot-tml-kafka-amd64
```

> **Tip**
>
> Wait about 10 seconds...

View The Real-Time Dashboard

Then, open up your favorite browser and enter this URL below:

```
http://localhost:9005/iot-failure-seneca.html?topic=iot-preprocess2,iot-preprocess&offset=-1&groupid=&rollbackoffset=500&topictype=prediction&append=0&secure=1
```

> **Tip**
>
> PRESS THE RED "START STREAMING" button in the top-left...

You should see this Dashboard in your browser start to populate with real-time preprocessed IOT data:

> Note
>
> The above dashboard is processing real-time data and streaming it directly from your container to your browser using websockets.

> Tip
>
> Hover over with your mouse on the map bubbles. You can also download all the table data by clicking "Download as CSV".

QUICK START: Another TML Soluton with Real-Time Entity Based Processing AND Machine Learning

Let's run another TML solution, but this time with machine learning models being created for each device entity.

Run this docker command

```
docker run -d -p 9006:9006 maadsdocker/uoft-iot-tml-kafka-amd64
```

> Tip
>
> Wait about 10 seconds...

View Another Real-Time Dashboard

Then, open up your favorite browser and enter this URL below:

```
https://localhost:9006/iot-failure-machinelearning-uoft.html?topic=iot-preprocess,iot-ml-prediction-results-output&offset=-1&groupid=&rollbackoffset=500&topictype=prediction&append=0&secure=1
```

> Tip
>
> PRESS THE RED "START STREAMING" button in the top-left...

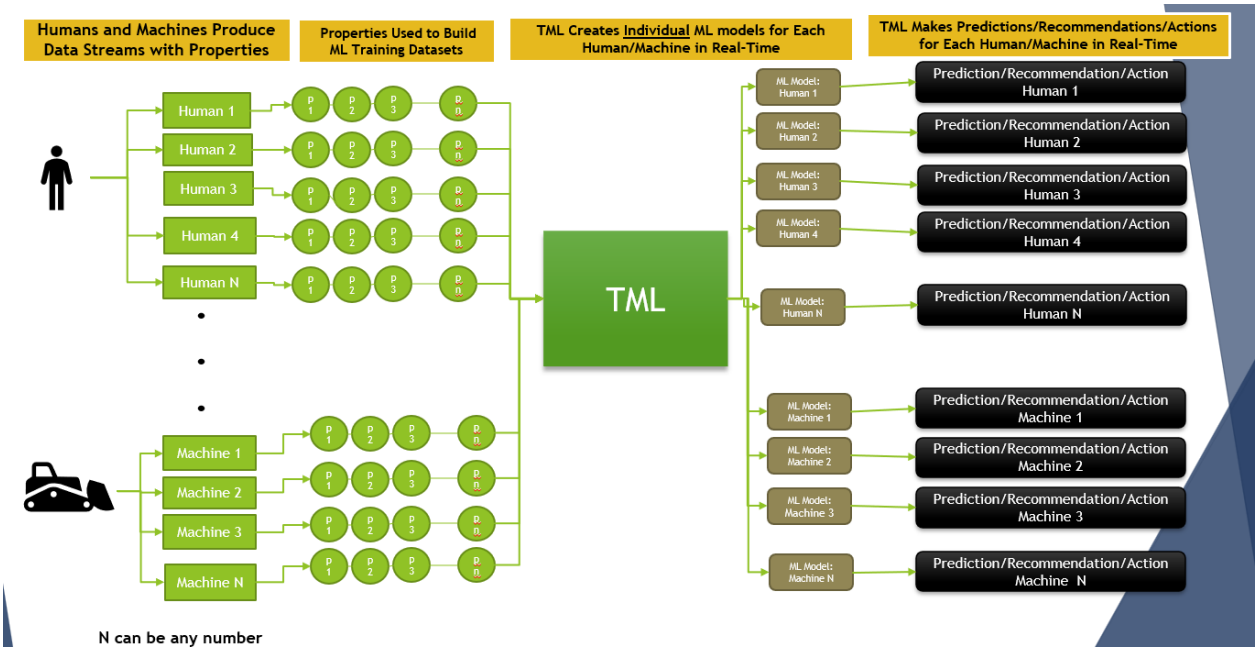
You should see this Dashboard in your browser start to populate with real-time entity based probability predictions of IOT device failures. **The figure below shows 43 machine learning models created for 43 devices!**

> **Tip**
>
> Press the TOGGLE button in the top-right of the dashboard.

TML Performs Entity Level Machine Learning and Processing

What does entity level processing and machine learning mean?

An entity is any device or object that produces real-time data. TML processes data from **individual devices or objects**. This means TML can create machine learning models for each invidual device.



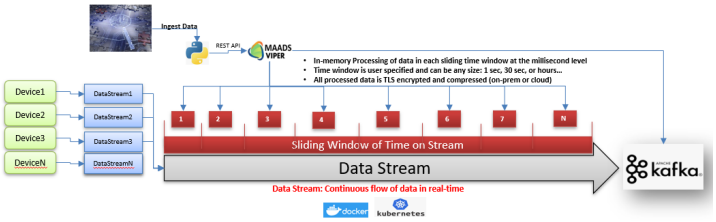
Important

The power of real-time entity based processing and machine learning means that if there are **1 million devices generating data, TML can create 1 million machine learning models for each device**.

Because each device or object operates in its own environment - by processing each device invidually - TML offers a much deeper understanding of the behaviour of that device, and therefore able to **predict the future behaviours** of that device more accurately.

TML uses :ref:`JSON PROCESSING` to process JSON messages streaming to Kafka.

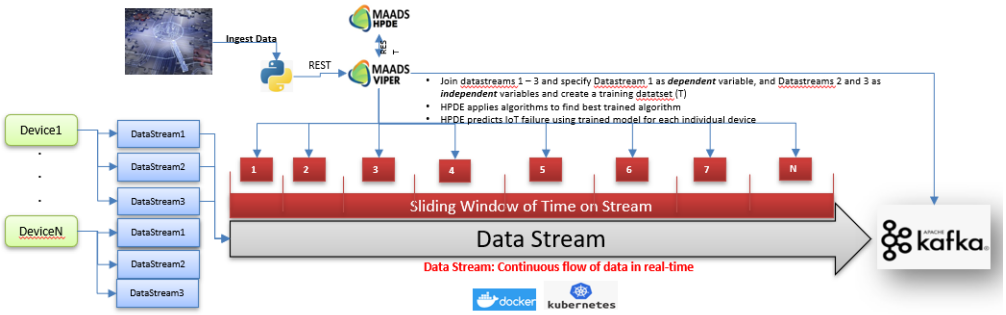
TML Processes Real-Time Data Using Sliding Time Windows



Attention!

- TML performs in-memory processing of data in the Kafka Topic using TWO components across all sliding time windows. See: :ref:`STEP 4: Preprocesing Data: tml-system-step-4-kafka-preprocess-dag`
- REST API connect MAADSTML python script to MAADS-VIPER
- 35+ different processing types: min, max, dataage, timediff, variance, anomaly prediction, outlier detection, etc…
- Apache Kafka is the central source of both input and output data – no external real-time database needed
- No SQL queries are made for processing and machine learning
- All TML solutions are containerized with Docker and scale with Kubernetes

TML Machine Learning Using Sliding Time Windows



Attention!

- TML performs in-memory machine learning of data in the Kafka Topic by joining data streams using THREE components across all sliding time windows:
- REST API connect MAADSTML python script to MAADS-VIPER and MAADS-HPDE
- 5 different algorithm types: logistic regression, linear regression, gradient boosting, neural networks, ridge regression
- Apache Kafka is the central source of both input and output data for estimated parameters – no external real-time database needed. See :ref:`STEP 5: Entity Based Machine Learning : tml-system-step-5-kafka-machine-learning-dag`
- TML auto-creates individual machine learning models for each Device at the “entity” level and joins datastreams 1-3 for each device and user specifies
- “Dependent” variable streams, and “Independent” variables streams

JSON PROCESSING

TML processes json data only. JSON (Java Script Object Notation) is a standard format used by all applications. Processing JSON data is highly efficient compared to SQL queries, due mainly to minimal data movements, faster compute, no database needed, and very low cost to process very large amounts of data.

To process your JSON data - you need to tell TML applicate the **JSON paths to the keys and values** to process.

> Tip
>
> Watch the YouTube video that shows you how to process your json.

TML requires the following - as shown in the table below. We will go though an example on how to apply this to your JSON data.

> Important
>
> Json processing and the **jsoncriteria** field will be required in :ref:`STEP 4: Preprocesing Data: tml-system-step-4-kafka-preprocess-dag.py`

Jsoncriteria TML Fields	**Description**
uid	This needs to be a unique id for the json message. For example, if processing IOT device data, this would be the device's serial number. You can also specify multiple UID by separating by a pipe "|" . See example below :ref:`JSON Message In A Payload`
filter	There is a filter field in the jsoncriteria. This allows you to either: 1. filter:allrecords 2. filter:<key>=<value, or allrecords> 3. filter:<key>=<value, or allrecords>,payload=<json path to payload key>
subtopics	This is the json path to the variable name that identifies the type of data. For example, if processing IOT, this could be json path to **voltage**
values	This needs to be the json path to the value of the subtopics. For example, for IOT voltage, it needs to be the path to the voltage value.
identifiers	This is the path to some metadata about the values.

datetime	This is the path to any datetime field.
msgid	This is the path to any MSG ID of the json message or blank.
latlong	This needs to be path to latitude or longitude or blank. You can join the fields by semicolon.

Note

Lets take an example. Lets say we want to process the following JSON message:

{"metadata":{"oem_id":"32795e59","oem_model":"SQR141U1XXW","dsn":**"AC000W016399396"**, "property_name":**"Power"**,"display_name":**"Power (mW)"**,"base_type":"integer","event_type":"datapoint"}, lat: **"43.11"**, long: **"127.011"**, "datapoint": {"id":**"de3e8f0e-7faa-11ec-31cb- 6b3a1eb15a96"**,"updated_at":**"2022-01- 27T19:53:59Z"**,"created_at":"2022-01-27T19:53:59Z","echo":false,"closed":false,"value":**"0"**,"metadata":{},"created_at_from_device":"2022-01- 27T19:51:40Z","user_uuid":"f4d3b326-da9a-11eb-87af-0a580ae966af","discarded":false,"scope":"user","direction":"output"}}

The JSONCRITERIA specifies the JSON PATH to each TML Field separated by ~:

jsoncriteria='uid=metadata.dsn,filter:allrecords~subtopics=metadata.property_name~ values=datapoint.value~ identifiers=metadata.display_name~datetime=datapoint.updated_at~msgid=datapoint.id~ latlong=lat:long'

uid=metadata.dsn will retrieve the value **AC000W016399396**

**filter:allrecords

subtopics=metadata.property_name will retrieve the value **Power**

values=datapoint.value will retrieve the value **0**

identifiers=metadata.display_name will retrieve the value **Power**

datetime=datapoint.updated_at will retrieve the value **2022-01-27T19:53:59Z**

msgid=datapoint.id will retrieve the value **de3e8f0e-7faa-11ec-31cb-6b3a1eb15a96**

latlong=lat:long will retrieve the value **43.11:127.011**

Important

Json path are an important aspect of instructing TML where to find the data (**value**) in the Json **key** for processing and machine learning.

The number of JSON paths in the **subtopics** MUST equal the number of JSON paths in the **values**. Multiple JSON paths can be separated by a comma.

TML requires the Json path of data to be processed. Use the "." (dot) to go into deeper levels of the Json.

Json Path Example

Lets consider the following example.

Sample Json message	Json Paths

{
"metadata": {
"oem_id": "32795e59",
"oem_model": "SQR141U1XXW",
"dsn": "AC000W016399396",
"property_name": "Power",
"display_name": "Power (mW)",
"base_type": "integer",
"event_type": "datapoint"
},
"datapoint": {
"id": "de3e8f0e-7faa-11ec-31cb-6b3a1eb15a96",
"updated_at": "2022-01- 27T19:53:59Z",
"created_at": "2022-01-27T19:53:59Z",
"echo": false,
"closed": false,
"value": "0",
"metadata": {},
"created_at_from_device": "2022-01-27T19:51:40Z",
"user_uuid": "f4d3b326-da9a-11eb-87af-0a580ae966af",
"discarded": false,
"scope": "user",
"direction": "output"
}, "lat": 29.22,
"long": -141.22
}
	The Json Path to the variable: **dsn** is **metadata.dsn**
	The Json Path to the key: **value** is datapoint.value
	The Json criteria will be:
	jsoncriteria=
	uid= metadata.dsn,filter:allrecords~
	subtopics= metadata.property_name~
	values= datapoint.value~
	identifiers= metadata.display_name~
	datetime= datapoint.updated_at~
	msgid= datapoint.id~
	latlong=lat:long
	Note: ~ and are just string delimiter
	and continuation characters, respectively.

Say you have a value you want to extract from a Json array: **"code": {** "coding": [**{** "system": "http://snomed.info/sct", "code": "84489001", "display": "Mold (organism)" }]},	The Json Path to the variable array: **code** is **code.coding.0.code**, 0 is the first element of the array.

JSON Message In A Payload

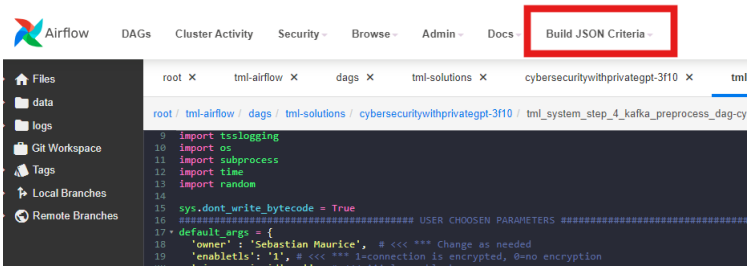
> Important
>
> If your JSON message comes as a **payload**, in the **filter** field you can specify jsoncriteria as follows:
>
> **jsoncriteria='uid=code.coding.0.code|code.coding.1.code|component.0.code.coding.0.code|**
> component.1.code.coding.0.code,
>
> **filter**:resourceType=allrecords, payload=payload.payload~
>
> **subtopics=code.coding.0.code,component.0.code.coding.0.code,**
> component.1.code.coding.0.code,medicationCodeableConcept.coding.0.code~
>
> **values=valueQuantity.value,component.0.valueQuantity.value,**
> component.1.valueQuantity.value,medicationCodeableConcept.coding.0.display~
>
> **identifiers=code.coding.0.display,component.0.code.coding.0.display,**
> component.1.code.coding.0.display,medicationCodeableConcept.coding.text~
>
> datetime=effectiveDateTime~
>
> msgid=subject.reference~
>
> **latlong=**
> address.0.extension.0.extension.0.valueDecimal:address.0.extension.0.extension.1.valueDecimal'
>
> # add + to join fields

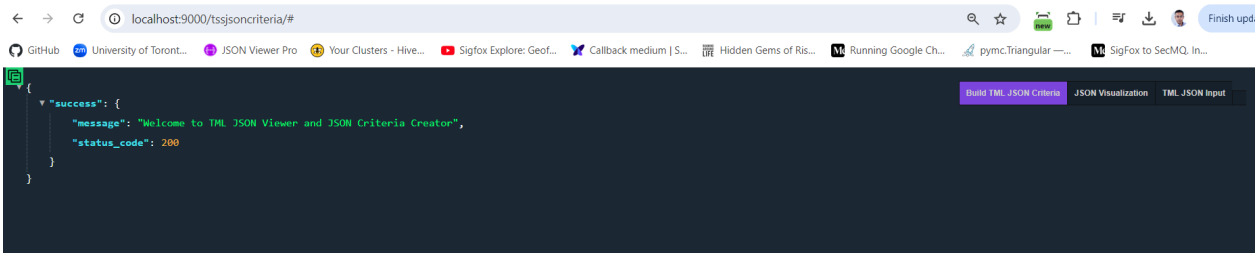
JSON Criteria Creator

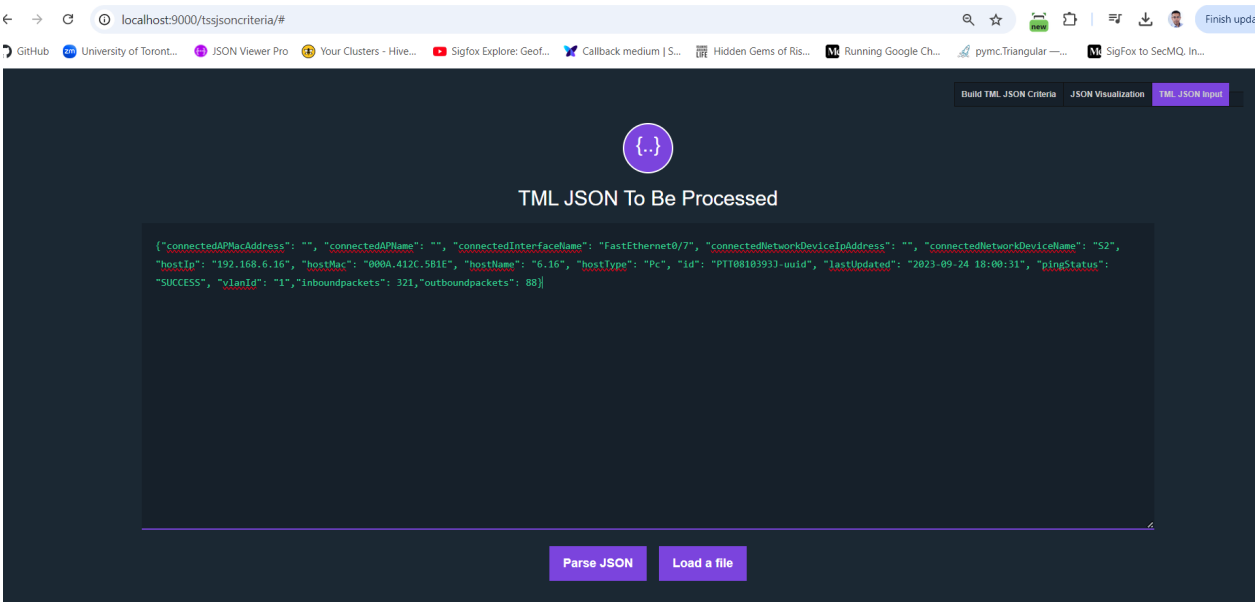
> Important
>
> The Json criteria creator is a simple, powerful and visual way to process your json.

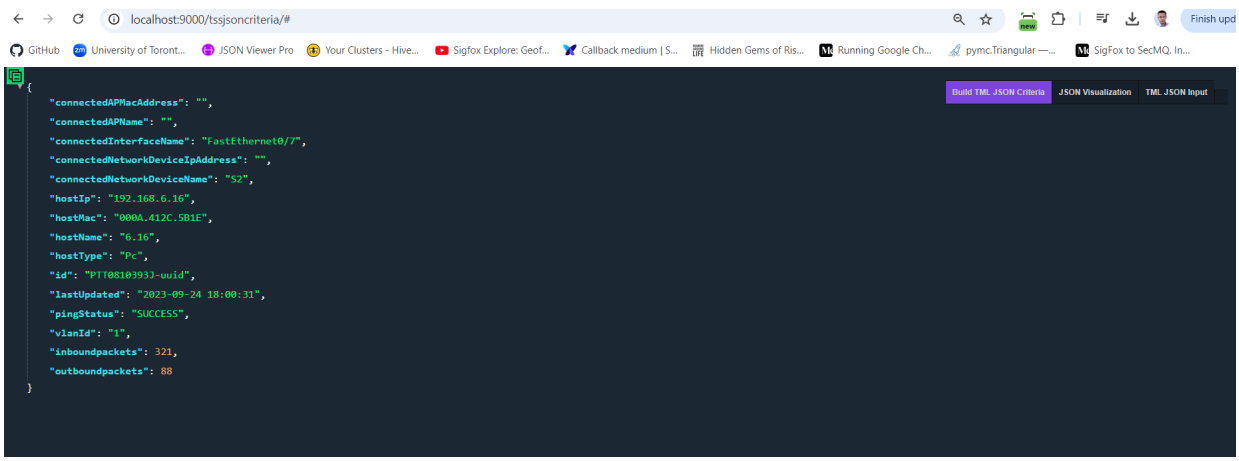
YouTube video

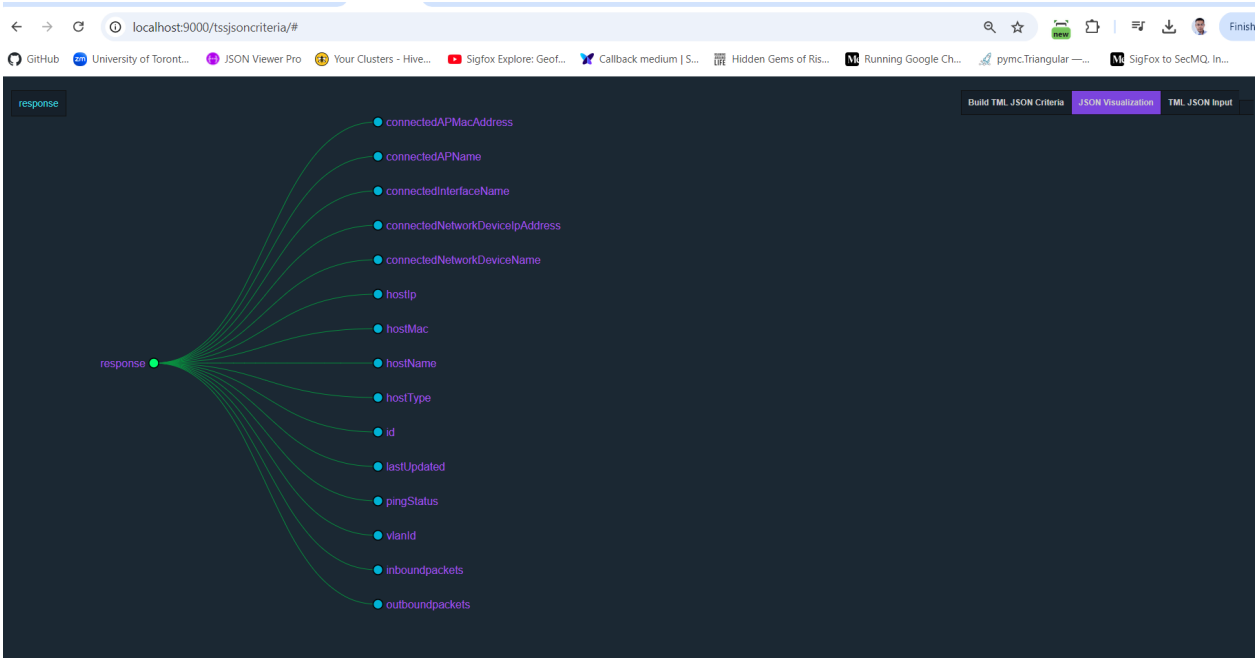
Here are the screenshots of the TML JSON creator.

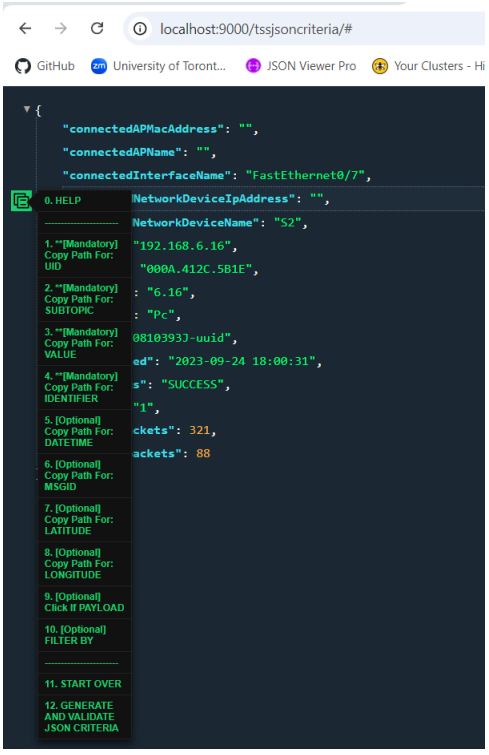


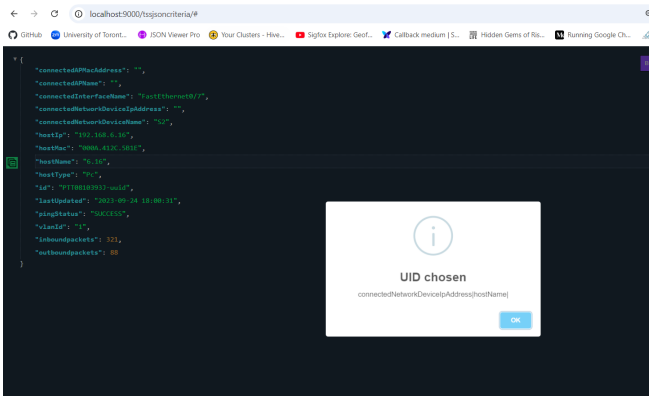


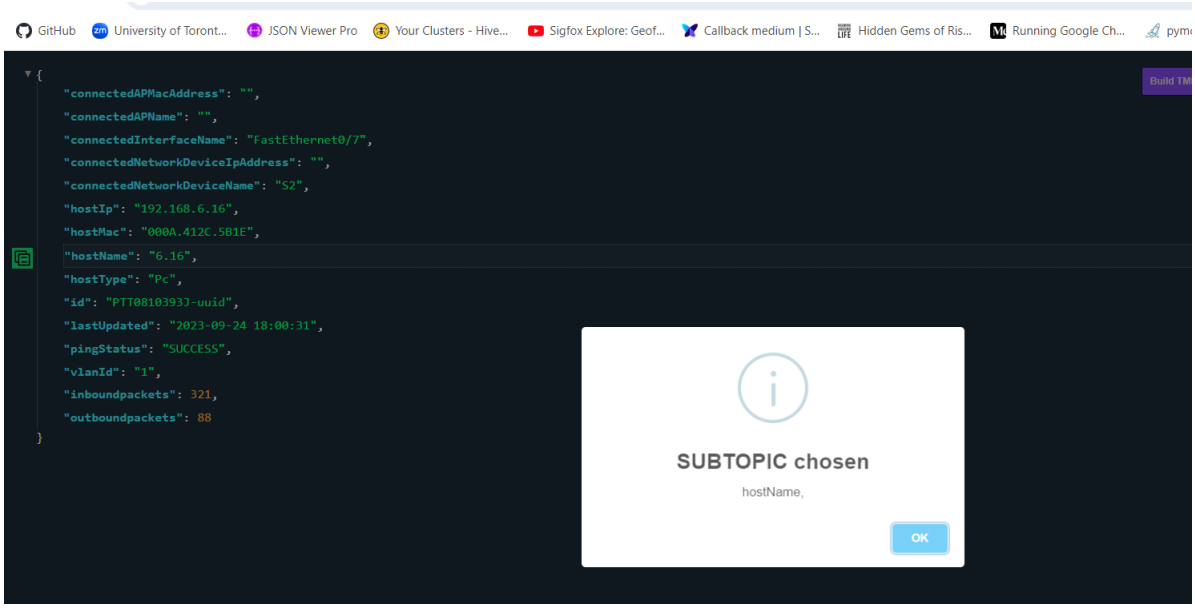


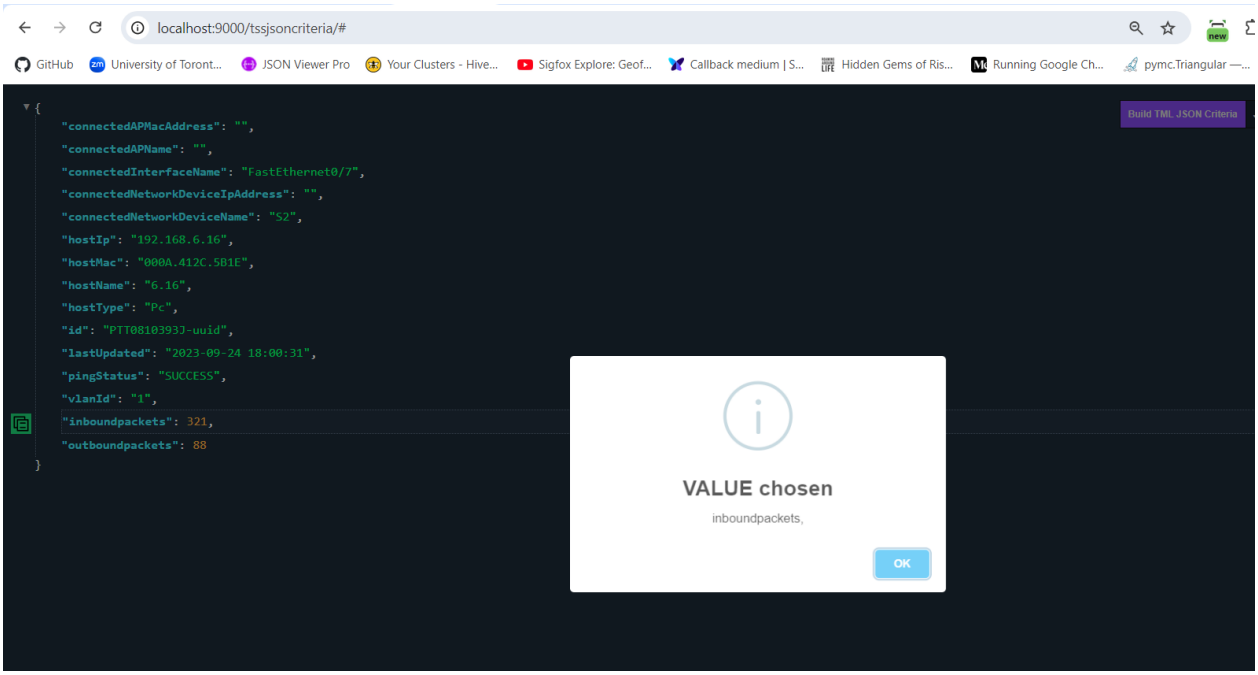


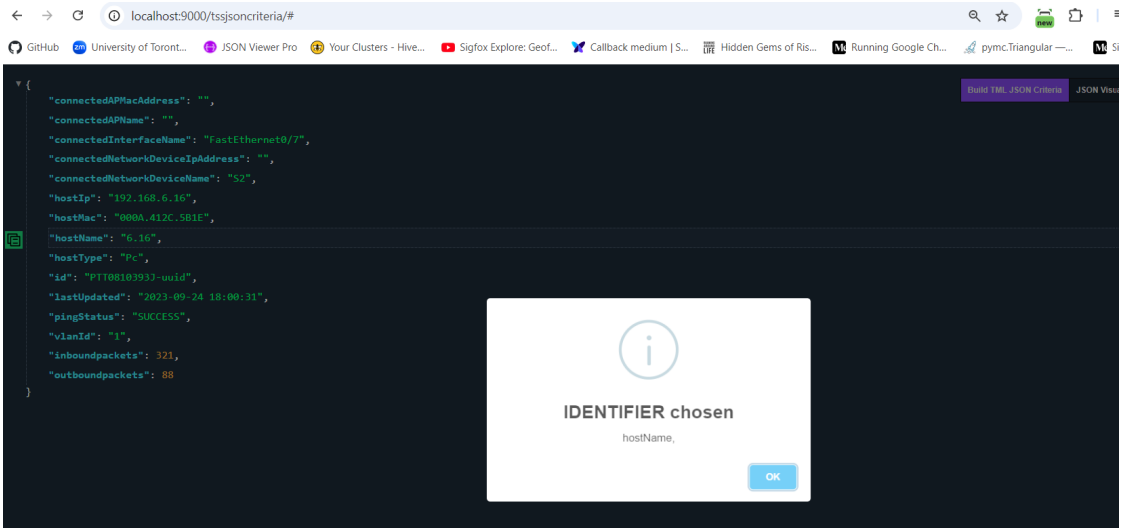

localhost:9000/tssjsoncriteria/#
response
Build TML JSON Criteria
JSON Visualization
TML JSON Input
connectedAPMacAddress
connectedAPName
connectedInterfaceName
connectedNetworkDeviceIpAddress
connectedNetworkDeviceName
hostIp
hostMac
hostName
hostType
id
lastUpdated
pingStatus
vlanId
inboundpackets
outboundpackets

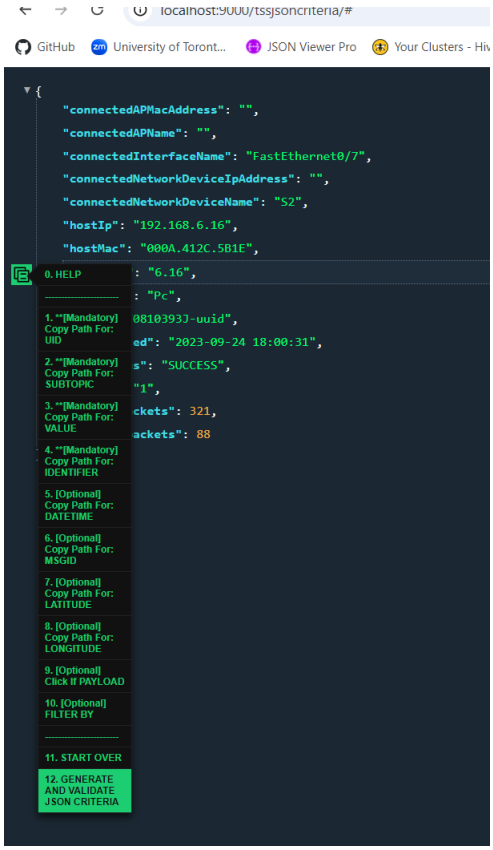

localhost:9000/tssjsoncriteria/#
"connectedAPMacAddress": "",
"connectedAPName": "",
"connectedInterfaceName": "FastEthernet0/7",
0. HELP
1. **[Mandatory] Copy Path For: UID
2. **[Mandatory] Copy Path For: SUBTOPIC
3. **[Mandatory] Copy Path For: VALUE
4. **[Mandatory] Copy Path For: IDENTIFIER
5. [Optional] Copy Path For: DATETIME
6. [Optional] Copy Path For: MSGID
7. [Optional] Copy Path For: LATITUDE
8. [Optional] Copy Path For: LONGITUDE
9. [Optional] Click If PAYLOAD
10. [Optional] FILTER BY
11. START OVER
12. GENERATE AND VALIDATE JSON CRITERIA

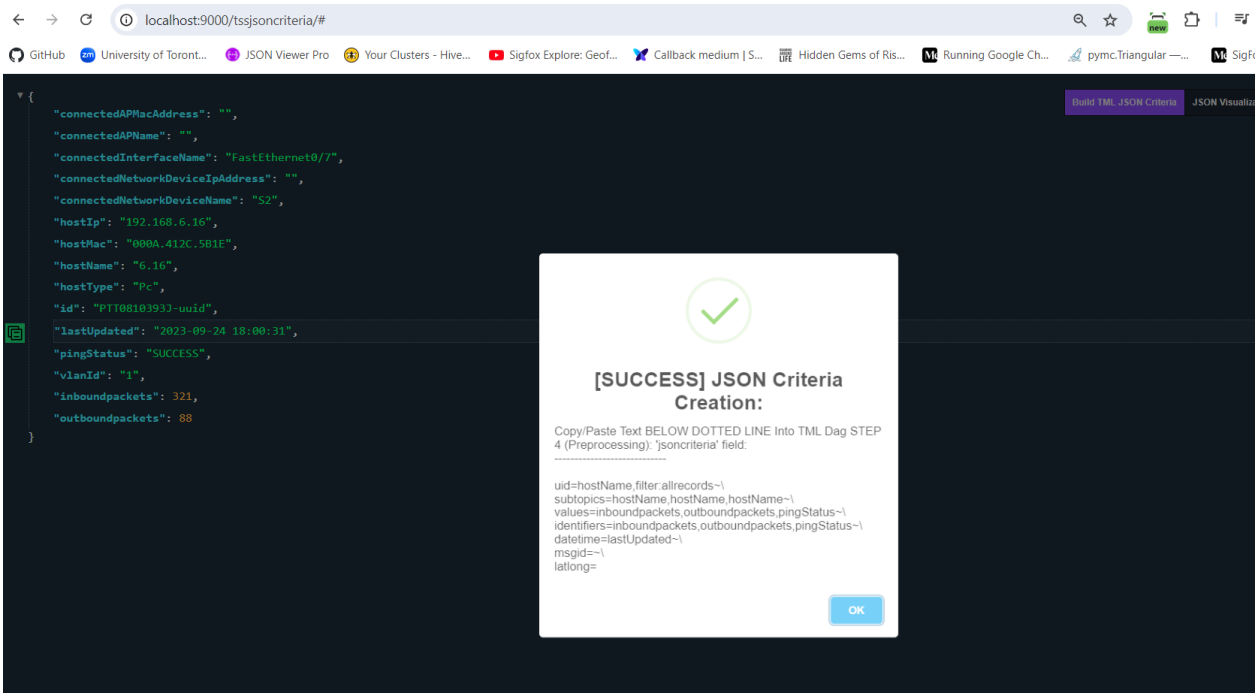

UID chosen
OK

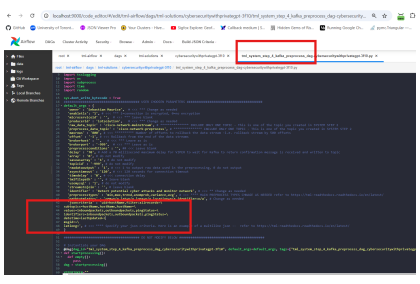

GitHub
University of Toront...
JSON Viewer Pro
Your Clusters - Hive...
Sigfox Explore: Geof...
Callback medium | S...
Hidden Gems of Ris...
Running Google Ch...
{
"connectedAPMacAddress": "",
"connectedAPName": "",
"connectedInterfaceName": "FastEthernet0/7",
"connectedNetworkDeviceIpAddress": "",
"connectedNetworkDeviceName": "S2",
"hostIp": "192.168.6.16",
"hostMac": "000A.412C.5B1E",
"hostName": "6.16",
"hostType": "Pc",
"id": "PTT08103933-uuid",
"lastUpdated": "2023-09-24 18:00:31",
"pingStatus": "SUCCESS",
"vlanId": "1",
"inboundpackets": 321,
"outboundpackets": 88
}
SUBTOPIC chosen
hostName,
OK


localhost:9000/tssjsoncriteria/#
GitHub
University of Toront...
JSON Viewer Pro
Your Clusters - Hive...
Sigfox Explore: Geof...
Callback medium | S...
Hidden Gems of Ris...
Running Google Ch...
pymc.Triangular —...
Build TML JSON Criteria
{
"connectedAPMacAddress": "",
"connectedAPName": "",
"connectedInterfaceName": "FastEthernet0/7",
"connectedNetworkDeviceIpAddress": "",
"connectedNetworkDeviceName": "S2",
"hostIp": "192.168.6.16",
"hostMac": "000A.412C.5B1E",
"hostName": "6.16",
"hostType": "Pc",
"id": "PTT08103933-uuid",
"lastUpdated": "2023-09-24 18:00:31",
"pingStatus": "SUCCESS",
"vlanId": "1",
"inboundpackets": 321,
"outboundpackets": 88
}
VALUE chosen
inboundpackets,
OK


localhost:9000/tssjsoncriteria/#
GitHub
University of Toront...
JSON Viewer Pro
Your Clusters - Hive...
Sigfox Explore: Geof...
Callback medium | S...
Hidden Gems of Ris...
Running Google Ch...
pymc.Triangular —...
Build TML JSON Criteria
{
"connectedAPMacAddress": "",
"connectedAPName": "",
"connectedInterfaceName": "FastEthernet0/7",
"connectedNetworkDeviceIpAddress": "",
"connectedNetworkDeviceName": "S2",
"hostIp": "192.168.6.16",
"hostMac": "000A.412C.5B1E",
"hostName": "6.16",
"hostType": "Pc",
"id": "PTT08103933-uuid",
"lastUpdated": "2023-09-24 18:00:31",
"pingStatus": "SUCCESS",
"vlanId": "1",
"inboundpackets": 321,
"outboundpackets": 88
}
IDENTIFIER chosen
hostName,
OK


localhost:9000/tssjsoncriteria/#
GitHub
University of Toront...
JSON Viewer Pro
Your Clusters - Hiv
"connectedAPMacAddress": "",
"connectedAPName": "",
"connectedInterfaceName": "FastEthernet0/7",
"connectedNetworkDeviceIpAddress": "",
"connectedNetworkDeviceName": "S2",
"hostIp": "192.168.6.16",
"hostMac": "000A.412C.5B1E",
0. HELP
1. **[Mandatory] Copy Path For: UID
2. **[Mandatory] Copy Path For: SUBTOPIC
3. **[Mandatory] Copy Path For: VALUE
4. **[Mandatory] Copy Path For: IDENTIFIER
5. [Optional] Copy Path For: DATETIME
6. [Optional] Copy Path For: MSGID
7. [Optional] Copy Path For: LATITUDE
8. [Optional] Copy Path For: LONGITUDE
9. [Optional] Click If PAYLOAD
10. [Optional] FILTER BY
11. START OVER
12. GENERATE AND VALIDATE JSON CRITERIA


localhost:9000/tssjsoncriteria/#
GitHub
University of Toront...
JSON Viewer Pro
Your Clusters - Hive...
Sigfox Explore: Geof...
Callback medium | S...
Hidden Gems of Ris...
Running Google Ch...
pymc.Triangular —...
Build TML JSON Criteria
JSON Visualiz
"connectedAPMacAddress": "",
"connectedAPName": "",
"connectedInterfaceName": "FastEthernet0/7",
"connectedNetworkDeviceIpAddress": "",
"connectedNetworkDeviceName": "S2",
"hostIp": "192.168.6.16",
"hostMac": "000A.412C.5B1E",
"hostName": "6.16",
"hostType": "Pc",
"id": "PTT0810393J-uuid",
"lastUpdated": "2023-09-24 18:00:31",
"pingStatus": "SUCCESS",
"vlanId": "1",
"inboundpackets": 321,
"outboundpackets": 88
[SUCCESS] JSON Criteria Creation:
Copy/Paste Text BELOW DOTTED LINE Into TML Dag STEP 4 (Preprocessing): 'jsoncriteria' field:
uid=hostName,filter:allrecords~\
subtopics=hostName,hostName,hostName~\
values=inboundpackets,outboundpackets,pingStatus~\
identifiers=inboundpackets,outboundpackets,pingStatus~\
datetime=lastUpdated~\
msgid=~\
latlong=
OK



TML Solution Examples

All examples simply follow the steps here :ref:`Lets Start Building a TML Solution`

> **Tip**
>
> You can watch the development and running of the examples below in this Youtube video.

TSS Starter Examples
1. :ref:`Real-Time IoT Data Preprocessing Example` This will process IoT data for anomalies, stream the results to a dashboard, build and push the container to docker hub, automatically create the solution documentation on Readthdocs, and auto-commit your solution to GitHub. Data Used is here (unzip the file): Demo Data
2. :ref:`Real-Time IoT Data Preprocessing With Secondary Preprocessing and With Map Example` This will process IoT data for anomalies, stream the results to a dashboard, but will also perform a **secondary processing** using :ref:`STEP 4b: Preprocesing 2 Data: tml-system-step-4b-kafka-preprocess-dag` to build and push the container to docker hub, automatically create the solution documentation on Readthdocs, and auto-commit your solution to GitHub.
2b. :ref:`Real-Time IoT Data Preprocessing With RESTAPI, Secondary Preprocessing and With Map Example` This solution is the same as 2, BUT this will solution will ingest real-time data from a RESTAPI client. To run this solution you must: 1. Have this solution running in Kubernetes 2. Download the IoTData.zip demo data 3. Download and Run the client RESTAPI file 4. Store these files to your local computer
2c. :ref:`Real-Time IoT Data Preprocessing With gRPC, Secondary Preprocessing and With Map Example` This solution is the same as 2, BUT this will solution will ingest real-time data from a gRPC client. To run this solution you must: 1. Have this solution running in Kubernetes 2. Download the IoTData.zip demo data 3. Download and Run the client gRPC file 4. Store these files to your local computer

3. :ref:`Real-Time IoT Data Preprocessing and Machine Learning Example`
This will process IoT data, and perform entity level machine learning (TML)
on the data, stream the results to a dashboard,
build and push the container to docker hub, automatically create the
solution documentation on Readthdocs, and auto-commit your solution
to GitHub.
Data Used is here (unzip the file): Demo Data

4. :ref:`Cybersecurity Solution with PrivateGPT, MQTT, HiveMQ`
This will process cybersecurity data for anomalies,
and further process the TML anomalies
data by sending the results to privateGPT container. It will stream
the results to a dashboard,
build and push the container to docker hub, automatically create the
solution documentation on Readthdocs, and auto-commit your solution
to GitHub.
MQTT Client File here is: MQTT Client
Data Used is here (unzip the file): Demo Data

4b. To Scale this Solution Watch the YouTube Video: here

Note

While we are using a local file for the demos, this is to make it easy for users to build and run these solutions themselves. Ideally, in industry, we would use APIs like MQTT, REST and gRPC to ingest data from devices and stream it to TML solutions. TSS allows you to build solutions with these APIs with pre-written DAGs here:

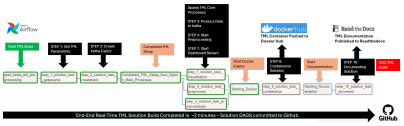
- :ref:`STEP 3a: Produce Data Using MQTT: tml-read-MQTT-step-3-kafka-producetotopic-dag`
  - With pre-built client library: :ref:`STEP 3a.i: MQTT CLIENT`
- :ref:`STEP 3b: Produce Data Using RESTAPI: tml-read-RESTAPI-step-3-kafka-producetotopic-dag`
  - With pre-built client library: :ref:`STEP 3b.i: REST API CLIENT`
- :ref:`STEP 3c: Produce Data Using gRPC: tml-read-gRPC-step-3-kafka-producetotopic-dag`
  - With pre-built client library: :ref:`STEP 3c.i: gRPC API CLIENT`

Real-Time IoT Data Preprocessing Example

:ref:`Solution DAG Code: solution_preprocessing_dag-myawesometmlsolution-3f10`

This IoT Data Preprocessing Solution DAG: **solution_preprocessing_dag-myawesometmlsolution-3f10** reads local file data in **/rawdata/IoTdata.txt** and streams it to Kafka. The streaming data are then processed with TML binary Viper and the output data are streamed to a browser that runs the dashboard: dashboard.html that is located in /Viperviz/viperviz/views.

The solution will automatically build and push the solution container to docker hub, automatically create documentation on READTHEDOCS.io and **automatically commits your solution code to Github, all in about 2 minutes.**



Solution DAG Code: solution_preprocessing_dag-myawesometmlsolution-3f10

The Python code below is the code representtion for the figure. **This code builds the entire end-end TML solution in about 2 minutes.**

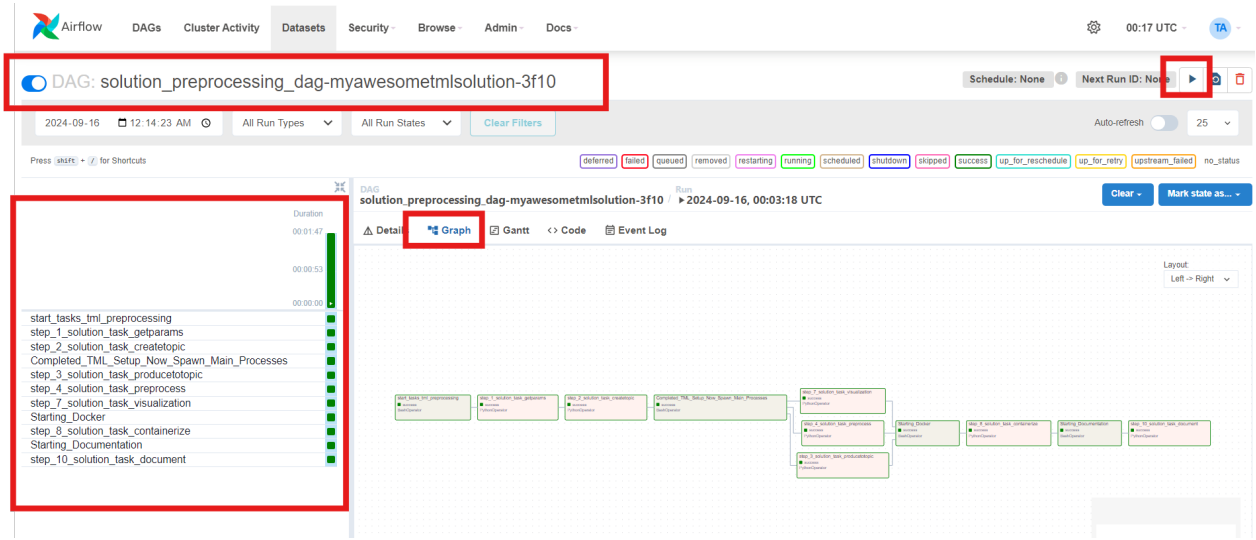
```
from __future__ import annotations

import pendulum
from airflow.decorators import task
from airflow.models.dag import DAG
from airflow.operators.bash import BashOperator
from airflow.sensors.external_task import ExternalTaskSensor
import tsslogging
import os
from datetime import datetime
import importlib
from airflow.operators.python import (
    ExternalPythonOperator,
    PythonOperator
)
step1 = importlib.import_module("tml-solutions.myawesometmlsolution-3f10.tml_system_step_1_getparams_dag-myawesometmlsolution-3f10")
step2 = importlib.import_module("tml-solutions.myawesometmlsolution-3f10.tml_system_step_2_kafka_createtopic_dag-myawesometmlsolution-3f10")
step3 = importlib.import_module("tml-solutions.myawesometmlsolution-3f10.tml_read_LOCALFILE_step_3_kafka_producetotopic_dag-myawesometmlsolution-3f10")
step4 = importlib.import_module("tml-solutions.myawesometmlsolution-3f10.tml_system_step_4_kafka_preprocess_dag-myawesometmlsolution-3f10")
step5 = importlib.import_module("tml-solutions.myawesometmlsolution-3f10.tml_system_step_5_kafka_machine_learning_dag-myawesometmlsolution-3f10")
step6 = importlib.import_module("tml-solutions.myawesometmlsolution-3f10.tml_system_step_6_kafka_predictions_dag-myawesometmlsolution-3f10")
step7 = importlib.import_module("tml-solutions.myawesometmlsolution-3f10.tml_system_step_7_kafka_visualization_dag-myawesometmlsolution-3f10")
step8 = importlib.import_module("tml-solutions.myawesometmlsolution-3f10.tml_system_step_8_deploy_solution_to_docker_dag-myawesometmlsolution-3f10")
step9 = importlib.import_module("tml-solutions.myawesometmlsolution-3f10.tml_system_step_9_privategpt_qdrant_dag-myawesometmlsolution-3f10")
step10 = importlib.import_module("tml-solutions.myawesometmlsolution-3f10.tml_system_step_10_documentation_dag-myawesometmlsolution-3f10")


with DAG(
    dag_id="solution_preprocessing_dag-myawesometmlsolution-3f10",
    start_date=datetime(2023, 1, 1),
    schedule=None,
) as dag:
  start_task = BashOperator(
    task_id="start_tasks_tml_preprocessing",
    bash_command="echo 'Start task'",
  )
# STEP 1: Get the Parameters
  sensor_A = PythonOperator(
            task_id="step_1_solution_task_getparams",
            python_callable=step1.getparams,
            provide_context=True,
  )

# STEP 2: Create the Kafka topics
  sensor_B = PythonOperator(
      task_id="step_2_solution_task_createtopic",
      python_callable=step2.setupkafkatopics,
      provide_context=True,
  )
# STEP 3: Produce data to topic
  sensor_C = PythonOperator(
      task_id="step_3_solution_task_producetotopic",
      python_callable=step3.startproducing,
      provide_context=True,
  )
# STEP 4: Preprocess the data
  sensor_D = PythonOperator(
      task_id="step_4_solution_task_preprocess",
      python_callable=step4.dopreprocessing,
      provide_context=True,
  )
# STEP 7: Containerize the solution
  sensor_E = PythonOperator(
      task_id="step_7_solution_task_visualization",
      python_callable=step7.startstreamingengine,
      provide_context=True,
  )
# STEP 8: Containerize the solution
  sensor_F = PythonOperator(
      task_id="step_8_solution_task_containerize",
      python_callable=step8.dockerit,
      provide_context=True,
  )
  start_task2 = BashOperator(
    task_id="Starting_Docker",
    bash_command="echo 'Start task Completed'",
  )
  start_task3 = BashOperator(
    task_id="Starting_Documentation",
    bash_command="echo 'Start task Completed'",
  )
  start_task4 = BashOperator(
    task_id="Completed_TML_Setup_Now_Spawn_Main_Processes",
    bash_command="echo 'Start task Completed'",
  )
# STEP 10: Document the solution
  sensor_G = PythonOperator(
      task_id="step_10_solution_task_document",
      python_callable=step10.generatedoc,
      provide_context=True,
  )

  start_task >> sensor_A >> sensor_B >> start_task4 >> [sensor_C, sensor_D, sensor_E] >> start_task2 >> sensor_F >> start_task3  >> sensor_G
```

TML Dag Parameter Changes To Be Made For:
solution_preprocessing_dag-myawesometmlsolution-3f10

> **Note**
>
> No TML Dag parameter changes are needed for this solution.

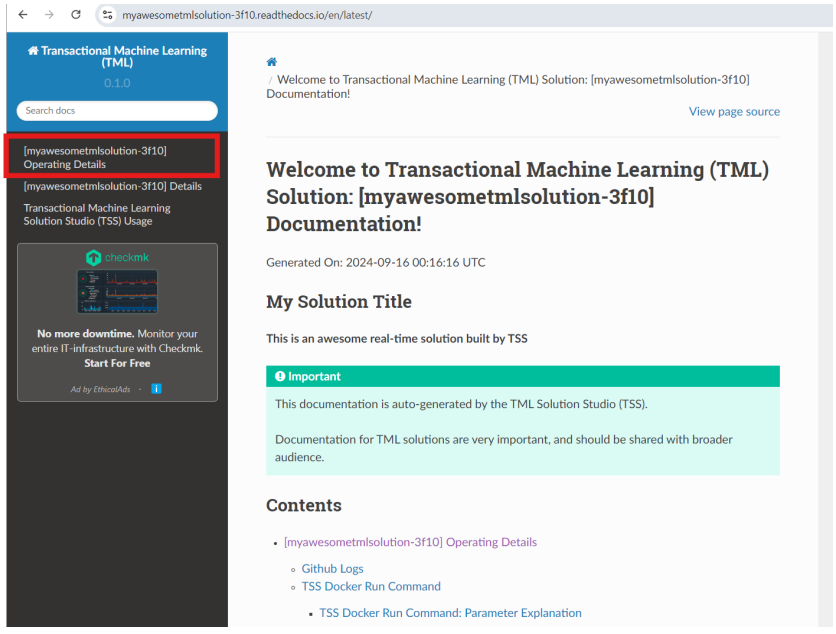
Successful Run Screen

Below the TSS/Airflow screen that shows a successful TML solution build. All colors should be green for all of the steps. If you see a red color, it means your DAG has an error.



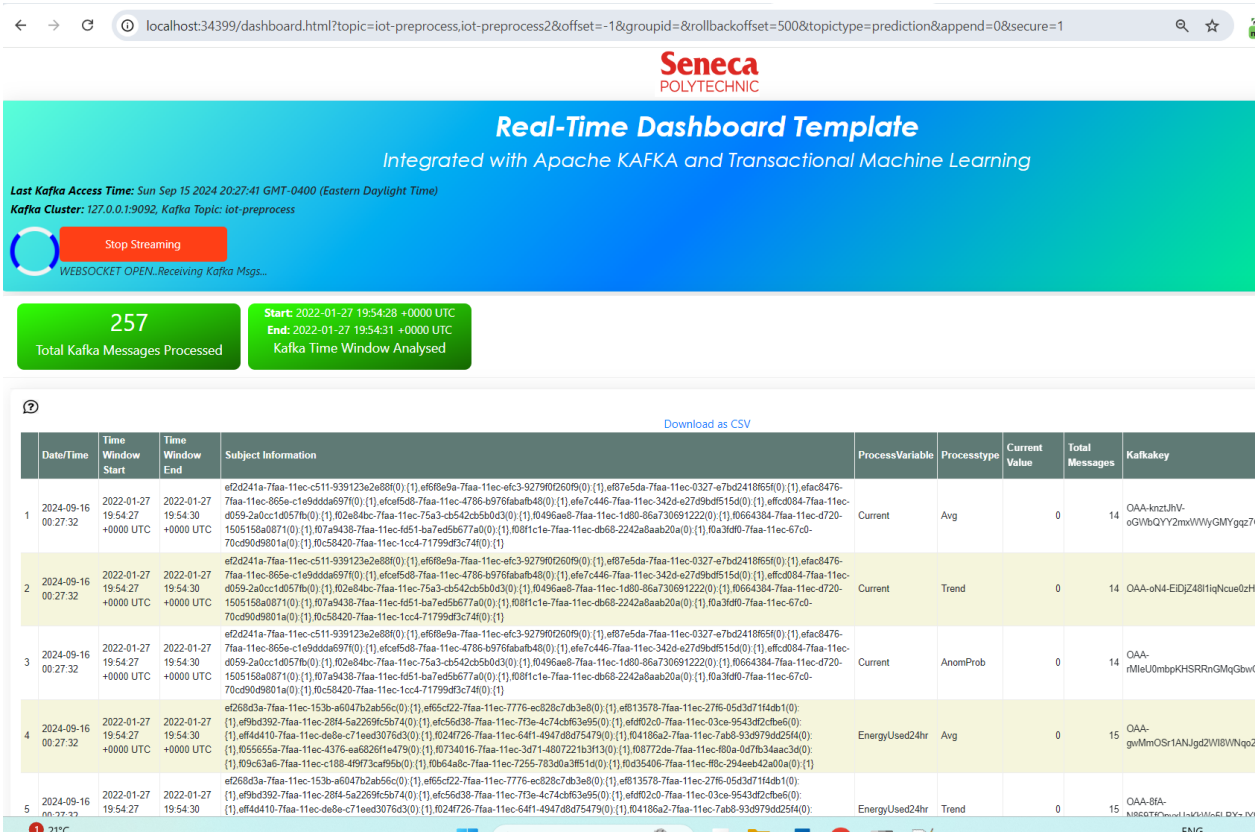
Solution Documentation Example

This is the solution documentation that is auto-generated by TSS. Every TML solution you create will have its own auto-generated documentation that will provide details on the entire solution.

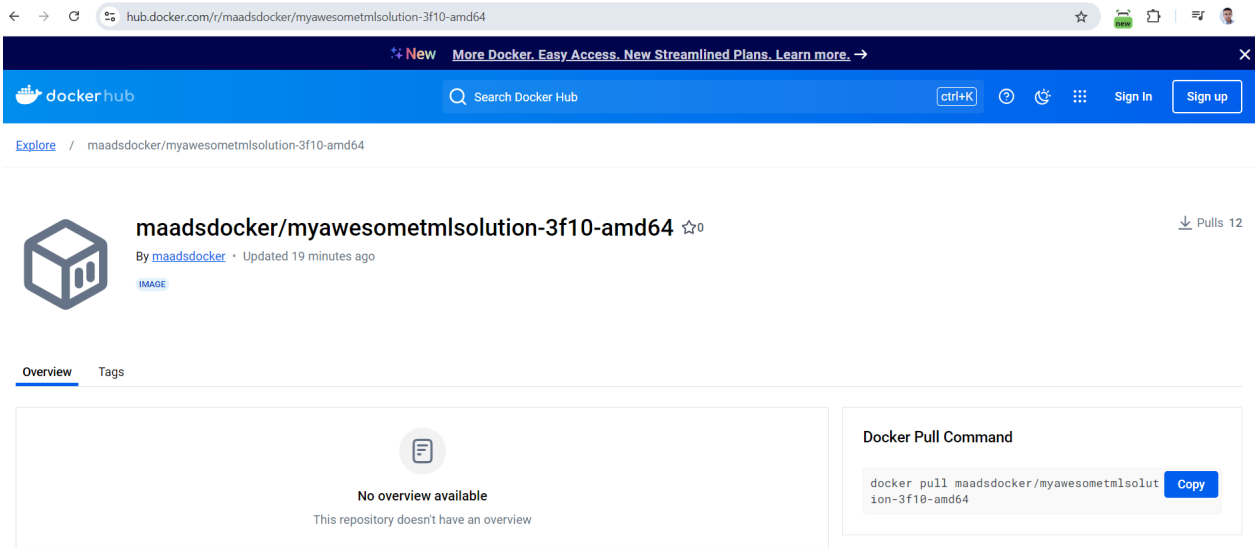


> **Important**
>
> You will need to run the solution in your own TSS environment for the links to work in this documentation. It is provided as an example of the powerful capabilities of TSS: https://myawesometmlsolution-3f10.readthedocs.io/

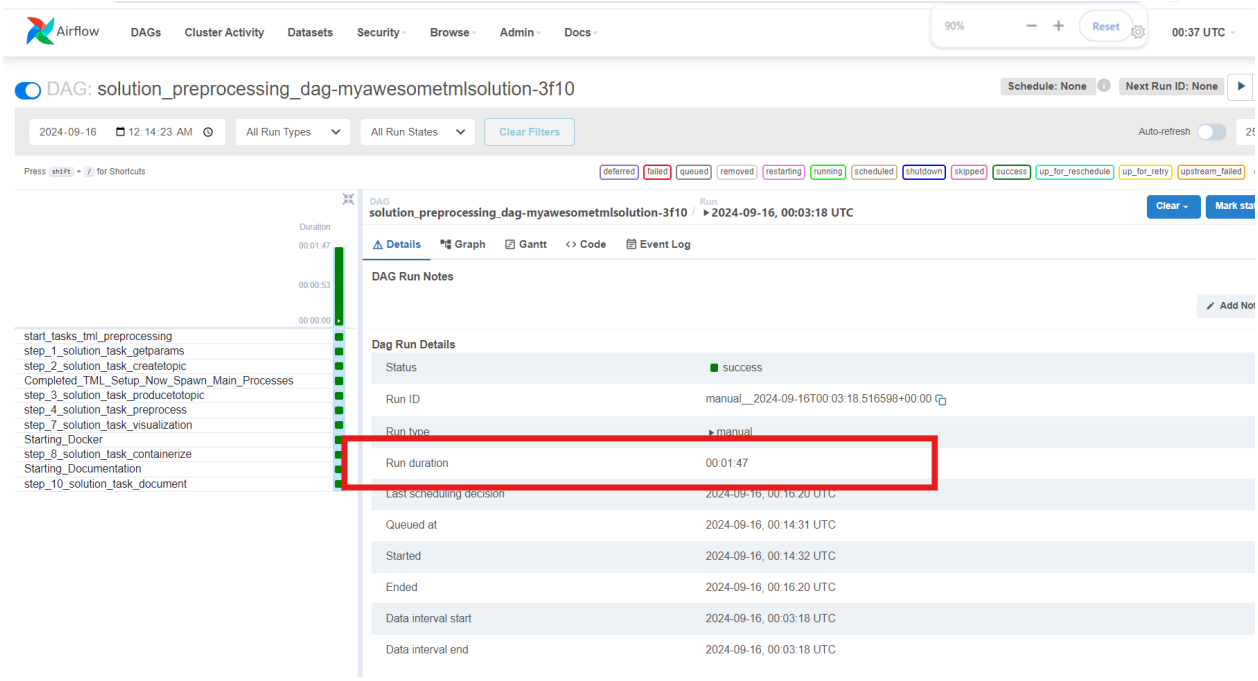
Here is the Solution Real-Time Dashboard:



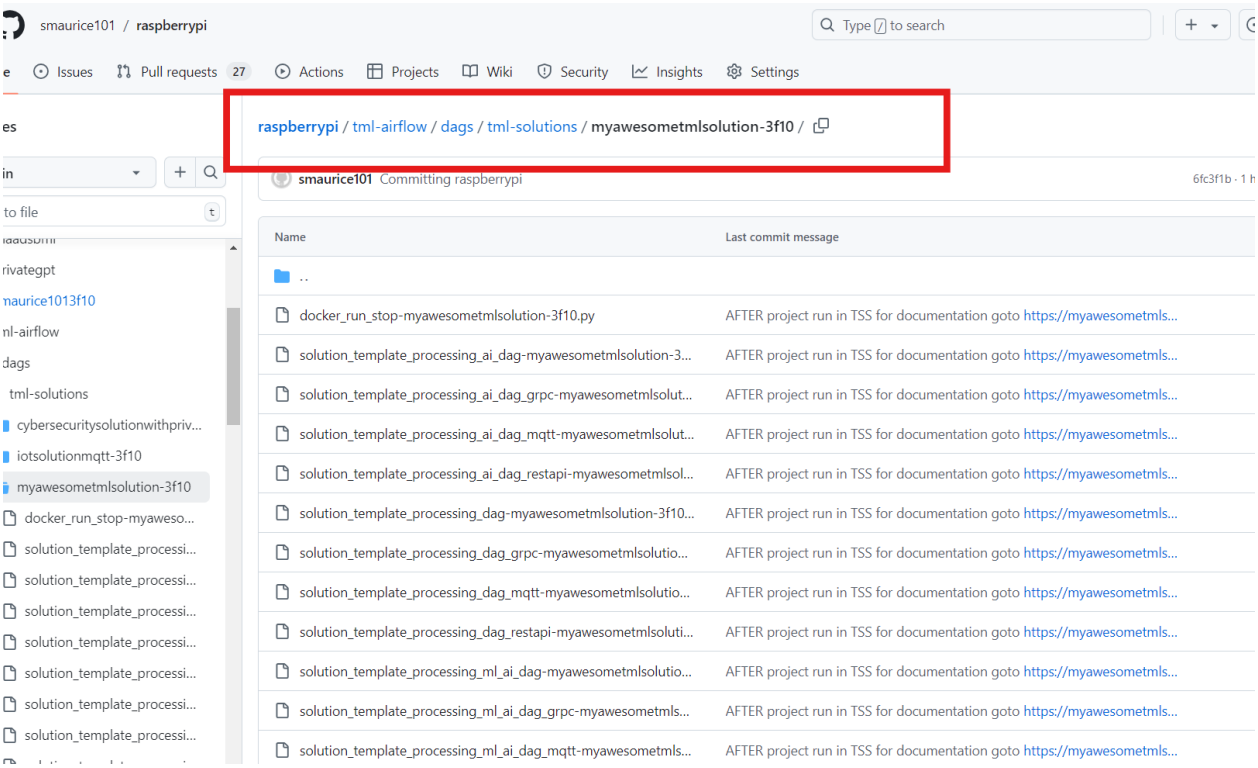
Here is the Solution Docker Run container:



The entire end-end real-time solution took less than 2 minutes to build:



Github Commits



Real-Time IoT Data Preprocessing With Secondary Preprocessing and With Map Example

:ref:`Solution DAG Code: solution_preprocessing_dag-iotsolution-3f10`

This IoT Data Preprocessing Solution DAG: **solution_preprocessing_dag-myawesometmlsolution-3f10** reads local file data in **/rawdata/IoTdata.txt** and streams it to Kafka. The streaming data are then processed TWICE with TML binary Viper and the output data are streamed to a browser that runs the dashboard with a map: iot-failure-seneca.html that is located in /Viperviz/viperviz/views.

The solution will automatically build and push the solution container to docker hub, automatically create documentation on READTHEDOCS.io and **automatically commits your solution code to Github, all in about 2 minutes.**

Solution DAG Code: solution_preprocessing_dag-iotsolution-3f10

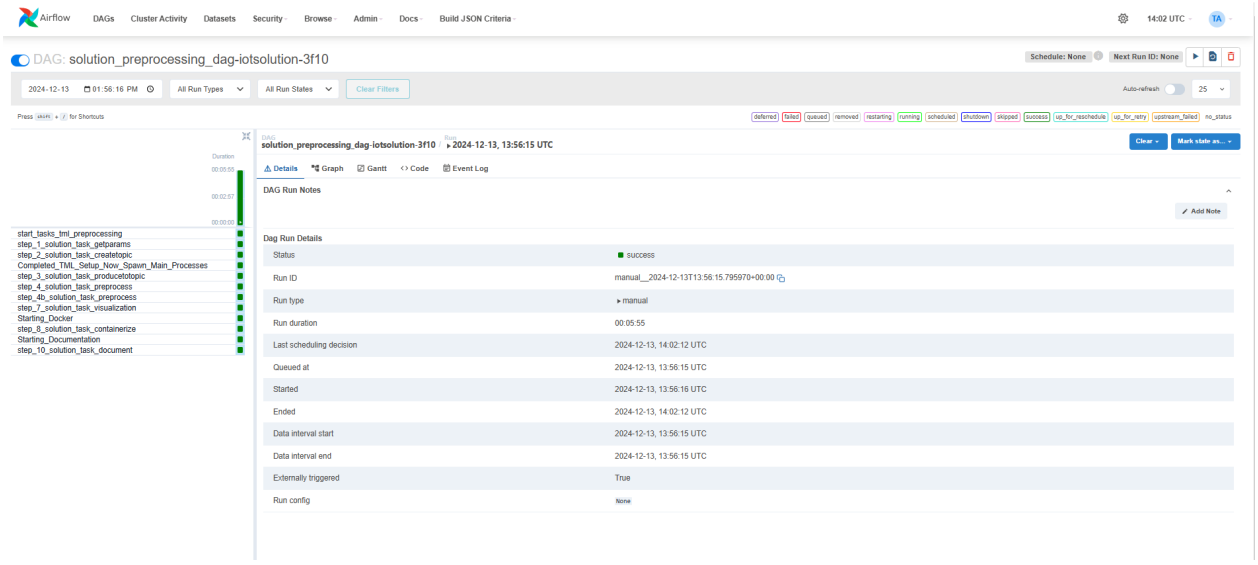
```
from __future__ import annotations

import pendulum
from airflow.decorators import task
from airflow.models.dag import DAG
from airflow.operators.bash import BashOperator
from airflow.sensors.external_task import ExternalTaskSensor
import tsslogging
import os
from datetime import datetime
import importlib
from airflow.operators.python import (
    ExternalPythonOperator,
    PythonOperator
)
step1 = importlib.import_module("tml-solutions.iotsolution-3f10.tml_system_step_1_getparams_dag-iotsolution-3f10")
step2 = importlib.import_module("tml-solutions.iotsolution-3f10.tml_system_step_2_kafka_createtopic_dag-iotsolution-3f10")
step3 = importlib.import_module("tml-solutions.iotsolution-3f10.tml_read_LOCALFILE_step_3_kafka_producetotopic_dag-iotsolution-3f10")
step4 = importlib.import_module("tml-solutions.iotsolution-3f10.tml_system_step_4_kafka_preprocess_dag-iotsolution-3f10")
step4b = importlib.import_module("tml-solutions.iotsolution-3f10.tml_system_step_4b_kafka_preprocess_dag-iotsolution-3f10")

step5 = importlib.import_module("tml-solutions.iotsolution-3f10.tml_system_step_5_kafka_machine_learning_dag-iotsolution-3f10")
step6 = importlib.import_module("tml-solutions.iotsolution-3f10.tml_system_step_6_kafka_predictions_dag-iotsolution-3f10")
step7 = importlib.import_module("tml-solutions.iotsolution-3f10.tml_system_step_7_kafka_visualization_dag-iotsolution-3f10")
step8 = importlib.import_module("tml-solutions.iotsolution-3f10.tml_system_step_8_deploy_solution_to_docker_dag-iotsolution-3f10")
step9 = importlib.import_module("tml-solutions.iotsolution-3f10.tml_system_step_9_privategpt_qdrant_dag-iotsolution-3f10")
step10 = importlib.import_module("tml-solutions.iotsolution-3f10.tml_system_step_10_documentation_dag-iotsolution-3f10")


with DAG(
    dag_id="solution_preprocessing_dag-iotsolution-3f10",
    start_date=datetime(2023, 1, 1),
    schedule=None,
) as dag:
  start_task = BashOperator(
    task_id="start_tasks_tml_preprocessing",
    bash_command="echo 'Start task'",
  )
# STEP 1: Get the Parameters
  sensor_A = PythonOperator(
            task_id="step_1_solution_task_getparams",
            python_callable=step1.getparams,
            provide_context=True,
  )

# STEP 2: Create the Kafka topics
  sensor_B = PythonOperator(
      task_id="step_2_solution_task_createtopic",
      python_callable=step2.setupkafkatopics,
      provide_context=True,
  )
# STEP 3: Produce data to topic
  sensor_C = PythonOperator(
      task_id="step_3_solution_task_producetotopic",
      python_callable=step3.startproducing,
      provide_context=True,
  )
# STEP 4: Preprocess the data
  sensor_D = PythonOperator(
      task_id="step_4_solution_task_preprocess",
      python_callable=step4.dopreprocessing,
      provide_context=True,
  )
    # STEP 4b: Preprocess the data
  sensor_D2 = PythonOperator(
      task_id="step_4b_solution_task_preprocess",
      python_callable=step4b.dopreprocessing,
      provide_context=True,
  )
# STEP 7: Containerize the solution
  sensor_E = PythonOperator(
      task_id="step_7_solution_task_visualization",
      python_callable=step7.startstreamingengine,
      provide_context=True,
  )
# STEP 8: Containerize the solution
  sensor_F = PythonOperator(
      task_id="step_8_solution_task_containerize",
      python_callable=step8.dockerit,
      provide_context=True,
  )
  start_task2 = BashOperator(
    task_id="Starting_Docker",
    bash_command="echo 'Start task Completed'",
  )
  start_task3 = BashOperator(
    task_id="Starting_Documentation",
    bash_command="echo 'Start task Completed'",
  )
  start_task4 = BashOperator(
    task_id="Completed_TML_Setup_Now_Spawn_Main_Processes",
    bash_command="echo 'Start task Completed'",
  )
# STEP 10: Document the solution
  sensor_G = PythonOperator(
      task_id="step_10_solution_task_document",
      python_callable=step10.generatedoc,
      provide_context=True,
  )

  start_task >> sensor_A >> sensor_B >> start_task4 >> [sensor_C, sensor_D, sensor_D2, sensor_E] >> start_task2 >> sensor_F >> start_task3  >> sensor_G
```

TML Dag Parameter Changes To Be Made For: solution_preprocessing_dag-iotsolution-3f10

> **Note**
>
> No TML Dag parameter changes are needed for this solution.

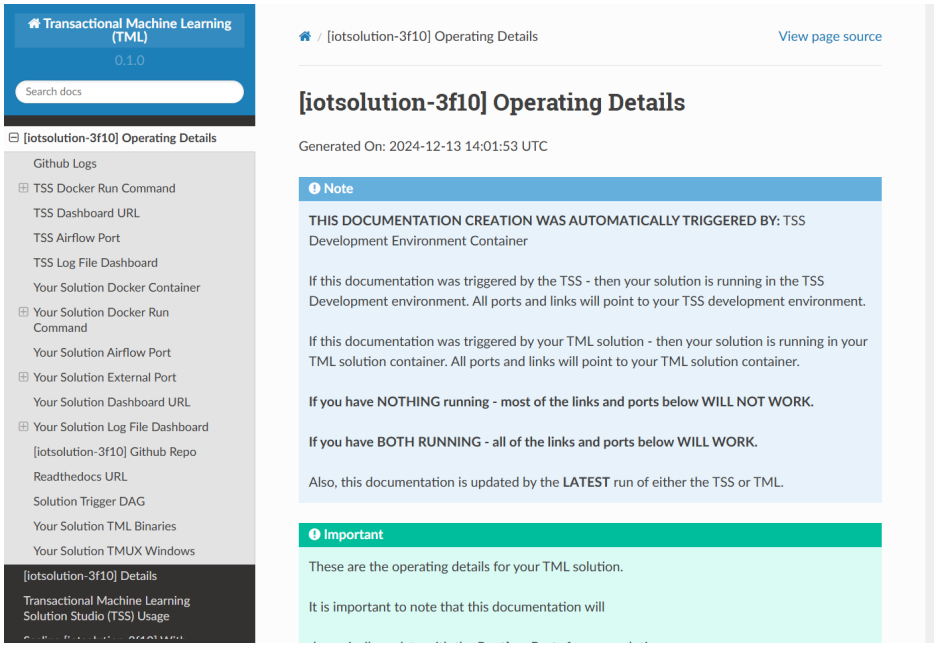
Successful Run Screen

Below the TSS/Airflow screen that shows a successful TML solution build. All colors should be green for all of the steps. If you see a red color, it means your DAG has an error.



Solution Documentation Example

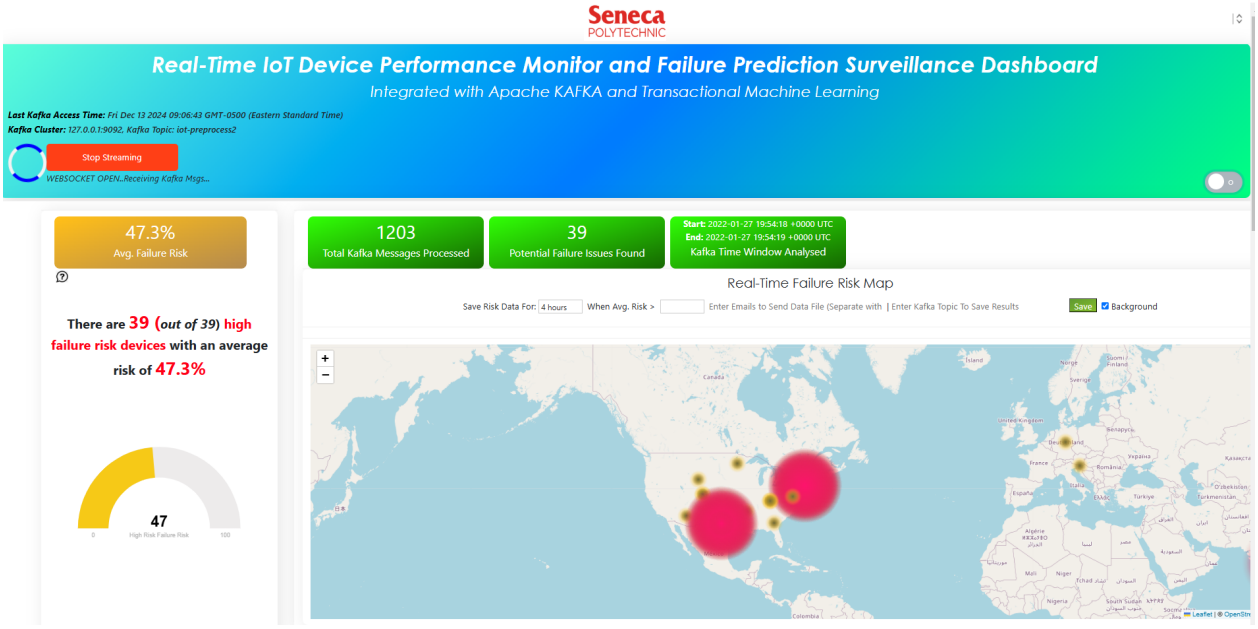
This is the solution documentation that is auto-generated by TSS. Every TML solution you create will have its own auto-generated documentation that will provide details on the entire solution.



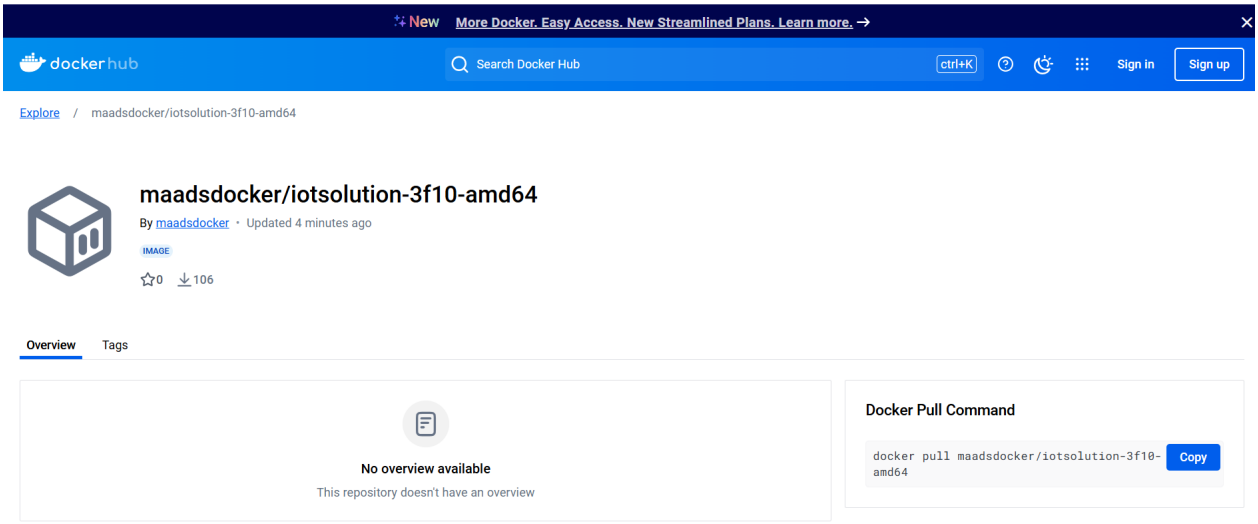
Important

You will need to run the solution in your own TSS environment for the links to work in this documentation. It is provided as an example of the powerful capabilities of TSS: https://iotsolution-3f10.readthedocs.io/

Here is the Solution Real-Time Dashboard:



Here is the Solution Docker Run container:



Real-Time IoT Data Preprocessing With RESTAPI, Secondary Preprocessing and With Map Example

:ref:`Solution DAG Code: solution_preprocessing_dag_restapi-iotsolution-restapi-3f10`

This IoT Data Preprocessing Solution DAG: **solution_preprocessing_dag_restapi-iotsolution-restapi-3f10** ingest data by REST API. Users can stream data using the client REST file into Kubernetes.

The streaming data are then processed TWICE with TML binary Viper and the output data are streamed to a browser that runs the dashboard with a map: iot-failure-seneca.html that is located in /Viperviz/viperviz/views.

The solution will automatically build and push the solution container to docker hub, automatically create documentation on READTHEDOCS.io and **automatically commits your solution code to Github, all in about 2 minutes.**

Solution DAG Code: solution_preprocessing_dag_restapi-iotsolution-restapi-3f10

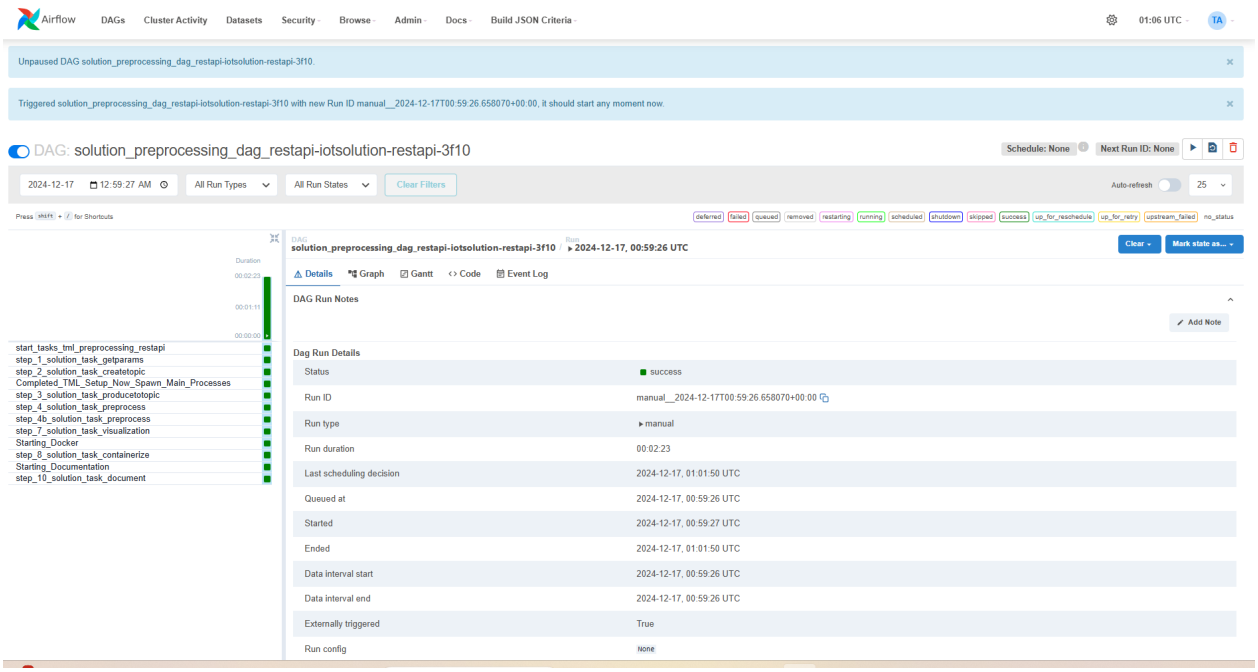
```
from __future__ import annotations

import pendulum
from airflow.decorators import task
from airflow.models.dag import DAG
from airflow.operators.bash import BashOperator
from airflow.sensors.external_task import ExternalTaskSensor
import tsslogging
import os
from datetime import datetime

import importlib
from airflow.operators.python import (
    ExternalPythonOperator,
    PythonOperator
)
step1 = importlib.import_module("tml-solutions.iotsolution-restapi-3f10.tml_system_step_1_getparams_dag-iotsolution-restapi-3f10")
step2 = importlib.import_module("tml-solutions.iotsolution-restapi-3f10.tml_system_step_2_kafka_createtopic_dag-iotsolution-restapi-3f10")
step3 = importlib.import_module("tml-solutions.iotsolution-restapi-3f10.tml_read_RESTAPI_step_3_kafka_producetotopic_dag-iotsolution-restapi-3f10")
step4 = importlib.import_module("tml-solutions.iotsolution-restapi-3f10.tml_system_step_4_kafka_preprocess_dag-iotsolution-restapi-3f10")
step4b = importlib.import_module("tml-solutions.iotsolution-restapi-3f10.tml_system_step_4b_kafka_preprocess_dag-iotsolution-restapi-3f10")

step5 = importlib.import_module("tml-solutions.iotsolution-restapi-3f10.tml_system_step_5_kafka_machine_learning_dag-iotsolution-restapi-3f10")
step6 = importlib.import_module("tml-solutions.iotsolution-restapi-3f10.tml_system_step_6_kafka_predictions_dag-iotsolution-restapi-3f10")
step7 = importlib.import_module("tml-solutions.iotsolution-restapi-3f10.tml_system_step_7_kafka_visualization_dag-iotsolution-restapi-3f10")
step8 = importlib.import_module("tml-solutions.iotsolution-restapi-3f10.tml_system_step_8_deploy_solution_to_docker_dag-iotsolution-restapi-3f10")
step9 = importlib.import_module("tml-solutions.iotsolution-restapi-3f10.tml_system_step_9_privategpt_qdrant_dag-iotsolution-restapi-3f10")
step10 = importlib.import_module("tml-solutions.iotsolution-restapi-3f10.tml_system_step_10_documentation_dag-iotsolution-restapi-3f10")

with DAG(
    dag_id="solution_preprocessing_dag_restapi-iotsolution-restapi-3f10",
    start_date=datetime(2023, 1, 1),
    schedule=None,
) as dag:
  start_task = BashOperator(
    task_id="start_tasks_tml_preprocessing_restapi",
    bash_command="echo 'Start task'",
  )
# STEP 1: Get the Parameters
  sensor_A = PythonOperator(
            task_id="step_1_solution_task_getparams",
            python_callable=step1.getparams,
            provide_context=True,
  )

# STEP 2: Create the Kafka topics
  sensor_B = PythonOperator(
      task_id="step_2_solution_task_createtopic",
      python_callable=step2.setupkafkatopics,
      provide_context=True,
  )
# STEP 3: Produce data to topic
  sensor_C = PythonOperator(
      task_id="step_3_solution_task_producetotopic",
      python_callable=step3.startproducing,
      provide_context=True,
  )
# STEP 4: Preprocess the data
  sensor_D = PythonOperator(
      task_id="step_4_solution_task_preprocess",
      python_callable=step4.dopreprocessing,
      provide_context=True,
  )
# STEP 4b: Preprocess the data
  sensor_D2 = PythonOperator(
      task_id="step_4b_solution_task_preprocess",
      python_callable=step4b.dopreprocessing,
      provide_context=True,
  )

# STEP 7: Containerize the solution
  sensor_E = PythonOperator(
      task_id="step_7_solution_task_visualization",
      python_callable=step7.startstreamingengine,
      provide_context=True,
  )
# STEP 8: Containerize the solution
  sensor_F = PythonOperator(
      task_id="step_8_solution_task_containerize",
      python_callable=step8.dockerit,
      provide_context=True,
  )
  start_task2 = BashOperator(
    task_id="Starting_Docker",
    bash_command="echo 'Start task Completed'",
  )
  start_task3 = BashOperator(
    task_id="Starting_Documentation",
    bash_command="echo 'Start task Completed'",
  )
  start_task4 = BashOperator(
    task_id="Completed_TML_Setup_Now_Spawn_Main_Processes",
    bash_command="echo 'Start task Completed'",
  )
# STEP 10: Document the solution
  sensor_G = PythonOperator(
      task_id="step_10_solution_task_document",
      python_callable=step10.generatedoc,
      provide_context=True,
  )

  start_task >> sensor_A >> sensor_B >> start_task4 >> [sensor_C, sensor_D, sensor_D2, sensor_E] >> start_task2 >> sensor_F >> start_task3  >> sensor_G
```

TML Dag Parameter Changes To Be Made For: solution_preprocessing_dag_restapi-iotsolution-restapi-3f10

> **Note**
>
> No TML Dag parameter changes are needed for this solution.

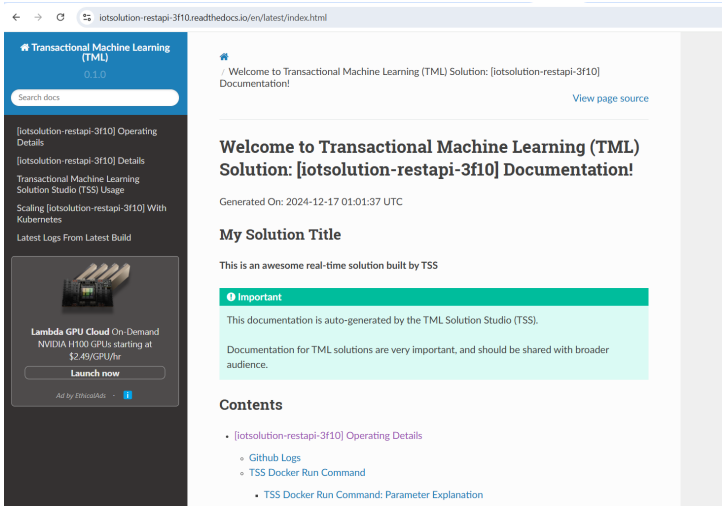
Successful Run Screen

Below the TSS/Airflow screen that shows a successful TML solution build. All colors should be green for all of the steps. If you see a red color, it means your DAG has an error.



Solution Documentation Example

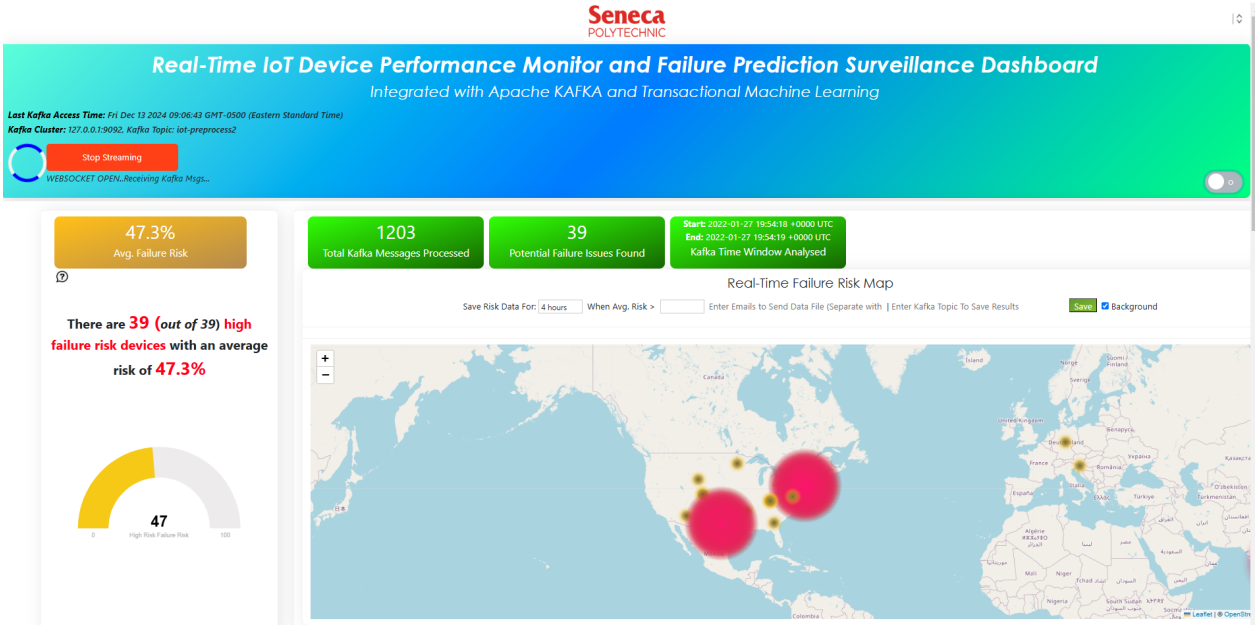
This is the solution documentation that is auto-generated by TSS. Every TML solution you create will have its own auto-generated documentation that will provide details on the entire solution.



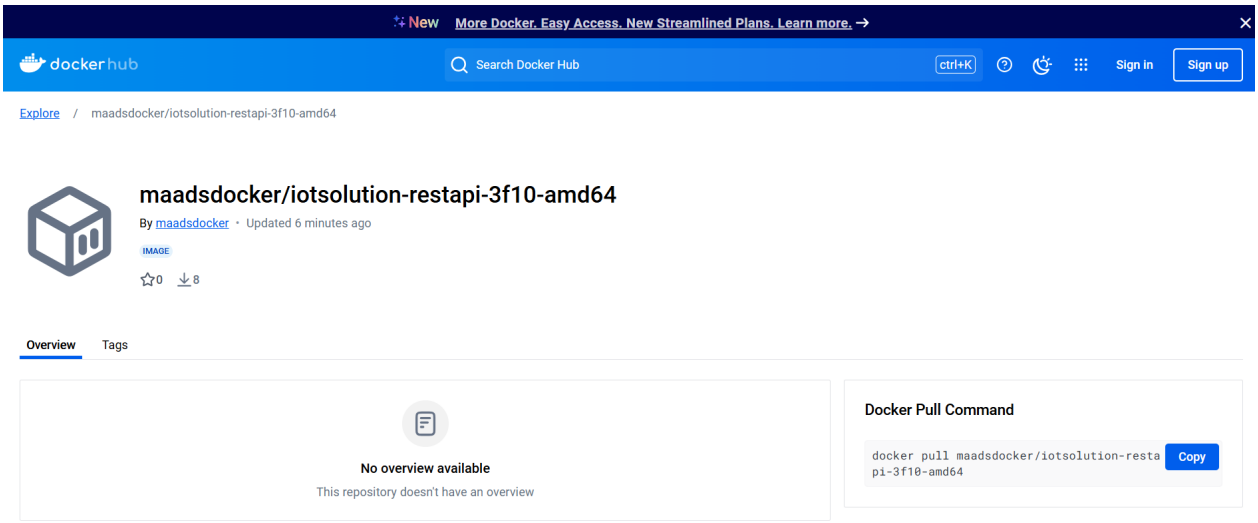
Important

You will need to run the solution in your own TSS environment for the links to work in this documentation. It is provided as an example of the powerful capabilities of TSS: `https://iotsolution-restapi-3f10.readthedocs.io/en/latest/index.html>`_

Here is the Solution Real-Time Dashboard:



Here is the Solution Docker Run container:



Real-Time IoT Data Preprocessing With gRPC, Secondary Preprocessing and With Map Example

:ref:`Solution DAG Code: solution_preprocessing_dag_grpc-iotsolution-grpc-3f10`

This IoT Data Preprocessing Solution DAG: **solution_preprocessing_dag_grpc-iotsolution-grpc-3f10** ingest data by gRPC API. Users can stream data using the client gRPC file into Kubernetes.

The streaming data are then processed TWICE with TML binary Viper and the output data are streamed to a browser that runs the dashboard with a map: iot-failure-seneca.html that is located in /Viperviz/viperviz/views.

The solution will automatically build and push the solution container to docker hub, automatically create documentation on READTHEDOCS.io and **automatically commits your solution code to Github, all in about 2 minutes.**

Solution DAG Code: solution_preprocessing_dag_grpc-iotsolution-grpc-3f10

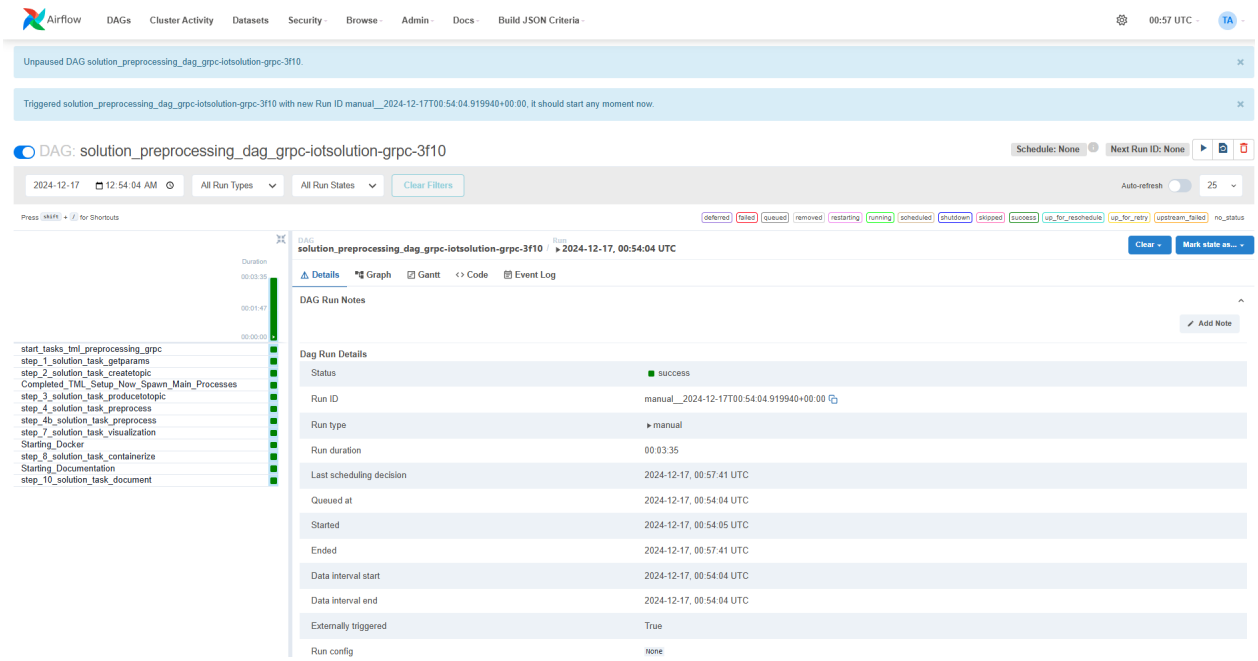
```
from __future__ import annotations

import pendulum
from airflow.decorators import task
from airflow.models.dag import DAG
from airflow.operators.bash import BashOperator
from airflow.sensors.external_task import ExternalTaskSensor
import tsslogging
import os
from datetime import datetime

import importlib
from airflow.operators.python import (
    ExternalPythonOperator,
    PythonOperator
)
step1 = importlib.import_module("tml-solutions.iotsolution-grpc-3f10.tml_system_step_1_getparams_dag-iotsolution-grpc-3f10")
step2 = importlib.import_module("tml-solutions.iotsolution-grpc-3f10.tml_system_step_2_kafka_createtopic_dag-iotsolution-grpc-3f10")
step3 = importlib.import_module("tml-solutions.iotsolution-grpc-3f10.tml_read_gRPC_step_3_kafka_producetotopic_dag-iotsolution-grpc-3f10")
step4 = importlib.import_module("tml-solutions.iotsolution-grpc-3f10.tml_system_step_4_kafka_preprocess_dag-iotsolution-grpc-3f10")
step4b = importlib.import_module("tml-solutions.iotsolution-grpc-3f10.tml_system_step_4b_kafka_preprocess_dag-iotsolution-grpc-3f10")

step5 = importlib.import_module("tml-solutions.iotsolution-grpc-3f10.tml_system_step_5_kafka_machine_learning_dag-iotsolution-grpc-3f10")
step6 = importlib.import_module("tml-solutions.iotsolution-grpc-3f10.tml_system_step_6_kafka_predictions_dag-iotsolution-grpc-3f10")
step7 = importlib.import_module("tml-solutions.iotsolution-grpc-3f10.tml_system_step_7_kafka_visualization_dag-iotsolution-grpc-3f10")
step8 = importlib.import_module("tml-solutions.iotsolution-grpc-3f10.tml_system_step_8_deploy_solution_to_docker_dag-iotsolution-grpc-3f10")
step9 = importlib.import_module("tml-solutions.iotsolution-grpc-3f10.tml_system_step_9_privategpt_qdrant_dag-iotsolution-grpc-3f10")
step10 = importlib.import_module("tml-solutions.iotsolution-grpc-3f10.tml_system_step_10_documentation_dag-iotsolution-grpc-3f10")

with DAG(
    dag_id="solution_preprocessing_dag_grpc-iotsolution-grpc-3f10",
    start_date=datetime(2023, 1, 1),
    schedule=None,
) as dag:
  start_task = BashOperator(
    task_id="start_tasks_tml_preprocessing_grpc",
    bash_command="echo 'Start task'",
  )
# STEP 1: Get the Parameters
  sensor_A = PythonOperator(
            task_id="step_1_solution_task_getparams",
            python_callable=step1.getparams,
            provide_context=True,
  )

# STEP 2: Create the Kafka topics
  sensor_B = PythonOperator(
      task_id="step_2_solution_task_createtopic",
      python_callable=step2.setupkafkatopics,
      provide_context=True,
  )
# STEP 3: Produce data to topic
  sensor_C = PythonOperator(
      task_id="step_3_solution_task_producetotopic",
      python_callable=step3.startproducing,
      provide_context=True,
  )
# STEP 4: Preprocess the data
  sensor_D = PythonOperator(
      task_id="step_4_solution_task_preprocess",
      python_callable=step4.dopreprocessing,
      provide_context=True,
  )
# STEP 4b: Preprocess the data
  sensor_D2 = PythonOperator(
      task_id="step_4b_solution_task_preprocess",
      python_callable=step4b.dopreprocessing,
      provide_context=True,
  )

# STEP 7: Containerize the solution
  sensor_E = PythonOperator(
      task_id="step_7_solution_task_visualization",
      python_callable=step7.startstreamingengine,
      provide_context=True,
  )
# STEP 8: Containerize the solution
  sensor_F = PythonOperator(
      task_id="step_8_solution_task_containerize",
      python_callable=step8.dockerit,
      provide_context=True,
  )
  start_task2 = BashOperator(
    task_id="Starting_Docker",
    bash_command="echo 'Start task Completed'",
  )
  start_task3 = BashOperator(
    task_id="Starting_Documentation",
    bash_command="echo 'Start task Completed'",
  )
  start_task4 = BashOperator(
    task_id="Completed_TML_Setup_Now_Spawn_Main_Processes",
    bash_command="echo 'Start task Completed'",
  )
# STEP 10: Document the solution
  sensor_G = PythonOperator(
      task_id="step_10_solution_task_document",
      python_callable=step10.generatedoc,
      provide_context=True,
  )

  start_task >> sensor_A >> sensor_B >> start_task4 >> [sensor_C, sensor_D, sensor_D2, sensor_E] >> start_task2 >> sensor_F >> start_task3  >> sensor_G
```

TML Dag Parameter Changes To Be Made For: solution_preprocessing_dag_grpc-iotsolution-grpc-3f10

> **Note**
>
> No TML Dag parameter changes are needed for this solution.

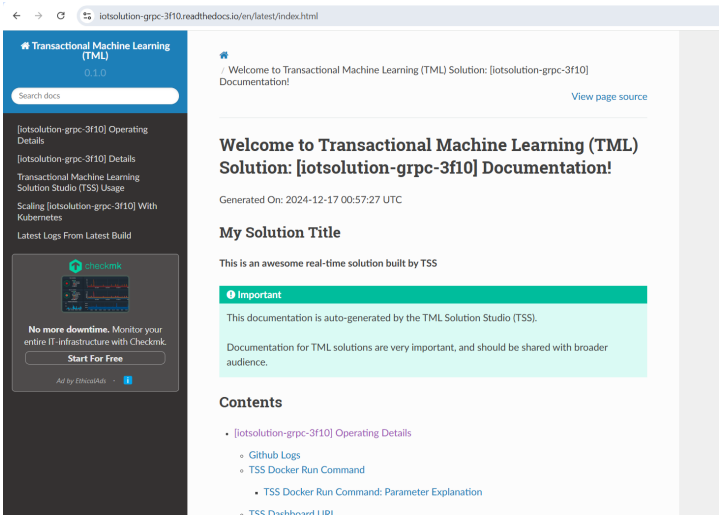
Successful Run Screen

Below the TSS/Airflow screen that shows a successful TML solution build. All colors should be green for all of the steps. If you see a red color, it means your DAG has an error.



Solution Documentation Example

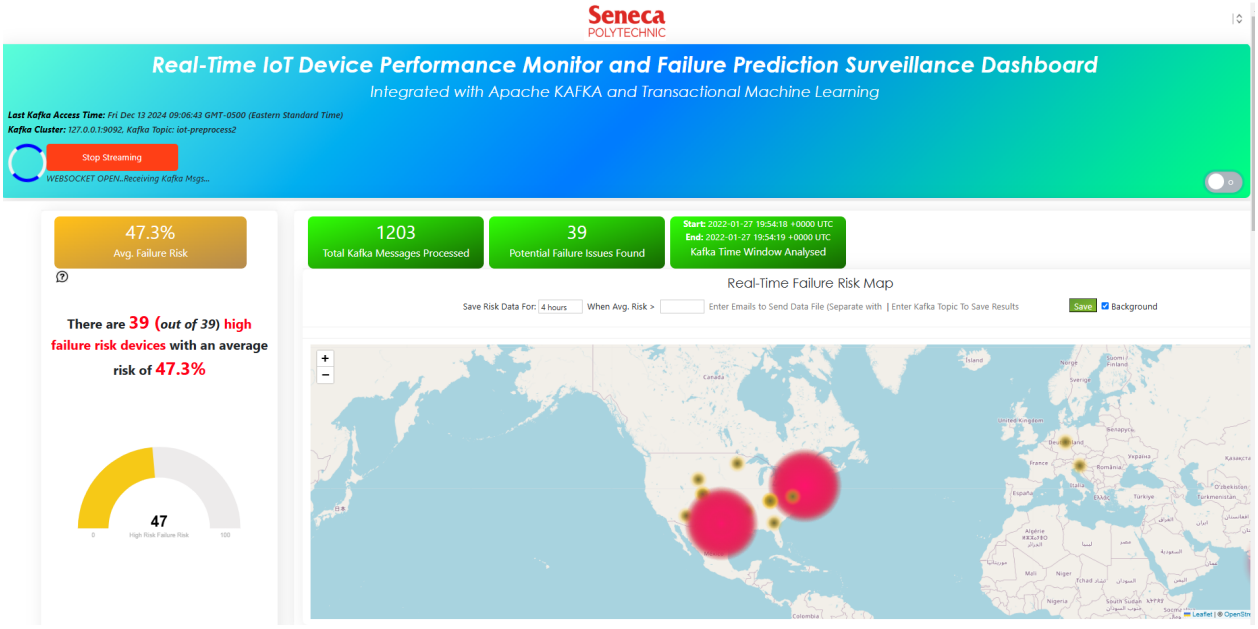
This is the solution documentation that is auto-generated by TSS. Every TML solution you create will have its own auto-generated documentation that will provide details on the entire solution.



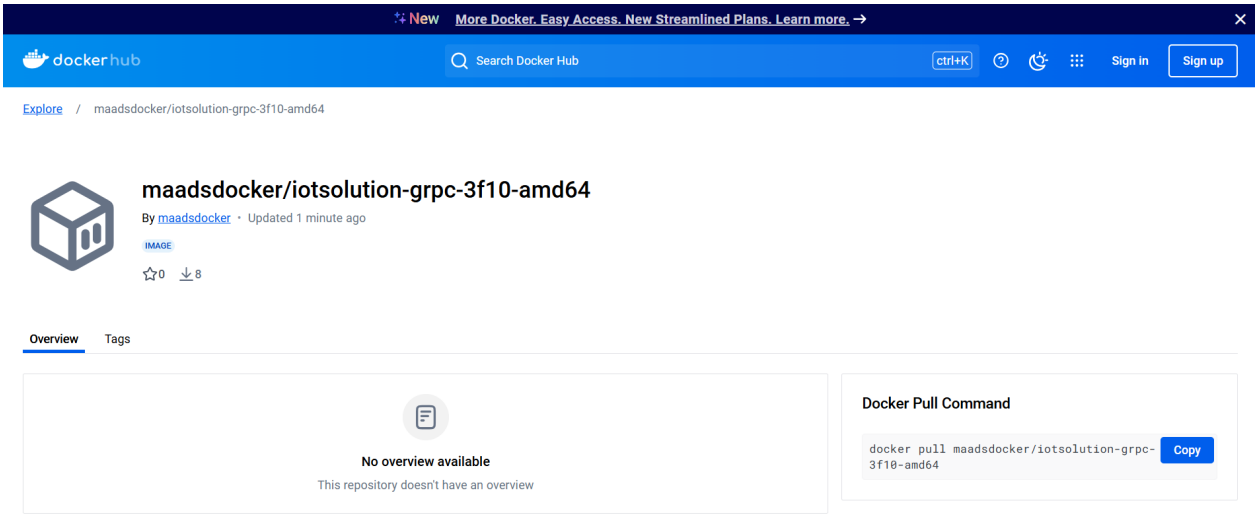
Important

You will need to run the solution in your own TSS environment for the links to work in this documentation. It is provided as an example of the powerful capabilities of TSS: `https://iotsolution-grpc-3f10.readthedocs.io/en/latest/index.html>`_

Here is the Solution Real-Time Dashboard:



Here is the Solution Docker Run container:



Real-Time IoT Data Preprocessing and Machine Learning Example

:ref:`Solution DAG Code: solution_preprocessing_ml_dag-myawesometmlsolutionml-3f10`

This IoT Data Preprocessing and Machine Learning Solution DAG: **solution_preprocessing_ml_dag-myawesometmlsolutionml-3f10** reads local file data in /rawdata/IoTdata.txt and streams it to Kafka. **The streaming data are then processed and entity level**

machine learning is performed with TML binaries Viper and HPDE, the output data are streamed to a browser that runs the dashboard: iot-failure-machinelearning.html, that is located in /Viperviz/viperviz/views.

The solution will automatically build and push the solution container to docker hub, automatically create documentation on READTHEDOCS.io and automatically commit your solution code to Github, all in about 2 minutes.

Solution DAG Code: solution_preprocessing_ml_dag-myawesometmlsolutionml-3f10

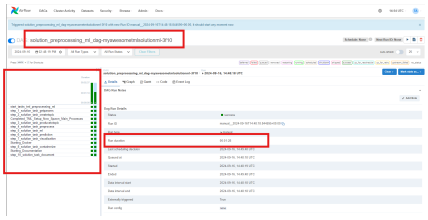
```
from __future__ import annotations

import pendulum
from airflow.decorators import task
from airflow.models.dag import DAG
from airflow.operators.bash import BashOperator
from airflow.sensors.external_task import ExternalTaskSensor
import tsslogging
import os
from datetime import datetime
import importlib
from airflow.operators.python import (
    ExternalPythonOperator,
    PythonOperator
)
step1 = importlib.import_module("tml-solutions.myawesometmlsolutionml-3f10.tml_system_step_1_getparams_dag-myawesometmlsolutionml-3f10")
step2 = importlib.import_module("tml-solutions.myawesometmlsolutionml-3f10.tml_system_step_2_kafka_createtopic_dag-myawesometmlsolutionml-3f10")
step3 = importlib.import_module("tml-solutions.myawesometmlsolutionml-3f10.tml_read_LOCALFILE_step_3_kafka_producetotopic_dag-myawesometmlsolutionml-3f10")
step4 = importlib.import_module("tml-solutions.myawesometmlsolutionml-3f10.tml_system_step_4_kafka_preprocess_dag-myawesometmlsolutionml-3f10")
step5 = importlib.import_module("tml-solutions.myawesometmlsolutionml-3f10.tml_system_step_5_kafka_machine_learning_dag-myawesometmlsolutionml-3f10")
step6 = importlib.import_module("tml-solutions.myawesometmlsolutionml-3f10.tml_system_step_6_kafka_predictions_dag-myawesometmlsolutionml-3f10")
step7 = importlib.import_module("tml-solutions.myawesometmlsolutionml-3f10.tml_system_step_7_kafka_visualization_dag-myawesometmlsolutionml-3f10")
step8 = importlib.import_module("tml-solutions.myawesometmlsolutionml-3f10.tml_system_step_8_deploy_solution_to_docker_dag-myawesometmlsolutionml-3f10")
step9 = importlib.import_module("tml-solutions.myawesometmlsolutionml-3f10.tml_system_step_9_privategpt_qdrant_dag-myawesometmlsolutionml-3f10")
step10 = importlib.import_module("tml-solutions.myawesometmlsolutionml-3f10.tml_system_step_10_documentation_dag-myawesometmlsolutionml-3f10")

with DAG(
    dag_id="solution_preprocessing_ml_dag-myawesometmlsolutionml-3f10",
    start_date=datetime(2023, 1, 1),
    schedule=None,
) as dag:
  start_task = BashOperator(
    task_id="start_tasks_tml_preprocessing_ml",
    bash_command="echo 'Start task'",
  )
# STEP 1: Get the Parameters
  sensor_A = PythonOperator(
            task_id="step_1_solution_task_getparams",
            python_callable=step1.getparams,
            provide_context=True,
  )

# STEP 2: Create the Kafka topics
  sensor_B = PythonOperator(
      task_id="step_2_solution_task_createtopic",
      python_callable=step2.setupkafkatopics,
      provide_context=True,
  )
# STEP 3: Produce data to topic
  sensor_C = PythonOperator(
      task_id="step_3_solution_task_producetotopic",
      python_callable=step3.startproducing,
      provide_context=True,
  )
# STEP 4: Preprocess the data
  sensor_D = PythonOperator(
      task_id="step_4_solution_task_preprocess",
      python_callable=step4.dopreprocessing,
      provide_context=True,
  )
# STEP 5: ML
  sensor_E = PythonOperator(
      task_id="step_5_solution_task_ml",
      python_callable=step5.startml,
      provide_context=True,
  )
# STEP 6: Predictions
  sensor_F = PythonOperator(
      task_id="step_6_solution_task_prediction",
      python_callable=step6.startpredictions,
      provide_context=True,
  )

# STEP 7: Visualization the solution
  sensor_G = PythonOperator(
      task_id="step_7_solution_task_visualization",
      python_callable=step7.startstreamingengine,
      provide_context=True,
  )
# STEP 8: Containerize the solution
  sensor_H = PythonOperator(
      task_id="step_8_solution_task_containerize",
      python_callable=step8.dockerit,
      provide_context=True,
  )
  start_task2 = BashOperator(
    task_id="Starting_Docker",
    bash_command="echo 'Start task Completed'",
  )
  start_task3 = BashOperator(
    task_id="Starting_Documentation",
    bash_command="echo 'Start task Completed'",
  )
  start_task4 = BashOperator(
    task_id="Completed_TML_Setup_Now_Spawn_Main_Processes",
    bash_command="echo 'Start task Completed'",
  )
# STEP 10: Document the solution
  sensor_J = PythonOperator(
      task_id="step_10_solution_task_document",
      python_callable=step10.generatedoc,
      provide_context=True,
  )

  start_task >> sensor_A >> sensor_B >> start_task4 >> [sensor_C, sensor_D, sensor_E, sensor_F, sensor_G] >> start_task2 >> sensor_H >> start_task3 >> sensor_J
```

TML Dag Parameter Changes To Be Made For:
solution_preprocessing_ml_dag-myawesometmlsolutionml-3f10

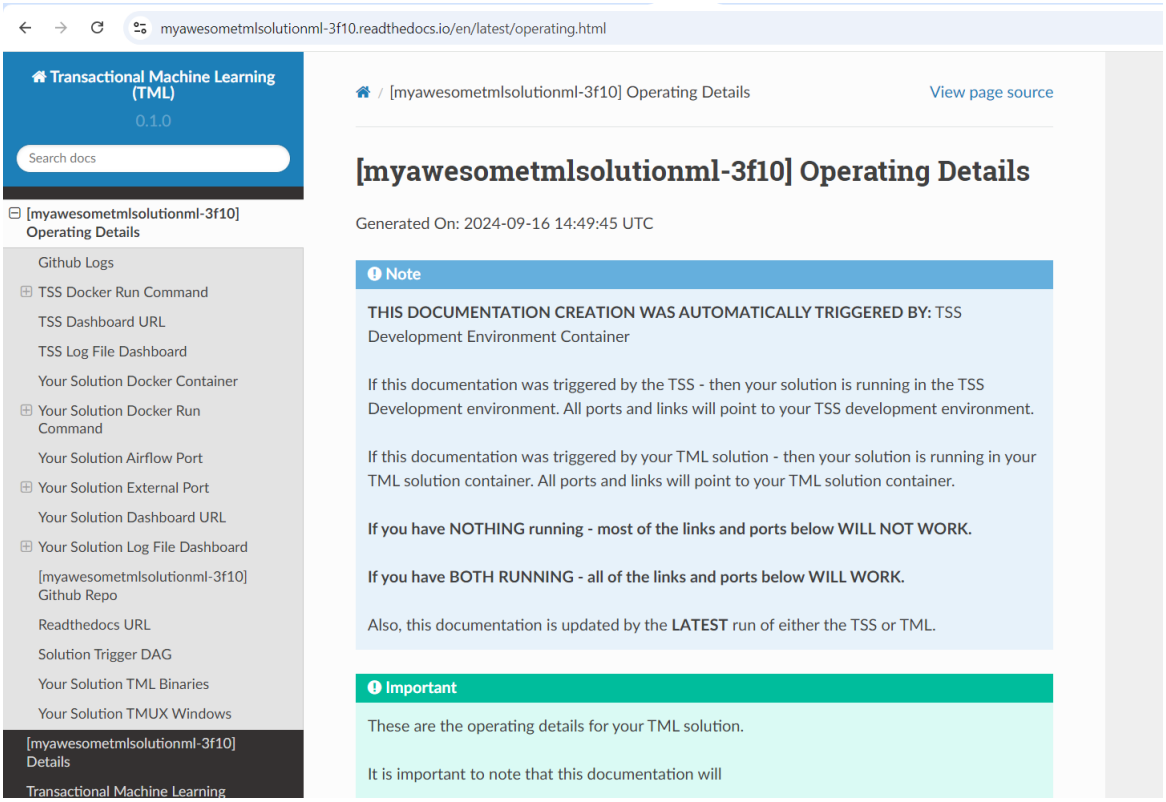
> Tip
>
> This is the same that is located here: solution_preprocessing_ml_dag-myawesometmlsolutionml-3f10

TML Dag Default_args Parameter To Change To New Value
TML Dag: tml_system_step_2_kafka_createtopic_dag-myawesometmlsolutionml-3f10.py **Current Value:** 'numpartitions': '1' **New Value:** 'numpartitions': '3'
TML Dag: tml_system_step_5_kafka_machine_learning_dag-myawesometmlsolutionml-3f10.py **Current Value:** 'islogistic' : '0' **New Value:** 'islogistic' : '1'
TML Dag: tml_system_step_5_kafka_machine_learning_dag-myawesometmlsolutionml-3f10.py **Current Value:** 'dependentvariable' : '' **New Value:** 'dependentvariable' : 'failure'
TML Dag: tml_system_step_5_kafka_machine_learning_dag-myawesometmlsolutionml-3f10.py **Current Value:** 'independentvariables': '' **New Value:** 'independentvariables': 'Power_preprocessed_Trend,Voltage_preprocessed_Trend,Current_preprocessed_Trend'
TML Dag: tml_system_step_5_kafka_machine_learning_dag-myawesometmlsolutionml-3f10.py **Current Value:** 'fullpathtotrainingdata' : '/Viper-ml/viperlogs/<choose foldername>', **New Value:** 'fullpathtotrainingdata' : '/Viper-ml/viperlogs/iotlogistic'
TML Dag: tml_system_step_5_kafka_machine_learning_dag-myawesometmlsolutionml-3f10.py **Current Value:** 'processlogic' : '' **New Value:** processlogic': 'classification_name=failure_prob:Power_preprocessed_Trend=-n,0:Voltage_preprocessed_Trend=-n,0:Current_preprocessed_Trend=-n,0'
TML Dag: tml_system_step_6_kafka_predictions_dag-myawesometmlsolutionml-3f10.py **Current Value:** 'consumefrom' : '' **New Value:** 'consumefrom' : 'ml-data'
TML Dag: tml_system_step_6_kafka_predictions_dag-myawesometmlsolutionml-3f10.py **Current Value:** 'pathtoalgos' : '/Viper-ml/viperlogs/<choose foldername>' **New Value:** 'pathtoalgos' : '/Viper-ml/viperlogs/iotlogistic'
TML Dag: tml_system_step_7_kafka_visualization_dag-myawesometmlsolutionml-3f10.py **Current Value:** 'topic' : 'iot-preprocess,iot-preprocess2' **New Value:** 'topic' : 'iot-preprocess,iot-ml-prediction-results-output'
TML Dag: tml_system_step_7_kafka_visualization_dag-myawesometmlsolutionml-3f10.py **Current Value:** 'dashboardhtml': 'dashboard.html' **New Value:** 'dashboardhtml': 'iot-failure-machinelearning.html'

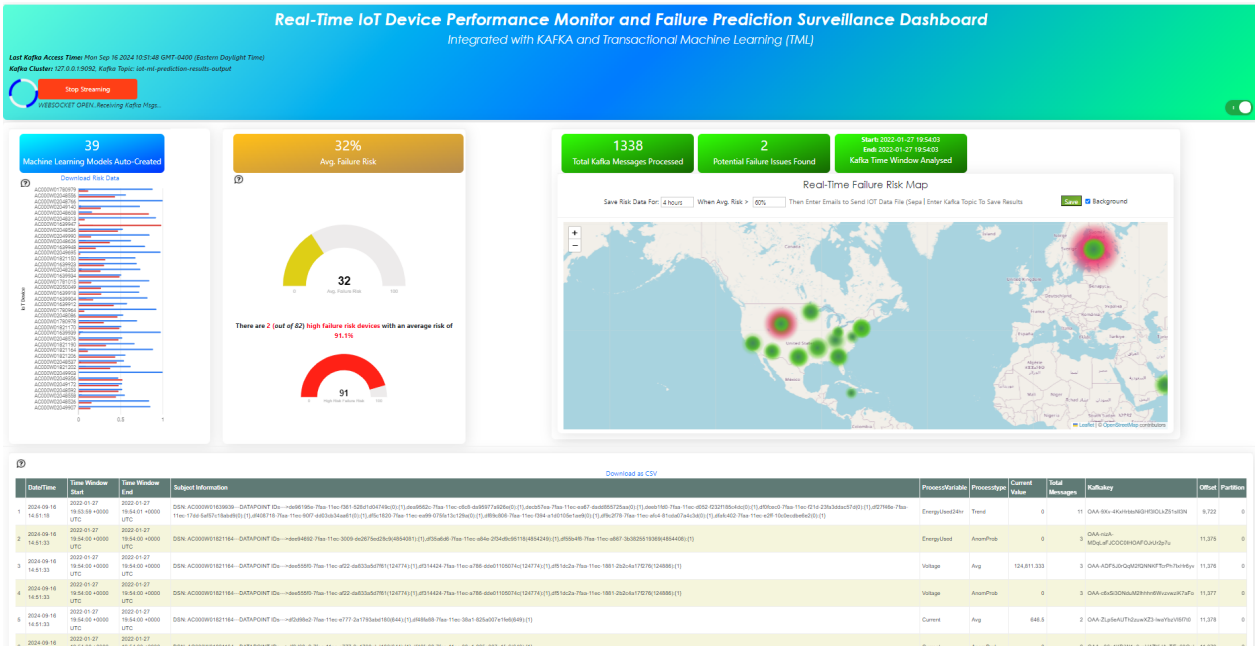
Here is the TSS successful run:



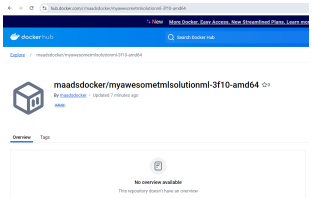
Here is the automated readthedocs documentation



This is the real-time dashboard generated:



Here is the docker container that was automatically built and pushed to Docker hub:



Cybersecurity Solution with PrivateGPT, MQTT, HiveMQ

:ref:`Solution DAG Code: solution_preprocessing_ai_mqtt_dag-cybersecuritywithprivategpt-3f10`

This Cybersecurity Data Preprocessing with GenAI Solution DAG: **solution_preprocessing_ai_dag-cybersecuritysolutionwithprivategpt-3f10** reads local file data in /rawdata/cisco_network_data.txt and streams it to Kafka. **The streaming data are then processed, the processed output data sent to the privateGPT container and Qdrant vector DB for further analysis.** Processing is done by Viper and AI is performed by privateGPT, the output data are streamed to a browser that runs the dashboard: tml-cisco-network-privategpt-monitor.html, that is located in /Viperviz/viperviz/views.

The solution will automatically build and push the solution container to docker hub, automatically create documentation on READTHEDOCS.io and automatically commit your solution code to Github, all in about 2 minutes.

Note also the solution will start the privateGPT and Qdrant containers automatically for you.

Solution DAG Code: solution_preprocessing_ai_mqtt_dag-cybersecuritywithprivategpt-3f10

```
from __future__ import annotations

import pendulum
from airflow.decorators import task
from airflow.models.dag import DAG
from airflow.operators.bash import BashOperator
from airflow.sensors.external_task import ExternalTaskSensor
import tsslogging
import os
from datetime import datetime
import importlib
from airflow.operators.python import (
    ExternalPythonOperator,
    PythonOperator
)
step1 = importlib.import_module("tml-solutions.cybersecuritywithprivategpt-3f10.tml_system_step_1_getparams_dag-cybersecuritywithprivategpt-3f10")
step2 = importlib.import_module("tml-solutions.cybersecuritywithprivategpt-3f10.tml_system_step_2_kafka_createtopic_dag-cybersecuritywithprivategpt-3f10")
step3 = importlib.import_module("tml-solutions.cybersecuritywithprivategpt-3f10.tml_read_MQTT_step_3_kafka_producetotopic_dag-cybersecuritywithprivategpt-3f10")
step4 = importlib.import_module("tml-solutions.cybersecuritywithprivategpt-3f10.tml_system_step_4_kafka_preprocess_dag-cybersecuritywithprivategpt-3f10")
step5 = importlib.import_module("tml-solutions.cybersecuritywithprivategpt-3f10.tml_system_step_5_kafka_machine_learning_dag-cybersecuritywithprivategpt-3f10")
step6 = importlib.import_module("tml-solutions.cybersecuritywithprivategpt-3f10.tml_system_step_6_kafka_predictions_dag-cybersecuritywithprivategpt-3f10")
step7 = importlib.import_module("tml-solutions.cybersecuritywithprivategpt-3f10.tml_system_step_7_kafka_visualization_dag-cybersecuritywithprivategpt-3f10")
step8 = importlib.import_module("tml-solutions.cybersecuritywithprivategpt-3f10.tml_system_step_8_deploy_solution_to_docker_dag-cybersecuritywithprivategpt-3f10")
step9 = importlib.import_module("tml-solutions.cybersecuritywithprivategpt-3f10.tml_system_step_9_privategpt_qdrant_dag-cybersecuritywithprivategpt-3f10")
step10 = importlib.import_module("tml-solutions.cybersecuritywithprivategpt-3f10.tml_system_step_10_documentation_dag-cybersecuritywithprivategpt-3f10")


with DAG(
    dag_id="solution_preprocessing_ai_mqtt_dag-cybersecuritywithprivategpt-3f10",
    start_date=datetime(2023, 1, 1),
    schedule=None,
) as dag:
  start_task = BashOperator(
    task_id="start_tasks_tml_preprocessing_ai_mqtt",
    bash_command="echo 'Start task'",
  )
# STEP 1: Get the Parameters
  sensor_A = PythonOperator(
            task_id="step_1_solution_task_getparams",
            python_callable=step1.getparams,
            provide_context=True,
  )

# STEP 2: Create the Kafka topics
  sensor_B = PythonOperator(
      task_id="step_2_solution_task_createtopic",
      python_callable=step2.setupkafkatopics,
      provide_context=True,
  )
# STEP 3: Produce data to topic
  sensor_C = PythonOperator(
      task_id="step_3_solution_task_producetotopic",
      python_callable=step3.startproducing,
      provide_context=True,
  )
# STEP 4: Preprocess the data
  sensor_D = PythonOperator(
      task_id="step_4_solution_task_preprocess",
      python_callable=step4.dopreprocessing,
      provide_context=True,
  )
# STEP 7: Containerize the solution
  sensor_E = PythonOperator(
      task_id="step_7_solution_task_visualization",
      python_callable=step7.startstreamingengine,
      provide_context=True,
  )
# STEP 8: Containerize the solution
  sensor_F = PythonOperator(
      task_id="step_8_solution_task_containerize",
      python_callable=step8.dockerit,
      provide_context=True,
  )
# STEP 9: PrivateGPT
  sensor_I = PythonOperator(
      task_id="step_9_solution_task_ai",
      python_callable=step9.startprivategpt,
      provide_context=True,
  )
  start_task2 = BashOperator(
    task_id="Starting_Docker",
    bash_command="echo 'Start task Completed'",
  )
  start_task3 = BashOperator(
    task_id="Starting_Documentation",
    bash_command="echo 'Start task Completed'",
  )
  start_task4 = BashOperator(
    task_id="Completed_TML_Setup_Now_Spawn_Main_Processes",
    bash_command="echo 'Start task Completed'",
  )
# STEP 10: Document the solution
  sensor_G = PythonOperator(
      task_id="step_10_solution_task_document",
      python_callable=step10.generatedoc,
      provide_context=True,
  )

  start_task >> sensor_A >> sensor_B  >> start_task4 >> [sensor_I, sensor_C, sensor_D, sensor_E] >> start_task2 >> sensor_F >> start_task3  >> sensor_G
```

TML Dag Parameter Changes To Be Made For: solution_preprocessing_ai_mqtt_dag-cybersecuritywithprivategpt-3f10

Tip

This is the same that is located here: solution_preprocessing_ai_mqtt_dag-cybersecuritywithprivategpt-3f10

TML Dag Default_args Parameter To Change To New Value
TML Dag: tml_system_step_2_kafka_createtopic_dag-cybersecuritywithprivategpt-3f10.py **Current Value:** 'raw_data_topic' : 'iot-raw-data' **New Value:** 'raw_data_topic' : 'cisco-network-mainstream'
TML Dag: tml_system_step_2_kafka_createtopic_dag-cybersecuritywithprivategpt-3f10.py **Current Value:** 'preprocess_data_topic' : 'iot-preprocess,iot-preprocess2' **New Value:** 'preprocess_data_topic' : 'cisco-network-preprocess'
TML Dag: tml_read_MQTT_step_3_kafka_producetotopic_dag-cybersecuritywithprivategpt-3f10.py **Current Value:** 'topics' : 'iot-raw-data' **New Value:** 'topics' : 'cisco-network-mainstream'
TML Dag: tml_read_MQTT_step_3_kafka_producetotopic_dag-cybersecuritywithprivategpt-3f10.py **Current Value:** 'mqtt_broker' : '' **New Value:** 'mqtt_broker' : '<ENTER YOUR HIVEMQ BROKER>' For example - HIVEMQ broker should look similar to this: **b526253c5560459da5337e561c142369.s1.eu.hivemq.cloud**
TML Dag: tml_read_MQTT_step_3_kafka_producetotopic_dag-cybersecuritywithprivategpt-3f10.py **Current Value:** 'mqtt_port' : '' **New Value:** 'mqtt_port' : '8883',
TML Dag: tml_read_MQTT_step_3_kafka_producetotopic_dag-cybersecuritywithprivategpt-3f10.py **Current Value:** 'mqtt_subscribe_topic' : '' **New Value:** 'mqtt_subscribe_topic' : 'tml/cybersecurity'
TML Dag: tml_read_MQTT_step_3_kafka_producetotopic_dag-cybersecuritywithprivategpt-3f10.py **Current Value:** 'mqtt_enabletls': '0' **New Value:** 'mqtt_enabletls': '1'
TML Dag: tml_system_step_4_kafka_preprocess_dag-cybersecuritywithprivategpt-3f10.py **Curent Value:** 'raw_data_topic' : 'iot-raw-data' **New Value:** 'raw_data_topic' : 'cisco-network-mainstream'
TML Dag: tml_system_step_4_kafka_preprocess_dag-cybersecuritywithprivategpt-3f10.py **Current Value:** 'preprocess_data_topic' : 'iot-preprocess' **New Value:** 'preprocess_data_topic' : 'cisco-network-preprocess'
TML Dag: tml_system_step_4_kafka_preprocess_dag-cybersecuritywithprivategpt-3f10.py **Current Value:** 'identifier' : 'IoT device performance and failures' **New Value:** 'identifier' : 'Detect potential cyber attacks and monitor network'
TML Dag: tml_system_step_4_kafka_preprocess_dag-cybersecuritywithprivategpt-3f10.py **Current Value:** 'preprocesstypes' : 'anomprob,trend,avg' **New Value:** 'preprocesstypes' : 'min,max,trend,anomprob,variance,avg'

TML Dag: tml_system_step_4_kafka_preprocess_dag-cybersecuritywithprivategpt-3f10.py
Current Value: 'jsoncriteria' : 'uid=metadata.dsn,filter:allrecords~
subtopics=metadata.property_name~
values=datapoint.value~
identifiers=metadata.display_name~
datetime=datapoint.updated_at~
msgid=datapoint.id~
latlong=lat:long',
New Value: 'jsoncriteria' : 'uid=hostName,filter:allrecords~
subtopics=hostName,hostName,hostName~
values=inboundpackets,outboundpackets,pingStatus~
identifiers=inboundpackets,outboundpackets,pingStatus~
datetime=lastUpdated~
msgid=~
latlong=',

TML Dag: tml_system_step_7_kafka_visualization_dag-cybersecuritywithprivategpt-3f10.py
Current Value: 'topic' : 'iot-preprocess,iot-preprocess2'
New Value: 'topic' : 'cisco-network-preprocess,cisco-network-privategpt'

TML Dag: tml_system_step_7_kafka_visualization_dag-cybersecuritywithprivategpt-3f10.py
Current Value: 'dashboardhtml' : 'dashboard.html'
New Value: 'dashboardhtml': 'tml-cisco-network-privategpt-monitor.html'

TML Dag: tml_system_step_9_privategpt_qdrant_dag-cybersecuritywithprivategpt-3f10.py
NOTE: THIS CHANGE IS ONLY FOR USERS WITHOUT ACCESS TO A NVIDIA GPU CARD
Current Value: 'pgptcontainername' : 'maadsdocker/tml-privategpt-with-gpu-nvidia-amd64',
New Value: 'pgptcontainername' : 'maadsdocker/tml-privategpt-no-gpu-amd64'

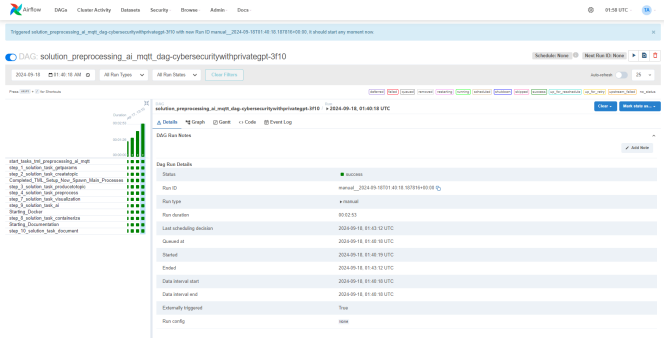
TML Dag: tml_system_step_9_privategpt_qdrant_dag-cybersecuritywithprivategpt-3f10.py
Current Value: 'consumefrom' : 'iot-preprocess',
New Value: 'consumefrom' : 'cisco-network-preprocess'

TML Dag: tml_system_step_9_privategpt_qdrant_dag-cybersecuritywithprivategpt-3f10.py
Current Value: 'prompt': 'Do the device data show any malfunction or defects?'
New Value: 'prompt': 'Do any of the values of the inbound or outbound packets look abnormal?'

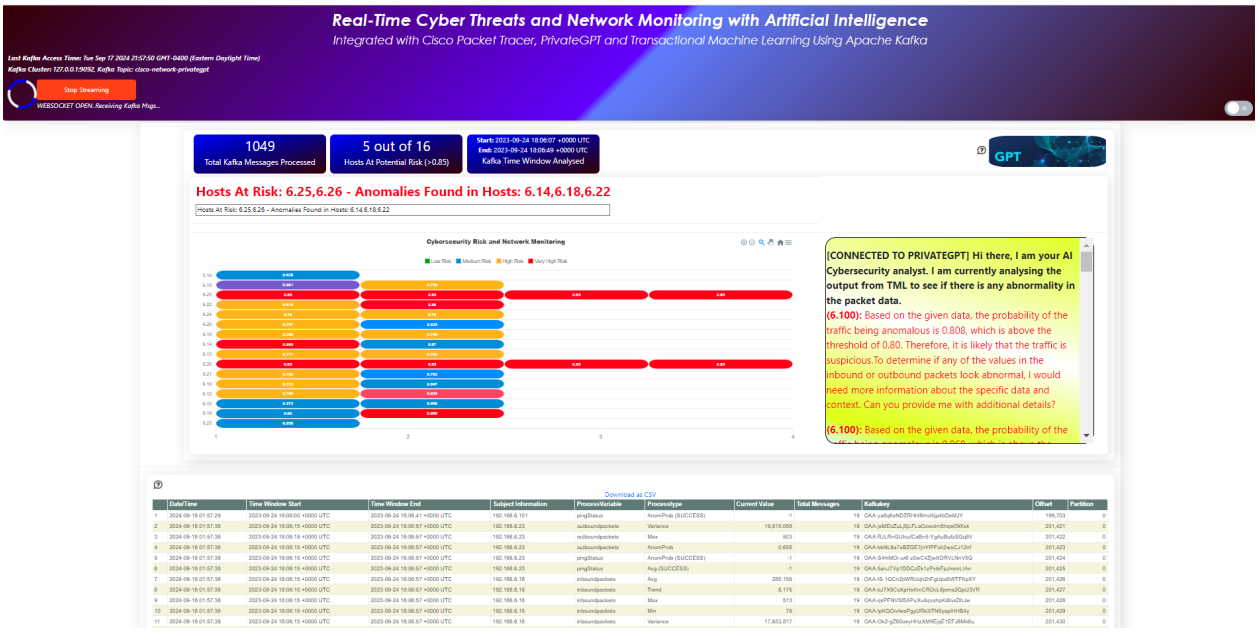
TML Dag: tml_system_step_9_privategpt_qdrant_dag-cybersecuritywithprivategpt-3f10.py
Current Value: 'context' : 'This is IoT data from devices. The data are
anomaly probabilities for each IoT device. If voltage or current
probabilities are low, it is likely the device is not working properly.'
New Value: 'context' : 'These data are anomaly probabilities of suspicious data traffic.
A high probability of over 0.80 is likely suspicious.'

TML Dag: tml_system_step_9_privategpt_qdrant_dag-cybersecuritywithprivategpt-3f10.py
Current Value: 'keyattribute' : 'Voltage,current'
New Value: 'keyattribute' : 'outboundpackets,inboundpackets'

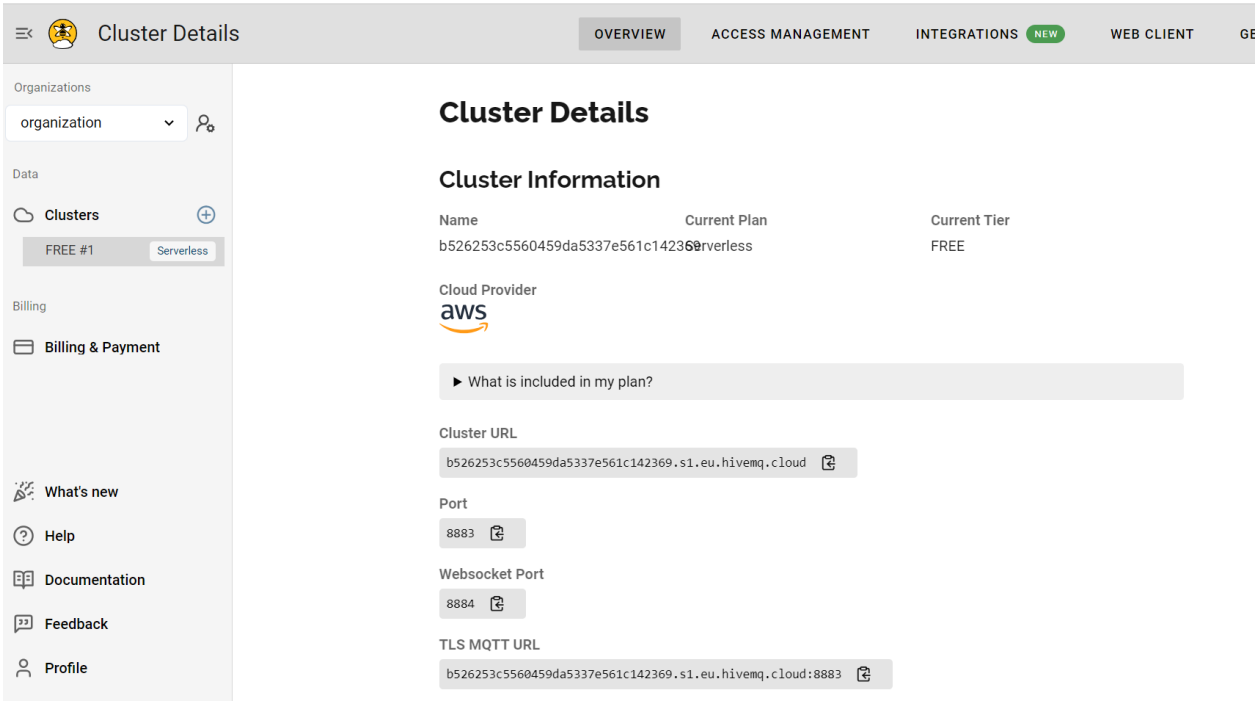
DAG Successful Run



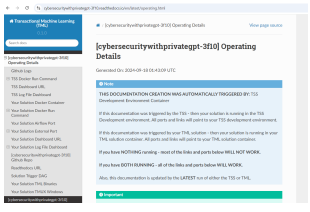
The Dashboard with PrivateGPT



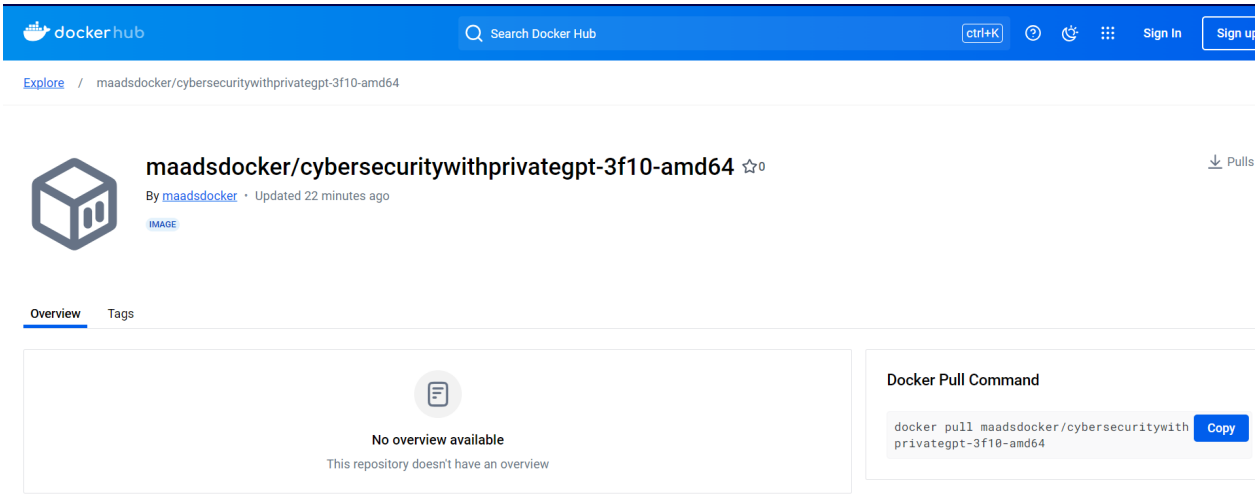
The HiveMQ Cluster



Solution Documentation



Solution Docker Container



TML and Generative AI

TML uses privateGPT containers (discussed below) for secure, fast, and distributed, AI.

Attention!

1. **These containers are dependent on the NVidia GPU cards up to 5090.**
2. Containers are compatible with CUDA versions upto 12.8.
3. Containers will run on AMD64 and ARM64 chip architectures.
4. They also require Qdrant Vector DB - Here is the Qdrant Docker Run Command to Install Qdrant Vector DB locally with TML integration:

```
docker run -d -p 6333:6333 -v $(pwd)/qdrant_storage:/qdrant/storage:z qdrant/qdrant
```

Note

TML Solution developers have several privateGPT container options. These range from the small to large models with varying GPU memory requirements.

These models and containers are listed in the table below.

Tip

These models all follow a llama2 prompt style. See here for more details.

PrivateGPT Special Containers

TML-privateGPT Container	GPU Suggested Requirements

AMD64: Basic Model Version 1

- ARM64 container
- LLM: TheBloke/Mistral-7B-Instruct-v0.1-GGUF
- Embedding: BAAI/bge-small-en-v1.5
- Vector Dimension: 384
- **Docker Run Command for AMD64 Container:**

```
docker run -d -p 8001:8001 --net=host --gpus all \
--env PORT=8001 --env TSS=0 --env GPU=1 \
--env COLLECTION=tml --env WEB_CONCURRENCY=2 \
--env CUDA_VISIBLE_DEVICES=0 --env TOKENIZERS_PARALLELISM=false \
--env temperature=0.1 --env vectorsearchtype=cosine \
--env contextwindowsize=4096 --env vectordimension=384 \
--env mainmodel="TheBloke/Mistral-7B-Instruct-v0.1-GGUF" \
--env mainembedding="BAAI/bge-small-en-v1.5" \
-v /var/run/docker.sock:/var/run/docker.sock:z \
maadsdocker/tml-privategpt-with-gpu-nvidia-amd64:latest
```

1. Suggested VRAM/GPU should be around 20GB
2. SSD 2-3 TB
3. Suggested Machine: On-demand 1x NVIDIA A10
4. Suggested Cost GPU/Hour: $0.75/GPU/h

AMD64: Mid Model Version 2

- ARM64 container
- LLM: mistralai/Mistral-7B-Instruct-v0.2
- Embedding: BAAI/bge-small-en-v1.5
- Vector Dimension: 384
- **Docker Run Command for AMD64 Container:**

```
docker run -d -p 8001:8001 --net=host --gpus all \
--env PORT=8001 --env TSS=0 --env GPU=1 \
--env COLLECTION=tml --env WEB_CONCURRENCY=2 \
--env CUDA_VISIBLE_DEVICES=0 --env TOKENIZERS_PARALLELISM=false \
--env temperature=0.1 --env vectorsearchtype=cosine \
--env contextwindowsize=4096 --env vectordimension=384 \
--env mainmodel="mistralai/Mistral-7B-Instruct-v0.2" \
--env mainembedding="BAAI/bge-small-en-v1.5" \
-v /var/run/docker.sock:/var/run/docker.sock:z \
maadsdocker/tml-privategpt-with-gpu-nvidia-amd64-v2:latest
```

1. Suggested VRAM/GPU should be around 24GB
2. SSD 2-3 TB
3. Suggested Machine: On-demand 1x NVIDIA A10
4. Suggested Cost GPU/Hour: $0.75/GPU/h

AMD64: Advanced Model Version 3

- ARM64 container
- LLM: mistralai/Mistral-7B-Instruct-v0.3
- Embedding: BAAI/bge-base-en-v1.5
- Vector Dimension: 768
- **Docker Run Command for AMD64 Container:**

```
docker run -d -p 8001:8001 --net=host --gpus all \
--env PORT=8001 --env TSS=0 --env GPU=1 \
--env COLLECTION=tml --env WEB_CONCURRENCY=2 \
--env CUDA_VISIBLE_DEVICES=0 --env TOKENIZERS_PARALLELISM=false \
--env temperature=0.1 --env vectorsearchtype=cosine \
--env contextwindowsize=4096 --env vectordimension=768 \
--env mainmodel="mistralai/Mistral-7B-Instruct-v0.3" \
--env mainembedding="BAAI/bge-base-en-v1.5" \
-v /var/run/docker.sock:/var/run/docker.sock:z \
maadsdocker/tml-privategpt-with-gpu-nvidia-amd64-v3
```

1. Suggested VRAM/GPU should be around 24GB
2. SSD 2-3 TB
3. Suggested Machine: On-demand 1x NVIDIA A10
4. Suggested Cost GPU/Hour: $0.75/GPU/h

AMD64: Large Advanced Model Version 3
 	
• ARM64 container
• LLM: mistralai/Mistral-7B-Instruct-v0.3
• Embedding: BAAI/bge-m3
• Vector Dimension: 1024
• **Docker Run Command for AMD64 Container:**
 	
`docker run -d -p 8001:8001 --net=host --gpus all \`
`--env PORT=8001 --env TSS=0 --env GPU=1 \`
`--env COLLECTION=tml --env WEB_CONCURRENCY=2 \`
`--env CUDA_VISIBLE_DEVICES=0 --env TOKENIZERS_PARALLELISM=false \`
`--env temperature=0.1 --env vectorsearchtype=cosine \`
`--env contextwindowsize=4096 --env vectordimension=1024 \`
`--env mainmodel="mistralai/Mistral-7B-Instruct-v0.3" \`
`--env mainembedding="BAAI/bge-m3" \`
`-v /var/run/docker.sock:/var/run/docker.sock:z \`
`maadsdocker/tml-privategpt-with-gpu-nvidia-amd64-v3-large`
	1. Suggested VRAM/GPU should be around 40GB
	2. SSD 2-3 TB
	3. Suggested Machine: On-demand 1x NVIDIA A6000 or A100
	4. Suggested Cost GPU/Hour: $0.80 - $1.30/GPU/h

TML and Agentic AI Special Container

For TML and Agentic AI solutions users must you the following container

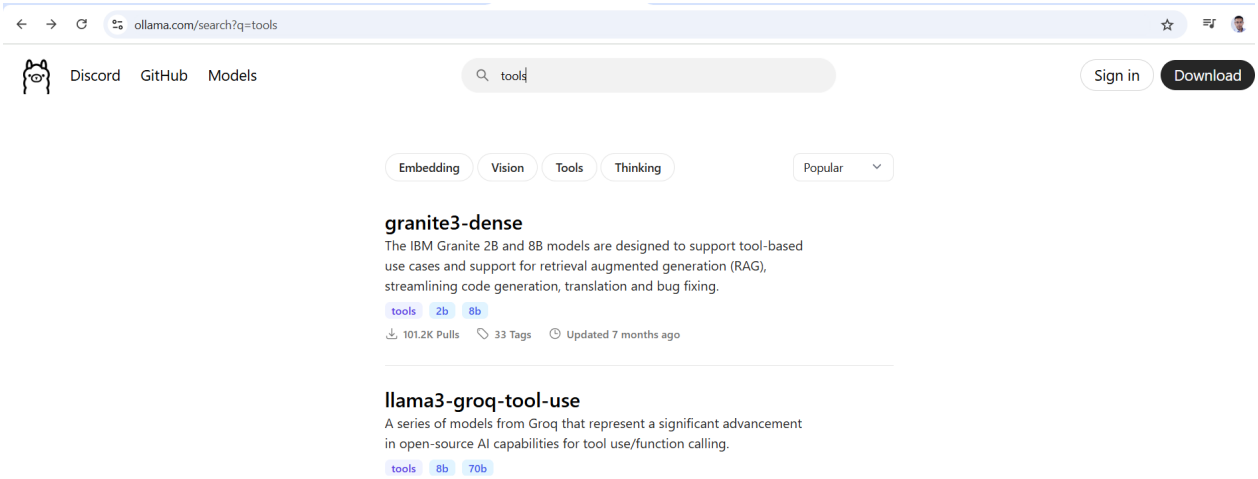
-
  - AMD64: Agentic AI Llama3 with Ollama Server
    - ARM64 container
    - LLM: Llama 3.1 OR Llama 3.2 OR ANY OTHER TOOL' MODELS
    - Embedding: nomic-embed-text
    - Vector Dimension: n/a
    - **Docker Run Command for AMD64 Container:**

```
docker run -d -p 8001:8001 --net=host --gpus all --env PORT=8001 \
--env TSS=0 \
--env GPU=1 \
--env COLLECTION=tml \
--env WEB_CONCURRENCY=2 \
--env CUDA_VISIBLE_DEVICES=0 \
--env TOKENIZERS_PARALLELISM=false \
--env temperature=0.1 \
--env vectorsearchtype=cosine \
--env contextwindowsize=4096 \
--env vectordimension=384 \
--env mainembedding="nomic-embed-text" \
-v /var/run/docker.sock:/var/run/docker.sock:z \
--env LLAMAMODEL=llama3.2 \
--env OLLAMASERVERPORT="http://localhost:11434" \
maadsdocker/tml-privategpt-with-gpu-nvidia-amd64-llama3-tools
```

-
  1. Suggested VRAM/GPU should be around 20GB
  2. SSD 2-3 TB
  3. Suggested Machine: On-demand 1x NVIDIA A10
  4. Suggested Cost GPU/Hour: $0.75/GPU/h

> **Tip**
>
> You can switch between Llama 3.1 and Llama 3.2 models by updating the:
>
> - **--env LLAMAMODEL=llama3.2**
> - You can also use ANY other TOOLS models from Ollama.com (see figure below)
>
> Ollama server host and port can be updated by updating the:
>
> - --env OLLAMASERVERPORT="http://localhost:11434"

To use models other models go to Ollama.com and search **tools**



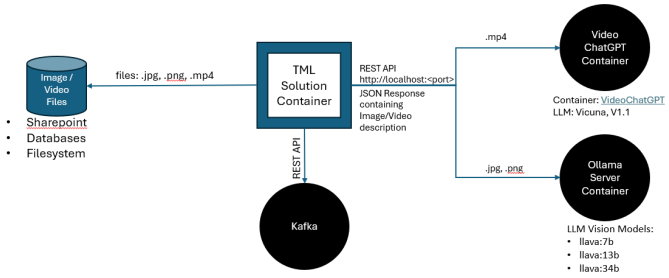
TML and Vision Models

You can use the Llava vision models by setting the **--env LLAMAMODEL=** with the following:

- **--env LLAMAMODEL=llava:7b**
- **--env LLAMAMODEL=llava:13b**
- **--env LLAMAMODEL=llava:34b**

The general reference architecture shows how TML connects to Ollama server container and Video ChatGPT in real-time to process images and Videos:

> **Note**
>
> VideoChatGPT uses Vicuna v1.1



Note

All images must be base64 decoded - see code below in section :ref:`TML and Vision Models: Sample Code`

You must have the Ollama server container running:

```
docker run -d -p 8001:8001 --net=host --gpus all --env PORT=8001 \
--env TSS=0 \
--env GPU=1 \
--env COLLECTION=tml \
--env WEB_CONCURRENCY=2 \
--env CUDA_VISIBLE_DEVICES=0 \
--env TOKENIZERS_PARALLELISM=false \
--env temperature=0.1 \
--env vectorsearchtype=cosine \
--env contextwindowsize=4096 \
--env vectordimension=384 \
--env mainembedding="nomic-embed-text" \
-v /var/run/docker.sock:/var/run/docker.sock:z \
--env LLAMAMODEL=llava:7b \
--env OLLAMASERVERPORT="http://localhost:11434" \
maadsdocker/tml-privategpt-with-gpu-nvidia-amd64-llama3-tools
```

TML and Vision Models: Sample Code

```
import base64
import requests

def base64encodeimage(imagefile):
     with open(imagefile, "rb") as image_file:
         data = base64.b64encode(image_file.read())

     return data

def base64ToString(b):
    return b.decode("utf-8")


def describeimage(imgname):

    imgdata=base64encodeimage(imgname)

    headers = {
        'Content-Type': 'application/x-www-form-urlencoded',
    }

    data = '{\n  "model": "llava:7b",\n  "prompt":"What is in this picture?",\n  "stream": false,\n  "images": ["'+base64ToString(imgdata)+'"]\n}'

    response = requests.post('http://localhost:11434/api/generate', headers=headers, data=data)

    print(response.text)

    return response

describeimage("./image2.png")
```

TML and Video ChatGPT

Users can call video chatGPT container as follows:

Docker Run Command

```
docker run --gpus all -d -p 7900:7900 \
--net=host --env CUDA_VISIBLE_DEVICES=0 \
--env VIDEOGPTPORT=7900 \
-v /mnt/c/sample_videos:/VideoChatGPT/videofile:z \
--env VIDEOGPTFOLDER=/VideoChatGPT/videofile maadsdocker/tml-videochatgpt-nvidia-gpu-amd64
```

Note

NOTE: Details on the Docker run command:

- -p 7900:7900: This is port forwarding port 7900 host port to the container port 7900
- VIDEOGPTPORT=7900: This enables the API to connect to video chatgpt on port 7900
- -v /mnt/c/sample_videos:/VideoChatGPT/videofile:z: All your video files need to be stored on the host machine, the Docker container maps this host folder to the container folder for video retrieval
- VIDEOGPTFOLDER=/VideoChatGPT/videofile : This is the container video folder
- NOTE: You need to drop the mp4 files on your host folder that is mapped to the container folder.

Video ChatGPT Sample Code

```
import maadstml # import the maadstml python library: pip install maadstml

#################### NOTE: This will only work if Video Chatgpt is running on a machine with NVidia GPU and Cuda toolkit installed

def videochat(videofilename):

    url='http://127.0.0.1'            # IP video chatgpt is listening on

    port='7900'                       # Port Video chatgpt is listening on

    filename=videofilename          # Video file name

    responsefolder='sample-videos'   # folder, video chatgpt will write out the responses to

    temperature=0.1                   # temperature - varies between 0-1, closer to 0 more conservative the responses

    max_output_tokens=512             #  max tokens or words returned

    prompt='What is this video about? Is there anythin strange about this video?'  # prompts to ask video chatgot about the video

#Load video chatgpt
    ret=maadstml.videochatloadresponse(url,port,filename,prompt,responsefolder,temperature,max_output_tokens)
    print(ret)

#CALL Video chat gpt container - you can put this in a loop and analyse several videos at once with multiple containers
videofilename = 'sample_6.mp4'

ret = videochat(videofilename) # returns the response file name
print(ret)
```

TML API for GenAI Using MAADSTML Python Library

TML solutions can be built to access GPT technology in real-time using the MAADSTML python library functions:

MAADSTML Python Function	Description
pgptingestdocs	Set Context for PrivateGPT by ingesting PDFs or text documents. All responses will then use these documents for context.

pgptgetingestedembeddings	After documents are ingested, you can retrieve the embeddings for the ingested documents. These embeddings allow you to filter the documents for specific context.
pgptchat	Send any prompt to privateGPT (with or without context) and get back a response.
pgptdeleteembeddings	Delete embeddings.
pgpthealth	Check the health of the privateGPT http server.

GenAI With STEP 9

Several powerful, real-time, AI analysis can be performed with :ref:`STEP 9: PrivateGPT and Qdrant Integration: tml-system-step-9-privategpt_qdrant-dag`

These are the following:

1. Perform post-analyis on TML output with GenAI
2. Use Qdrant vector DB, to use local documents, for querying with GenAI
3. Scale GenAI with privateGPT for secure, local, and quality AI analysis.

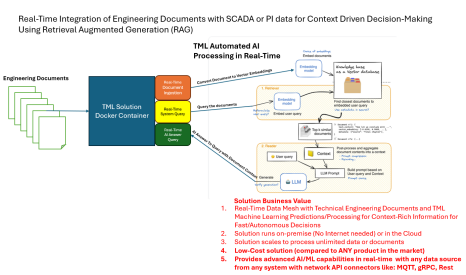
> **Tip**
>
> Take a look here :ref:`TML, PrivateGPT and Qdrant Example Scenarios` for more information.

TML and RAG: A Powerful Combination

TML using :ref:`STEP 9: PrivateGPT and Qdrant Integration: tml-system-step-9-privategpt_qdrant-dag` can perform RAG (Retrieval-augmented Generation) with a few simple configurations.

Below is a figure to show Advanced RAG model (inspiration from huggingface blog) to ingest Engineering documents for real-time prompting using one of the privateGPT containers. Together with Qdrant vector DB, users can analyse local files with TML in real-time with no-code just configurations of Step 9.

> **Important**
>
> This would be very useful especially for Cybersecurity uses cases where you want to cross-reference source IP address with web log files to determine if there are any "authentication failures" or "wrong passwords" in the log files associated to the source IP address.
>
> Together with Qdrant vector DB, users can analyse local files with TML in real-time with no-code just configurations of Step 9, in few seconds.



The incorporation of RAG with TML for real-time cybersecurity analysis of log files is demonstrated in :ref:`Cybersecurity Solution with PrivateGPT, MQTT, HiveMQ`

Private GPT Container

More privateGPT containers can be found here: :ref:`PrivateGPT Special Containers`. The container will require a NVIDIA GPU.

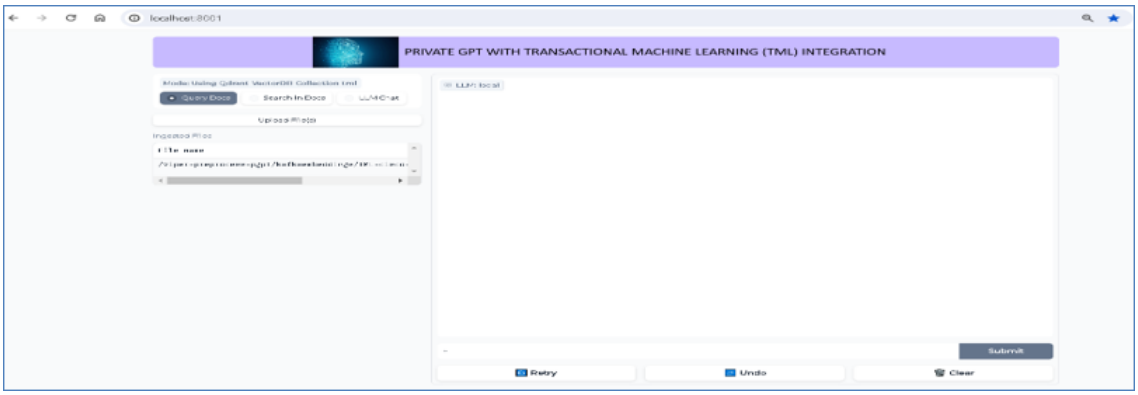
```
docker pull maadsdocker/tml-privategpt-with-gpu-nvidia-amd64
```

```
docker run -d -p 8001:8001 --net=host --gpus all \
--env PORT=8001 --env TSS=0 --env GPU=1 \
--env COLLECTION=tml --env WEB_CONCURRENCY=2 \
--env CUDA_VISIBLE_DEVICES=0 --env TOKENIZERS_PARALLELISM=false \
--env temperature=0.1 --env vectorsearchtype=cosine \
--env contextwindowsize=4096 --env vectordimension=384 \
--env mainmodel="TheBloke/Mistral-7B-Instruct-v0.1-GGUF" \
--env mainembedding="BAAI/bge-small-en-v1.5" \
maadsdocker/tml-privategpt-with-gpu-nvidia-amd64:latest
```

Tip

To check if privateGPT is running enter this in your browser: http://localhost:8001

You should see the private GPT website below.



Note

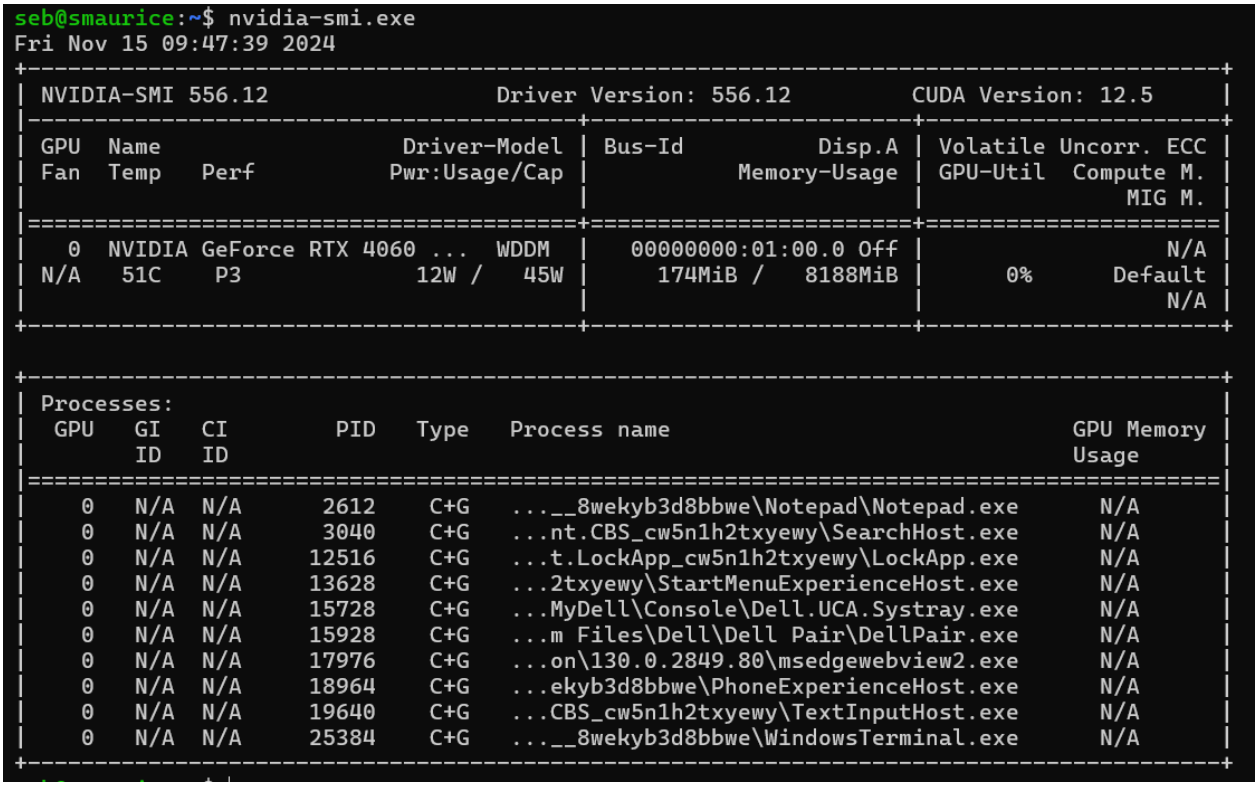
If you set WEB_CONCURRENCY greater than 1, you will need Qdrant Vector DB running (see below)

PrivateGPT Container With NO GPU

> **Tip**
>
> If you do not have a Nvidia GPU you can use the docker container with NO GPU:
>
> docker run -d -p 8001:8001 --env PORT=8001 --env GPU=0 --env CUDA_VISIBLE_DEVICES=0 maadsdocker/tml-privategpt-no-gpu-amd64

Installing CUDA For NVIDIA GPU

> **Important**
>
> It is highly recommended that users run the privateGPT container using the NVIDIA GPU for FASTER performance.
>
> If you have a NVIDIA GPU you must install the CUDA Software Development Kit in your Linux environment.
>
> To confirm your GPU card is recognized in Linux type: **nvidia-smi** - You should see an image similar to below.



NVIDIA Common Issues

Important

If you run Docker or Minikube with the **--gpus all** flag and see an ERROR message like:

docker: Error response from daemon: could not select device driver "" with capabilities: [[gpu]].

Then run the following:

```
sudo nvidia-ctk runtime configure --runtime=docker

sudo systemctl restart docker
```

Attention!

Make sure to STOP the TSS Container and other containers before running Kubernetes/Minikube.

If you get the following WARNING from Kubernetes:

Warning FailedScheduling 13m default-scheduler 0/1 nodes are available: 1 Insufficient nvidia.com/gpu. preemption: 0/1 nodes are available: 1 No preemption victims found for incoming pod.

Issue the commands below:

```
sudo apt update && sudo apt install -y nvidia-docker2

sudo nvidia-ctk runtime configure --runtime=docker

sudo systemctl restart docker
```

To Enable GPU in Kubernetes

You can apply the following YML file to the Kubernetes cluster to enable GPU support.

```
kubectl create -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.12.3/nvidia-device-plugin.yml
```

Also see section: :ref:`NVIDIA GPU On Windows WSL`

Accessing PrivateGPT With MAADSTML Python API

Once you have the PrivateGPT container running you can access it using the maadstml API. Here is some sample Python code to access the privateGPT container:

Note

Since PrivateGPT is compatible with REST API, you can use any programming language, and take advantage of free, and fast AI.

```
import maadstml
import json

def sendpromptgpt(prompt,pgptip,pgptport):
  pgptendpoint="/v1/completions"
  includesources=False
  docfilter=""
  context=False

  try:
    response=maadstml.pgptchat(prompt,context,docfilter,pgptport,includesources,pgptip,pgptendpoint)
    jb=json.loads(response)
    response=jb['choices'][0]['message']['content']

  except Exception as e:
   print("ERROR: connecting to PrivateGPT=",e)
   return ""

  return response

def setupprompt():
     pgptip="http://127.0.0.1"
     pgptport="8001"

     prompt="Who is the prime minister of Canada?"
     message=sendpromptgpt(prompt,pgptip,pgptport)
```

Details of LLM Used in privateGPT Container
llm_load_print_meta: format = GGUF V2
llm_load_print_meta: arch = llama
llm_load_print_meta: vocab type = SPM
llm_load_print_meta: n_vocab = 32000
llm_load_print_meta: n_merges = 0
llm_load_print_meta: n_ctx_train = 32768
llm_load_print_meta: n_embd = 4096
llm_load_print_meta: n_head = 32
llm_load_print_meta: n_head_kv = 8
llm_load_print_meta: n_layer = 32
llm_load_print_meta: n_rot = 128
llm_load_print_meta: n_gqa = 4
llm_load_print_meta: f_norm_eps = 0.0e+00
llm_load_print_meta: f_norm_rms_eps = 1.0e-05
llm_load_print_meta: f_clamp_kqv = 0.0e+00
llm_load_print_meta: f_max_alibi_bias = 0.0e+00
llm_load_print_meta: n_ff = 14336
llm_load_print_meta: rope scaling = linear
llm_load_print_meta: freq_base_train = 10000.0
llm_load_print_meta: freq_scale_train = 1
llm_load_print_meta: n_yarn_orig_ctx = 32768
llm_load_print_meta: rope_finetuned = unknown

llm_load_print_meta: model type = 7B
llm_load_print_meta: model ftype = mostly Q4_K - Medium
llm_load_print_meta: model params = 7.24 B
llm_load_print_meta: model size = 4.07 GiB (4.83 BPW)
llm_load_print_meta: general.name = mistralai_mistral-7b-instruct-v0.2
llm_load_print_meta: BOS token = 1 ''
llm_load_print_meta: EOS token = 2 ''
llm_load_print_meta: UNK token = 0 ''
llm_load_print_meta: LF token = 13 '<0x0A>'
llm_load_tensors: ggml ctx size = 0.11 MB
llm_load_tensors: mem required = 4165.47 MB

Qdrant Vector Database

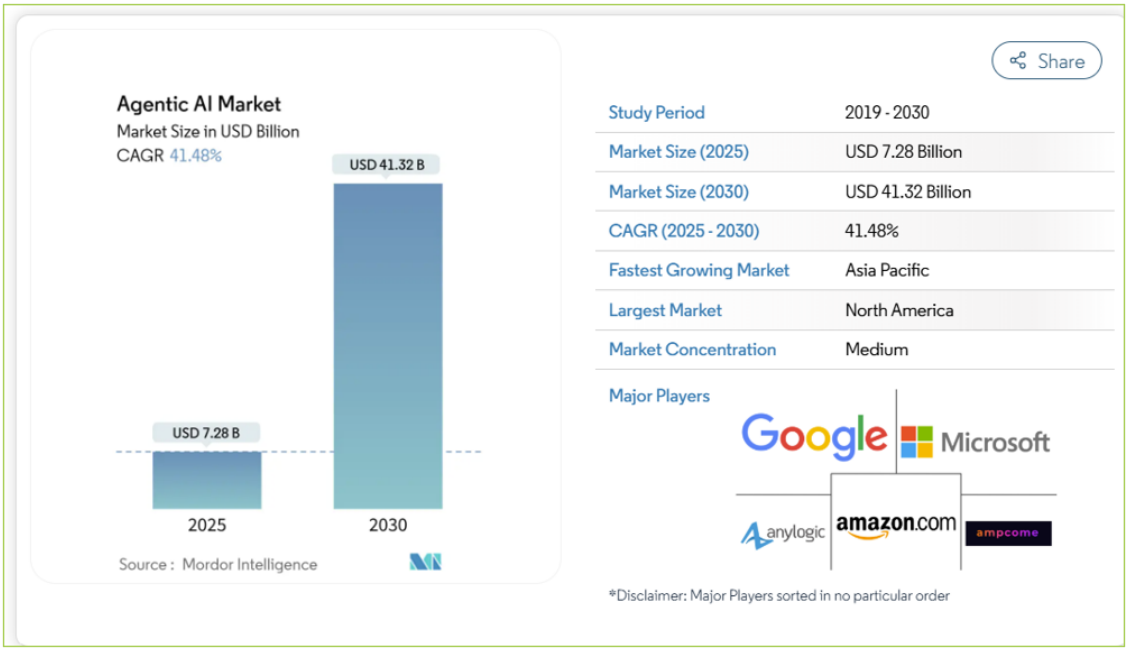
The privateGPT is also integrated with Qdrant Vector DB

```
docker run -d -p 6333:6333 -v $(pwd)/qdrant_storage:/qdrant/storage:z qdrant/qdrant
```

> Tip
>
> After running the container, to access the Qdrant dashboard enter the following URL in your browser:
>
> ```
> http://localhost:6333/dashboard
> ```

TML and Agentic AI

- The growth of Agent based computing integrated with GenAI is the next evolution of AI
- Agents are now able to reason and perform the appropriate tasks on their own - without human intervention
- The maket for Agentic AI is expected to aggressively grow in the next 5 years with a CAGR of 41%
- This growth will be driven by demand for automation of simple and complex tasks



In the above Figure:

- Agents are a natural evolution of the AI market
- Agentic AI market will experience tremendous growth as AI LLM get smarter with a CAGR of 41.48%
- agents are the way to incorporate LLMs in advanced real-time workflows
- Mutli-agents allow for a community of agents with more advanced capabilities
- For more information read this blog

What is an Agent?

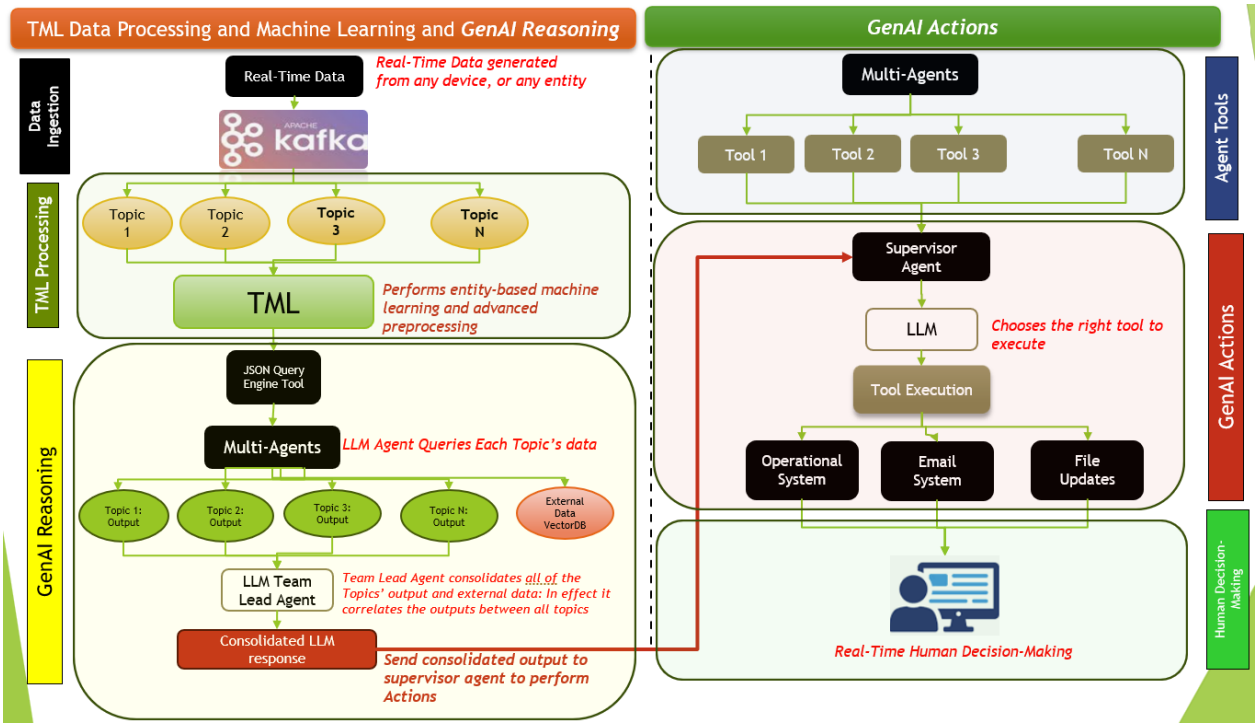
> **Important**
>
> A computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives [Wooldridge and Jennings 1995].

TML and Agentic AI: A Powerful Combination for Real-Time Data

- Real-time agent-based systems for entity level processing and machine learning is a ground-breaking capability currently “unparalleled” in the global AI market
- Integrating Agentic AI in TML solutions opens up tremendous opportunities for real-time workflow solutions that can incorporate multiple data sources that can be processed by Multiple Agents at the same time
- TML solutions are already a “Single” agent
- **The more advanced use of TML and Agents is in a multi-agent framework**
    - This will be the powerful capability in TML

TML and Multi-Agentic AI Process Flow in Real-Time: GenAI Reasoning and Actions

The figure below shows the TML and Multi-Agentic AI process flow for real-time data processing. Integrating multi-agentic framework with TML for real-time data processing with GenAI reasoning and actions offers tremendous opportunities for machines to perform advanced reasoning and actions for real-time decision making by humans.

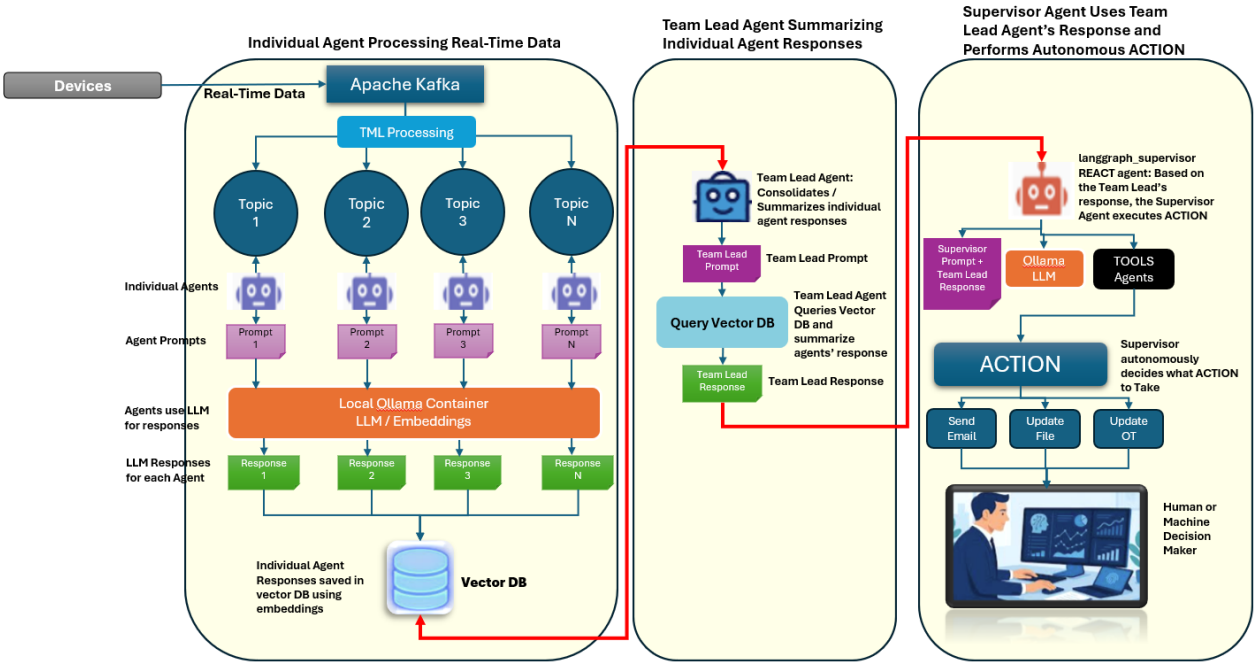


TML and Multi-Agentic AI Solution Reference

The figure below shows how the TML agentic AI solution processes real-time data with multi-agents:

- Individual agents monitor the Kafka topics containing TML processed real-time data
- The individual agents are prompted with questions to analyse the real-time data
- The individual agent responses are writte to a Vector DB
- The Team Lead agent summarizes the individual agents' responses by querying the vector DB. This allows for a "meshing" of the information from individual agents, and provides for a "fuller" analysis of all of the agents' data.
- The Supervisor agent takes the Team Lead's response summary and generates ACTION
- The autonomous actions could be send email, update file or update some operational technology

- Human or machine decision maker then receives the outcome of this ACTION.



Sample Output from TML Multi-Agentic AI Solution

Below is sample output from a TML agentic solution, that implements the above solution reference architecture, for monitoring IoT device topics. Some things to note:

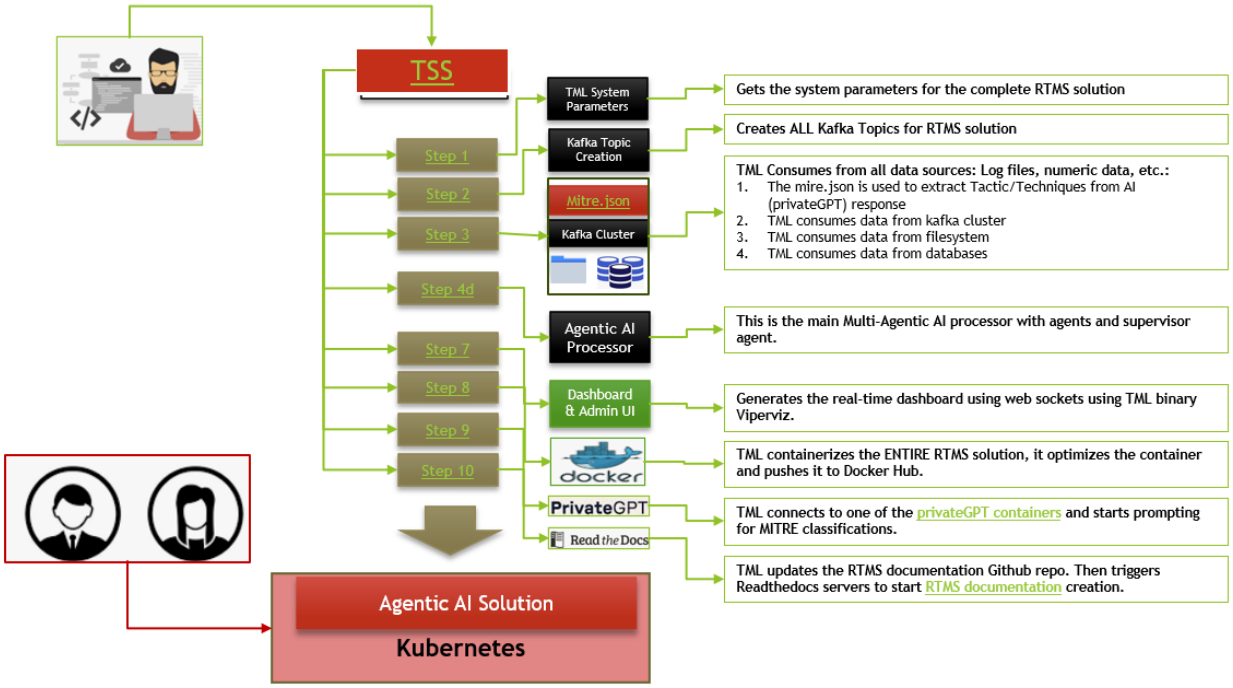
> **Important**
>
> This sample output is generated from STEP 9b: Multi-Agentic Agentic A: tml-system-step-9b-agenticai-dag

- **Topic_Agent** are the agents monitoring the Kafka topic for any anomalies. In this example we habe 3 topic agents monitoring, in real-time, three different topics.
- **Team_Lead_Agent** analyzes all of the respnses from the Topic_Agents and summarizes the information for hand off to the supervisor agent
- **Supervisor_Agent**, based on the Team_Lead_Agent summary, decides what tool to route the information to for an ACTION. In this example the supervisor routes the request to the send_email agent who sends an email to a human(s).

Building TML and Agentic AI Solutions with TML Solution Studio (TSS)

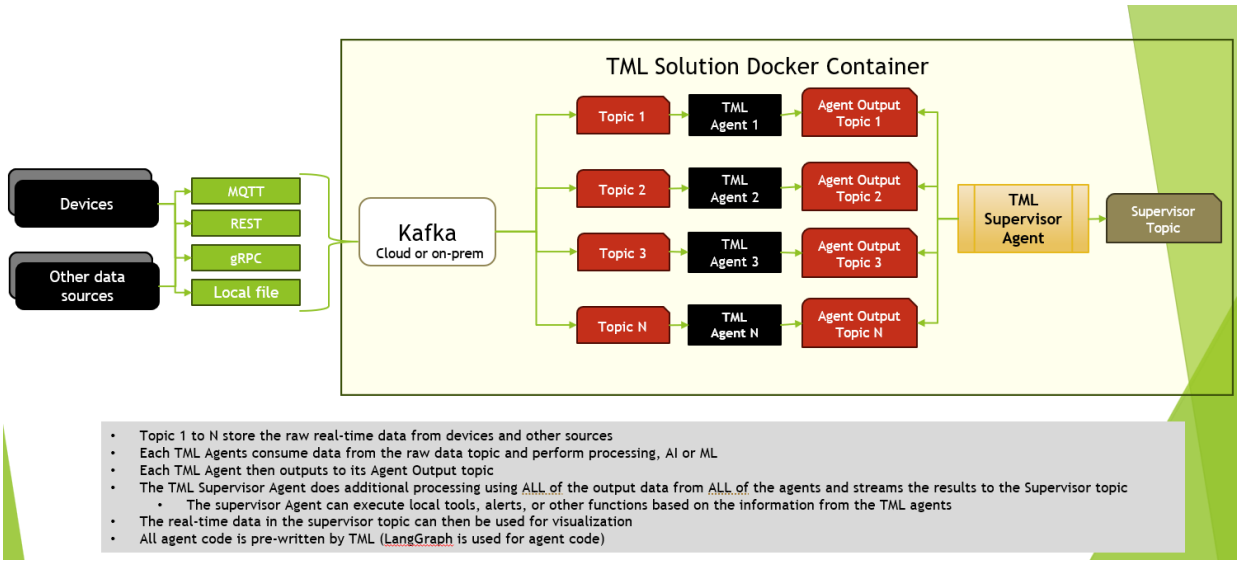
- All TML solutions are built with the TSS
- TSS enforces a process driven approach to build TML solution in a few minutes
- The image on the left shows an example of a TML solution build process
- Every TML solution are built with NO-CODE

- Each TML solution takes less than 5 minutes to develop
- The output of a TSS solution build is a docker container



TML and (Multi) Agentic AI Architecture

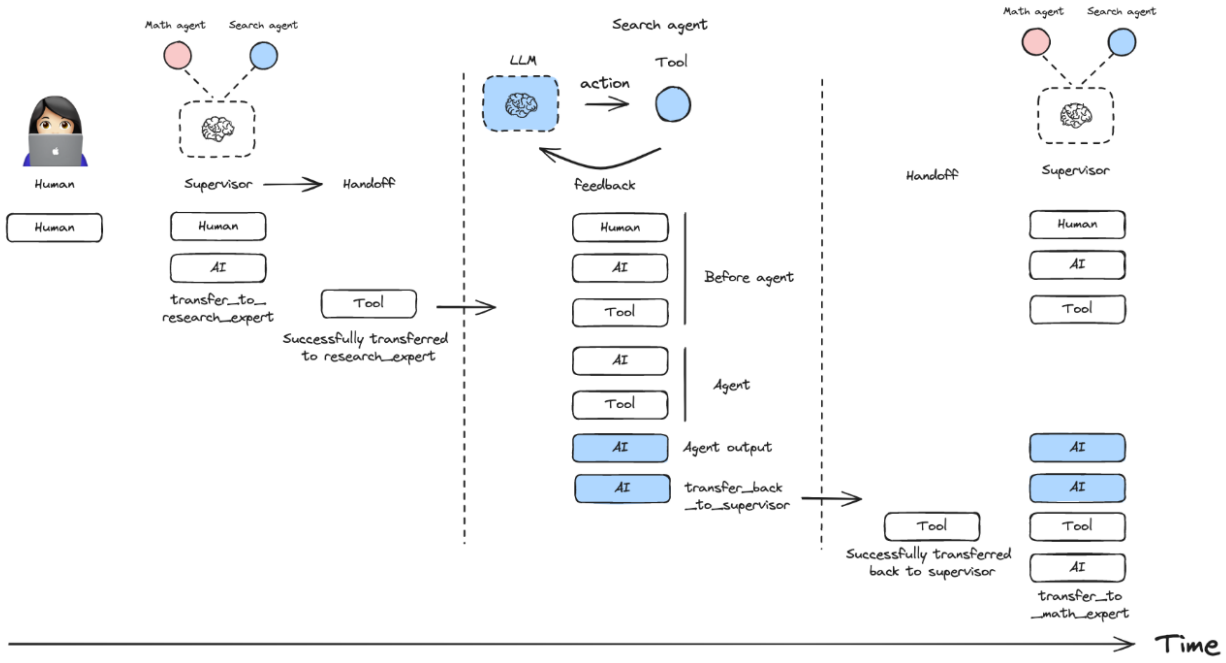
- The TML and Agentic Architecture is very simple: Agents can be configured in the TSS
- With NO-CODE - users can advanced agent based solutions that process real-time data and perform tasks in real-time
- **The AI integration is with the TML privateGPT Agentic AI containers**
    - This local container uses Ollama server for API based Agentic workflow automation
    - Two LLM models can be used: Llama 3.1 or Llama 3.2
- Using local GenAI containers drastically reduces the cost of Agentic solution for large scale data processing



- Topic 1 to N store the raw real-time data from devices and other sources
- Each TML Agents consume data from the raw data topic and perform processing, AI or ML
- Each TML Agent then outputs to its Agent Output topic
- The TML Supervisor Agent does additional processing using ALL of the output data from ALL of the agents and streams the results to the Supervisor topic
    - The supervisor Agent can execute local tools, alerts, or other functions based on the information from the TML agents
- The real-time data in the supervisor topic can then be used for visualization
- All agent code is pre-written by TML (LangGraph is used for agent code)

Implementing Complex Workflows with TML and (Multi) Agentic AI

Implementing complex real-time workflows to automate complex tasks is possible with TML and Agentic AI, as shown below. In fact, this is would be a new skill set for Busniness analysts but focused on Agentic AI solutions:



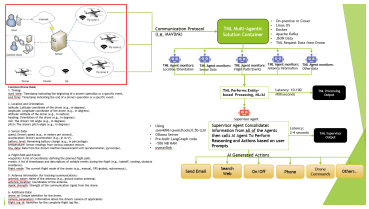
TML and TSS use LangGraph for (multi) Agent based code. TML agents can execute tools autonomously. Tools are out of the box, or users can build their own custom tools and integrate with their TML solutions, easily.

Advantages of TML with Agentic AI

- Real-Time entity-based Agent computing can offer finer-grained insights that could improve the quality of real-time decisions for many uses in IoT, Cybersecurity, Finance, Manufacturing, Energy etc.
- By processing data from multiple data sources by individual agents, and then combining the output (supervisor agent) increases the level intelligences extracted from the data leading to higher dimensional, entity-level, intelligence in real-time
- Ability to perform complex workflow tasks in real-time offers greater, and faster, visibility on critical operational functions
- COST: Drastic reduction in costs using TML and Agentic AI. Because TML uses local Agentic AI container API calls are FREE. This leads to a drastic reduction in costs for TML and Agentic AI solutions, immediately.

EXAMPLE: TML Agentic AI For Drones

Below is an example solution architecture applying TML and Agentic AI to Drones using MAVLink as the communication prootocol.



Can I Run TML Binaries in Standalone?

Note

Yes, the :ref:`TML Solution Components` can be run as standalone - but this is not advised as it may become difficult to maintain and support. TSS is the preferred, and easiest, way. However, if you want to learn more please view the 4-part Youtube videos:

1. TML Crash Course Part 1
2. TML Crash Course Part 2
3. TML Crash Course Part 3
4. TML Crash Course Part 4

MAADSTML Python Library API

Multi-Agent Accelerator for Data Science Using Transactional Machine Learning (MAADSTML)

Source: MAADSTML Python Library

Revolutionizing Data Stream Science with Transactional Machine Learning

> Important
>
> The MAADSTML python library is a very powerful library that accelerates TML solution builds. While this library includes many functions, the TSS automatically calls the core functions for your TML solution - considerably reducing the effort on your part.
>
> Refer to :ref:`Where Do I Start?`

Overview

MAADSTML combines Artificial Intelligence, ChatGPT, PrivateGPT, Auto Machine Learning with Data Streams Integrated with Apache Kafka (or Redpanda) to create frictionless and elastic machine learning solutions.

This library allows users to harness the power of agent-based computing using hundreds of advanced linear and non-linear algorithms. Users can easily integrate Predictive Analytics, Prescriptive Analytics, Pre-Processing, and Optimization in any data stream solution by wrapping additional code around the functions below. It connects with **Apache KAFKA brokers** for cloud based computing using Kafka (or Redpanda) as the data backbone.

If analysing MILLIONS of IoT devices, you can easily deploy thousands of VIPER/HPDE instances in Kubernetes Cluster in AWS/GCP/Azure.

It uses VIPER as a **KAFKA connector and seamlessly combines Auto Machine Learning, with Real-Time Machine Learning, Real-Time Optimization and Real-Time Predictions** while publishing these insights in to a Kafka cluster in real-time at scale, while allowing users to consume these insights from anywhere, anytime and in any format.

It also HPDE as the AutoML technology for TML. Linux/Windows/Mac versions can be downloaded from [Github](https://github.com/smaurice101/transactionalmachinelearning)

It uses VIPERviz to visualize streaming insights over HTTP(S). Linux/Windows/Mac versions can be downloaded from [Github](https://github.com/smaurice101/transactionalmachinelearning)

MAADSTML details can be found in the book: [Transactional Machine Learning with Data Streams and AutoML](https://www.amazon.com/Transactional-Machine-Learning-Streams-AutoML/dp/1484270223)

To install this library a request should be made to **support@otics.ca** for a username and a MAADSTOKEN. Once you have these credentials then install this Python library.

Compatibility

- Python 3.8 or greater
- Minimal Python skills needed

Copyright

- Author: Sebastian Maurice, PhD

Installation

- At the command prompt write: **pip install maadstml** - This assumes you have [Downloaded Python](https://www.python.org/downloads/) and installed it on your computer.

MAADS-VIPER Connector to Manage Apache KAFKA:

- MAADS-VIPER python library connects to VIPER instances on any servers; VIPER manages Apache Kafka. VIPER is REST based and cross-platform that can run on windows, linux, MAC, etc.. It also fully supports SSL/TLS encryption in Kafka brokers for producing and consuming.

****TML is integrated with PrivateGPT (https://github.com/imartinez/privateGPT), which is a production ready GPT, that is 100% Local, 100% Secure and 100% FREE GPT Access.**

- Users need to PULL and RUN one of the privateGPT Docker containers:
- 1. Docker Hub: maadsdocker/tml-privategpt-no-gpu-amd64 (without NVIDIA GPU for AMD64 Chip)
- 2. Docker Hub: maadsdocker/tml-privategpt-with-gpu-amd64 (with NVIDIA GPU for AMD64 Chip)
- 3. Docker Hub: maadsdocker/tml-privategpt-no-gpu-arm64 (without NVIDIA GPU for ARM64 Chip)
- 4. Docker Hub: maadsdocker/tml-privategpt-with-gpu-arm64 (with NVIDIA GPU for ARM64 Chip)
- Additional details are here: https://github.com/smaurice101/raspberrypi/tree/main/privategpt
- TML accesses privateGPT container using REST API.
- For PrivateGPT production deployments it is recommended that machines have the NVIDIA GPU as this will lead to significant performance improvements.

- **pgptingestdocs** - Set Context for PrivateGPT by ingesting PDFs or text documents. All responses will then use these documents for context.
- **pgptgetingestedembeddings** - After documents are ingested, you can retrieve the embeddings for the ingested documents. These embeddings allow you to filter the documents for specific context.
- **pgptchat** - Send any prompt to privateGPT (with or without context) and get back a response.
- **pgptdeleteembeddings** - Delete embeddings.
- **pgpthealth** - Check the health of the privateGPT http server.
- **vipermirrorbrokers** - Migrate data streams from (mutiple) brokers to (multiple) brokers FAST! In one simple function you have the

  power to migrate from hundreds of brokers with hundreds of topics and partitions to any other brokers

  with ease. Viper ensures no duplication of messages and translates offsets from last committed. Every transaction is logged, making data validation and auditability a snap. You can also increase or decrease partitions and apply filter to topics to copy over.
- **viperstreamquery** - Query multiple streams with conditional statements. For example, if you preprocessed multiple streams you can

  query them in real-time and extract powerful insights. You can use >, <, =, AND, OR.
- **viperstreamquerybatch** - Query multiple streams with conditional statements. For example, if you preprocessed multiple streams you can

  query them in real-time and extract powerful insights. You can use >, <, =, AND, OR. Batch allows you to query

  multiple IDs at once.
- **viperlisttopics** - List all topics in Kafka brokers

- **viperdeactivatetopic** - Deactivate topics in kafka brokers and prevent unused algorithms from consuming storage and computing resources that cost money
- **viperactivatetopic** - Activate topics in Kafka brokers
- **vipercreatetopic** - Create topics in Kafka brokers
- **viperstats** - List all stats from Kafka brokers allowing VIPER and KAFKA admins with a end-end view of who is producing data to algorithms, and who is consuming the insights from the algorithms including date/time stamp on the last reads/writes to topics, and how many bytes were read and written to topics and a lot more
- **vipersubscribeconsumer** - Admins can subscribe consumers to topics and consumers will immediately receive insights from topics. This also gives admins more control of who is consuming the insights and allows them to ensures any issues are resolved quickly in case something happens to the algorithms.
- **viperunsubscribeconsumer** - Admins can unsubscribe consumers from receiving insights, this is important to ensure storage and compute resources are always used for active users. For example, if a business user leaves your company or no longer needs the insights, by unsubscribing the consumer, the algorithm will STOP producing the insights.
- **viperhpdetraining** - Users can do real-time machine learning (RTML) on the data in Kafka topics. This is very powerful and useful for "transactional learnings" on the fly using our HPDE technology. HPDE will find the optimal algorithm for the data in less than 60 seconds.
- **viperpreprocessrtms** - Users can use this function to mesh TEXT data with TML machine learning output to cross-reference entities with TEXT files like log files. This function is a

  very powerful function that incorporates "past memory" of real-time events using sliding time windows. For details see: How TML incorporates real-time memory using sliding time windows
- **viperhpdetrainingbatch** - Users can do real-time machine learning (RTML) on the data in Kafka topics. This is very powerful and useful for "transactional learnings" on the fly using

  our HPDE technology. HPDE will find the optimal algorithm for the data in less than 60 seconds. Batch allows you to perform ML on multiple IDs at once.
- **viperhpdepredict** - Using the optimal algorithm - users can do real-time predictions from streaming data into Kafka Topics.
- **viperhpdepredictprocess** - Using the optimal algorithm you can determine object ranking based on input data. For example, if you want to know which human or machine is the

  best or worst given input data then this function will return the best or worst human or machine.
- **viperhpdepredictbatch** - Using the optimal algorithm - users can do real-time predictions from streaming data into Kafka Topics. Batch allows you to perform predictions

  on multiple IDs at once.
- **viperhpdeoptimize** - Users can even do optimization to MINIMIZE or MAXIMIZE the optimal algorithm to find the BEST values for the independent variables that will minimize or maximize the dependent variable.
- **viperhpdeoptimizebatch** - Users can even do optimization to MINIMIZE or MAXIMIZE the optimal algorithm to find the BEST values for the independent variables that will minimize or maximize the dependent

  variable. Batch allows you to optimize multiple IDs at once.
- **viperproducetotopic** - Users can produce to any topics by injesting from any data sources.
- **viperproducetotopicbulk** - Users can produce to any topics by injesting from any data sources. Use this function to write bulk transactions at high speeds. With the right architecture and network you can stream 1 million transactions per second (or more).
- **viperconsumefromtopic** - Users can consume from any topic and graph the data.

- **viperconsumefromtopicbatch** - Users can consume from any topic and graph the data. Batch allows you to consume from multiple IDs at once.
- **viperconsumefromstreamtopic** - Users can consume from a multiple stream of topics at once
- **vipercreateconsumergroup** - Admins can create a consumer group made up of any number of consumers. You can add as many partitions for the group in the Kafka broker as well as specify the replication factor to ensure high availaibility and no disruption to users who consume insights from the topics.
- **viperconsumergroupconsumefromtopic** - Users who are part of the consumer group can consume from the group topic.
- **viperproducetotopicstream** - Users can join multiple topic streams and produce the combined results to another topic.
- **viperpreprocessproducetotopicstream** - Users can pre-process data streams using the following functions: MIN, MAX, AVG, COUNT, COUNTSTR, DIFF, DIFFMARGIN, SUM, MEDIAN, VARIANCE, OUTLIERS, OUTLIERSX-Y,VARIED,

ANOMPROB,ANOMPROBX-Y,ENTROPY, AUTOCORR, TREND, CONSISTENCY, IQR (InterQuartileRange), Midhinge, GM (Geometric mean), HM (Harmonic mean), Trimean,

CV (coefficient of Variation),Mad (Mean absolute deviation), Skewness, Kurtosis, Spikedetect, Unique, Uniquestr, Timediff: time should be in this layout:2006-01-02T15:04:05, Timediff returns the difference in seconds between the first date/time and last datetime. Avgtimediff returns the

average time in seconds between consecutive dates.. Spikedetect uses a Zscore method to detect

spikes in the data using lag of 5, StD of 3.5 from mean and influence of 0.5. Geodiff (returns distance in Kilometers between two lat/long points)

Dataage_[UTC offset]_[timetype], dataage can be used to check the last update time of the data in the data stream from

current local time. You can specify the UTC offset to adjust the current time to match the timezone of the data stream. You can specify timetype as millisecond, second, minute, hour, day. For example, if Dataage_1_minute, then this processtype will compare the last timestamp in the data stream, to the local UTC time offset +1 and compute the time difference between the data stream timestamp and current local time and return the difference in minutes. This is a very powerful processtype for data quality and data assurance programs for any number of data streams.

Unique Checks numeric data for duplication. Returns 1 if no data duplication (unique), 0 otherwise.

Uniquestr Checks string data for duplication. Returns 1 if no data duplication (unique), 0 otherwise.

Uniquecount Checks numeric data for duplication. Returns count of unique numbers.

Uniquestrcount Checks string data for duplication. Returns count of unique strings.

CONSISTENCY checks if the data all have consistent data types. Returns 1 for consistent data types, 0 otherwise.

Meanci95 or Meanci99 - returns a 95% or 99% confidence interval: mean, low, high

RAW for no processing.

ANOMPROB=Anomaly Probability, it will run several algorithms on the data stream window to determine a probability percentage of

anomalous behaviour. This can be cross-referenced with other process types. This is very useful if you want to extract aggregate values that you can then use to build TML models and/or make decisions to prevent issues. ENTROPY will compute the amount of

information in the data stream. AUTOCORR will run a autocorrelation regression: Y = Y (t-1), to indicate how previous value correlates with future

value. TREND will run a linear regression of Y = f(Time), to determine if the data in the stream are increasing or decreasing.

ANOMPROBX-Y (similar to OUTLIERSX-Y), where X and Y are numbers or "n", if "n" means examine all anomalies for recurring patterns.

They allow you to check if the anomalies in the streams are truly anomalies and not some

pattern. For example, if a IoT device shuts off and turns on again routinely, this may be picked up as an anomaly when in fact it is normal behaviour. So, to ignore these cases, if ANOMPROB2-5, this tells Viper, check anomalies with patterns of 2-5 peaks. If the stream has two classes and these two classes are like 0 and 1000, and show a pattern, then they should not be considered an anomaly. Meaning, class=0, is the device shutting down, class=1000 is the device turning back on. If ANOMPROB3-10, Viper will check for patterns of classes 3 to 10 to see if they recur routinely. This is very helpful to reduce false positives and false negatives.

- **viperpreprocessbatch** - This function is similar to *viperpreprocessproducetotopicstream* the only difference is you can specify multiple

  tmlids in Topicid field. This allows you to batch process multiple tmlids at once. This is very useful if using

  kubernetes architecture.

- **vipercreatejointopicstreams** - Users can join multiple topic streams
- **vipercreatetrainingdata** - Users can create a training data set from the topic streams for Real-Time Machine Learning (RTML) on the fly.
- **vipermodifyconsumerdetails** - Users can modify consumer details on the topic. When topics are created an admin must indicate name, email, location and description of the topic. This helps to better manage the topic and if there are issues, the admin can contact the individual consuming from the topic.
- **vipermodifytopicdetails** - Users can modify details on the topic. When topics are created an admin must indicate name, email, location and description of the topic. This helps to better manage the topic and if there are issues, the admin can contact the developer of the algorithm and resolve issue quickly to ensure disruption to consumers is minimal.
- **vipergroupdeactivate** - Admins can deactive a consumer group, which will stop all insights being delivered to consumers in the group.
- **vipergroupactivate** - Admins can activate a group to re-start the insights.
- **viperdeletetopics** - Admins can delete topics in VIPER database and Kafka clusters.
- **viperanomalytrain** - Perform anomaly/peer group analysis on text or numeric data stream using advanced unsupervised learning. VIPER automatically joins

  streams, and determines the peer group of "usual" behaviours using proprietary algorithms, which are then used to predict anomalies with

  viperanomalypredict in real-time. Users can use several parameters to fine tune the peer groups.

  VIPER is one of the very few, if not only, technology to do anomaly/peer group analysis using unsupervised learning on data streams with Apache Kafka.

- **viperanomalytrainbatch** - Batch allows you to perform anomaly training on multiple IDs at once.
- **viperanomalypredict** - Predicts anomalies for text or numeric data using the peer groups found with *viperanomalytrain*. VIPER automatically joins streams and compares each value with the peer groups and determines if a value is anomalous in real-time. Users can use several parameters to fine tune the analysis.

  VIPER is one of the very few, if not only, technology to do anomaly detection/predictions using unsupervised learning on data streams with Apache Kafka.

- **viperanomalypredictbatch** - Batch allows you to perform anomaly prediction on multiple IDs at once.
- **viperstreamcorr** - Performs streaming correlations by joining multiple data streams with 2 variables. Also performs cross-correlations with 4 variables.

    This is a powerful function and can offer important correlation signals between variables. Will also correlate TEXT using natural language processing (NLP).
- **viperpreprocesscustomjson** - Immediately start processing ANY RAW JSON data in minutes. This is useful if you want to start processing data quickly.
- **viperstreamcluster** - Perform cluster analysis on streaming data. This uses K-Means clustering with Euclidean or EuclideanSquared algorithms to compute

    distance. It is a very useful function if you want to determine common behaviours between devices, patients, or other entities.

        Users can also setup email alerts if specific clusters are found.
- **vipersearchanomaly** - Perform advanced analysis for user search. This function is useful if you want to monitor what people are searching for, and determine

    if the searches are anamolous and differ from the peer group of "normal" search behaviour.
- **vipernlp** - Perform advanced natural language summary of PDFs.
- **viperchatgpt** - Start a conversation with ChatGPT in real-time and stream responses.
- **viperexractpdffields** - Extracts fields from PDF file
- **viperexractpdffieldbylabel** - Extracts fields from PDF file by label name.
- **videochatloadresponse** - Analyse videos with video chatgpt. This is a powerful GPT LLM that will understand and reason with videos frame by frame.

    It will also understand the spatio-temporal frames in the video. Video gpt runs in a container.
- **areyoubusy** - If deploying thousands of VIPER/HPDE binaries in a Kubernetes cluster - you can broadcast a 'areyoubusy' message to all VIPER and HPDE

    binaries, and they will return back the HOST/PORT if they are NOT busy with other tasks. This is very convenient for dynamically managing

        enormous load among VIPER/HPDE and allows you to dynamically assign HOST/PORT to **non-busy** VIPER/HPDE microservices.

First import the Python library.

import maadstml

1. maadstml.viperstats(vipertoken,host,port=-999,brokerhost='',brokerport=-999,microserviceid='')

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

brokerhost : string, optional

- Address where Kafka broker is running - if none is specified, the Kafka broker address in the VIPER.ENV file will be used.

brokerport : int, optional

- Port on which Kafka is listenting.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: A JSON formatted object of all the Kafka broker information.

****2. maadstml.vipersubscribeconsumer(vipertoken,host,port,topic,companyname,contactname,contactemail,**

location,description,brokerhost='',brokerport=-999,groupid='',microserviceid='')**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topic : string, required

- Topic to subscribe to in Kafka broker

companyname : string, required

- Company name of consumer

contactname : string, required

- Contact name of consumer

contactemail : string, required

- Contact email of consumer

location : string, required

- Location of consumer

description : string, required

- Description of why consumer wants to subscribe to topic

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

groupid : string, optional

- Subscribe consumer to group

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Consumer ID that the user must use to receive insights from topic.

****3. maadstml.viperunsubscribeconsumer(vipertoken,host,port,consumerid,brokerhost='',brokerport=-999,**

microserviceid='')**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

consumerid : string, required

- Consumer id to unsubscribe

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

RETURNS: Success/failure

****4. maadstml.viperproducetotopic(vipertoken,host,port,topic,producerid,enabletls=0,delay=100,inputdata='',maadsalgokey='',**

maadstoken='',getoptimal=0,externalprediction='',subtopics='',topicid=-999,identifier='',array=0,brokerhost='',brokerport=-999,microserviceid='')**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topic : string, required

- Topic or Topics to produce to. You can separate multiple topics by a comma. If using multiple topics, you must have the same number of producer ids (separated by commas), and same number of externalprediction (separated by commas). Producing to multiple topics at once is convenient for synchronizing the timing of streams for machine learning.

subtopic : string, optional

- Enter sub-topic streams. This is useful if you want to reduce the number of topics/partitions in Kafka by adding sub-topics in the main topic.

topicid : int, optional

- Topicid represents an id for some entity. For example, if you have 1000 IoT devices, with 10 subtopic streams you can assign a Topicid to each IoT device and each of the 10 subtopics will be associated to each IoT device. This way, you do not create 10,000 streams, but just 1 Main Topic stream, and VIPER will add the 10,000 streams in the one topic. This will also drastically reduce the partition costs. You can also create custom machine learning models, predictions, and optimization for each 1000 IoT devices quickly: **It is very powerful.**

"array* : int, optional

- You can stream multiple variables at once, and use array=1 to specify that the streams are an array. This is similar to streaming 1 ROW in a database, and useful if you want to synchonize variables for machine learning. For example, if a device produces 3 streams: stream A, stream B, stream C, and rather than streaming A, B, C separately you can add them to subtopic="A,B,C", and externalprediction="value_FOR_A,value_FOR_B,value_FOR_C", then specify array=1, then when you do machine learning on this data, the variables A, B, C are date/time synchronized and you can choose which variable is the depdendent variable in viperhpdetraining function.

identifier : string, optional

- You can add any string identifier for the device. For examaple, DSN ID, IoT device id etc..

producerid : string, required

- Producer ID of topic to produce to in the Kafka broker

enabletls : int, optional

- Set to 1 if Kafka broker is enabled with SSL/TLS encryption, otherwise 0 for plaintext.

delay: int, optional

- Time in milliseconds from VIPER backsout from writing messages

inputdata : string, optional

- This is the inputdata for the optimal algorithm found by MAADS or HPDE

maadsalgokey : string, optional

- This should be the optimal algorithm key returned by maadstml.dotraining function.

maadstoken : string, optional - If the topic is the name of the algorithm from MAADS, then a MAADSTOKEN must be specified to access the algorithm in the MAADS server

getoptimal: int, optional - If you used the maadstml.OPTIMIZE function to optimize a MAADS algorithm, then if this is 1 it will only retrieve the optimal results in JSON format.

externalprediction : string, optional - If you are using your own custom algorithms, then the output of your algorithm can be still used and fed into the Kafka topic.

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns the value produced or results retrieved from the optimization.

****4.1. maadstml.viperproducetotopicbulk(vipertoken,host,port,topic,producerid,inputdata,partitionsize=100,enabletls=1,delay=100,**

brokerhost='',brokerport=-999,microserviceid='')**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topic : string, required

- Topic or Topics to produce to. You can separate multiple topics by a comma. If using multiple topics, you must have the same number of producer ids (separated by commas), and same number of externalprediction (separated by commas). Producing to multiple topics at once is convenient for synchronizing the timing of streams for machine learning.

producerid : string, required

- Producer ID of topic to produce to in the Kafka broker. Separate multiple producer ids with comma.

inputdata : string, required

- You can write multiple transactions to each topic. Each group of transactions must be separated by a tilde. Each transaction in the group must be separate by a comma. The number of groups must match the producerids and topics. For example, if you are writing to two topics: topic1,topic2, then the inputdata should be: trans1,transn2,...,transnN~trans1,transn2,...,transnN. The number of transactions and topics can be any number. This function can be very powerful if you need to analyse millions or billions of transactions very quickly.

partitionsize : int, optional

- This is the number of partitions of the inputdata. For example, if your transactions=10000, then VIPER will create partitions of size 100 (if partitionsize=100) resulting in 100 threads for concurrency. The higher the partitionsize, the lower the number of threads. If you want to streams lots of data fast, then a partitionzie of 1 is the fastest but will come with overhead because more RAM and CPU will be consumed.

enabletls : int, optional

- Set to 1 if Kafka broker is enabled with SSL/TLS encryption, otherwise 0 for plaintext.

delay: int, optional

- Time in milliseconds from VIPER backsout from writing messages

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: None

****5. maadstml.viperconsumefromtopic(vipertoken,host,port,topic,consumerid,companyname,partition=-1,enabletls=0,delay=100,offset=0,**

brokerhost='',brokerport=-999,microserviceid='',topicid='-999',rollbackoffsets=0,preprocesstype='')****

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topic : string, required

- Topic to consume from in the Kafka broker

preprocesstype : string, optional

- If you only want to search for record that have a particular processtype, you can enter: MIN, MAX, AVG, COUNT, COUNTSTR, DIFF, DIFFMARGIN, SUM, MEDIAN, VARIANCE, OUTLIERS, OUTLIERSX-Y, VARIED, ANOMPROB,ANOMPROBX-Y,ENTROPY, AUTOCORR, TREND, CONSISTENCY, Unique, Uniquestr, Geodiff (returns distance in Kilometers between two lat/long points) IQR (InterQuartileRange), Midhinge, GM (Geometric mean), HM (Harmonic mean), Trimean, CV (coefficient of Variation), Mad (Mean absolute deviation), Skewness, Kurtosis, Spikedetect,

Timediff: time should be in this layout:2006-01-02T15:04:05, Timediff returns the difference in seconds between the first date/time and last datetime. Avgtimediff returns the average time in seconds between consecutive dates. Spikedetect uses a Zscore method to detect spikes in the data using lag of 5, StD of 3.5 from mean and influence of 0.5.

Dataage_[UTC offset]_[timetype], dataage can be used to check the last update time of the data in the data stream from current local time. You can specify the UTC offset to adjust the current time to match the timezone of the data stream. You can specify timetype as millisecond, second, minute, hour, day. For example, if Dataage_1_minute, then this processtype will compare the last timestamp in the data stream, to the local UTC time offset +1 and compute the time difference between the data stream timestamp and current local time and return the difference in minutes. This is a very powerful processtype for data quality and data assurance programs for any number of data streams.

Unique Checks numeric data for duplication. Returns 1 if no data duplication (unique), 0 otherwise.

Uniquestr Checks string data for duplication. Returns 1 if no data duplication (unique), 0 otherwise.

Uniquecount Checks numeric data for duplication. Returns count of unique numbers.

Uniquestrcount Checks string data for duplication. Returns count of unique strings.

CONSISTENCY checks if the data all have consistent data types. Returns 1 for consistent data types, 0 otherwise.

Meanci95 or Meanci99 - returns a 95% or 99% confidence interval: mean, low, high

RAW for no processing.

ANOMPROB=Anomaly probability, it will run several algorithms on the data stream window to determine a probaility of anomalous behaviour. This can be cross-refenced with OUTLIERS. It can be very powerful way to detection issues with devices.

ANOMPROBX-Y (similar to OUTLIERSX-Y), where X and Y are numbers, or "n". If "n", means examine all anomalies for patterns. They allow you to check if the anomalies in the streams are truly anomalies and not some pattern. For example, if a IoT device shuts off and turns on again routinely, this may be picked up as an anomaly when in fact it is normal behaviour. So, to ignore these cases, if ANOMPROB2-5, this tells Viper, check anomalies with patterns of 2-5 peaks. If the stream has two classes and these two classes are like 0 and 1000, and show a pattern, then they should not be considered an anomaly. Meaning, class=0, is the device shutting down, class=1000 is the device turning back on. If ANOMPROB3-10, Viper will check for patterns of classes 3 to 10 to see if they recur routinely. This is very helpful to reduce false positives and false negatives.

topicid : string, optional

- Topicid represents an id for some entity. For example, if you have 1000 IoT devices, you can consume on a per device by entering its topicid that you gave when you produced the topic stream. Or, you can read from multiple topicids at the same time. For example, if you have 10 ids, then you can specify each one separated by a comma: 1,2,3,4,5,6,7,8,9,10 VIPER will read topicids in parallel. This can drastically speed up consumption of messages but will require more CPU.

rollbackoffsets : int, optional, enter value between 0 and 100

- This will rollback the streams by this percentage. For example, if using topicid, the main stream is rolled back by this percentage amount.

consumerid : string, required

- Consumer id associated with the topic

companyname : string, required

- Your company name

partition : int, optional

- set to Kafka partition number or -1 to autodect

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encrypted traffic, otherwise set to 0 for plaintext.

delay: int, optional

- Time in milliseconds before VIPER backsout from reading messages

offset: int, optional

- Offset to start the reading from..if 0 then reading will start from the beginning of the topic. If -1, VIPER will automatically go to the last offset. Or, you can extract the LastOffet from the returned JSON and use this offset for your next call.

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the contents read from the topic.

****5.1 maadstml.viperconsumefromtopicbatch(vipertoken,host,port,topic,consumerid,companyname, partition=-1,enabletls=0,delay=100,offset=0,**

brokerhost='',brokerport=-999,microserviceid='',topicid='-999',rollbackoffsets=0,preprocesstype='',timedelay=0,asyncti

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

asynctimeout : int, optional

-This is the timeout in seconds for the Python library async function.

timedelay : int, optional

- Timedelay is in SECONDS. Because batch runs continuously in the background, this will cause Viper to pause *timedelay* seconds when reading and writing to Kafka. For example, if the raw data is being generated every 3600 seconds, it may make sense to set timedelay=3600

topic : string, required

- Topic to consume from in the Kafka broker

preprocesstype : string, optional

- If you only want to search for record that have a particular processtype, you can enter: MIN, MAX, AVG, COUNT, COUNTSTR, DIFF, DIFFMARGIN, SUM, MEDIAN, VARIANCE, OUTLIERS, OUTLIERSX-Y, VARIED, ANOMPROB,ANOMPROBX-Y,ENTROPY, AUTOCORR, TREND, IQR (InterQuartileRange), Midhinge, CONSISTENCY, GM (Geometric mean), HM (Harmonic mean), Trimean, CV (coefficient of Variation), Mad (Mean absolute deviation), Skewness, Kurtosis, Spikedetect, Unique, Uniquestr, Timediff: time should be in this layout:2006-01-02T15:04:05, Timediff returns the difference in seconds between the first date/time and last datetime. Avgtimediff returns the average time in seconds between consecutive dates. Spikedetect uses a Zscore method to detect spikes in the data using lag of 5, StD of 3.5 from mean and influence of 0.5. Geodiff (returns distance in

Kilometers between two lat/long points) Unique Checks numeric data for duplication. Returns 1 if no data duplication (unique), 0 otherwise.

Dataage_[UTC offset]_[timetype], dataage can be used to check the last update time of the data in the data stream from current local time. You can specify the UTC offset to adjust the current time to match the timezone of the data stream. You can specify timetype as millisecond, second, minute, hour, day. For example, if Dataage_1_minute, then this processtype will compare the last timestamp in the data stream, to the local UTC time offset +1 and compute the time difference between the data stream timestamp and current local time and return the difference in minutes. This is a very powerful processtype for data quality and data assurance programs for any number of data streams.

Uniquestr Checks string data for duplication. Returns 1 if no data duplication (unique), 0 otherwise.

Uniquecount Checks numeric data for duplication. Returns count of unique numbers.

Uniquestrcount Checks string data for duplication. Returns count of unique strings.

CONSISTENCY checks if the data all have consistent data types. Returns 1 for consistent data types, 0 otherwise.

Meanci95 or Meanci99 - returns a 95% or 99% confidence interval: mean, low, high

RAW for no processing.

ANOMPROB=Anomaly probability, it will run several algorithms on the data stream window to determine a probaility of anomalous behaviour. This can be cross-refenced with OUTLIERS. It can be very powerful way to detection issues with devices.

ANOMPROBX-Y (similar to OUTLIERSX-Y), where X and Y are numbers, or "n". If "n", means examine all anomalies for patterns. They allow you to check if the anomalies in the streams are truly anomalies and not some pattern. For example, if a IoT device shuts off and turns on again routinely, this may be picked up as an anomaly when in fact it is normal behaviour. So, to ignore these cases, if ANOMPROB2-5, this tells Viper, check anomalies with patterns of 2-5 peaks. If the stream has two classes and these two classes are like 0 and 1000, and show a pattern, then they should not be considered an anomaly. Meaning, class=0, is the device shutting down, class=1000 is the device turning back on. If ANOMPROB3-10, Viper will check for patterns of classes 3 to 10 to see if they recur routinely. This is very helpful to reduce false positives and false negatives.

topicid : string, required

- Topicid represents an id for some entity. For example, if you have 1000 IoT devices, you can consume on a per device by entering its topicid that you gave when you produced the topic stream. Or, you can read from multiple topicids at the same time. For example, if you have 10 ids, then you can specify each one separated by a comma: 1,2,3,4,5,6,7,8,9,10 VIPER will read topicids in parallel. This can drastically speed up consumption of messages but will require more CPU. VIPER will consume continously from topic ids.

rollbackoffsets : int, optional, enter value between 0 and 100

- This will rollback the streams by this percentage. For example, if using topicid, the main stream is rolled back by this percentage amount.

consumerid : string, required

- Consumer id associated with the topic

companyname : string, required

- Your company name

partition : int, optional

- set to Kafka partition number or -1 to autodect

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encrypted traffic, otherwise set to 0 for plaintext.

delay: int, optional

- Time in milliseconds before VIPER backsout from reading messages

offset: int, optional

- Offset to start the reading from..if 0 then reading will start from the beginning of the topic. If -1, VIPER will automatically go to the last offset. Or, you can extract the LastOffet from the returned JSON and use this offset for your next call.

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the contents read from the topic.

****6. maadstml.viperhpdepredict(vipertoken,host,port,consumefrom,produceto,companyname,consumerid,producerid,**

hpdehost,inputdata,maxrows=0,algokey='',partition=-1,offset=-1,enabletls=1,delay=1000,hpdeport=-999,brokerhost='' brokerport=-999,timeout=120,usedeploy=0,microserviceid='',topicid=-999, maintopic='', streamstojoin='', array=0,pathtoalgos='')**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topicid : int, optional

- Topicid represents an id for some entity. For example, if you have 1000 IoT devices, with 10 subtopic streams you can assign a Topicid to each IoT device and each of the 10 subtopics will be associated to each IoT device. This way, you can do predictions for each IoT using its own custom ML model.

pathtoalgos : string, required

- Enter the full path to the root folder where the algorithms are stored.

maintopic : string, optional

- This is the name of the topic that contains the sub-topic streams.

array : int, optional

- Set array=1 if you produced data (from viperproducetotopic) as an array.

streamstojoin : string, optional

- These are the sub-topics you are streaming into maintopic. To do predictions, VIPER will automatically join these streams to create the input data for predictions for each Topicid.

consumefrom : string, required

- Topic to consume from in the Kafka broker

produceto : string, required

- Topic to produce results of the prediction to

companyname : string, required

- Your company name

consumerid: string, required

- Consumerid associated with the topic to consume from

producerid: string, required

- Producerid associated with the topic to produce to

inputdata: string, required

- This is a comma separated list of values that represent the independent variables in your algorithm. The order must match the order of the independent variables in your algorithm. OR, you can enter a data stream that contains the joined topics from *vipercreatejointopicstreams*.

maxrows: int, optional

- Use this to rollback the stream by maxrows offsets. For example, if you want to make 1000 predictions then set maxrows=1000, and make 1000 predictions from the current offset of the independent variables.

algokey: string, optional

- If you know the algorithm key that was returned by VIPERHPDETRAIING then you can specify it here. Specifying the algokey can drastically speed up the predictions.

partition : int, optional

- If you know the kafka partition used to store data then specify it here. Most cases Kafka will dynamically store data in partitions, so you should use the default of -1 to let VIPER find it.

offset : int, optional

- Offset to start consuming data. Usually you can use -1, and VIPER will get the last offset.

hpdehost: string, required

- Address of HPDE

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encryted traffic, otherwise 0 for plaintext.

delay: int, optional

- Time in milliseconds before VIPER backsout from reading messages

hpdeport: int, required

- Port number HPDE is listening on

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

timeout : int, optional

- Number of seconds that VIPER waits when trying to make a connection to HPDE.

usedeploy : int, optional

- If 0 will use algorithm in test, else if 1 use in production algorithm.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the prediction.

6.1 maadstml.viperhpdepredictbatch(vipertoken,host,port,consumefrom,produceto,companyname, consumerid,producerid,

hpdehost,inputdata,maxrows=0,algokey='',partition=-1,offset=-1,enabletls=1,delay=1000,hpdeport=-999,brokerhost='' brokerport=-999,timeout=120,usedeploy=0,microserviceid='',topicid="-999", maintopic='', streamstojoin='', array=0,timedelay=0,asynctimeout=120,pathtoalgos='')**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

asynctimeout : int, optional

-This is the timeout in seconds for the Python library async function.

timedelay : int, optional

- Timedelay is in SECONDS. Because batch runs continuously in the background, this will cause Viper to pause *timedelay* seconds when reading and writing to Kafka. For example, if the raw data is being generated every 3600 seconds, it may make sense to set timedelay=3600

topicid : string, required

- Topicid represents an id for some entity. For example, if you have 1000 IoT devices, with 10 subtopic streams you can assign a Topicid to each IoT device and each of the 10 subtopics will be associated to each IoT device. This way, you can do predictions for each IoT using its own custom ML model. Separate multiple topicids by a comma. For example, topicid="1,2,3,4,5" and viper will process at once.

pathtoalgos : string, required

- Enter the full path to the root folder where the algorithms are stored.

maintopic : string, optional

- This is the name of the topic that contains the sub-topic streams.

array : int, optional

- Set array=1 if you produced data (from viperproducetotopic) as an array.

streamstojoin : string, optional

- These are the sub-topics you are streaming into maintopic. To do predictions, VIPER will automatically join these streams to create the input data for predictions for each Topicid.

consumefrom : string, required

- Topic to consume from in the Kafka broker

produceto : string, required

- Topic to produce results of the prediction to

companyname : string, required

- Your company name

consumerid: string, required

- Consumerid associated with the topic to consume from

producerid: string, required

- Producerid associated with the topic to produce to

inputdata: string, required

- This is a comma separated list of values that represent the independent variables in your algorithm. The order must match the order of the independent variables in your algorithm. OR, you can enter a data stream that contains the joined topics from *vipercreatejointopicstreams*.

maxrows: int, optional

- Use this to rollback the stream by maxrows offsets. For example, if you want to make 1000 predictions then set maxrows=1000, and make 1000 predictions from the current offset of the independent variables.

algokey: string, optional

- If you know the algorithm key that was returned by VIPERHPDETRAIING then you can specify it here. Specifying the algokey can drastically speed up the predictions.

partition : int, optional

- If you know the kafka partition used to store data then specify it here. Most cases Kafka will dynamically store data in partitions, so you should use the default of -1 to let VIPER find it.

offset : int, optional

- Offset to start consuming data. Usually you can use -1, and VIPER will get the last offset.

hpdehost: string, required

- Address of HPDE

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encryted traffic, otherwise 0 for plaintext.

delay: int, optional

- Time in milliseconds before VIPER backsout from reading messages

hpdeport: int, required

- Port number HPDE is listening on

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

timeout : int, optional

- Number of seconds that VIPER waits when trying to make a connection to HPDE.

usedeploy : int, optional

- If 0 will use algorithm in test, else if 1 use in production algorithm.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the prediction.

****6.2. maadstml.viperhpdepredictprocess(vipertoken,host,port,consumefrom,produceto,companyname,consumerid,producerid,hpdehost,inputdata,processtype,maxrows=0,**

algokey='',partition=-1,offset=-1,enabletls=1,delay=1000,hpdeport=-999,brokerhost='',brokerport=9092, timeout=120,usedeploy=0,microserviceid='',topicid=-999, maintopic='', streamstojoin='',array=0,pathtoalgos='')**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topicid : int, optional

- Topicid represents an id for some entity. For example, if you have 1000 IoT devices, with 10 subtopic streams you can assign a Topicid to each IoT device and each of the 10 subtopics will be associated to each IoT device. This way, you can do predictions for each IoT using its own custom ML model.

pathtoalgos : string, required

- Enter the full path to the root folder where the algorithms are stored.

maintopic : string, optional

- This is the name of the topic that contains the sub-topic streams.

array : int, optional

- Set array=1 if you produced data (from viperproducetotopic) as an array.

streamstojoin : string, optional

- These are the sub-topics you are streaming into maintopic. To do predictions, VIPER will automatically join these streams to create the input data for predictions for each Topicid.

consumefrom : string, required

- Topic to consume from in the Kafka broker

produceto : string, required

- Topic to produce results of the prediction to

companyname : string, required

- Your company name

consumerid: string, required

- Consumerid associated with the topic to consume from

producerid: string, required

- Producerid associated with the topic to produce to

inputdata: string, required

- This is a comma separated list of values that represent the independent variables in your algorithm. The order must match the order of the independent variables in your algorithm. OR, you can enter a data stream that contains the joined topics from *vipercreatejointopicstreams*.

processtype: string, required

- This must be: max, min, avg, median, trend, all. For example, to find the maximum or the best human or machine. Trend will compute the predictions are trending. Avg is the average of all predictions. Median is the median of predictions. All will produce all predictions.

maxrows: int, optional

- Use this to rollback the stream by maxrows offsets. For example, if you want to make 1000 predictions then set maxrows=1000, and make 1000 predictions from the current offset of the independent variables.

algokey: string, optional

- If you know the algorithm key that was returned by VIPERHPDETRAIING then you can specify it here. Specifying the algokey can drastically speed up the predictions.

partition : int, optional

- If you know the kafka partition used to store data then specify it here. Most cases Kafka will dynamically store data in partitions, so you should use the default of -1 to let VIPER find it.

offset : int, optional

- Offset to start consuming data. Usually you can use -1, and VIPER will get the last offset.

hpdehost: string, required

- Address of HPDE

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encryted traffic, otherwise 0 for plaintext.

delay: int, optional

- Time in milliseconds before VIPER backsout from reading messages

hpdeport: int, required

- Port number HPDE is listening on

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

timeout : int, optional

- Number of seconds that VIPER waits when trying to make a connection to HPDE.

usedeploy : int, optional

- If 0 will use algorithm in test, else if 1 use in production algorithm.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the prediction.

****7. maadstml.viperhpdeoptimize(vipertoken,host,port,consumefrom,produceto,companyname,consumerid,producerid,**

hpdehost,partition=-1,offset=-1,enabletls=0,delay=100,hpdeport=-999,usedeploy=0,ismin=1,constraints='best',

stretchbounds=20,constrainttype=1,epsilon=10,brokerhost='',brokerport=-999,timeout=120,microserviceid='',topicid=-999)**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

consumefrom : string, required

- Topic to consume from in the Kafka broker

topicid : int, optional

- Topicid represents an id for some entity. For example, if you have 1000 IoT devices, you can perform mathematical optimization for each of the 1000 IoT devices using their specific algorithm.

produceto : string, required

- Topic to produce results of the prediction to

companyname : string, required

- Your company name

consumerid: string, required

- Consumerid associated with the topic to consume from

producerid: string, required

- Producerid associated with the topic to produce to

hpdehost: string, required

- Address of HPDE

partition : int, optional

- If you know the kafka partition used to store data then specify it here. Most cases Kafka will dynamically store data in partitions, so you should use the default of -1 to let VIPER find it.

offset : int, optional

- Offset to start consuming data. Usually you can use -1, and VIPER will get the last offset.

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encrypted traffic, otherwise set to 0 for plaintext.

delay: int, optional

- Time in milliseconds before VIPER backsout from reading messages

hpdeport: int, required

- Port number HPDE is listening on

usedeploy **:** *int, optional*

- If 0 will use algorithm in test, else if 1 use in production algorithm.

ismin : int, optional - If 1 then function is minimized, else if 0 the function is maximized

constraints: string, optional

- If "best" then HPDE will choose the best values of the independent variables to minmize or maximize the dependent variable. Users can also specify their own constraints for each variable and must be in the following format: varname1:min:max,varname2:min:max,...

stretchbounds: int, optional

- A number between 0 and 100, this is the percentage to stretch the bounds on the constraints.

constrainttype: int, optional

- If 1 then HPDE uses the min/max of each variable for the bounds, if 2 HPDE will adjust the min/max by their standard deviation, if 3 then HPDE uses stretchbounds to adjust the min/max for each variable.

epsilon: int, optional

- Once HPDE finds a good local minima/maxima, it then uses this epsilon value to find the Global minima/maxima to ensure you have the best values of the independent variables that minimize or maximize the dependent variable.

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

timeout : int, optional

- Number of seconds that VIPER waits when trying to make a connection to HPDE.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the optimization details and optimal values.

****7.1 maadstml.viperhpdeoptimizebatch(vipertoken,host,port,consumefrom,produceto,companyname,consumerid,producerid,**

hpdehost,partition=-1,offset=-1,enabletls=0,delay=100,hpdeport=-999,usedeploy=0,ismin=1,constraints='best',

stretchbounds=20,constrainttype=1,epsilon=10,brokerhost='',brokerport=-999,timeout=120,microserviceid='',topicid="-999", timedelay=0,asynctimeout=120)**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

asynctimeout : int, optional

-This is the timeout in seconds for the Python library async function.

timedelay : int, optional

- Timedelay is in SECONDS. Because batch runs continuously in the background, this will cause Viper to pause *timedelay* seconds when reading and writing to Kafka. For example, if the raw data is being generated every 3600 seconds, it may make sense to set timedelay=3600

consumefrom : string, required

- Topic to consume from in the Kafka broker

topicid : string, required

- Topicid represents an id for some entity. For example, if you have 1000 IoT devices, you can perform mathematical optimization for each of the 1000 IoT devices using their specific algorithm. Separate multiple topicids by a comma.

produceto : string, required

- Topic to produce results of the prediction to

companyname : string, required

- Your company name

consumerid: string, required

- Consumerid associated with the topic to consume from

producerid: string, required

- Producerid associated with the topic to produce to

hpdehost: string, required

- Address of HPDE

partition : int, optional

- If you know the kafka partition used to store data then specify it here. Most cases Kafka will dynamically store data in partitions, so you should use the default of -1 to let VIPER find it.

offset : int, optional

- Offset to start consuming data. Usually you can use -1, and VIPER will get the last offset.

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encrypted traffic, otherwise set to 0 for plaintext.

delay: int, optional

- Time in milliseconds before VIPER backsout from reading messages

hpdeport: int, required

- Port number HPDE is listening on

***usedeploy* :** *int, optional*

- If 0 will use algorithm in test, else if 1 use in production algorithm.

ismin : int, optional - If 1 then function is minimized, else if 0 the function is maximized

constraints: string, optional

- If "best" then HPDE will choose the best values of the independent variables to minmize or maximize the dependent variable. Users can also specify their own constraints for each variable and must be in the following format: varname1:min:max,varname2:min:max,...

stretchbounds: int, optional

- A number between 0 and 100, this is the percentage to stretch the bounds on the constraints.

constrainttype: int, optional

- If 1 then HPDE uses the min/max of each variable for the bounds, if 2 HPDE will adjust the min/max by their standard deviation, if 3 then HPDE uses stretchbounds to adjust the min/max for each variable.

epsilon: int, optional

- Once HPDE finds a good local minima/maxima, it then uses this epsilon value to find the Global minima/maxima to ensure you have the best values of the independent variables that minimize or maximize the dependent variable.

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

timeout : int, optional

- Number of seconds that VIPER waits when trying to make a connection to HPDE.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the optimization details and optimal values.

****8. maadstml.viperhpdetraining(vipertoken,host,port,consumefrom,produceto,companyname,consumerid,producerid,**

hpdehost,viperconfigfile,enabletls=1,partition=-1,deploy=0,modelruns=50,modelsearchtuner=80,hpdeport=-999,

offset=-1,islogistic=0,brokerhost='',
brokerport=-999,timeout=120,microserviceid='',topicid=-999,maintopic='',

independentvariables='',dependentvariable='',rollbackoffsets=0,fullpathtotrainingdata='',processlogic='',

identifier='',array=0,transformtype='',sendcoefto='',coeftoprocess='',coefsubtopicnames='')**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

transformtype : string, optional

- You can transform the dependent and independent variables using: log-log, log-lin, lin-log, lin=linear, log=natural log This may be useful if you want to compute price or demand elasticities.

sendcoefto : string, optional

- This is the name of the kafka topic that you want to stream the estimated parameters to.

coeftoprocess : string, optional

- This is the indexes of the estimated parameters. For example, if the ML model has a constant and two estimated parameters, then coeftoprocess="0,1,2" means stream constant term (at index 0) and the two estmiated parameters at index 1, and 2.

coefsubtopicnames : string, optional

- This is the names for the estimated parameters. For example, "constant,elasticity,elasticity2" would be streamed as kafka topics for *coeftoprocess*

topicid : int, optional

- Topicid represents an id for some entity. For example, if you have 1000 IoT devices, you can create individual Machine Learning models for each IoT device in real-time. This is a core functionality of TML solutions.

array : int, optional

- Set array=1 if the data you are consuming from is an array of multiple streams that you produced from viperproducetotopic in an effort to synchronize data for training.

maintopic : string, optional

- This is the maintopic that contains the sub-topc streams.

independentvariables : string, optional

- These are the independent variables that are the subtopics.

dependentvariable : string, optional

- This is the dependent variable in the subtopic streams.

rollbackoffsets: int, optional

- This is the rollback percentage to create the training dataset. VIPER will automatically create a training dataset using the independent and dependent variable streams.

fullpathtotrainingdata: string, optional

- This is the FULL path where you want to store the training dataset. VIPER will write file to disk. Make sure proper permissions are granted to VIPER. For example, **c:/myfolder/mypath**

processlogic : string, optional

- You can dynamically build a classification model by specifying how you want to classify the dependent variable by indicating your conditions in the processlogic variable (this will take effect if islogistic=1). For example:

  processlogic='classification_name=my_prob:temperature=20.5,30:humidity=50,55', means the following:

  1. The name of the dependent variable is specified by **classification_name**
  2. Then you can specify the conditions on the streams. If your stream is Temperature and humidity, if Temperature is between 20.5 and 30, then my_prob=1, otherwise my_prob=0, and

     if Humidity is between 50 and 55, then my_prob=1, otherwise my_prob=0
  3. If you want to specify no upperbound you can use *n*, or *-n* for no lowerbound. For example, if **temperature=20.5,n**, means temperature >=20.5 then my_prob=1

     If **humidity=-n,55**, means humidity<=55 then my_prob=1

- This allows you to classify the dependent with any number of variables all in real-time!

consumefrom : string, required

- Topic to consume from in the Kafka broker

produceto : string, required

- Topic to produce results of the prediction to

companyname : string, required

- Your company name

consumerid: string, required

identifier: string, optional

- You can add any name or identifier like DSN ID
- Consumerid associated with the topic to consume from

producerid: string, required

- Producerid associated with the topic to produce to

hpdehost: string, required

- Address of HPDE

viperconfigfile : string, required

- Full path to VIPER.ENV configuration file on server.

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encrypted traffic, otherwise set to 0 for plaintext.

partition: int, optional

- Partition used by kafka to store data. NOTE: Kafka will dynamically store data in partitions. Unless you know for sure the partition, you should use the default of -1 to let VIPER determine where your data is.

deploy: int, optional

- If deploy=1, this will deploy the algorithm to the Deploy folder. This is useful if you do not want to use this algorithm in production, and just testing it. If just testing, then set deploy=0 (default).

modelruns: int, optional

- Number of iterations for model training

modelsearchtuner: int, optional

- An integer between 0-100, this variable will attempt to fine tune the model search space. A number close to 0 means you will have lots of models but their quality may be low, a number close to 100 (default=80) means you will have fewer models but their quality will be higher

hpdeport: int, required

- Port number HPDE is listening on

offset : int, optional

- If 0 will use the training data from the beginning of the topic

islogistic: int, optional

- If is 1, the HPDE will switch to logistic modeling, else continous.

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

timeout : int, optional

- Number of seconds that VIPER waits when trying to make a connection to HPDE.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the optimal algorithm that best fits your data.

****8.1 maadstml.viperhpdetrainingbatch(vipertoken,host,port,consumefrom,produceto,companyname, consumerid,producerid,**

hpdehost,viperconfigfile,enabletls=1,partition=-1,deploy=0,modelruns=50,modelsearchtuner=80, hpdeport=-999,

offset=-1,islogistic=0,brokerhost='',
brokerport=-999,timeout=120,microserviceid='',topicid="-999",maintopic='',

independentvariables='',dependentvariable='',rollbackoffsets=0,fullpathtotrainingdata='',processlogic='',

identifier='',array=0,timedelay=0,asynctimeout=120)**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

asynctimeout : int, optional

-This is the timeout in seconds for the Python library async function.

timedelay : int, optional

- Timedelay is in SECONDS. Because batch runs continuously in the background, this will cause Viper to pause *timedelay* seconds when reading and writing to Kafka. For example, if the raw data is being generated every 3600 seconds, it may make sense to set timedelay=3600

topicid : string, required

- Topicid represents an id for some entity. For example, if you have 1000 IoT devices, you can create individual Machine Learning models for each IoT device in real-time. This is a core functionality of TML solutions. Separate multiple topic ids by comma.

array : int, optional

- Set array=1 if the data you are consuming from is an array of multiple streams that you produced from viperproducetotopic in an effort to synchronize data for training.

maintopic : string, optional

- This is the maintopic that contains the sub-topc streams.

independentvariables : string, optional

- These are the independent variables that are the subtopics.

dependentvariable : string, optional

- This is the dependent variable in the subtopic streams.

rollbackoffsets: int, optional

- This is the rollback percentage to create the training dataset. VIPER will automatically create a training dataset using the independent and dependent variable streams.

fullpathtotrainingdata: string, optional

- This is the FULL path where you want to store the training dataset. VIPER will write file to disk. Make sure proper permissions are granted to VIPER. For example, **c:/myfolder/mypath**

processlogic : string, optional

- You can dynamically build a classification model by specifying how you want to classify the dependent variable by indicating your conditions in the processlogic variable (this will take effect if islogistic=1). For example:

  processlogic='classification_name=my_prob:temperature=20.5,30:humidity=50,55', means the following:

  1. The name of the dependent variable is specified by **classification_name**
  2. Then you can specify the conditions on the streams. If your stream is Temperature and humidity, if Temperature is between 20.5 and 30, then my_prob=1, otherwise my_prob=0, and

     if Humidity is between 50 and 55, then my_prob=1, otherwise my_prob=0
  3. If you want to specify no upperbound you can use *n*, or *-n* for no lowerbound. For example, if **temperature=20.5,n**, means temperature >=20.5 then my_prob=1

     If **humidity=-n,55**, means humidity<=55 then my_prob=1
- This allows you to classify the dependent with any number of variables all in real-time!

consumefrom : string, required

- Topic to consume from in the Kafka broker

produceto : string, required

- Topic to produce results of the prediction to

companyname : string, required

- Your company name

consumerid: string, required

identifier: string, optional

- You can add any name or identifier like DSN ID
- Consumerid associated with the topic to consume from

producerid: string, required

- Producerid associated with the topic to produce to

hpdehost: string, required

- Address of HPDE

viperconfigfile : string, required

- Full path to VIPER.ENV configuration file on server.

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encrypted traffic, otherwise set to 0 for plaintext.

partition: int, optional

- Partition used by kafka to store data. NOTE: Kafka will dynamically store data in partitions. Unless you know for sure the partition, you should use the default of -1 to let VIPER determine where your data is.

deploy: int, optional

- If deploy=1, this will deploy the algorithm to the Deploy folder. This is useful if you do not want to use this algorithm in production, and just testing it. If just testing, then set deploy=0 (default).

modelruns: int, optional

- Number of iterations for model training

modelsearchtuner: int, optional

- An integer between 0-100, this variable will attempt to fine tune the model search space. A number close to 0 means you will have lots of models but their quality may be low, a number close to 100 (default=80) means you will have fewer models but their quality will be higher

hpdeport: int, required

- Port number HPDE is listening on

offset : int, optional

- If 0 will use the training data from the beginning of the topic

islogistic: int, optional

- If is 1, the HPDE will switch to logistic modeling, else continous.

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

timeout : int, optional

- Number of seconds that VIPER waits when trying to make a connection to HPDE.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the optimal algorithm that best fits your data.

****9. maadstml.viperproducetotopicstream(vipertoken,host,port,topic,producerid,offset,maxrows=0,enabletls=0,delay=100,**

brokerhost='',brokerport=-999,microserviceid='',topicid=-999,mainstreamtopic='',streamstojoin='')**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topic : string, required

- Topics to produce to in the Kafka broker - this is a topic that contains multiple topics, VIPER will consume from each topic and write results to the produceto topic

topicid : int, optional

- Topicid represents an id for some entity. For example, if you have 1000 IoT devices, you can join these streams and produce it to one stream,

mainstreamtopic: string, optional

- This is the main stream topic that contain the subtopic streams.

streamstojoin: string, optional

- These are the streams you want to join and produce to mainstreamtopic.

producerid : string, required

- Producerid of the topic producing to

offset : int

- If 0 will use the stream data from the beginning of the topics, -1 will automatically go to last offset

maxrows : int, optional

- If offset=-1, this number will rollback the streams by maxrows amount i.e. rollback=lastoffset-maxrows

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encrypted traffic, otherwise 0 for plaintext

delay: int, optional

- Time in milliseconds before VIPER backsout from reading messages

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the optimal algorithm that best fits your data.

****10. maadstml.vipercreatetrainingdata(vipertoken,host,port,consumefrom,produceto,dependentvariable,**

independentvariables,consumerid,producerid,companyname,partition=-1,enabletls=0,delay=100, brokerhost='',brokerport=-999,microserviceid='',topicid=-999)**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

consumefrom : string, required

- Topic to consume from

topicid : int, optional

- Topicid represents an id for some entity. For example, if you have 1000 IoT devices, with 10 subtopic streams you can assign a Topicid to each IoT device and each of the 10 subtopics will be associated to each IoT device. You can create training dataset for each device.

produceto : string, required

- Topic to produce to

dependentvariable : string, required

- Topic name of the dependentvariable

independentvariables : string, required

- Topic names of the independentvariables - VIPER will automatically read the data streams. Separate multiple variables by comma.

consumerid : string, required

- Consumerid of the topic to consume to

producerid : string, required

- Producerid of the topic producing to

partition : int, optional

- This is the partition that Kafka stored the stream data. Specifically, the streams you joined from function *viperproducetotopicstream* will be stored in a partition by Kafka, if you want to create a training dataset from these data, then you should use this partition. This ensures you are using the right data to create a training dataset.

companyname : string, required

- Your company name

enabletls: int, optional

- Set to 1 if Kafka broker is enabled for SSL/TLS encrypted traffic, otherwise set to 0 for plaintext.

delay: int, optional

- Time in milliseconds before VIPER backout from reading messages

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the training data set.

11. maadstml.vipercreatetopic(vipertoken,host,port,topic,companyname,contactname,contactemail,location, description,enabletls=0,brokerhost='',brokerport=-999,numpartitions=1,replication=1,microserviceid='')

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topic : string, required

- Topic to create

companyname : string, required

- Company name of consumer

contactname : string, required

- Contact name of consumer

contactemail : string, required

- Contact email of consumer

location : string, required

- Location of consumer

description : string, required

- Description of why consumer wants to subscribe to topic

enabletls : int, optional

- Set to 1 if Kafka is SSL/TLS enabled for encrypted traffic, otherwise 0 for no encryption (plain text)

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

numpartitions: int, optional

- Number of the parititons to create in the Kafka broker - more parititons the faster Kafka will produce results.

replication: int, optional

- Specificies the number of brokers to replicate to - this is important for failover

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the producer id for the topic.

****12. maadstml.viperconsumefromstreamtopic(vipertoken,host,port,topic,consumerid,companyname,partition=-1,**

enabletls=0,delay=100,offset=0,brokerhost='',brokerport=-999,microserviceid='',topicid=-999)**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topic : string, required

- Topic to consume from

topicid : int, optional

- Topicid represents an id for some entity. For example, if you have 1000 IoT devices, you can consume for each device.

consumerid : string, required

- Consumerid associated with topic

companyname : string, required

- Your company name

partition: int, optional

- Set to a kafka partition number, or -1 to autodetect partition.

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encrypted traffic, otherwise set to 0 for plaintext.

delay: int, optional

- Time in milliseconds before VIPER backsout from reading messages

offset : int, optional

- Offset to start reading from ..if 0 VIPER will read from the beginning

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the contents of all the topics read

****13. maadstml.vipercreatejointopicstreams(vipertoken,host,port,topic,topicstojoin,companyname,contactname,contactemail,**

description,location,enabletls=0,brokerhost='',brokerport=-999,replication=1,numpartitions=1,microserviceid='', topicid=-999)**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topic : string, required

- Topic to consume from

topicid : int, optional

- Topicid represents an id for some entity. Create a joined topic stream per topicid.

topicstojoin : string, required

- Enter two or more topics separated by a comma and VIPER will join them into one topic

companyname : string, required

- Company name of consumer

contactname : string, required

- Contact name of consumer

contactemail : string, required

- Contact email of consumer

location : string, required

- Location of consumer

description : string, required

- Description of why consumer wants to subscribe to topic

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled, otherwise set to 0 for plaintext.

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

numpartitions : int, optional

- Number of partitions

replication : int, optional

- Replication factor

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the producerid of the joined streams

****14. maadstml.vipercreateconsumergroup(vipertoken,host,port,topic,groupname,companyname,contactname,contactemail,**

description,location,enabletls=1,brokerhost='',brokerport=-999,microserviceid='')**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topic : string, required

- Topic to dd to the group, multiple (active) topics can be separated by comma

groupname : string, required

- Enter the name of the group

companyname : string, required

- Company name of consumer

contactname : string, required

- Contact name of consumer

contactemail : string, required

- Contact email of consumer

location : string, required

- Location of consumer

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled, otherwise set to 0 for plaintext.

description : string, required

- Description of why consumer wants to subscribe to topic

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the groupid of the group.

****15. maadstml.viperconsumergroupconsumefromtopic(vipertoken,host,port,topic,consumerid,groupid,companyname,**

partition=-1,enabletls=0,delay=100,offset=0,rollbackoffset=0,brokerhost='',brokerport=-999,microserviceid='')**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topic : string, required

- Topic to dd to the group, multiple (active) topics can be separated by comma

consumerid : string, required

- Enter the consumerid associated with the topic

groupid : string, required

- Enter the groups id

companyname : string, required

- Enter the company name

partition: int, optional

- set to Kakfa partition number or -1 to autodetect

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled, otherwise set to 0 for plaintext.

delay: int, optional

- Time in milliseconds before VIPER backsout from reading messages

offset : int, optional

- Offset to start reading from. If 0, will read from the beginning of topic, or -1 to automatically go to end of topic.

rollbackoffset : int, optional

- The number of offsets to rollback the data stream.

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the contents of the group.

16. maadstml.vipermodifyconsumerdetails(vipertoken,host,port,topic,companyname,consumerid,contactname='', contactemail='',location='',brokerhost='',brokerport=9092,microserviceid='')

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topic : string, required

- Topic to dd to the group, multiple (active) topics can be separated by comma

consumerid : string, required

- Enter the consumerid associated with the topic

companyname : string, required

- Enter the company name

contactname : string, optional

- Enter the contact name

contactemail : string, optional - Enter the contact email

location : string, optional

- Enter the location

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns success/failure

****17. maadstml.vipermodifytopicdetails(vipertoken,host,port,topic,companyname,partition=0,enabletls=1,** isgroup=0,contactname='',contactemail='',location='',brokerhost='',brokerport=9092,microserviceid='')**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topic : string, required

- Topic to dd to the group, multiple (active) topics can be separated by comma

companyname : string, required

- Enter the company name

partition : int, optional

- You can change the partition in the Kafka topic.

enabletls : int, optional

- If enabletls=1, then SSL/TLS is enables in Kafka, otherwise if enabletls=0 it is not.

isgroup : int, optional

- This tells VIPER whether this is a group topic if isgroup=1, or a normal topic if isgroup=0

contactname : string, optional

- Enter the contact name

contactemail : string, optional - Enter the contact email

location : string, optional

- Enter the location

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns success/failure

18. maadstml.viperactivatetopic(vipertoken,host,port,topic,microserviceid='')

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topic : string, required

- Topic to activate

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns success/failure

19. maadstml.viperdeactivatetopic(vipertoken,host,port,topic,microserviceid='')

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topic : string, required

- Topic to deactivate

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns success/failure

20. maadstml.vipergroupactivate(vipertoken,host,port,groupname,groupid,microserviceid='')

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

groupname : string, required

- Name of the group

groupid : string, required

- ID of the group

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns success/failure

21. maadstml.vipergroupdeactivate(vipertoken,host,port,groupname,groupid,microserviceid='')

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

groupname : string, required

- Name of the group

groupid : string, required

- ID of the group

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns success/failure

22. maadstml.viperdeletetopics(vipertoken,host,port,topic,enabletls=1,brokerhost='',brokerport=9092,microserviceid='')

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topic : string, required

- Topic to delete. Separate multiple topics by a comma.

enabletls : int, optional

- If enabletls=1, then SSL/TLS is enable on Kafka, otherwise if enabletls=0, it is not.

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

microserviceid : string, optional

- microservice to access viper

23. maadstml.balancebigdata(localcsvfile,numberofbins,maxrows,outputfile,bincutoff,distcutoff,startcolumn=0)

Parameters:

localcsvfile : string, required

- Local file, must be CSV formatted.

numberofbins : int, required

- The number of bins for the histogram. You can set to any value but 10 is usually fine.

maxrows : int, required

- The number of rows to return, which will be a subset of your original data.

outputfile : string, required

- Your new data will be writted as CSV to this file.

bincutoff : float, required.

- This is the threshold percentage for the bins. Specifically, the data in each variable is allocated to bins, but many times it will not fall in ALL of the bins. By setting this percentage between 0 and 1, MAADS will choose variables that exceed this threshold to determine which variables have data that are well distributed across bins. The variables with the most distributed values in the bins will drive the selection of the rows in your dataset that give the best distribution - this will be very important for MAADS training. Usually 0.7 is good.

distcutoff : float, required.

- This is the threshold percentage for the distribution. Specifically, MAADS uses a Lilliefors statistic to determine whether the data are well distributed. The lower the number the better. Usually 0.45 is good.

startcolumn : int, optional

- This tells MAADS which column to start from. If you have DATE in the first column, you can tell MAADS to start from 1 (columns are zero-based)

RETURNS: Returns a detailed JSON object and new balaced dataset written to outputfile.

****24. maadstml.viperanomalytrain(vipertoken,host,port,consumefrom,produceto,producepeergroupto,produceridpeergroup,consumeridproduceto,**

streamstoanalyse,companyname,consumerid,producerid,flags,hpdehost,viperconfigfile, enabletls=1,partition=-1,hpdeport=-999,topicid=-999,maintopic='',rollbackoffsets=0,fullpathtotrainingdata='',

brokerhost='',brokerport=9092,delay=1000,timeout=120,microserviceid='')**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

consumefrom : string, required

- Topic to consume from in the Kafka broker

produceto : string, required

- Topic to produce results of the prediction to

topicid : int, optional

- Topicid represents an id for some entity. For example, if you have 1000 IoT devices, you can perform anomaly detection/predictions for each device.

maintopic : string, optional

- This is the maintopic that contains the subtopic streams.

rollbackoffsets: int, optional

- This is the percentage to rollback the streams that you are analysing: streamstoanalyse

fullpathtotrainingdata: string, optional

- This is the full path to the training dataset to use to find peer groups.

producepeergroupto : string, required

- Topic to produce the peer group for anomaly comparisons

produceridpeergroup : string, required

- Producerid for the peer group topic

consumeridproduceto : string, required

- Consumer id for the Produceto topic

streamstoanalyse : string, required

- Comma separated list of streams to analyse for anomalies

flags : string, required

- These are flags that will be used to select the peer group for each stream. The flags must have the following format: *topic=[topic name],topictype=[numeric or string],threshnumber=[a number between 0 and 10000, i.e. 200], lag=[a number between 1 and 20, i.e. 5],zthresh=[a number between 1 and 5, i.e. 2.5],influence=[a number between 0 and 1 i.e. 0.5]*

  threshnumber: decimal number to determine usual behaviour - only for numeric streams, numbers are compared to the centroid number, a standardized distance is taken and all numbers below the thresholdnumeric are deemed as usual i.e. thresholdnumber=200, any value below is close to the centroid - you need to experiment with this number.

  lag: number of lags for the moving mean window, works to smooth the function i.e. lag=5

  zthresh: number of standard deviations from moving mean i.e. 3.5

  influence: strength in identifying outliers for both stationary and non-stationary data, i.e. influence=0 ignores outliers when recalculating the new threshold, influence=1 is least robust. Influence should be between (0,1), i.e. influence=0.5

  Flags must be provided for each topic. Separate multiple flags by ~

companyname : string, required

- Your company name

consumerid: string, required

- Consumerid associated with the topic to consume from

producerid: string, required

- Producerid associated with the topic to produce to

hpdehost: string, required

- Address of HPDE

viperconfigfile : string, required

- Full path to VIPER.ENV configuration file on server.

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encrypted traffic, otherwise set to 0 for plaintext.

partition: int, optional

- Partition used by kafka to store data. NOTE: Kafka will dynamically store data in partitions. Unless you know for sure the partition, you should use the default of -1 to let VIPER determine where your data is.

hpdeport: int, required

- Port number HPDE is listening on

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

delay : int, optional

- delay parameter to wait for Kafka to respond - in milliseconds.

timeout : int, optional

- Number of seconds that VIPER waits when trying to make a connection to HPDE.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the peer groups for all the streams.

****24.1 maadstml.viperanomalytrainbatch(vipertoken,host,port,consumefrom,produceto,producepeergroupto,produceridpeergroup,consumeridproduceto,**

streamstoanalyse,companyname,consumerid,producerid,flags,hpdehost,viperconfigfile, enabletls=1,partition=-1,hpdeport=-999,topicid="-999",maintopic='',rollbackoffsets=0,fullpathtotrainingdata='',

brokerhost='',brokerport=9092,delay=1000,timeout=120,microserviceid='',timedelay=0,asynctimeout=120)**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

asynctimeout : int, optional

-This is the timeout in seconds for the Python library async function.

timedelay : int, optional

- Timedelay is in SECONDS. Because batch runs continuously in the background, this will cause Viper to pause *timedelay* seconds when reading and writing to Kafka. For example, if the raw data is being generated every 3600 seconds, it may make sense to set timedelay=3600

consumefrom : string, required

- Topic to consume from in the Kafka broker

produceto : string, required

- Topic to produce results of the prediction to

topicid : string, required

- Topicid represents an id for some entity. For example, if you have 1000 IoT devices, you can perform anomaly detection/predictions for each device. Separate multiple topicids by a comma.

maintopic : string, optional

- This is the maintopic that contains the subtopic streams.

rollbackoffsets: int, optional

- This is the percentage to rollback the streams that you are analysing: streamstoanalyse

fullpathtotrainingdata: string, optional

- This is the full path to the training dataset to use to find peer groups.

producepeergroupto : string, required

- Topic to produce the peer group for anomaly comparisons

produceridpeergroup : string, required

- Producerid for the peer group topic

consumeridproduceto : string, required

- Consumer id for the Produceto topic

streamstoanalyse : string, required

- Comma separated list of streams to analyse for anomalies

flags : string, required

- These are flags that will be used to select the peer group for each stream. The flags must have the following format: *topic=[topic name],topictype=[numeric or string],threshnumber=[a number between 0 and 10000, i.e. 200], lag=[a number between 1 and 20, i.e. 5],zthresh=[a number between 1 and 5, i.e. 2.5],influence=[a number between 0 and 1 i.e. 0.5]*

  threshnumber: decimal number to determine usual behaviour - only for numeric streams, numbers are compared to the centroid number, a standardized distance is taken and all numbers below the thresholdnumeric are deemed as usual i.e. thresholdnumber=200, any value below is close to the centroid - you need to experiment with this number.

  lag: number of lags for the moving mean window, works to smooth the function i.e. lag=5

  zthresh: number of standard deviations from moving mean i.e. 3.5

  influence: strength in identifying outliers for both stationary and non-stationary data, i.e. influence=0 ignores outliers when recalculating the new threshold, influence=1 is least robust. Influence should be between (0,1), i.e. influence=0.5

  Flags must be provided for each topic. Separate multiple flags by ~

companyname : string, required

- Your company name

consumerid: string, required

- Consumerid associated with the topic to consume from

producerid: string, required

- Producerid associated with the topic to produce to

hpdehost: string, required

- Address of HPDE

viperconfigfile : string, required

- Full path to VIPER.ENV configuration file on server.

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encrypted traffic, otherwise set to 0 for plaintext.

partition: int, optional

- Partition used by kafka to store data. NOTE: Kafka will dynamically store data in partitions. Unless you know for sure the partition, you should use the default of -1 to let VIPER determine where your data is.

hpdeport: int, required

- Port number HPDE is listening on

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

delay : int, optional

- delay parameter to wait for Kafka to respond - in milliseconds.

timeout : int, optional

- Number of seconds that VIPER waits when trying to make a connection to HPDE.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the peer groups for all the streams.

****25. maadstml.viperanomalypredict(vipertoken,host,port,consumefrom,produceto,consumeinputstream,produceinputstreamtest,produceridinputstreamtest,**

streamstoanalyse,consumeridinputstream,companyname,consumerid,producerid,flags,hpdehost,viperconfigfile,

enabletls=1,partition=-1,hpdeport=-999,topicid=-999,maintopic='',rollbackoffsets=0,fullpathtopeergroupdata='',

brokerhost='',brokerport=9092,delay=1000,timeout=120,microserviceid='')**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

consumefrom : string, required

- Topic to consume from in the Kafka broker

produceto : string, required

- Topic to produce results of the prediction to

consumeinputstream : string, required

- Topic of the input stream to test for anomalies

produceinputstreamtest : string, required

- Topic to store the input stream data for analysis

produceridinputstreamtest : string, required

- Producer id for the produceinputstreamtest topic

streamstoanalyse : string, required

- Comma separated list of streams to analyse for anomalies

flags : string, required

- These are flags that will be used to select the peer group for each stream. The flags must have the following format: *riskscore=[a number between 0 and 1]~complete=[and, or, pvalue i.e. p50 means streams over 50% that have an anomaly]~type=[and,or this will determine what logic to apply to v and sc],topic=[topic name],topictype=[numeric or string],v=[v>some value, v<some value, or valueany], sc=[sc>some number, sc<some number - this is the score for the anomaly test]

  if using strings, the specify flags: type=[and,or],topic=[topic name],topictype=string,stringcontains=[0 or 1 - 1 will do a substring test, 0 will equate the strings],v2=[any text you want to test - use | for OR or ^ for AND],sc=[score value, sc<some value, sc>some value]

  riskscore: this the riskscore threshold. A decimal number between 0 and 1, use this as a threshold to flag anomalies.

  complete : If using multiple streams, this will test each stream to see if the computed riskscore and perform an AND or OR on each risk value and take an average of the risk scores if using AND. Otherwise if at least one stream exceeds the riskscore it will return.

  type: AND or OR - if using v or sc, this is used to apply the appropriate logic between v and sc. For example, if type=or, then VIPER will see if a test value is less than or greater than V, OR, standarzided value is less than or greater than sc.

  sc: is a standarized variavice between the peer group value and test value.

  v1: is a user chosen value which can be used to test for a particular value. For example, if you want to flag values less then 0, then choose v<0 and VIPER will flag them as anomolous.

  v2: if analysing string streams, v2 can be strings you want to check for. For example, if I want to check for two strings: Failed and Attempt Failed, then set v2=Failed^Attempt Failed, where ^ tells VIPER to perform an AND operation. If I want either to exist, 2=Failed|Attempt Failed, where | tells VIPER to perform an OR operation.

  stringcontains : if using string streams, and you want to see if a particular text value exists and flag it - then if stringcontains=1, VIPER will test for substrings, otherwise it will equate the strings.

  Flags must be provided for each topic. Separate multiple flags by ~

consumeridinputstream : string, required

- Consumer id of the input stream topic: consumeinputstream

companyname : string, required

- Your company name

consumerid: string, required

- Consumerid associated with the topic to consume from

producerid: string, required

- Producerid associated with the topic to produce to

hpdehost: string, required

- Address of HPDE

viperconfigfile : string, required

- Full path to VIPER.ENV configuration file on server.

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encrypted traffic, otherwise set to 0 for plaintext.

partition: int, optional

- Partition used by kafka to store data. NOTE: Kafka will dynamically store data in partitions. Unless you know for sure the partition, you should use the default of -1 to let VIPER determine where your data is.

hpdeport: int, required

- Port number HPDE is listening on

topicid : int, optional

- Topicid represents an id for some entity. For example, if you have 1000 IoT devices, you can perform anomaly prediction for each device.

maintopic : string, optional

- This is the maintopic that contains the subtopic streams.

rollbackoffsets: int, optional

- This is the percentage to rollback the streams that you are analysing: streamstoanalyse

fullpathtopeergroupdata: string, optional

- This is the full path to the peer group you found in viperanomalytrain; this will be used for anomaly detection.

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

delay : int, optional

- delay parameter to wait for Kafka to respond - in milliseconds.

timeout : int, optional

- Number of seconds that VIPER waits when trying to make a connection to HPDE.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the peer groups for all the streams.

****25.1 maadstml.viperanomalypredictbatch(vipertoken,host,port,consumefrom,produceto,consumeinputstream,produceinputstreamtest,produceridinputstreamtest,**

streamstoanalyse,consumeridinputstream,companyname,consumerid,producerid,flags,hpdehost,viperconfigfile,

enabletls=1,partition=-1,hpdeport=-999,topicid="-999",maintopic='',rollbackoffsets=0,fullpathtopeergroupdata='',

brokerhost='',brokerport=9092,delay=1000,timeout=120,microserviceid='',timedelay=0,asynctimeout=120)**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

asynctimeout : int, optional

-This is the timeout in seconds for the Python library async function.

timedelay : int, optional

- Timedelay is in SECONDS. Because batch runs continuously in the background, this will cause Viper to pause *timedelay* seconds when reading and writing to Kafka. For example, if the raw data is being generated every 3600 seconds, it may make sense to set timedelay=3600

consumefrom : string, required

- Topic to consume from in the Kafka broker

produceto : string, required

- Topic to produce results of the prediction to

consumeinputstream : string, required

- Topic of the input stream to test for anomalies

produceinputstreamtest : string, required

- Topic to store the input stream data for analysis

produceridinputstreamtest : string, required

- Producer id for the produceinputstreamtest topic

streamstoanalyse : string, required

- Comma separated list of streams to analyse for anomalies

flags : string, required

- These are flags that will be used to select the peer group for each stream. The flags must have the following format: *riskscore=[a number between 0 and 1]~complete=[and, or, pvalue i.e. p50 means streams over 50% that have an anomaly]~type=[and,or this will determine what logic to apply to v and sc],topic=[topic name],topictype=[numeric or string],v=[v>some value, v<some value, or valueany], sc=[sc>some number, sc<some number - this is the score for the anomaly test]

  if using strings, the specify flags: type=[and,or],topic=[topic name],topictype=string,stringcontains=[0 or 1 - 1 will do a substring test, 0 will equate the strings],v2=[any text you want to test - use | for OR or ^ for AND],sc=[score value, sc<some value, sc>some value]

  riskscore: this the riskscore threshold. A decimal number between 0 and 1, use this as a threshold to flag anomalies.

  complete : If using multiple streams, this will test each stream to see if the computed riskscore and perform an AND or OR on each risk value and take an average of the risk scores if using AND. Otherwise if at least one stream exceeds the riskscore it will return.

  type: AND or OR - if using v or sc, this is used to apply the appropriate logic between v and sc. For example, if type=or, then VIPER will see if a test value is less than or greater than V, OR, standarzided value is less than or greater than sc.

  sc: is a standarized variavice between the peer group value and test value.

  v1: is a user chosen value which can be used to test for a particular value. For example, if you want to flag values less then 0, then choose v<0 and VIPER will flag them as anomolous.

  v2: if analysing string streams, v2 can be strings you want to check for. For example, if I want to check for two strings: Failed and Attempt Failed, then set v2=Failed^Attempt Failed, where ^ tells VIPER to perform an AND operation. If I want either to exist, 2=Failed|Attempt Failed, where | tells VIPER to perform an OR operation.

  stringcontains : if using string streams, and you want to see if a particular text value exists and flag it - then if stringcontains=1, VIPER will test for substrings, otherwise it will equate the strings.

  Flags must be provided for each topic. Separate multiple flags by ~

consumeridinputstream : string, required

- Consumer id of the input stream topic: consumeinputstream

companyname : string, required

- Your company name

consumerid: string, required

- Consumerid associated with the topic to consume from

producerid: string, required

- Producerid associated with the topic to produce to

hpdehost: string, required

- Address of HPDE

viperconfigfile : string, required

- Full path to VIPER.ENV configuration file on server.

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encrypted traffic, otherwise set to 0 for plaintext.

partition: int, optional

- Partition used by kafka to store data. NOTE: Kafka will dynamically store data in partitions. Unless you know for sure the partition, you should use the default of -1 to let VIPER determine where your data is.

hpdeport: int, required

- Port number HPDE is listening on

topicid : string, required

- Topicid represents an id for some entity. For example, if you have 1000 IoT devices, you can perform anomaly prediction for each device. Separate multiple topic ids by a comma.

maintopic : string, optional

- This is the maintopic that contains the subtopic streams.

rollbackoffsets: int, optional

- This is the percentage to rollback the streams that you are analysing: streamstoanalyse

fullpathtopeergroupdata: string, optional

- This is the full path to the peer group you found in viperanomalytrain; this will be used for anomaly detection.

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

delay : int, optional

- delay parameter to wait for Kafka to respond - in milliseconds.

timeout : int, optional

- Number of seconds that VIPER waits when trying to make a connection to HPDE.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: Returns a JSON object of the peer groups for all the streams.

****26. maadstml.viperpreprocessproducetotopicstream(VIPERTOKEN,host,port,topic,producerid,offset,maxrows=0,enabletls=0,delay=100,**

brokerhost='',brokerport=-999,microserviceid='',topicid=-999,streamstojoin='',preprocesslogic='',
preprocessconditions='',identifier='',preprocesstopic='',array=0,saveasarray=0,rawdataoutput=0)**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topic : string, required

- **Topics to produce to in the Kafka broker - this is a topic that contains multiple topics, VIPER will consume from each**

  topic and write the aggregated results back to this stream.

array : int, optional

- Set array=1 if you produced data (from viperproducetotopic) as an array.

rawdataoutput : int, optional

- Set rawdataoutput=1 and the raw data used for preprocessing will be added to the output json.

preprocessconditions : string, optional

- You can set conditions to aggregate functions: MIN, MAX, AVG, COUNT, COUNTSTR, DIFF, DIFFMARGIN, SUM, MEDIAN, VARIANCE, OUTLIERS, OUTLIERSX-Y, VARIED, ANOMPROB,ANOMPROBX-Y, CONSISTENCY, ENTROPY, AUTOCORR, TREND, IQR (InterQuartileRange), Midhinge, GM (Geometric mean), HM (Harmonic mean), Trimean, CV (coefficient of Variation), Mad (Mean absolute deviation),Skewness, Kurtosis, Spikedetect, Unique, Uniquestr, Timediff: time should be in this layout:2006-01-02T15:04:05, Timediff returns the difference in seconds between the first date/time and last datetime. Avgtimediff returns the average time in seconds between consecutive dates. Spikedetect uses a Zscore method to detect spikes in the data using lag of 5, StD of 3.5 from mean and influence of 0.5. Geodiff (returns distance in Kilometers between two lat/long points) Unique Checks numeric data for duplication. Returns 1 if no data duplication (unique), 0 otherwise.

  Dataage_[UTC offset]_[timetype], dataage can be used to check the last update time of the data in the data stream from current local time. You can specify the UTC offset to adjust the current time to match the timezone of the data stream. You can specify timetype as millisecond, second, minute, hour, day. For example, if Dataage_1_minute, then this processtype will compare the last timestamp in the data stream, to the local UTC time offset +1 and compute the time difference between the data stream timestamp and current local time and return the difference in minutes. This is a very powerful processtype for data quality and data assurance programs for any number of data streams.

  Uniquestr Checks string data for duplication. Returns 1 if no data duplication (unique), 0 otherwise.

  Uniquecount Checks numeric data for duplication. Returns count of unique numbers.

  Uniquestrcount Checks string data for duplication. Returns count of unique strings.

  CONSISTENCY checks if the data all have consistent data types. Returns 1 for consistent data types, 0 otherwise.

  Meanci95 or Meanci99 - returns a 95% or 99% confidence interval: mean, low, high

  RAW for no processing.

  ANOMPROB=Anomaly Probability, it will run several algorithms on the data stream window to determine a probaility of anomalous behaviour. This can be cross-refenced with OUTLIERS. It can be very powerful way to detection issues with devices. VARIED will determine if the values in the window

are all the same, or varied: it will return 1 for varied, 0 if values are all the same. This is useful if you want to know if something changed in the stream.

ANOMPROBX-Y (similar to OUTLIERSX-Y), where X and Y are numbers or "n". If "n" means examine all anomalies for patterns. They allow you to check if the anomalies in the streams are truly anomalies and not some pattern. For example, if a IoT device shuts off and turns on again routinely, this may be picked up as an anomaly when in fact it is normal behaviour. So, to ignore these cases, if ANOMPROB2-5, this tells Viper, check anomalies with patterns of 2-5 peaks. If the stream has two classes and these two classes are like 0 and 1000, and show a pattern, then they should not be considered an anomaly. Meaning, class=0, is the device shutting down, class=1000 is the device turning back on. If ANOMPROB3-10, Viper will check for patterns of classes 3 to 10 to see if they recur routinely. This is very helpful to reduce false positives and false negatives.

For example, preprocessconditions='humidity=55,60:temperature=34,n', and preprocesslogic='max,count', means Get the MAX value of values in humidity if humidity is between [55,60], and Count values in temperature if temperature >=34.

preprocesstopic : string, optional

- You can specify a topic for the preprocessed message. VIPER will automatically dump the preprocessed results to this topic.

identifier : string, optional

- Add any identifier like DSN ID.

producerid : string, required

- Producerid of the topic producing to

offset : int, optional

- If 0 will use the stream data from the beginning of the topics, -1 will automatically go to last offset

saveasarray : int, optional

- Set to 1, to save the preprocessed jsons as a json array. This is very helpful if you want to do machine learning or further query the preprocessed json because each processed json are time synchronized. For example, if you want to compare different preprocessed streams the date/time of the data is synchronized to give you impacts of one stream on another.

maxrows : int, optional

- If offset=-1, this number will rollback the streams by maxrows amount i.e. rollback=lastoffset-maxrows

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encrypted traffic, otherwise 0 for plaintext

delay: int, optional

- Time in milliseconds before VIPER backsout from reading messages

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

topicid : int, optional

- This represents the IoT device number or any entity

streamstojoin : string, optional

- If you entered topicid, you need to enter the streams you want to pre-process

preprocesslogic : string, optional

- Here you need to specify how you want to pre-process the streams. You can perform the following operations: MAX, MIN, AVG, COUNT, COUNTSTR, SUM, DIFF, DIFFMARGIN, VARIANCE, MEDIAN, OUTLIERS, OUTLIERSX-Y, VARIED, ANOMPROB, ANOMPROBX-Y, ENTROPY, AUTOCORR, TREND, CONSISTENCY, Unique, Uniquestr, Geodiff (returns distance in Kilometers between two lat/long points), IQR (InterQuartileRange), Midhinge, GM (Geometric mean), HM (Harmonic mean), Trimean, CV (coefficient of Variation), Mad (Mean absolute deviation), Skewness, Kurtosis, Spikedetect, Timediff: time should be in this layout:2006-01-02T15:04:05, Timediff returns the difference in seconds between the first date/time and last datetime. Avgtimediff returns the average time in seconds between consecutive dates. Uniquecount Checks numeric data for duplication. Returns count of unique numbers.

  Dataage_[UTC offset]_[timetype], dataage can be used to check the last update time of the data in the data stream from current local time. You can specify the UTC offset to adjust the current time to match the timezone of the data stream. You can specify timetype as millisecond, second, minute, hour, day. For example, if Dataage_1_minute, then this processtype will compare the last timestamp in the data stream, to the local UTC time offset +1 and compute the time difference between the data stream timestamp and current local time and return the difference in minutes. This is a very powerful processtype for data quality and data assurance programs for any number of data streams.

  Uniquestrcount Checks string data for duplication. Returns count of unique strings.

  Meanci95 or Meanci99 - returns a 95% or 99% confidence interval: mean, low, high

  RAW for no processing.

  Spikedetect uses a Zscore method to detect spikes in the data using lag of 5, StD of 3.5 from mean and influence of 0.5.

  The order of the operation must match the order of the stream. If you specified topicid, you can perform TML on the new preprocessed stream append appending: _preprocessed_processlogic For example, if streamstojoin="stream1,stream2,streams3", and preprocesslogic="min,max,diff", the new streams will be: stream1_preprocessed_Min, stream2_preprocessed_Max, stream3_preprocessed_Diff.

RETURNS: Returns preprocessed JSON.

27. maadstml.areyoubusy(host,port)

Parameters:

host : string, required

- You can get the host by determining all the hosts that are listening in your machine. You use this code: https://github.com/smaurice101/transactionalmachinelearning/blob/main/checkopenports

port : int, required

- You can get the port by determining all the ports that are listening in your machine. You use this code: https://github.com/smaurice101/transactionalmachinelearning/blob/main/checkopenports

RETURNS: Returns a list of available VIPER and HPDE with their HOST and PORT.

****28. maadstml.viperstreamquery(VIPERTOKEN,host,port,topic,producerid,offset=-1,maxrows=0,enabletls=1,delay=100,brokerhost='',**

brokerport=-999,microserviceid='',topicid=-999,streamstojoin='',preprocessconditions='', identifier='',preprocesstopic='',description='',array=0)**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topic : string, required

- **Topics to produce to in the Kafka broker - this is a topic that contains multiple topics, VIPER will consume from each**

  topic and write the aggregated results back to this stream.

producerid : string, required

- Producer id of topic

offset : int, optional

- If 0 will use the stream data from the beginning of the topics, -1 will automatically go to last offset

maxrows : int, optional

- If offset=-1, this number will rollback the streams by maxrows amount i.e. rollback=lastoffset-maxrows

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encrypted traffic, otherwise 0 for plaintext

delay: int, optional

- Time in milliseconds before VIPER backsout from reading messages

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

topicid : int, optional

- This represents the IoT device number or any entity

streamstojoin : string, required

- Identify multiple streams to join, separate by comma. For example, if you preprocessed Power, Current, Voltage:

  streamstojoin="Power_preprocessed_Avg,Current_preprocessed_Min,Voltage_preprocessed_Avg,Current_preprocessed_Trend"

preprocessconditions : string, required

- You apply strict conditions to a MAX of 3 streams. You can use >, <, =, AND, OR. You can add as many conditions as you like. Separate multiple conditions by semi-colon. You **cannot mix** AND and OR. For example,

  preprocessconditions='Power_preprocessed_Avg > 139000:Power_preprocessed_Avg < 1000 or Voltage_preprocessed_Avg > 120000 or Current_preprocessed_Min=0:Voltage_preprocessed_Avg > 120000 and Current_preprocessed_Trend>0'

identifier: string, optional

- Add an identifier text to the result. This is a label, and useful if you want to identify the result for some IOT device.

preprocesstopic : string, optional

- The topic to produce the query results to.

description : string, optional

- You can give each query condition a description. Separate multiple desction by semi-colon.

array : int, optional

- Set to 1 if you are reading a JSON ARRAY, otherwise 0.

RETURNS: 1 if the condition is TRUE (condition met), 0 if false (condition not met)

****28.1 maadstml.viperstreamquerybatch(VIPERTOKEN,host,port,topic,producerid,offset=-1,maxrows=0,enabletls=1,delay=100,brokerhost='',**

brokerport=-999,microserviceid='',topicid="-999",streamstojoin='',preprocessconditions='', identifier='',preprocesstopic='',description='',array=0,timedelay=0,asynctimeout=120)**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

asynctimeout : int, optional

-This is the timeout in seconds for the Python library async function.

timedelay : int, optional

- Timedelay is in SECONDS. Because batch runs continuously in the background, this will cause Viper to pause *timedelay* seconds when reading and writing to Kafka. For example, if the raw data is being generated every 3600 seconds, it may make sense to set timedelay=3600

topic : string, required

- **Topics to produce to in the Kafka broker - this is a topic that contains multiple topics, VIPER will consume from each**

  topic and write the aggregated results back to this stream.

producerid : string, required

- Producer id of topic

offset : int, optional

- If 0 will use the stream data from the beginning of the topics, -1 will automatically go to last offset

maxrows : int, optional

- If offset=-1, this number will rollback the streams by maxrows amount i.e. rollback=lastoffset-maxrows

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encrypted traffic, otherwise 0 for plaintext

delay: int, optional

- Time in milliseconds before VIPER backsout from reading messages

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

topicid : string, required

- This represents the IoT device number or any entity. Separate multiple topic ids by a comma.

streamstojoin : string, required

- Identify multiple streams to join, separate by comma. For example, if you preprocessed Power, Current, Voltage:

  streamstojoin="Power_preprocessed_Avg,Current_preprocessed_Min,Voltage_preprocessed_Avg,Current_preprocessed_Trend"

preprocessconditions : string, required

- You apply strict conditions to a MAX of 3 streams. You can use >, <, =, AND, OR. You can add as many conditions as you like. Separate multiple conditions by semi-colon. You **cannot mix** AND and OR. For example,

  preprocessconditions='Power_preprocessed_Avg > 139000:Power_preprocessed_Avg < 1000 or Voltage_preprocessed_Avg > 120000 or Current_preprocessed_Min=0:Voltage_preprocessed_Avg > 120000 and Current_preprocessed_Trend>0'

identifier: string, optional

- Add an identifier text to the result. This is a label, and useful if you want to identify the result for some IOT device.

preprocesstopic : string, optional

- The topic to produce the query results to.

description : string, optional

- You can give each query condition a description. Separate multiple desction by semi-colon.

array : int, optional

- Set to 1 if you are reading a JSON ARRAY, otherwise 0.

RETURNS: 1 if the condition is TRUE (condition met), 0 if false (condition not met)

****29. maadstml.viperpreprocessbatch(VIPERTOKEN,host,port,topic,producerid,offset,maxrows=0,enabletls=0,delay=100,**

brokerhost='',brokerport=-999,microserviceid='',topicid="-999",streamstojoin='',preprocesslogic='',

preprocessconditions='',identifier='',preprocesstopic='',array=0,saveasarray=0,timedelay=0,asynctimeout=120,rav

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

asynctimeout : int, optional

-This is the timeout in seconds for the Python library async function.

rawdataoutput : int, optional

-Set rawdataoutput=1 to output the raw preprocessing data to the Json.

timedelay : int, optional

- Timedelay is in SECONDS. Because batch runs continuously in the background, this will cause Viper to pause *timedelay* seconds when reading and writing to Kafka. For example, if the raw data is being generated every 3600 seconds, it may make sense to set timedelay=3600

topic : string, required

- **Topics to produce to in the Kafka broker - this is a topic that contains multiple topics, VIPER will consume from each**

  topic and write the aggregated results back to this stream.

array : int, optional

- Set array=1 if you produced data (from viperproducetotopic) as an array.

preprocessconditions : string, optional

- You can set conditions to aggregate functions: MIN, MAX, AVG, COUNT, COUNTSTR, DIFF, SUM, MEDIAN, VARIANCE, OUTLIERS, OUTLIERSX-Y, VARIED, ANOMPROB,ANOMPROBX-Y, ENTROPY, AUTOCORR, TREND, IQR (InterQuartileRange), Midhinge, GM (Geometric mean), HM (Harmonic mean), Trimean, CV (coefficient of Variation), Mad (Mean absolute deviation),Skewness, Kurtosis, Spikedetect, Unique, Uniquestr, Timediff: time should be in this layout:2006-01-02T15:04:05, Timediff returns the difference in seconds between the first date/time and last datetime. Avgtimediff returns the average time in seconds between consecutive dates. Spikedetect uses a Zscore method to detect spikes in the data using lag of 5, StD of 3.5 from mean and influence of 0.5. Geodiff (returns distance in Kilometers between two lat/long points).

  Dataage_[UTC offset]_[timetype], dataage can be used to check the last update time of the data in the data stream from current local time. You can specify the UTC offset to adjust the current time to match the timezone of the data stream. You can specify timetype as millisecond, second, minute, hour, day. For example, if Dataage_1_minute, then this processtype will compare the last timestamp in the data stream, to the local UTC time offset +1 and compute the time difference between the data stream timestamp and current local time and return the difference in minutes. This is a very powerful processtype for data quality and data assurance programs for any number of data streams.

  Unique Checks numeric data for duplication. Returns 1 if no data duplication (unique), 0 otherwise.

  Uniquestr Checks string data for duplication. Returns 1 if no data duplication (unique), 0 otherwise. Uniquecount Checks numeric data for duplication. Returns count of unique numbers. Uniquestrcount Checks string data for duplication. Returns count of unique strings.

  Meanci95 or Meanci99 - returns a 95% or 99% confidence interval: mean, low, high

  ANOMPROB=Anomaly Probability, it will run several algorithms on the data stream window to determine a probaility of anomalous behaviour. This can be cross-refenced with OUTLIERS. It can be very powerful way to detection issues with devices. VARIED will determine if the values in the window are all the same, or varied: it will return 1 for varied, 0 if values are all the same. This is useful if you want to know if something changed in the stream.

  ANOMPROBX-Y (similar to OUTLIERSX-Y), where X and Y are numbers or "n". If "n" means examine all anomalies for patterns. They allow you to check if the anomalies in the streams are truly anomalies and not some pattern. For example, if a IoT device shuts off and turns on again routinely, this may be picked up as an anomaly when in fact it is normal behaviour. So, to ignore these cases, if ANOMPROB2-5, this tells Viper, check anomalies with patterns of 2-5 peaks. If the stream has two

classes and these two classes are like 0 and 1000, and show a pattern, then they should not be considered an anomaly. Meaning, class=0, is the device shutting down, class=1000 is the device turning back on. If ANOMPROB3-10, Viper will check for patterns of classes 3 to 10 to see if they recur routinely. This is very helpful to reduce false positives and false negatives.

For example, preprocessconditions='humidity=55,60:temperature=34,n', and preprocesslogic='max,count', means Get the MAX value of values in humidity if humidity is between [55,60], and Count values in temperature if temperature >=34.

preprocesstopic : string, optional

- You can specify a topic for the preprocessed message. VIPER will automatically dump the preprocessed results to this topic.

identifier : string, optional

- Add any identifier like DSN ID. Note, for multiple identifiers per topicid, you can separate by pipe "|".

producerid : string, required

- Producerid of the topic producing to

offset : int, optional

- If 0 will use the stream data from the beginning of the topics, -1 will automatically go to last offset

saveasarray : int, optional

- Set to 1, to save the preprocessed jsons as a json array. This is very helpful if you want to do machine learning or further query the preprocessed json because each processed json are time synchronized. For example, if you want to compare different preprocessed streams the date/time of the data is synchronized to give you impacts of one stream on another.

maxrows : int, optional

- If offset=-1, this number will rollback the streams by maxrows amount i.e. rollback=lastoffset-maxrows

enabletls: int, optional

- Set to 1 if Kafka broker is SSL/TLS enabled for encrypted traffic, otherwise 0 for plaintext

delay: int, optional

- Time in milliseconds before VIPER backsout from reading messages

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

topicid : string, required

- This represents the IoT device number or any entity. You can specify multiple ids separated by a comma: topicid="1,2,4,5".

streamstojoin : string, optional

- If you entered topicid, you need to enter the streams you want to pre-process

preprocesslogic : string, optional

- Here you need to specify how you want to pre-process the streams. You can perform the following operations: MAX, MIN, AVG, COUNT, COUNTSTR, SUM, DIFF, VARIANCE, MEDIAN, OUTLIERS,

OUTLIERSX-Y, VARIED, ANOMPROB, ANOMPROBX-Y, ENTROPY, AUTOCORR, TREND, IQR (InterQuartileRange), Midhinge, CONSISTENCY, GM (Geometric mean), HM (Harmonic mean), Trimean, CV (coefficient of Variation), Mad (Mean absolute deviation), Skewness, Kurtosis, Spikedetect, Unique, Uniquestr, Timediff: time should be in this layout:2006-01-02T15:04:05, Timediff returns the difference in seconds between the first date/time and last datetime. Avgtimediff returns the average time in seconds between consecutive dates. Geodiff (returns distance in Kilometers between two lat/long points). Spikedetect uses a Zscore method to detect spikes in the data using lag of 5, StD of 3.5 from mean and influence of 0.5. Uniquecount Checks numeric data for duplication. Returns count of unique numbers. Uniquestrcount Checks string data for duplication. Returns count of unique strings.

Dataage_[UTC offset]_[timetype], dataage can be used to check the last update time of the data in the data stream from current local time. You can specify the UTC offset to adjust the current time to match the timezone of the data stream. You can specify timetype as millisecond, second, minute, hour, day. For example, if Dataage_1_minute, then this processtype will compare the last timestamp in the data stream, to the local UTC time offset +1 and compute the time difference between the data stream timestamp and current local time and return the difference in minutes. This is a very powerful processtype for data quality and data assurance programs for any number of data streams.

Meanci95 or Meanci99 - returns a 95% or 99% confidence interval: mean, low, high

The order of the operation must match the order of the stream. If you specified topicid, you can perform TML on the new preprocessed stream append appending: _preprocessed_processlogic For example, if streamstojoin="stream1,stream2,streams3", and preprocesslogic="min,max,diff", the new streams will be: stream1_preprocessed_Min, stream2_preprocessed_Max, stream3_preprocessed_Diff.

RETURNS: None.

30. maadstml.viperlisttopics(vipertoken,host,port=-999,brokerhost='', brokerport=-999,microserviceid='')

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

brokerhost : string, optional

- Address where Kafka broker is running - if none is specified, the Kafka broker address in the VIPER.ENV file will be used.

brokerport : int, optional

- Port on which Kafka is listenting.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: A JSON formatted object of all the topics in the Kafka broker.

****31. maadstml.viperpreprocesscustomjson(VIPERTOKEN,host,port,topic,producerid,offset,jsoncriteria='',rawdataoutput=0,maxrows=0,**

enabletls=0,delay=100,brokerhost='',brokerport=-999,microserviceid='',topicid=-999,streamstojoin='',preprocesslogic=

preprocessconditions='',identifier='',preprocesstopic='',array=0,saveasarray=0,timedelay=0,asynctimeout=120, usemysql=0,tmlfilepath='',pathtotmlattrs='')**

Parameters:

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

topic : string, required

- Topic containing the raw data to consume.

producerid : string, required

- Id of the Topic.

offset : int, required

- Offset to consume from. Set to -1 if consuming the last offset of topic.

jsoncriteria : string, required

- This is the JSON path to the data you want to consume . It must be the following format:

    UID is path to the main id. For example, Patient ID

    filter is the path to something that filter the jsons

    subtopic is the path to the subtopics in the json (several paths can be specified)

    values is the path to the Values of the subtopics - Subtopic and Value must have 1-1 match

    identifiers is the path to any special identifiers for the subtopics

    datetime is the path to the datetime of the message

    msgid is the path to any msg id

For example:

jsoncriteria='uid=subject.reference,filter:resourceType=Observation~
subtopics=code.coding.0.code,component.0.code.coding.0.code,component.1.code.coding.0.code~values=valueQuantity.value,component.0.valueQuantity.value,component.1.valueQuantity.value~identifiers=code.coding.0.display,component.0.code.coding.0.display,component.1.code.coding.0.display~datetime=effectiveDateTime~msgid=id'

rawdataoutput : int, optional

- set to 1 if you want to output the raw data. Note: This could involve a lot of data and Kafka may refuse to write to the topic.

maxrows : int, optional

- Number of offsets or percentage to roll back the data stream

enabletls : int, optional

- Set to 1 for TLS encrpyted traffic

delay : int, optional

- Delay to wait for Kafka to finish writing to topic

topicid : int, optional

- Since you are consuming raw data, this is not needed. Topicid will be set for you.

streamstojoin : string, optional

- This is ignored for raw data.

preprocesslogic : string, optional

- Specify your preprocess algorithms. For example, You can set conditions to aggregate functions: MIN, MAX, AVG, COUNT, COUNTSTR, DIFF, DIFFMARGIN, SUM, MEDIAN, VARIANCE, OUTLIERS, OUTLIERSX-Y, VARIED, ANOMPROB,ANOMPROBX-Y, CONSISTENCY, ENTROPY, AUTOCORR,

TREND, IQR (InterQuartileRange), Midhinge, GM (Geometric mean), HM (Harmonic mean), Trimean, CV (coefficient of Variation), Mad (Mean absolute deviation),Skewness, Kurtosis, Spikedetect, Unique, Uniquestr, Timediff: time should be in this layout:2006-01-02T15:04:05, Timediff returns the difference in seconds between the first date/time and last datetime. Avgtimediff returns the average time in seconds between consecutive dates. Spikedetect uses a Zscore method to detect spikes in the data using lag of 5, StD of 3.5 from mean and influence of 0.5. Geodiff (returns distance in Kilometers between two lat/long points) Unique Checks numeric data for duplication. Returns 1 if no data duplication (unique), 0 otherwise.

Dataage_[UTC offset]_[timetype], dataage can be used to check the last update time of the data in the data stream from current local time. You can specify the UTC offset to adjust the current time to match the timezone of the data stream. You can specify timetype as millisecond, second, minute, hour, day. For example, if Dataage_1_minute, then this processtype will compare the last timestamp in the data stream, to the local UTC time offset +1 and compute the time difference between the data stream timestamp and current local time and return the difference in minutes. This is a very powerful processtype for data quality and data assurance programs for any number of data streams.

Uniquestr Checks string data for duplication. Returns 1 if no data duplication (unique), 0 otherwise.

Uniquecount Checks numeric data for duplication. Returns count of unique numbers.

Uniquestrcount Checks string data for duplication. Returns count of unique strings.

CONSISTENCY checks if the data all have consistent data types. Returns 1 for consistent data types, 0 otherwise.

Meanci95 or Meanci99 - returns a 95% or 99% confidence interval: mean, low, high

RAW for no processing.

preprocessconditions : string, optional

- Specify any preprocess conditions

identifier : string, optional

- Specify any text identifier

preprocesstopic : string, optional

- Specify the name of the topic to write preprocessed results.

array : int, optional

- Ignored for raw data - as jsoncriteria specifies json path

saveasarray : int, optional

- Set to 1 to save as json array

timedelay : int, optional

- Delay to wait for response from Kafka.

asynctimeout : int, optional

- Maximum delay for asyncio in Python library

usemysql : int, optional

- Set to 1 to specify whether MySQL is used to store TMLIDs. This will be needed to track individual objects.

tmlfilepath : string, optional

- Ignored.

pathtotmlattrs : string, optional

- Specifiy any attributes for the TMLID. Here you can specify OEM, Latitude, Longitude, and Location JSON paths:

pathtotmlattrs='oem=id,lat=subject.reference,long=component.0.code.coding.0.display,location=component.1.valueQuantity.value'

port : int, required

- Port on which VIPER is listenting.

brokerhost : string, optional

- Address where Kafka broker is running - if none is specified, the Kafka broker address in the VIPER.ENV file will be used.

brokerport : int, optional

- Port on which Kafka is listenting.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: null

****32. maadstml.viperstreamcorr(vipertoken,host,port,topic,producerid,offset=-1,maxrows=0,enabletls=1,delay=100,brokerhost='',**

brokerport=-999,microserviceid='',topicid=-999,streamstojoin='',
identifier='',preprocesstopic='',description='',array=0, wherecondition='',
wheresearchkey='PreprocessIdentifier',rawdataoutput=1,threshhold=0,pvalue=0,
identifierextractpos="",topcorrnum=5,jsoncriteria='',tmlfilepath='',usemysql=0,
pathtotmlattrs='',mincorrvectorlen=5,writecorrstotopic='',outputtopicnames=0,nlp=0,
correlationtype='',docrosscorr=0)**

Parameters: Perform Stream correlations

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

topic : string, required

- Topic containing the raw data to consume.

producerid : string, required

- Id of the Topic.

wherecondition : string, optional

- Specify the where condition. For example, if you want to filter the data on "males", enter males. You can specify exact match by using [males], or substring by using (males), or "not" includes by using {males}

correlationtype : string, optional

- Specify type of correlation you want to do. Valid values are: kendall,spearman,pearson,ks You can specify some, or all (leave blank and ALL will be done), separated by comma. ks=kolmogorov-Smirnov test.

docrosscorr : int, optional

- Set to 1 if you want to do cross-correlations with 4 variables, not the normal 2-variable.

wheresearchkey : string, optional

- Specify the where search key. This key will be searched for "males".

description : string, optional

- Specify a text description for this correlation.

identifierextractpos : string, optional

- If doing correlation on data you have already preprocessed, you can extract the identifier from the identifier field in the preprocessed json.

offset : int, required

- Offset to consume from. Set to -1 if consuming the last offset of topic.

mincorrvectorlen : int, optional

- Minimum length of the data variables you are correlating.

topcorrnum : int, optional

- Top number of sorted correlations to output

threshhold : int, optional

- Threshold for the correlation coefficient. Must range from 0-100. All correlations will be greater than this number.

pvalue : int, optional

- Pvalue threshold for the p-values. Must range from 0-100. All p-values will be below this number.

writecorrstotopic : string, optional

- This is the name of the topic that Viper will write "individual" correlation results to.

outputtopicnames : int, optional

- Set to 1 if you want to write out topic names.

nlp : int, optional

- Set to 1 if you want to correlate TEXT data by using natural language processing (NLP).

jsoncriteria : string, required

- This is the JSON path to the data you want to consume . It must be the following format:

UID is path to the main id. For example, Patient ID

filter is the path to something that filter the jsons

subtopic is the path to the subtopics in the json (several paths can be specified)

values is the path to the Values of the subtopics - Subtopic and Value must have 1-1 match

identifiers is the path to any special identifiers for the subtopics

datetime is the path to the datetime of the message

msgid is the path to any msg id

For example:

jsoncriteria='uid=subject.reference,filter:resourceType=Observation~

subtopics=code.coding.0.code,component.0.code.coding.0.code,component.1.code.coding.0.code~values=valueQuantity.value,component.0.valueQuantity.value,component.1.valueQuantity.value~identifiers=code.coding.0.display,component.0.code.coding.0.display,component.1.code.coding.0.display~datetime=effectiveDateTime~msgid=id'

rawdataoutput : int, optional

- set to 1 if you want to output the raw data. Note: This could involve a lot of data and Kafka may refuse to write to the topic.

maxrows : int, optional

- Number of offsets or percentage to roll back the data stream

enabletls : int, optional

- Set to 1 for TLS encrpyted traffic

delay : int, optional

- Delay to wait for Kafka to finish writing to topic

topicid : int, optional

- Since you are consuming raw data, this is not needed. Topicid will be set for you.

streamstojoin : string, optional

- This is ignored for raw data.

preprocesslogic : string, optional

- Specify your preprocess algorithms. For example, min, max, variance, trend, anomprob, outliers, etc..

preprocessconditions : string, optional

- Specify any preprocess conditions

identifier : string, optional

- Specify any text identifier

preprocesstopic : string, optional

- Specify the name of the topic to write preprocessed results.

array : int, optional

- Ignored for raw data - as jsoncriteria specifies json path

saveasarray : int, optional

- Set to 1 to save as json array

timedelay : int, optional

- Delay to wait for response from Kafka.

asynctimeout : int, optional

- Maximum delay for asyncio in Python library

usemysql : int, optional

- Set to 1 to specify whether MySQL is used to store TMLIDs. This will be needed to track individual objects.

tmlfilepath : string, optional

- Ignored.

pathtotmlattrs : string, optional

- Specifiy any attributes for the TMLID. Here you can specify OEM, Latitude, Longitude, and Location JSON paths:

  pathtotmlattrs='oem=id,lat=subject.reference,long=component.0.code.coding.0.display,location=component.1.valueQuantity.value'

port : int, required

- Port on which VIPER is listenting.

brokerhost : string, optional

- Address where Kafka broker is running - if none is specified, the Kafka broker address in the VIPER.ENV file will be used.

brokerport : int, optional

- Port on which Kafka is listenting.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

RETURNS: null

****33. maadstml.viperstreamcluster(vipertoken,host,port,topic,producerid,offset=-1,maxrows=0,enabletls=1,delay=100,brokerhost='',**

brokerport=-999,microserviceid='',topicid=-999,iterations=1000, numclusters=8, distancealgo=1,description='',rawdataoutput=0,valuekey='',filterkey='',groupkey='', identifier='',datetimekey='',valueidentifier='',msgid='',valuecondition='', identifierextractpos='',preprocesstopic='', alertonclustersize=0,alertonsubjectpercentage=50,sendalertemailsto='',emailfrequencyinseconds=0, companyname='',analysisdescription='',identifierextractposlatitude=-1, identifierextractposlongitude=-1,identifierextractposlocation=-1, identifierextractjoinedidentifiers=-1,pdfformat='',minimumsubjects=2)**

Parameters: Perform Stream correlations

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

topic : string, required

- Topic containing the raw data to consume.

port : int, required

- Port on which VIPER is listenting.

brokerhost : string, optional

- Address where Kafka broker is running - if none is specified, the Kafka broker address in the VIPER.ENV file will be used.

brokerport : int, optional

- Port on which Kafka is listenting.

alertonsubjectpercentage : int, optional

- Set a value between 0-100 that specifies the percentage of subjects that exceed a threshold.

identifierextractjoinedidentifiers : int, optional

- Position of additional text in identfier field.

pdfformat : string, optional

- Speficy format text of the PDF to generate and emailed to users. You can set title, signature, showpdfemaillist, and charttitle.

  pdfformat="title=This is a Transactional Machine Learning Auto-Generated PDF for Cluster Analysis For OTICS|signature=Created by: OTICS, Toronto|showpdfemaillist=1|charttitle=Chart Shows Clusters of Patients with Similar Symptoms"

minimumsubjects : int, optional

- Sepecify minimum subjects in the cluster analysis.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

maxrows : int, optional

- Number of offsets or percentage to roll back the data stream

enabletls : int, optional

- Set to 1 for TLS encrpyted traffic

delay : int, optional

- Delay to wait for Kafka to finish writing to topic

producerid : string, required

- Id of the Topic.

topicid : int, optional

- Ignored

iterations : int, optional

- Number of iterations to compute clusters

numclusters : int, optional

- Number of clusters you want. Maximum is 20.

distancealgo : int, optional

- Set to 1 for Euclidean, or 2 for EuclideanSquared.

valuekey : string, required

- JSON path to the value to cluster on

filterkey : string, optional

- JSON path to filter on. Ex. Preprocesstype=Pearson, gets value from Key=Preprocesstype, and checks for value=Pearson

groupkey : string, optional

- JSON path to group on a key. Ex. Topicid, to group on TMLIDs

valueidentifier : string, optional

- JSON path to text value IDs you correlated.

msgid : string, optional

- JSON path for a unique message id

valuecondition : string, optional

- A condition to filter numeric values on. Ex. valuecondition="> .5", if valuekey is correlations, then all correlation > 0.5 are taken.

identifierextractpos : string, optional

- The location of data to extract from the Identifier field. Ex. identifierextractpos="1,2", will extract data from position 1 and 2.

preprocesstopic : string, required

- Topic to produce results to

alertonclustersize : int, optional

- Size of the cluster to alert on. Ex. if this is 100, then when any cluster has more than 100 elements an email is sent.

sendalertemailsto: string, optional

- List of email addresses to send alert to

emailfrequencyinseconds : int, optional

- Seconds between emails. Ex. set to 3600, so emails will be sent every 1 hour if alert condition met.

companyname : string, optional

- Your company name

analysisdescription : string, optional

- A detailed description of the analysis. This will be added to the PDF.

identifierextractposlatitude : int, optional

- Position for latitude in the Identifier field

identifierextractposlongitude : int, optional

- Position for longitude in the Identifier field

identifierextractposlocation : int, optional

- Position for location in the Identifier field

RETURNS: null

****34. maadstml.vipersearchanomaly(vipertoken,host,port,topic,producerid,offset,jsoncriteria='',rawdataoutput=0,maxrows=0,enabletls=0,delay=100,**

brokerhost='',brokerport=-999,microserviceid='',topicid=-999,identifier='',preprocesstopic='', timedelay=0,asynctimeout=120,searchterms='',entitysearch='',tagsearch='',checkanomaly=1,testtopic='', includeexclude=1,anomalythreshold=0,sendanomalyalertemail='',emailfrequency=3600)**

Parameters: Perform Stream correlations

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

topic : string, required

- Topic containing the raw data to consume.

port : int, required

- Port on which VIPER is listenting.

brokerhost : string, optional

- Address where Kafka broker is running - if none is specified, the Kafka broker address in the VIPER.ENV file will be used.

brokerport : int, optional

- Port on which Kafka is listenting.

jsoncriteria : string, optional

- Enter the JSON path to the search fields

anomalythreshold : int, optional

- Threshold to meet to determine if search differs from the peer group. This is a number between 0-100. The lower the number the "more" this search differs from the peer group and likely anomalous.

includeexclude : int, optional

- Set to 1 if you want the search terms included in the user searches, 0 otherwise.

sendanomalyalertemail : string, optional

- List of email addresses to send alerts to: separate list by comma.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

maxrows : int, optional

- Number of offsets or percentage to roll back the data stream

enabletls : int, optional

- Set to 1 for TLS encrpyted traffic

delay : int, optional

- Delay to wait for Kafka to finish writing to topic

producerid : string, required

- Id of the Topic.

emailfrequency : int, optional

- Frequency in seconds, between alert emails.

testtopic : string, optional

- ignored

preprocesstopic : string, required

- Topic to produce results to

sendalertemailsto: string, optional

- List of email addresses to send alert to

tagsearch : string, optional

- Search for tags in the search. You can enter: 'superlative,noun,interjection,verb,pronoun'

entitysearch : string, optional

- Search for entities in the search. You can enter: 'person,gpe', where gpe=Geo-political entity

searchterms : string, optional

- You can specify your own search terms. Separate list by comma.

emailfrequencyinseconds : int, optional

- Seconds between emails. Ex. set to 3600, so emails will be sent every 1 hour if alert condition met.

companyname : string, optional

- Your company name

topicid : int, optional

- ignored

identifier : string, optional

- identifier text

checkanomaly : int, optional

- Set to 1 to check for search anomaly.

rawdataoutput : int, optional

- ignored

RETURNS: null

****35. maadstml.vipermirrorbrokers(VIPERTOKEN,host,port,brokercloudusernamepassfrom,brokercloudusernamepassto,**

enabletlsfrom,enabletlsto, replicationfactorfrom,replicationfactorto,compressionfrom,compressionto, saslfrom,saslto,partitions,brokerlistfrom,brokerlistto, topiclist,asynctimeout=300,microserviceid="",servicenamefrom="broker",

servicenameto="broker",partitionchangeperc=0,replicationchange=0,filter="",rollbackoffset=0)**

Parameters: Perform Data Stream migration across brokers - fast and simple.

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

brokercloudusernamepassfrom : string, required

- This is a comma separated list of source broker username:password. For multiple brokers separate with comma, for example for 3 brokers: username:password,username:password,username:password

brokercloudusernamepassto : string, required

- This is a comma separated list of destination broker username:password. For multiple brokers separate with comma, for example for 3 brokers: username:password,username:password,username:password. The number of source and destination brokers must match.

enabletlsfrom : string, required

- This is a colon separated list of whether source brokers require TLS: 1=TLS, 0=NoTLS. For multiple brokers separate with colon, for example for 3 brokers: 1:0:1. Some brokers may be On-Prem and do not need TLS.

enabletlsto : string, required

- This is a colon separated list of whether destination brokers require TLS: 1=TLS, 0=NoTLS. For multiple brokers separate with colon, for example for 3 brokers: 1:0:1. Some brokers may be On-Prem and do not need TLS.

replicationfactorfrom : string, optional

- This is a colon separated list of the replication factor of source brokers. For multiple brokers separate with colon, for example for 3 brokers: 3:4:3, or leave blank to let VIPER decide.

replicationfactorto : string, optional

- This is a colon separated list of the replication factor of destination brokers. For multiple brokers separate with colon, for example for 3 brokers: 3:4:3, or leave blank to let VIPER decide.

compressionfrom : string, required

- This is a colon separated list of the compression type of source brokers: snappy, gzip, lz4. For multiple brokers separate with colon, for example for 3 brokers: snappy:snappy:gzip.

compressionto : string, required

- This is a colon separated list of the compression type of destination brokers: snappy, gzip, lz4. For multiple brokers separate with colon, for example for 3 brokers: snappy:snappy:gzip.

saslfrom : string, required

- This is a colon separated list of the SASL type: None, Plain, SCRAM256, SCRAM512 of source brokers. For multiple brokers separate with colon, for example for 3 brokers: PLAIN:SCRAM256:SCRAM512.

saslto : string, required

- This is a colon separated list of the SASL type: None, Plain, SCRAM256, SCRAM512 of destination brokers. For multiple brokers separate with colon, for example for 3 brokers: PLAIN:SCRAM256:SCRAM512.

partitions : string, optional

- If you are manually migrating topics you will need to specify the partitions of the topics in *topiclist*. Otherwise, VIPER will automatically find topics and their partitions on the broker for you - this is recommended.

brokerlistfrom : string, required

- This is a list of source brokers: host:port. For multiple brokers separate with comma, for example for 3 brokers: host:port,host:port,host:port.

brokerlistto : string, required

- This is a list of destination brokers: host:port. For multiple brokers separate with comma, for example for 3 brokers: host:port,host:port,host:port.

topiclist : string, optional

- You can manually specify topics to migrate, separate multiple topics with a comma. Otherwise, Viper will automatically find topics on the broker for you - this is recommended.

partitionchangeperc : number, optional

- You can increase or decrease partitions on destination broker by specifying a percentage between 0-100, or -100-0. Minimum partition will always be 1.

replicationchange : ignored for now

- You can increase or decrease replication factor on destination broker by specifying a positive or negative number. Minimum partition will always be 2.

filter : string, optional

- You can specify a filter to choose only those topics that satisfy the filter. Filters must have the following format: "searchstring1,searchstring2,searchstring3,..:Logic=0 or 1:search position: 0,1,2". For example, Logic 0=AND, 1=OR, search position: 0=BeginsWith, 1=Any, 2=EndsWith

asynctimeout : number, optional

- This specifies the timeout in seconds for the python connection.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

servicenamefrom : string, optional

- You can specify the name of the source brokers.

servicenameto : string, optional

- You can specify the name of the destination brokers.

rollbackoffset: ignored

36. maadstml.vipernlp(filename,maxsummarywords,maxkeywords)

Parameters: Perform NLP summarization of PDFs

filename : string, required

- Filename of PDF to summarize.

maxsummarywords : int, required

- Maximum amount of words in the summary.

maxkeywords : int, required

- Maximum amount of keywords to extract.

RETURNS: JSON string of summary.

37. maadstml.viperchatgpt(openaikey,texttoanalyse,query, temperature,modelname)

Parameters: Start a conversation with ChatGPT

openaikey : string, required

- OpenAI API key

texttoanalyse : string, required

- Text you want ChatGPT to analyse

query : string, required

- Prompts for chatGPT. For example, "What are key points in this text? What are the concerns or issues?"

temperature : float, required

- Temperature for chatgpt, must be between 0-1 i.e. 0.7

modelname : string, required

- ChatGPT model to use. For example, text-davinci-002, text-curie-001, text-babbage-001.

RETURNS: ChatGPT response.

38. maadstml.viperexractpdffields(pdffilename)

Parameters: Extract data from PDF

pdffilename : string, required

- PDF filename

RETURNS: JSON of PDF and writes JSON and XML files of PDF to disk.

39. maadstml.viperexractpdffieldbylabel(pdffilename,labelname,arcotype)

Parameters: Extract data from PDF by PDF labels

pdffilename : string, required

- PDF filename

labelname : string, required

- Label name in the PDF filename to search for.

pdffilename,labelname,arcotype : string, required

- Acrobyte tag in PDF i.e. LTTextLineHorizontal

RETURNS: Value of the labelname - if any.

40. maadstml.pgptingestdocs(docname,doctype, pgptip,pgptport,pgptendpoint)

Parameters:

docname : string, required

- A full-path to a PDF, or text file.

doctype : string, required

- This can be: binary, or text.

pgptip : string, required

- Your container IP - this is usually: http://127.0.0.1

pgptport : string, required

- Your container Port - this is usually: 8001. This will be dependent on the docker run port forwarding command. See: https://github.com/smaurice101/raspberrypi/tree/main/privategpt

pgptendpoint : string, required

- This must be: /v1/ingest

RETURNS: JSON containing Document details, or ERROR.

41. maadstml.pgptgetingestedembeddings(docname,ip,port,endpoint)

Parameters:

docname : string, required

- A full-path to a PDF, or text file.

ip : string, required

- Your container IP - this is usually: http://127.0.0.1

port : string, required

- Your container Port - this is usually: 8001. This will be dependent on the docker run port forwarding command. See: https://github.com/smaurice101/raspberrypi/tree/main/privategpt

endpoint : string, required

- This must be: /v1/ingest/list

RETURNS: Three variables: docids,docstr,docidsstr; these are the embeddings related to docname. Or, ERROR.

42. maadstml.pgptchat(prompt,context,docfilter,port,includesources,ip,endpoint)

Parameters:

prompt : string, required

- A prompt for privateGPT.

context : bool, required

- This can be True or False. If True, privateGPT will use context, if False, it will not.

docfilter : string array, required

- This is docidsstr, and can be retrieved from pgptgetingestedembeddings. If context=True, and dockfilter is empty, then ALL documents are used for context.

port : string, required

- Your container Port - this is usually: 8001. This will be dependent on the docker run port forwarding command. See: https://github.com/smaurice101/raspberrypi/tree/main/privategpt

includesources : bool, required

- This can be True or False. If True, with context, privateGPT will return the sources in the response.

ip : string, required

- Your container IP - this is usually: http://127.0.0.1

endpoint : string, required

- This must be: /v1/completions

RETURNS: The response from privateGPT, or ERROR.

43. maadstml.pgptdeleteembeddings(docids, ip,port,endpoint)

Parameters:

docids : string array, required

- An array of doc ids. This can be retrieved from pgptgetingestedembeddings.

port : string, required

- Your container Port - this is usually: 8001. This will be dependent on the docker run port forwarding command. See: https://github.com/smaurice101/raspberrypi/tree/main/privategpt

ip : string, required

- Your container IP - this is usually: http://127.0.0.1

endpoint : string, required

- This must be: /v1/ingest/

RETURNS: Null if successful, or ERROR.

44. maadstml.pgpthealth(ip,port,endpoint)

Parameters:

port : string, required

- Your container Port - this is usually: 8001. This will be dependent on the docker run port forwarding command. See: https://github.com/smaurice101/raspberrypi/tree/main/privategpt

ip : string, required

- Your container IP - this is usually: http://127.0.0.1

endpoint : string, required

- This must be: /health

RETURNS: This will return a JSON of OK if the privateGPT server is running, or ERROR.

45. maadstml.videochatloadresponse(url,port,filename,prompt,responsefolder='videogpt_response',temperature=0.2,max_output_tokens=512)

Parameters:

url : string, required

- IP video chatgpt is listening on in the container - this is usually: http://127.0.0.1

port : string, required

- Port video chat gpt is listening on in the container i.e. 7800

filename : string, required

- This is the video filename to analyse i.e. with mp4 extension

prompt : string, required

- This is the prompt for video chat gpt. i.e. "what is the video about? Is there anaything strange in the video?"

responsefolder : string, optional

- This is the folder you want video chatgpt to write responses to

temperature : float, optional

- Temperature determines how conservative video chat gpt is i.e. closer to 0 very conservative in responses

max_output_tokens : int, optional

- max_output_tokens determines tokens to return

RETURNS: The file name the response was written to by video chatgpt.

****46. maadstml.viperpreprocessrtms(vipertoken,host,port,topic,producerid,offset,maxrows=0,enabletls=0,delay=100,brokerhost='',brokerport=-999,microserviceid='',**

topicid=-999,rtmsstream='',searchterms='',rememberpastwindows='',identifier='',
preprocesstopic='',patternwindowthreshold='',array=0,saveasarray=0,rawdataoutput=0,
rtmsscorethreshold='',rtmsscorethresholdtopic='',attackscorethreshold='',
attackscorethresholdtopic='',patternscorethreshold='',patternscorethresholdtopic=''):**

Parameters:

** : string, required

VIPERTOKEN : string, required

- A token given to you by VIPER administrator.

host : string, required

- Indicates the url where the VIPER instance is located and listening.

port : int, required

- Port on which VIPER is listenting.

topic : string, optional

- This is the topic containing preprocessed data for entities

producerid : string, required

- Producerid for the topic.

offset : int, required

- This is the offset to start reading from ususally -1

maxrows : int, required

- The number of offsets to rollback the datastream

enabletls : int, required

- if 0, no encryption, otherwise if 1 all data are encrypted

brokerhost : string, optional

- Address of Kafka broker - if none is specified it will use broker address in VIPER.ENV file

brokerport : int, optional

- Port Kafka is listening on - if none is specified it will use port in the VIPER.ENV file

delay : int, optional

- delay parameter to wait for Kafka to respond - in milliseconds.

microserviceid : string, optional

- If you are routing connections to VIPER through a microservice then indicate it here.

topicid : int, required

- Specifies how entities are processed.

rtmsstream : string, required

- Specifies Kafka topic to stream the TEXT data into. Separate multiple topics by comma.

searchterms : string, required

- Search terms to use to search the data in text data. Separate by semi-colon for

  for different searches for different rtmsstream topics.

rememberpastwindows : int, required

- How many sliding time windows for TML to remember.

identifier : string, optional

- Identifies this analysis.

preprocesstopic : string, required

- Kafka topic to store the output.

patternwindowthreshold : int, required

- Threshold number of windows for the occurence of patterns of the search terms.

rtmsscorethreshold: string, optional

- Threshold number for RTMS score between 0-1

rtmsscorethresholdtopic: string, optional

- Name of a kafka topic that will contain messages greater than rtmsthreshold

attackscorethreshold: string, optional

- Threshold number between 0-1

attackscorethresholdtopic: string, optional

- Name of a kafka topic that will contain messages greater than attackthreshold

patternscorethreshold: string, optional

- Threshold number between 0-1

patternscorethresholdtopic: string, optional

- Name of a kafka topic that will contain messages greater than patternthreshold

array : int, optional

- Process data as arrays

saveasarray : int, optional

- Save output as arrays.

rawdataoutput : int, optional

- Output raw data used in the TML processing

RETURNS: Null

TML Core Technology Integration

Attention!

- All TML solutions can be run on-premise or in the cloud using Apache Kafka.
- All TML solutions process data in-memory - no external databases are needed - ONLY Apache Kafka.
- All TML solutions use TLS encryption to encrypt real-time data.
- All TML solutions compress real-time data using advanced compression algorithms like: snappy, gzip, lz4
- All TML solutions use JSON processing - not SQL - for faster, more cost effective, processing of real-time data. Refer to :ref:`JSON PROCESSING`
- All TML solutions perform entity based processing and machine learning. Refer to :ref:`TML Performs Entity Level Machine Learning and Processing`
- All TML solutions are containerized with Docker and scale with Kubernetes.
- All TML solutions are developed in Python using the MAADSTML Python Library and DAGs. Refer to :ref:`MAADSTML Python Library API` and :ref:`TML Solution Building`
- All TML solutions use REST API.
- All TML solutions can have a real-time visualization dashboards using websockets enabled by TML binary: Viperviz. Refer to :ref:`TML Real-Time Dashboards`

Below are all the technologies TML integrates with for fast, scalable, cost-effective, real-time solutions.

1. Apache Kafka

Apache Kafka is the world's largest open source platform for storage of real-time data streams. TML integrates with Apache Kafka on-premise or in the cloud using AWS MSK or Confluent Cloud.

2. Apache Airflow

Apache Airflow is an open-source workflow management platform for data engineering pipelines. It started at Airbnb in October 2014[2] as a solution to manage the company's increasingly complex workflows.

TML Solution Studio Container uses Airflow to build highly advanced, scalable, real-time TML solutions. Refer to :ref:`TML Solution Studio Container` for more details.

3. TML Binaries

TML uses THREE (3) core binaries: Viper, HPDE, Viperviz, for TML solutions. More details here :ref:`1. TML Components: Three Binaries`

4. TML Python Library

TML solutions are built with the MAADSTML Python Libary. Refer to :ref:`MAADSTML Python Library API` for more details.

5. TML GenAI With PrivateGPT and Qdrant Vector DB

TML solutions integrate with GenAI using a special PrivateGPT docker container. This allows for very secure, private, and highly cost-effective LLM capabilities. Refer to :ref:`TML and Generative AI` for more details.

The PrivateGPT container is integrated with Qdrant vector DB for localized AI processing with LLMs.

6. TMUX (Terminal Multiplexing)

All TML solution use TMUX to optimize TML solutions in Linux to enhance support and maintenance of solutions.

7. MariaDB (MySQL)

All TML solution use MariaDB as a configuration database for TML solutions.

8. Docker

TML solutions are containerized using Docker.

9. Kubernetes

TML solution containers are scaled with Kubernetes.

10. Github

TML solutions are tightly integrated with Github and can commit code locally and to remote branches directly from the TML Solution Studio container. Refer to :ref:`TML Solution Studio's Tight Integration with GitHub`.

11. Python and DAGs (Directed Acylic Graphs)

All TML solutions are written using Pre-written Python DAGs: see the :ref:`DAG Table`. Refer to :ref:`TML Solutions Can Be Built In 10 Steps Using Pre-Written DAGs (Directed Acyclic Graphs)`.

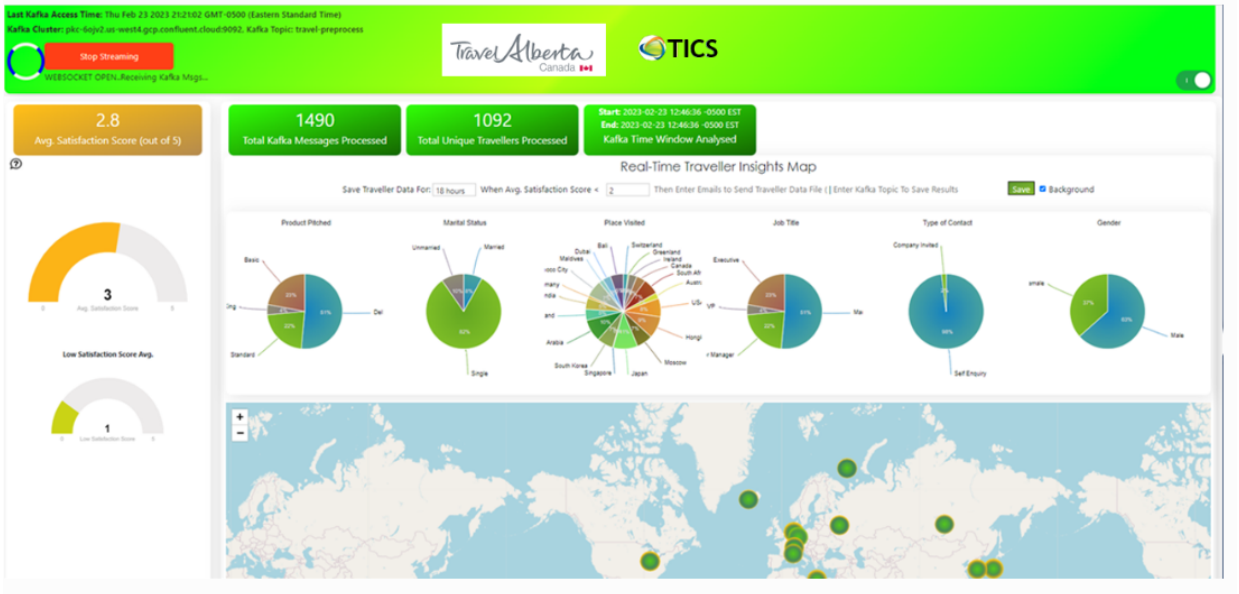
TML Real-Time Dashboards

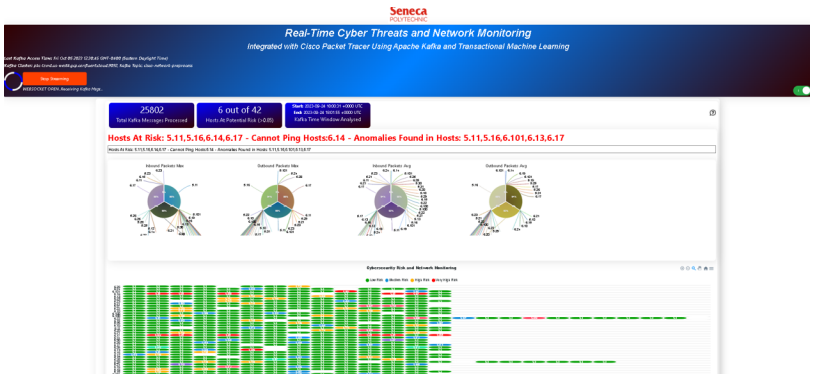
You can create real-time dashboards using TML binary called **Viperviz**. These dashboards are integrated with TML solutions and users can build very unique and powerful real-time dashboards using simple HTML and javascript.

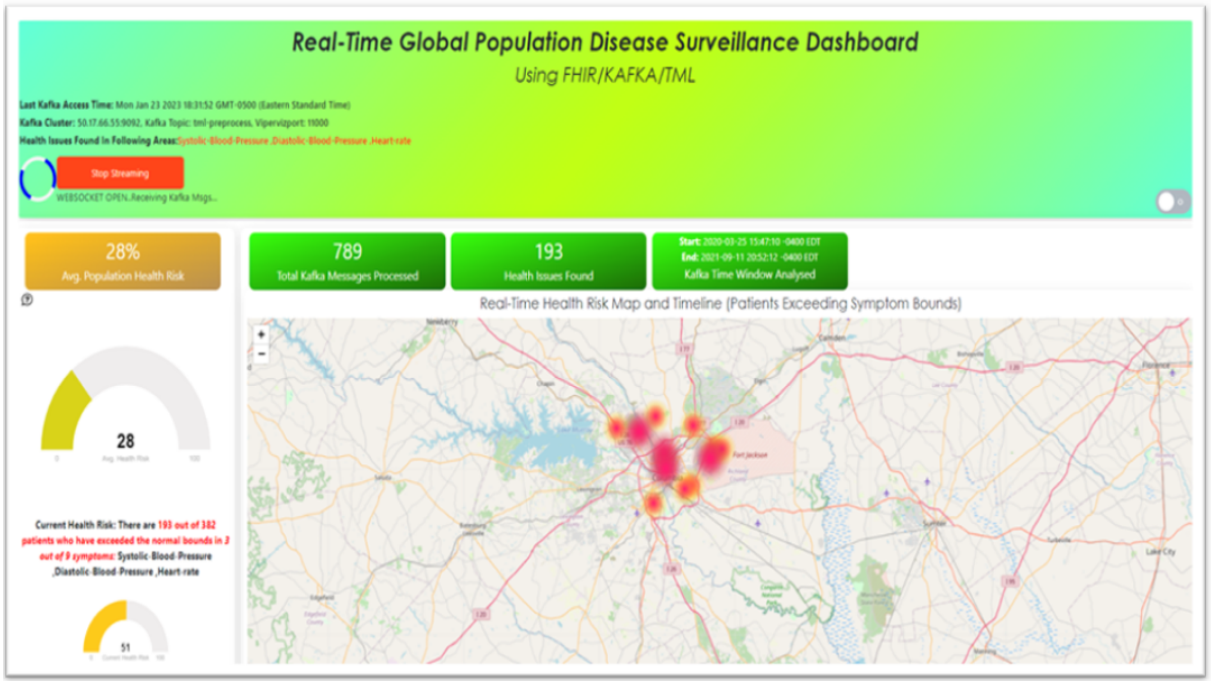
> **Tip**
>
> Watch the Youtube Video that shows how to create amazing TML real-time dashboards.

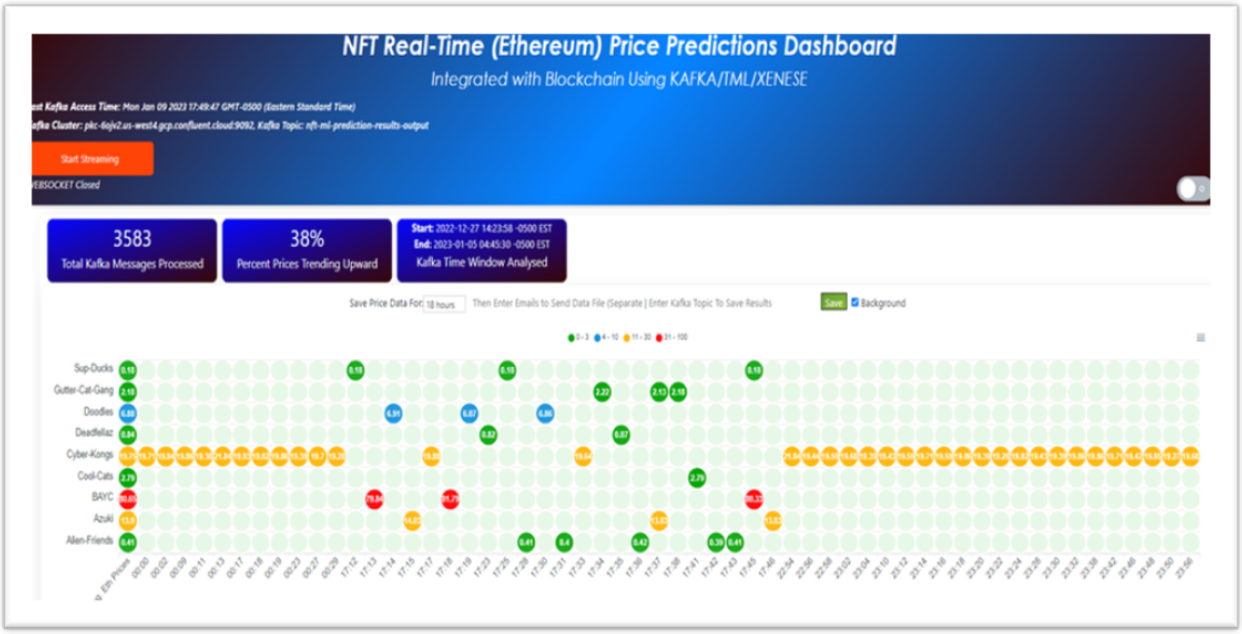
> **Note**
>
> Viperviz streams data **directly** from the **Docker Container** to the client browser. Viperviz binary uses websockets to stream data to the browser. This means you do not need a third-party visualization tool.

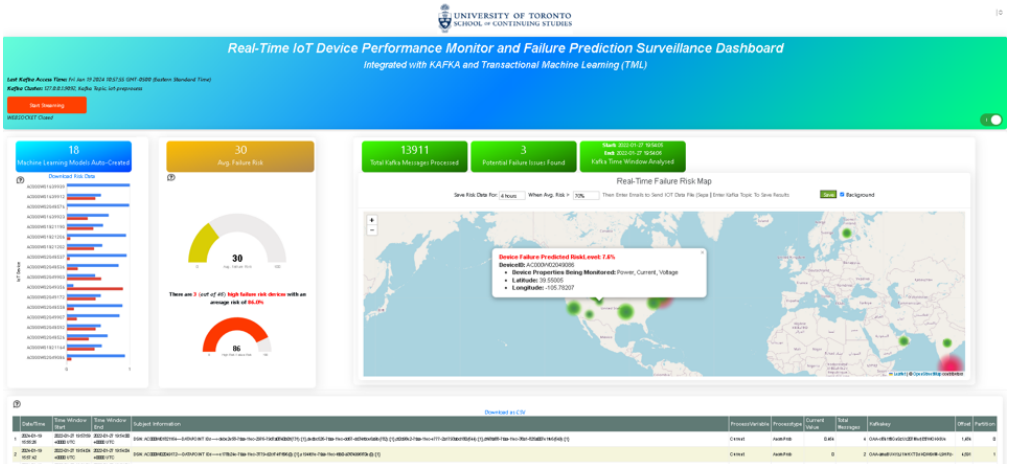
Some sample dashboards samples are below.

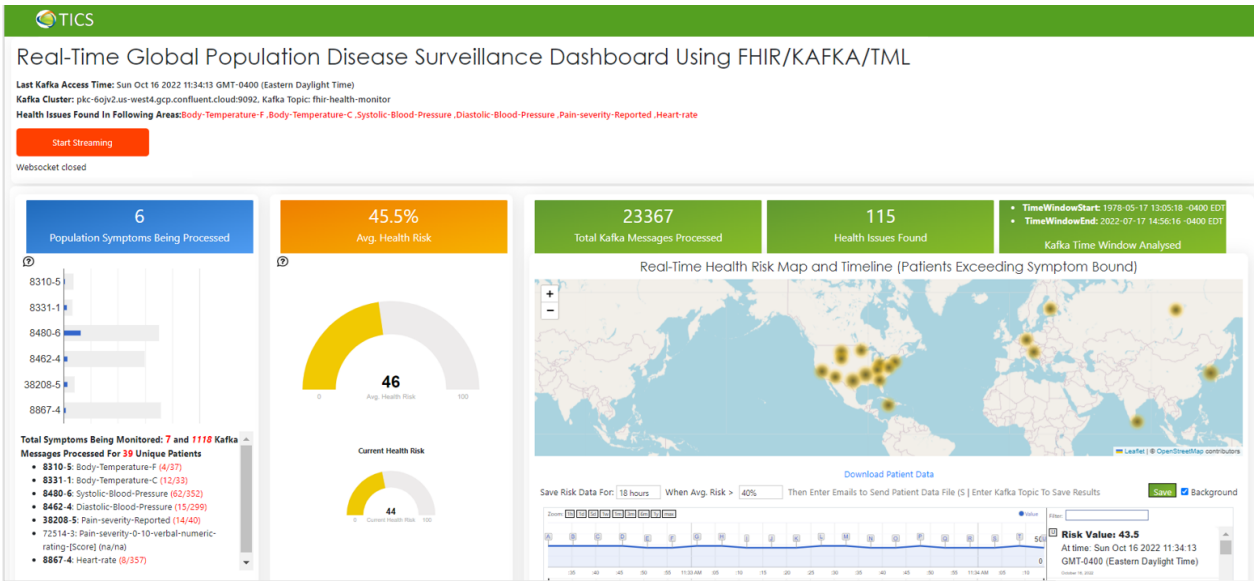





Real-Time Global Population Disease Surveillance Dashboard
Using FHIR/KAFKA/TML
Last Kafka Access Time: Mon Jan 23 2023 10:31:52 GMT-0500 (Eastern Standard Time)
Kafka Cluster: 50.17.66.55:9092, Kafka Topic: tml-preprocess, Vipervizport: 11000
Health Issues Found In Following Areas:Systolic Blood Pressure ,Diastolic Blood Pressure ,Heart rate
Stop Streaming
WEBSOCKET OPEN..Receiving Kafka Msgs...
28%
Avg. Population Health Risk
789
Total Kafka Messages Processed
193
Health Issues Found
Start: 2020-03-25 15:47:10 -0400 EDT
End: 2021-09-11 20:52:12 -0400 EDT
Kafka Time Window Analysed
Real-Time Health Risk Map and Timeline (Patients Exceeding Symptom Bounds)
28
Current Health Risk: There are 193 out of 382 patients who have exceeded the normal bounds in 2 out of 9 symptoms: Systolic Blood Pressure ,Diastolic Blood Pressure ,Heart rate
51


NFT Real-Time (Ethereum) Price Predictions Dashboard
Integrated with Blockchain Using KAFKA/TML/XENESE
Start Streaming
WEBSOCKET Closed
3583
Total Kafka Messages Processed
38%
Percent Prices Trending Upward
Start: 2022-12-27 14:23:58 -0500 EST
End: 2023-01-05 04:45:30 -0500 EST
Kafka Time Window Analysed
Save Price Data For: 18 hours
Then Enter Emails to Send Data File (Separate | Enter Kafka Topic To Save Results
Save
Background
Sup-Ducks
Gutter-Cat-Gang
Doodles
Deadfellaz
Cyber-Kongs
Cool-Cats
BAYC
Azuki
Allen-Friends


UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES
Real-Time IoT Device Performance Monitor and Failure Prediction Surveillance Dashboard
Integrated with KAFKA and Transactional Machine Learning (TML)
18
Machine Learning Models Auto-Created
30
Avg. Failure Risk
13911
Total Kafka Messages Processed
3
Potential Failure Issues Found
Real-Time Failure Risk Map
Device Failure Predicted Risk Level: 7.6%
DeviceID: AC000W020490096
Device Properties Being Monitored: Power, Current, Voltage
Latitude: 39.55005
Longitude: -105.78207

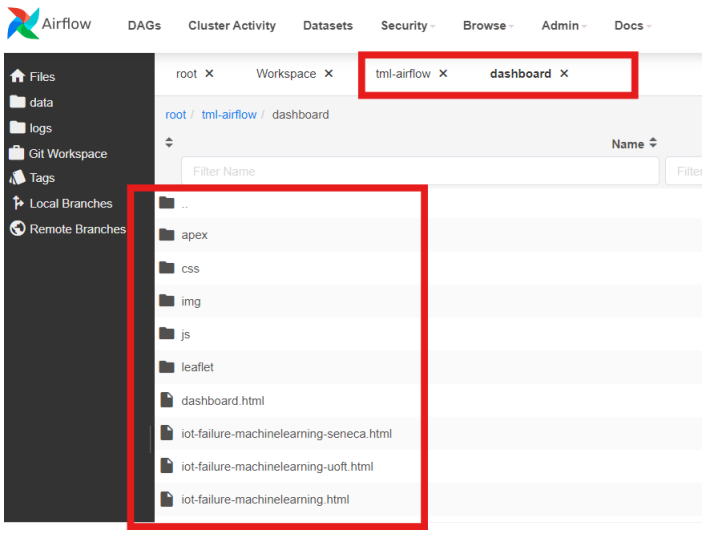


Running Dashboards

Creating Your Own Dashboards

Creating Real-Time Dashboard in TML are simple, yet very powerful and free. No third-party visualization tool is necessary.

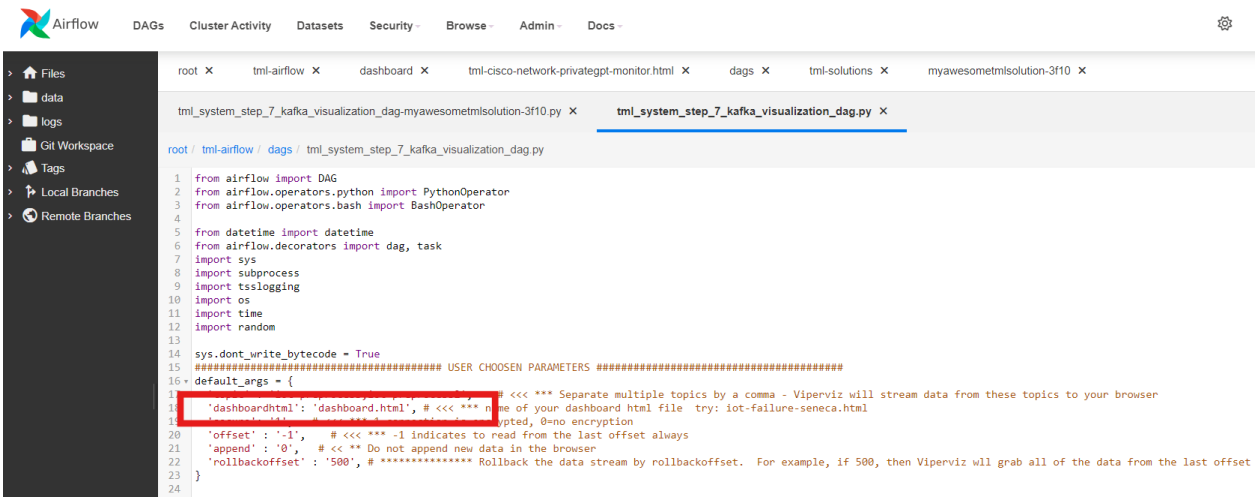
> ### Important
>
> All dashboards MUST be created and saved in your **<repo>/tml-airflow/dashboard** folder as shown in Figure below.
>
> You can also upload any images, javascript, css, etc. to the folders in this location.



Note

Refer to :ref:`STEP 7: Real-Time Visualization: tml-system-step-7-kafka-visualization-dag` for details.

To access your dashboard you must enter the filename of the HTML file (i.e. Your Dashboard file) in the **dashboardhtml** field in DAG 7.



Once you have created a dashboard to visualize TML data you enter a URL in your browser to run it.

Here is an example URL:

http://localhost:<PORT>/dashboard.html?topic=iot-preprocess&offset=-1&groupid=&rollbackoffset=500&topictype=prediction&append=0&secure=1

Note

When you run your TML solution in TSS by following the instructions here :ref:`Lets Start Building a TML Solution` a visualization URL will be generated for you in your TML solution documentation. A sample documentation is here. A PORT will be assigned to your dashboard at runtime.

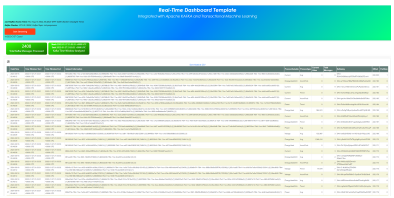
URL Key	Description
http://localhost:<PORT>	Almost all of the dashboard will point to the IP and Port that Viperviz is listening on. Viperviz has a built in webserver, so no setup is need, just plug and play. The above URL points to localhost and port 9005 for Viperviz
dashboard.html	TML Solution Studio (TSS) provides a template dashboard to get you up and running quickly. This is a base dashboard but will show you how real-time data from TML is analysed and processed. As shown in the above dashboards, you can create amazing dashboards with HTML and Javascript.

topic=iot-preprocess	In the topic key you specify the topic you want to consume data from. Viperviz will start consuming from this topic, i.e. iot-preprocess or whatever topic you have created to store your data. Note: You can specify more than one topic to consume from, just separate multiple topics with comma.
offset=-1	This tells Viperviz to start consuming from the latest data in the stream.
groupid=	ignored
rollbackoffset=500	This tells Viperviz to rollback the datastream by 500 offsets and send it to the browser. NOTE: While you can increase this number - use it with caution because it may overload your browser.
topictype=prediction	Leave as is
append=0	If this is 0, the dashboard will not keep appending new data because it may crash your browser as lots of data accumulates. If you set to 1, then data will append.
secure=1	Secure connection

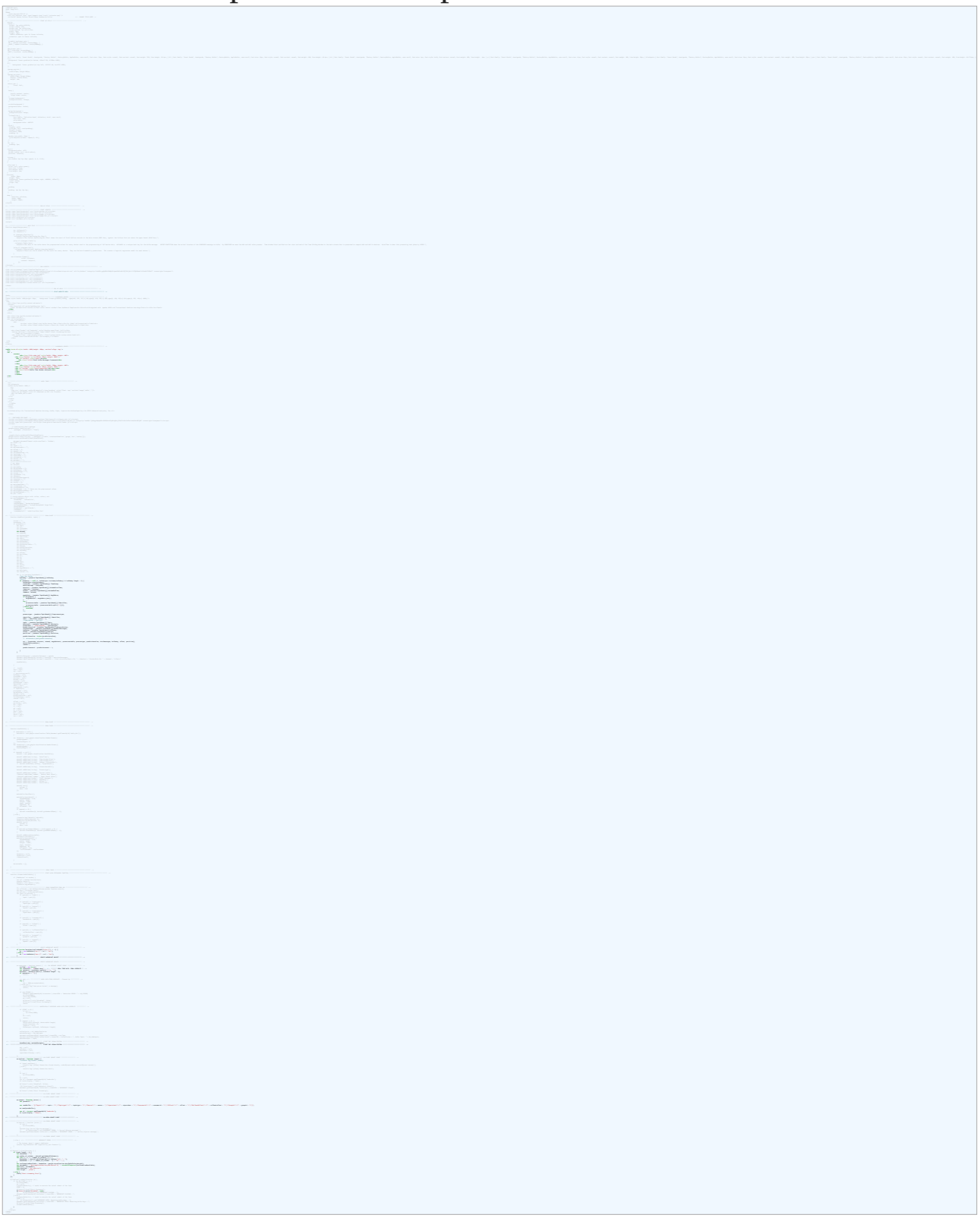
> Caution!
>
> Be careful streaming too many topics at once, and/or, setting the rollbackoffset to a high number because this will cause Viperviz to send a large amout of data to your computer browser, which could overload or crash your browser and/or your computer.

Dashboard Template

> Tip
>
> Here is the dashboard template - you can easily build on this to create your amazing real-time dashboards.
>
> This dashboard is a template that will immediately start to visualize your streaming data. You can add/edit/modify this dashboard to fit your business needs - it is written in simple HTML and Javascript. This template is a great way to quickly get you started with amazing real-time visualizations.
>
> You can modify it in TML Solution Studio (TSS) and commit changes to Github directly from the TSS.



Dashboard Template: Code Explanation



More Dashboard Examples

More examples are here: https://github.com/smaurice101/raspberrypi/tree/main/tml-airflow/dashboard

TML Solution Studio (TSS) Container

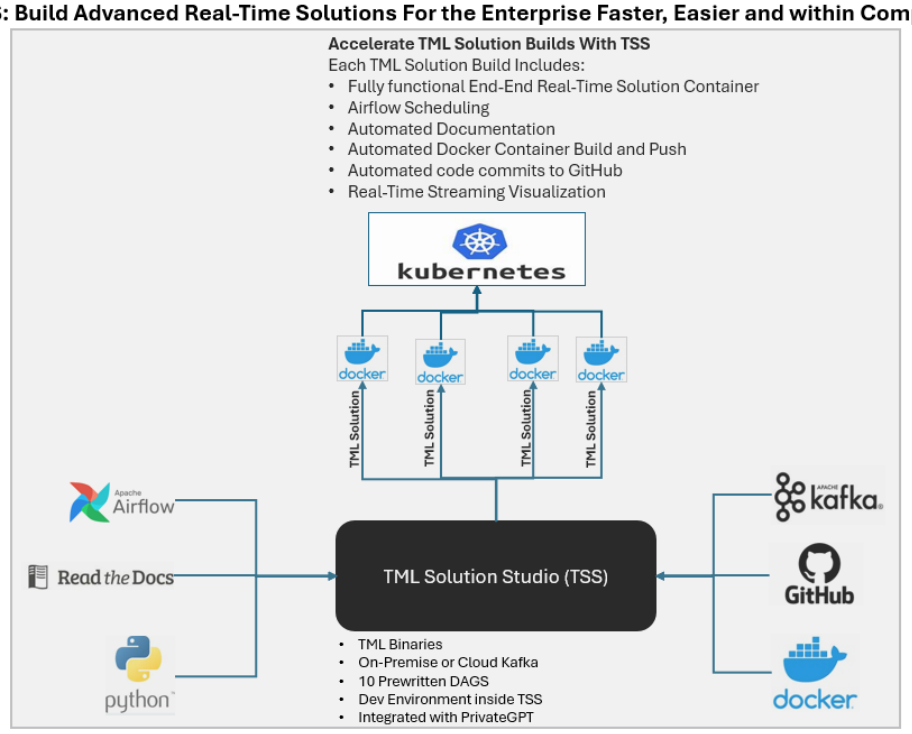
Before you can build TML solutions, you will need to run the TSS container for your OS. The TSS builds your TML solutions.

TSS Containers
Windows/Linux Users
Mac/Linux Users

> **Tip**
>
> Watch the TSS YouTube Video

> **Important**
>
> You MUST have the pre-requisites met before running this container: :ref:`TSS Pre-Requisites`

TSS: Build Advanced Real-Time Solutions For the Enterprise Faster, Easier and within Compliance



This is the main container that you need to use to build TML solutions. Below is the :ref:`TSS Docker Run Command` to run the container.

Important

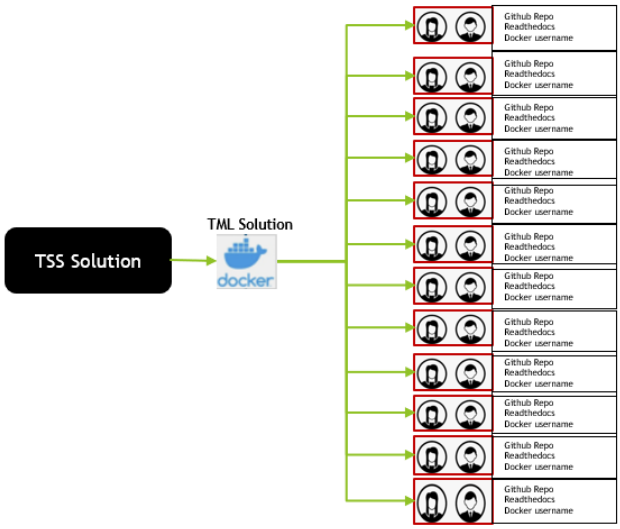
Use this TSS container and start building amazing, advanced and scalable real-time data streaming solutions - with real-time dashboards - auto deployment to Docker - auto solution documentation - integrated with AI - integrated with Github - integrated with Apache Airflow - integrated with Apache Kafka - in just a few minutes.

TML Solution Container Global Deployment

There are TWO ways to deploy TML solution containers:

1. **Docker Container:** Anyone around the world can run the container using the Solution's Docker Run Command
2. **Kubernetes:** Anyone around the world can run the TML Solution containers in kubernetes

For 1: see figure below



TML solution developer creates the TML solution container, any user around the world can run this solution with their own: 1. Github username/repo 2. Readthedocs token 3. Docker username

This means that ONE TML solution container is used by others with their OWN data, configurations parameters, and credentials without creating thousands of containers … think of this as FORKING in Github.

TSS Pre-Requisites

Attention!

The following prerequisites MUST be met before you run the TML Solution Studio Container using the :ref:`TSS Docker Run Command` :

1. You MUST Install Docker - in Linux Ubuntu run (if using Windows use WSL):

```
sudo apt update
sudo apt upgrade
```

```
sudo apt install docker.io
sudo docker --version
sudo docker run hello-world
sudo docker ps
```

2. You MUST have a Github Account
3. You MUST Clone Github Repo: https://github.com/smaurice101/raspberrypi.git
4. You MUST Create Github **Personal Access Token**. Refer to :ref:`Set Up Personal Access Tokens in Github`
5. You MUST have sign up for a Docker Hub account:
6. Create a Readthedocs account and get an API token: :ref:`Set Up Readthedocs`
7. Create a HiveMQ account - while this is OPTIONAL - if using MQTT and HiveMQ you will need this.
8. Kafka Cloud API keys from Confluent or AWS MSK - while this is OPTONAL - it will be for production, or large scale, deployments.

FOLLOW THE :ref:`How To Use the TML Solution Container` SECTION.

TSS Contains a TML Dev Environment

Important

Another powerful feature is the TSS TML Development environment which is contained directly inside the TSS container. TSS comes with all the TML solution components installed like: 1. Apache Kafka, 2. TML binaries, 3. MariaDB config db 4. your TML DAG scripts, 5. Viperviz for visualization

Once you are satisfied with your solution - you can use the :ref:`STEP 8: Deploy TML Solution to Docker : tml-system-step-8-deploy-solution-to-docker-dag` to deploy your solution to Docker.

Tip

TML developers can test each component or their entire TML solution inside the TSS before deploying the solution in the container. This is a very convenient way to make sure all the solution components are working before shipping your TML product.

TSS Docker Run Command

Note

If you are producing data using a local file, you need to add an extra -v volume map to the /rawdata folder in the container: Refer to :ref:`Producing Data Using a Local File`.

For example add **-v /your_localmachine/foldername:/rawdata:z**, where **your_localmachine/foldername** is a path in your local machine, and it is where you save your local file for processing.

Your file must contains JSON messages on each line. See Sample File

```
docker run -d --net="host" \
--env CHIP="AMD64" \
--env MAINHOST=127.0.0.1 \
--env TSS=1 \
--env SOLUTIONNAME=TSS \
--env AIRFLOWPORT=9000 \
--env VIPERVIZPORT=9005 \
--env EXTERNALPORT=-1 \
-v /var/run/docker.sock:/var/run/docker.sock:z \
-v /<your local dagsbackup folder>:/dagslocalbackup:z \
-v /your_localmachine/foldername:/rawdata:z \
--env READTHEDOCS='<Token>' \
--env GITREPOURL=<your github repo URL> \
--env  GITUSERNAME='<your github username>' \
--env GITPASSWORD='<Personal Access Token>' \
--env DOCKERUSERNAME='<your docker hub account>' \
--env DOCKERPASSWORD='<password>' \
--env MQTTUSERNAME='<enter MQTT username>' \
--env MQTTPASSWORD='<enter MQTT password>' \
--env KAFKACLOUDUSERNAME='' \
--env KAFKACLOUDPASSWORD='<Enter your API secret>' \
--env UPDATE=1 \
maadsdocker/tml-solution-studio-with-airflow-amd64
```

Parameter	Description
CHIP	Specifies the container OS. NOTE: If you are using MAC then change to CHIP=ARM64
MAINHOST=127.0.0.1	This is the IP address for the TML solution container. It will normally listen on 127.0.0.1
TSS	Do not modify.
SOLUTIONNAME	Do not modify.
AIRFLOWPORT=9000	This is the AIRFLOWPORT. This port will be needed to access the TML solution studio from your browser. For sample, enter: http://localhost:9000/ You will be asked for a username and password: enter **tml** for both.

VIPERVIZPORT	Choose a Viperviz port for visualization. For example, 9005
-v /<your local dagsbackup folder>:/dagslocalbackup:z	If you like, you can also backsup the dags to your local folder with this volume mapping
-v /your_localmachine/foldername:/rawdata:z	If you like, you can also map our local folder to the rawdata folder. This is neede if you will be processing local files with TML.
-v /var/run/docker.sock:/var/run/docker.sock:z	This maps the docker volume to the container - so TML studio can automatically build your solution container.
READTHEDOCS=<Token>	Create, copy and paste the Readthedocs token here. Refer to :ref:`Set Up Readthedocs`
GITREPOURL=<your github repo>	This is your Git repo you cloned from: https://github.com/smaurice101/raspberrypi.git.
GITUSERNAME=<your github username>	This is the username to your repo.
GITPASSWORD=<Personal Access Token>	This is the **Personal Access Token** for your repo. Look at the image below to find out how to generate this token.
DOCKERUSERNAME=<your docker hub account>	This is your Docker Hub username.
DOCKERPASSWORD=<password>	This is your password to Dockerhub account.
MQTTUSERNAME=<your MQTT username>	This is your MQTT username.
MQTTPASSWORD=<MQTT password>	This is your password to MQTT cluster.
UPDATE=1	This enables system updates if set to 1: meaning any updates to the system DAGS made by the TSS system maintainer will update all the user DAGS in all of the projects. This is a remote GitHub pull that keeps users ALWAYS with the updated Dags. You can Turn OFF system updates by setting to 0.

maadsdocker/tml-solution-studio-with-airflow-amd64	This is the official TML Solution Studio container for Window/Linux users with AMD64 chip architecture.
	If using MAC/Linux change: **amd64** to **arm64**
	For example:
	maadsdocker/tml-solution-studio-with-airflow-**arm64**
	`docker run -d --net="host" \`
	`--env CHIP="ARM64" \`
	`--env MAINHOST=127.0.0.1 \`
	`--env TSS=1 \`
	`--env SOLUTIONNAME=TSS \`
	`--env AIRFLOWPORT=9000 \`
	`--env VIPERVIZPORT=9005 \`
	`--env EXTERNALPORT=-1 \`
	`-v /var/run/docker.sock:/var/run/docker.sock:z \`
	`-v /<your local dagsbackup folder>:/dagslocalbackup:z \`
	`-v /your_localmachine/foldername:/rawdata:z \`
	`--env READTHEDOCS='<Token>' \`
	`--env GITREPOURL='<your git hub repo>' \`
	`--env  GITUSERNAME='<your github username>' \`
	`--env GITPASSWORD='<Personal Access Token>' \`
	`--env DOCKERUSERNAME='<your docker hub account>' \`
	`--env DOCKERPASSWORD='<password>' \`
	`--env MQTTUSERNAME='<enter MQTT username>' \`
	`--env MQTTPASSWORD='<enter MQTT password>' \`
	`--env KAFKACLOUDUSERNAME='' \`
	`--env KAFKACLOUDPASSWORD='<Enter your API secret>' \`
	`--env UPDATE=1 \`
	`maadsdocker/tml-solution-studio-with-airflow-arm64`

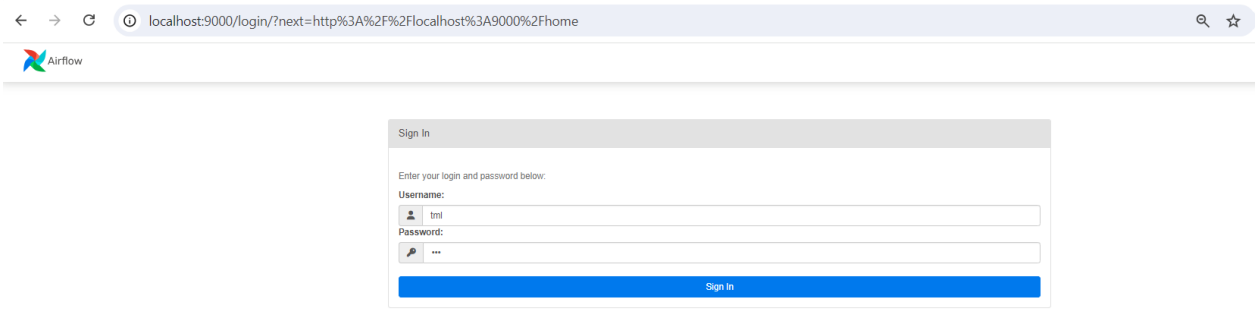
> Important
>
> It is highly recommended you map your local folder
>
> to the **dagslocalbackup** folder:
>
> **-v /<your local dagsbackup folder>:/dagslocalbackup:z**
>
> This ensures that if anything happens to Github you
>
> always have a local copy of all of your solution dags.

How To Use the TML Solution Container

> Tip
>
> Once you have the TML Solution container running you can go to your favourite browser and type the URL: http://localhost:9000

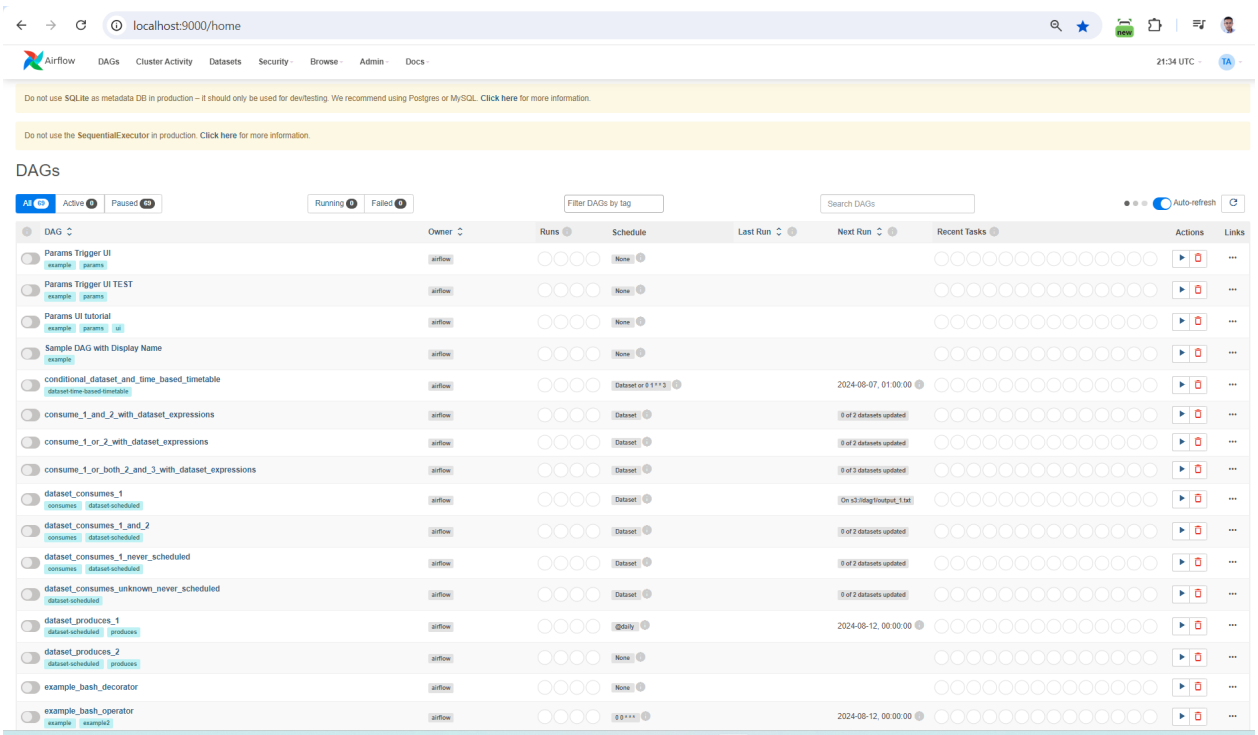
> Note
>
> The PORT number in the URL is what you specified in the Docker Run AIRFLOWPORT parameter i.e. **--env AIRFLOWPORT=9000**

After you enter the URL you will the following website:

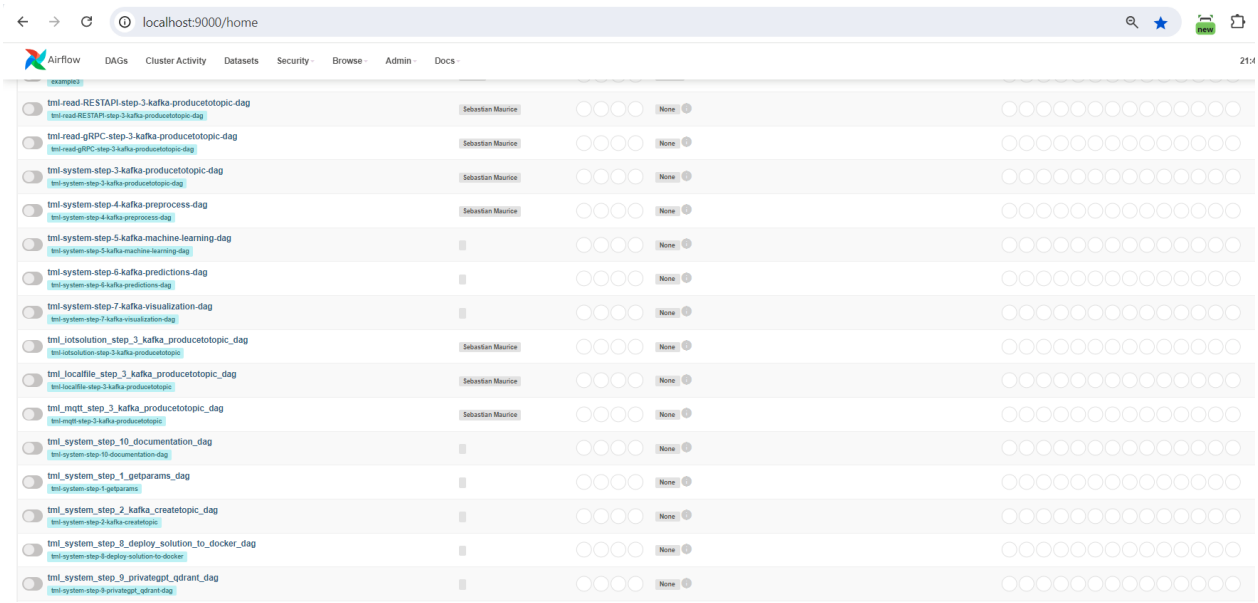


> Tip
>
> The username and password are both **tml**

After you have signed in successfully you will see the following screen with example DAGs:

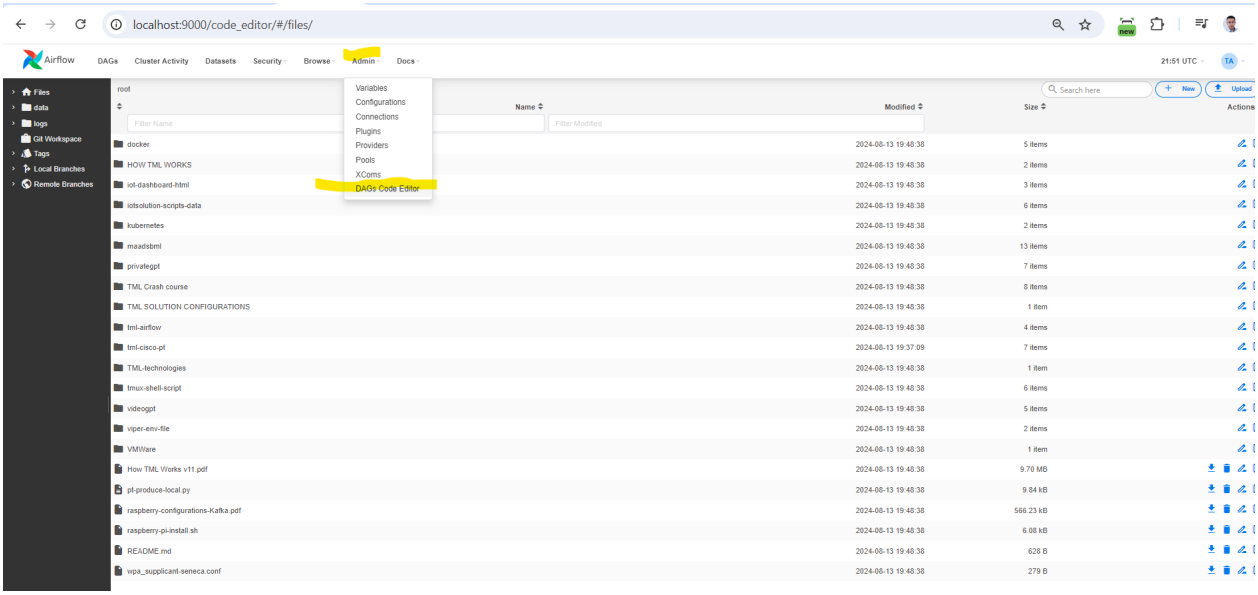


If you scroll down you will see the **TML DAGs** - as defined here: :ref:`DAG Table`. These are the DAGs you will use to build your TML Solutions:



TSS Code Editor

> Important
>
> Next go into the DAG Code Editor: Select Drop-down menu **Admin --> DAGs Code Editor**. Most of your TML Solution building will be done here. Note the DAGs solution process flows defined here: :ref:`Apache Airflow DAGs`



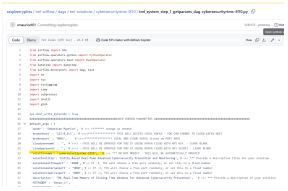
TSS Demo Github, Docker and Readthedocs Site Credentials

> **Important**
>
> To make it easier for users to run a TML solution quickly, a TML Demo Github site is provided here:
>
> - **GITHUB USERNAME:** tsstmldemo
> - **GITHUB REPO URL:** https://github.com/tsstmldemo/tsstmldemo
> - **GITHUB Personal Access Token:** <This will be retrieved for you from the OS IF using tsstmldemo repo>
> - **Readthedocs Token:** aefa71df39ad764ac2785b3167b77e8c1d7c553a
> - **Docker Username:** tsstmldocker
> - **Docker Password:** TssTml0828!?
>
> **Note: The above credentials are for demo purposes ONLY and are shared with anyone.** They ARE NOT meant to be used permanently. If you like TSS/TML and plan to use it seriously, then personal accounts and credentials must be used. Refer to TSS Pre-requisites on how to set these up.

Readthedocs Documentation URL

> **Important**
>
> The TSS is integrated with Readthedocs for automated documention via Github.
>
> All TML auto-generated documentaton can be found on readthedocs. The format of the URL is: https://<solution name>.readthedocs.io.
>
> For example, if you created a project name **cybersecurityrtms-3f10**, then the url will be:
>
> https://cybersecurityrtms-3f10.readthedocs.io
>
> The **solution name** can be found in the Step 1 task under your project name. As shown in the figure below, the TML solution name is **cybersecurityrtms-3f10**

> **Tip**
>
> You also add **-a stdout** to your solution Docker Run command. This will allow you to see the **READTHEDOCS URL** to your documentation.



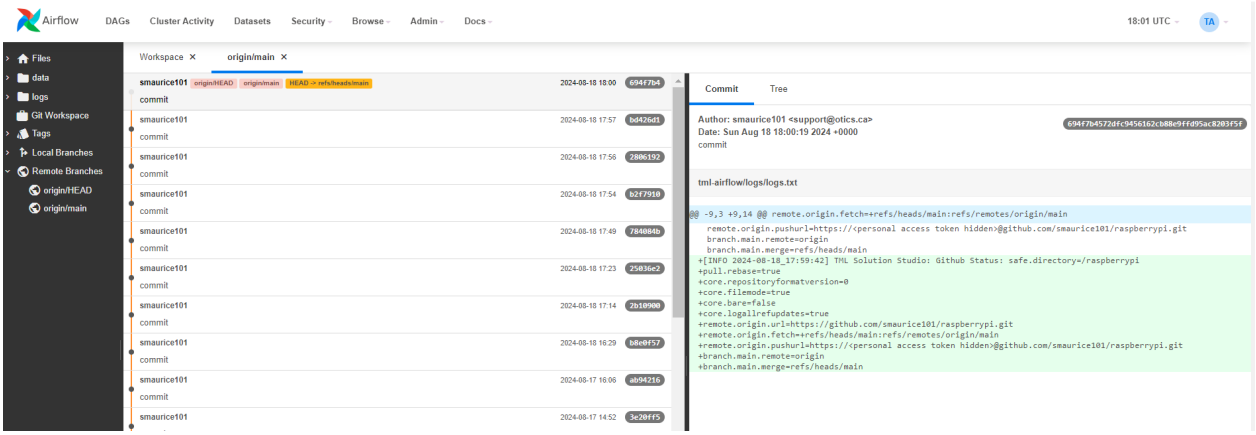
Common Docker and TMUX Commands

This is a list of common commands for Docker and Tmux.

Description	Command
List Docker containers	Type: **docker image ls**
Delete Docker containers	From: **docker image ls** and copy the **REPOSITORY** to delete Type: **docker rmi <REPOSITORY name> --force**
List Running Docker containers	Type: **docker ps**
Stop Running Docker containers	From **docker ps** copy the Container ID Type: **docker stop <paste container ID>**
Stop ALL Running Docker containers	Type: **docker stop $(docker ps -a -q)**
Go inside the Docker containers	From **docker ps** copy the Container ID Type: **docker exec -it <paste container ID> bash**
List the TMUX windows once inside the container	Type: **tmux ls**
Go inside TMUX windows	From **tmux ls** copy the window name you want to enter Type: **tmux a -t <window name>**
To scroll inside a TMUX window	Press: **CTRL+b, [**
To UN-scroll inside a TMUX window	Press: **CTRL + [**
To EXIT a TMUX window	Press: **CTRL + b, d**
To EXIT docker container	Type: **exit**
To Prune Docker System and Recover memory	System Prune type: **docker system prune**
To remove ALL containers	Type: **docker rm $(docker ps -a -q)**

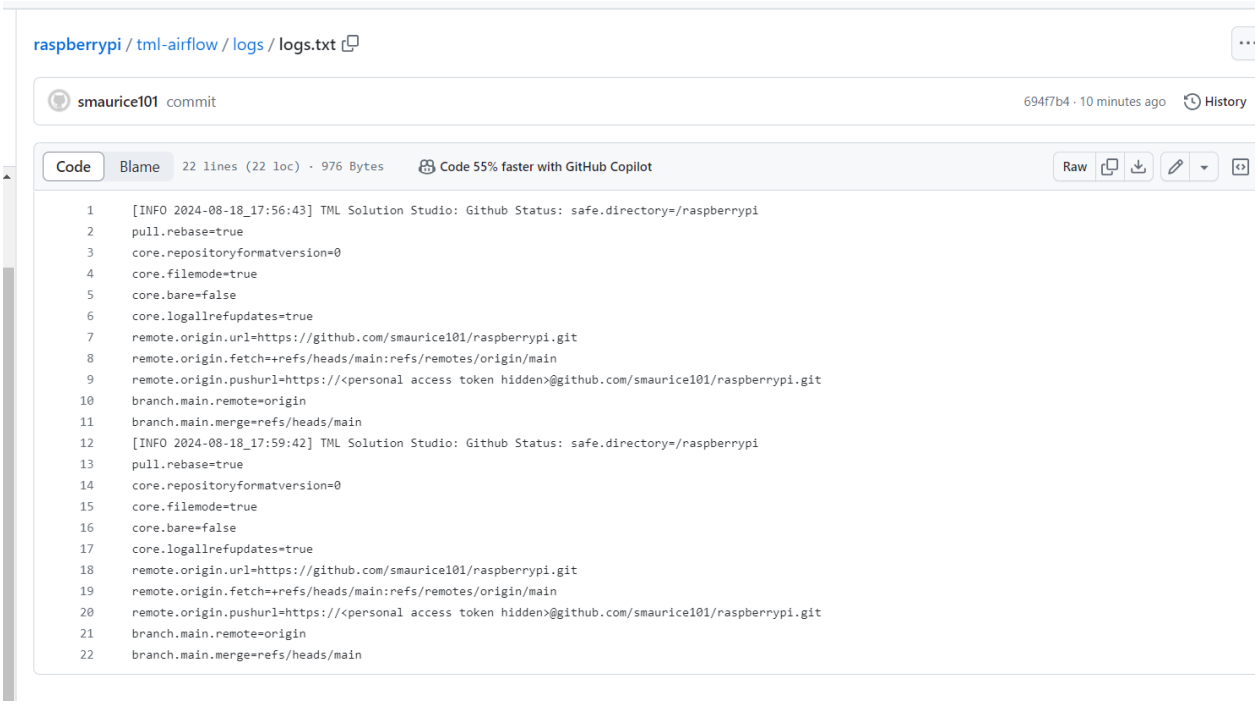
TSS Logging

The entire TSS solution build process is logged and committed to Github. This makes it very convenient to check for any errors in the TSS build process, and because errors are commited to the remote branch, the errors become visible to others to help in quickly rectifying any issues.



Tip

The logs are committed to your Github folder: **/tml-airflow/logs/logs.txt**



TML Solution Studio's Tight Integration with GitHub

TML Solution build process is tightly integrated with GitHub to maintain a seamless process of tracking TML solution changes. The figure below shows this integration. As TML solution code is updated in Airflow, it can be committed directly from the Airflow UI. This provides a high-level of convevience to ensure all code is properly checked in and committed to the main branch.

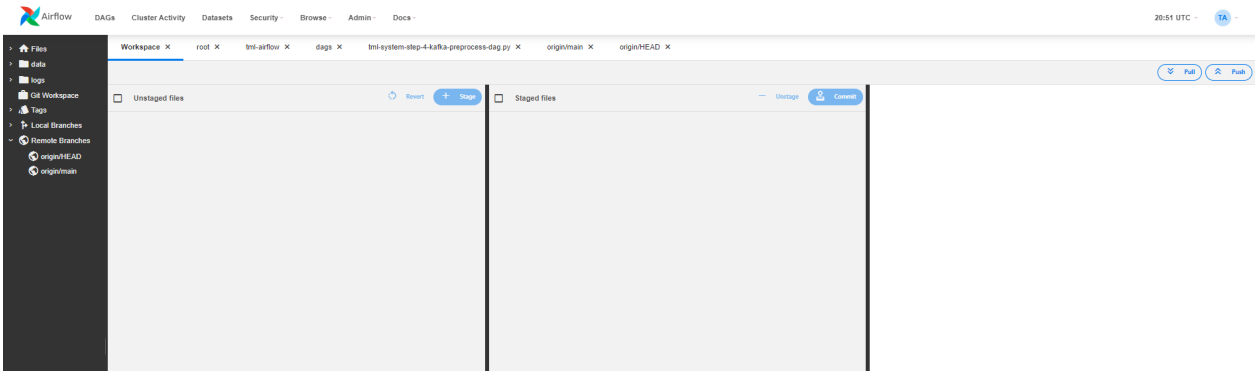
> **Tip**
>
> You integrate Github with TSS you need to generate a **Personal Access Token** from your Githib repo. Follow the instructions here: :ref:`Generating Personal Access Tokens in Github`. You then add this token in the :ref:`TSS Docker Run Command` in the field **GITPASSWORD**.

You can push and pull changes directly from the TML Solution Studio - integrated with Airflow.

> **Important**
>
> You can commit your code in the local and remote branches to ensure tightly controlled code changes. This is an important aspect of the TML Studio Studio (TSS) as well, all **git pushes** can be done directly from the TSS.
>
> NOTE: IT IS HIGHLY RECOMMENDED YOU WORK MAINLY IN THE TSS CODE EDITOR. IF YOU NEED TO MAKE CHANGES IN YOUR GITHUB REPO DIRECTLY - THEN MAKE THOSE CHANGES FIRST, THEN START WORK IN TSS...OTHERWISE THERE MAY BE A CONFLICT BETWEEN YOUR REMOTE AND LOCAL BRANCHES AND YOU MAY LOSE EDITS IN TSS EDITOR.

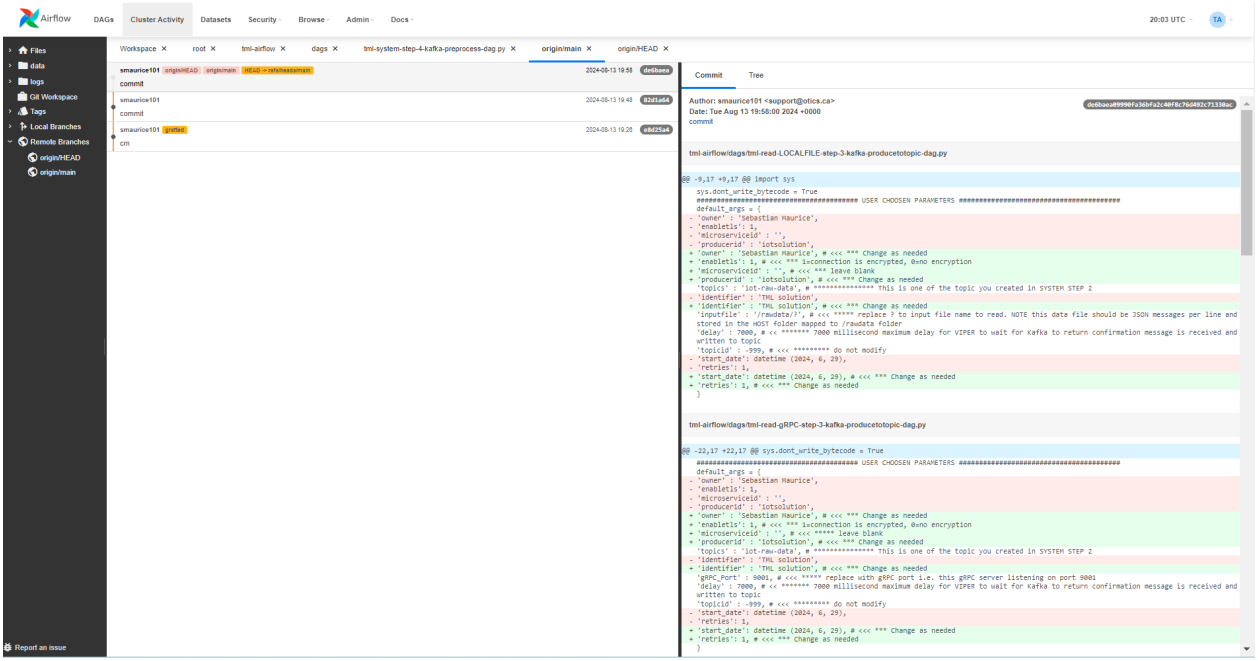
Push To and Pull From Local and Remote Github Branches

All changes are committed to local and remote branches by simple press of a button in the TML Studio Git Workspace.



Viewing Local and Remote Github Branches

You can view the local and remote branches conveniently inside the TML Solution Studio.

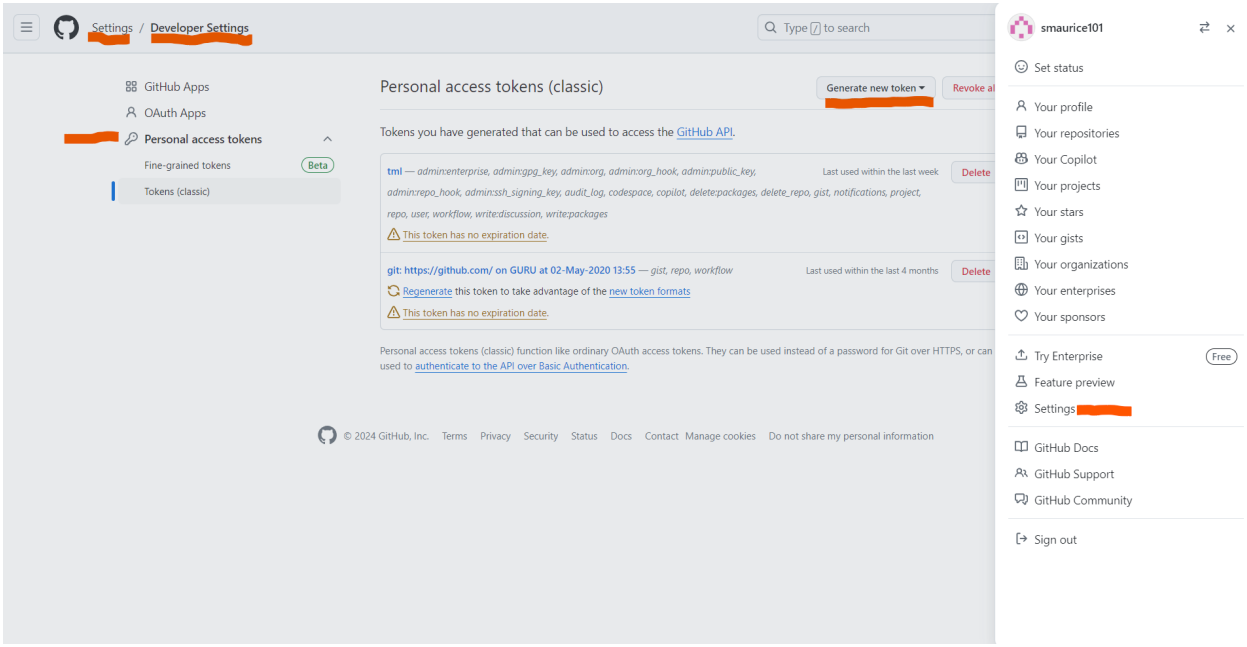


Github Logs

This is your main TSS Github logs. All TSS processes are committed to Github and logged.

> Important
>
> https://github.com/<your github username>/<repo you cloned>/blob/main/tml-airflow/logs/logs.txt

Set Up Personal Access Tokens in Github



Generating Personal Access Tokens in Github: Explanation

> **Tip**
>
> Follow these steps:
>
> 1. Log in to your Github account
> 2. In the Top-Right corner of your Github account click **Settings**
> 3. In the next screen, scroll all the way down and click **<> Developer settings**
> 4. Click **Personal access tokens**
> 5. Choose **Tokens (classic)**
> 6. Click **Generate new token** - Your token should start with **ghp_**.
>
>   **Give your Token Read/Write access**: see :ref:`Permissions For Your Token`
> 7. Copy and paste token in **GITPASSWORD** docker run command: :ref:`TSS Docker Run Command`

Permissions For Your Token

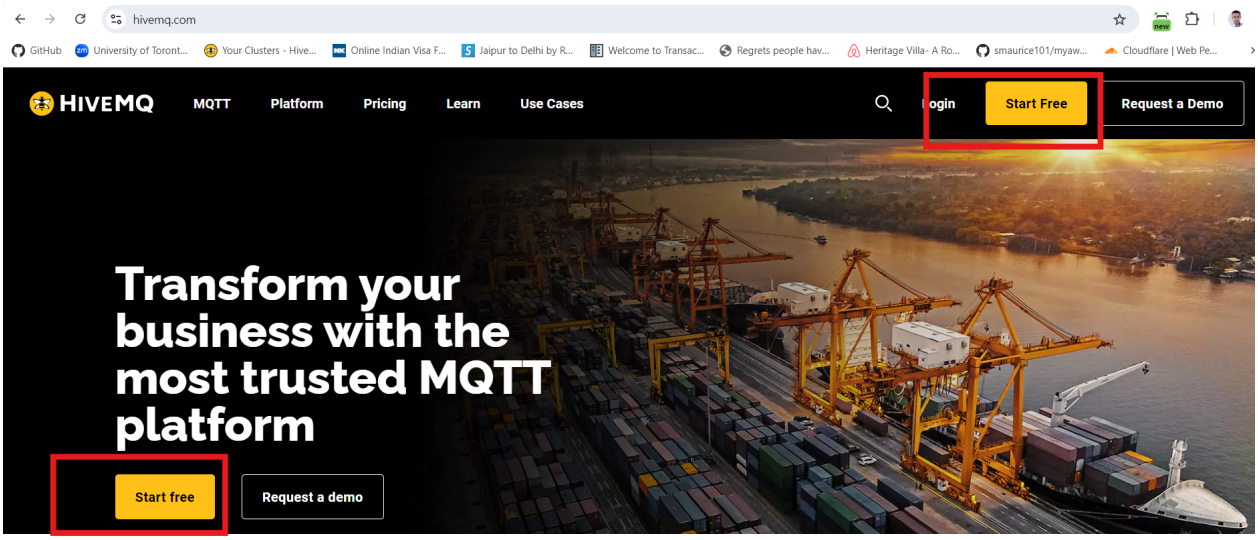
Your token permissions should look like this below:

Set Up HiveMQ

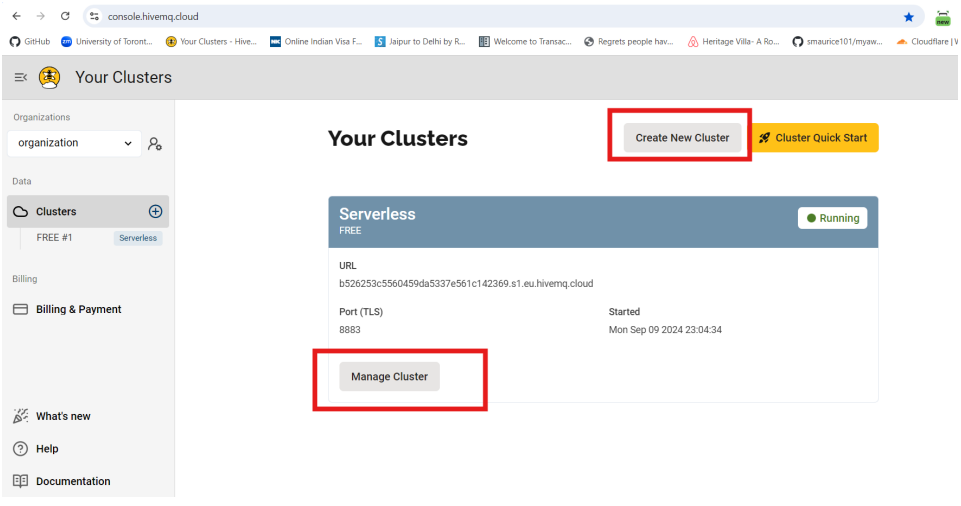
Setting up a HiveMQ cluster is a simle process and users can setup a HiveMQ cluster for free here at HiveMQ wesbite

> Important
>
> If you are planning on using the MQTT protocol in your TML solution then you will need to setup HiveMQ cluster.
>
> You will need to enter the **MQTTUSERNAME='<enter MQTT username>'** and **MQTTPASSWORD='<enter MQTT password>'** in the :ref:`TSS Docker Run Command`

Step 1: Setup a Free HiveMQ Account

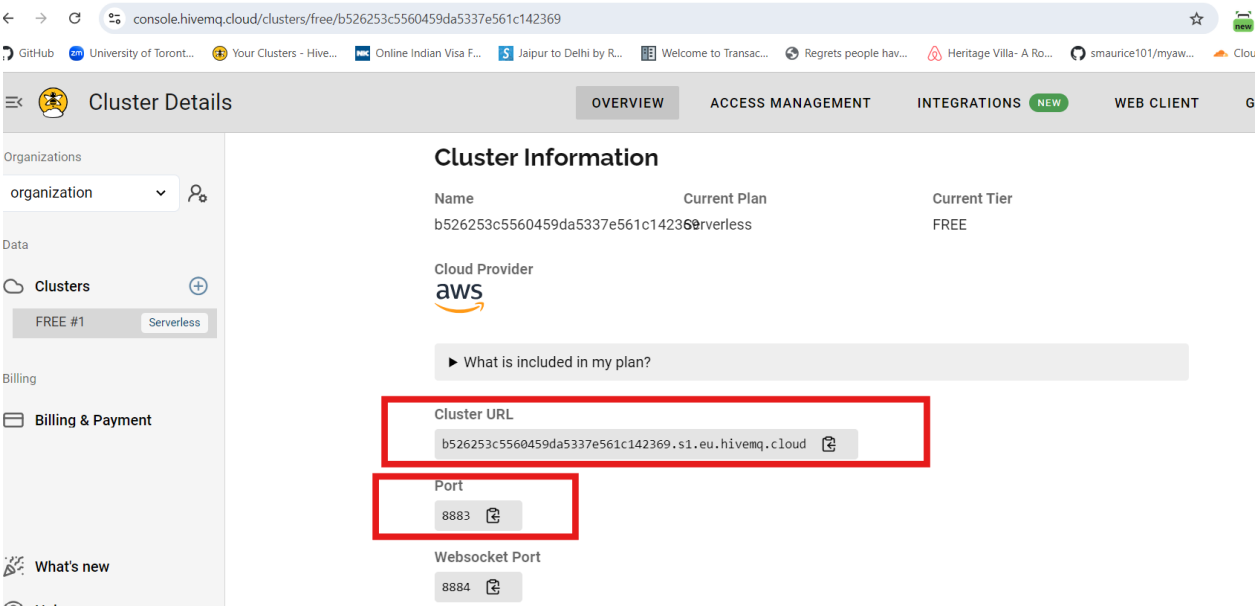


Step 2: Create a FREE Serverless Cluster



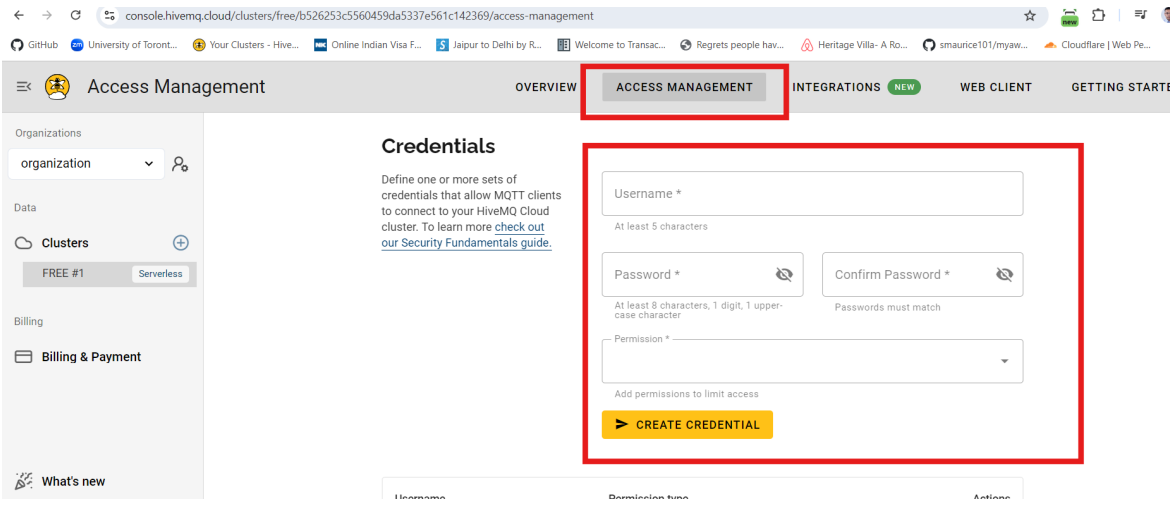
Step 3: Cluster Details

> **Note**
>
> Note the cluster URL and Port. You will need these details if publishing and subscribing from the HiveMQ cluster topic.



Step 4: Create MQTT Username and Password

> **Important**
>
> Give your username **Publishing and Subscribing** permission. These will be needed in the :ref:`TSS Docker Run Command`

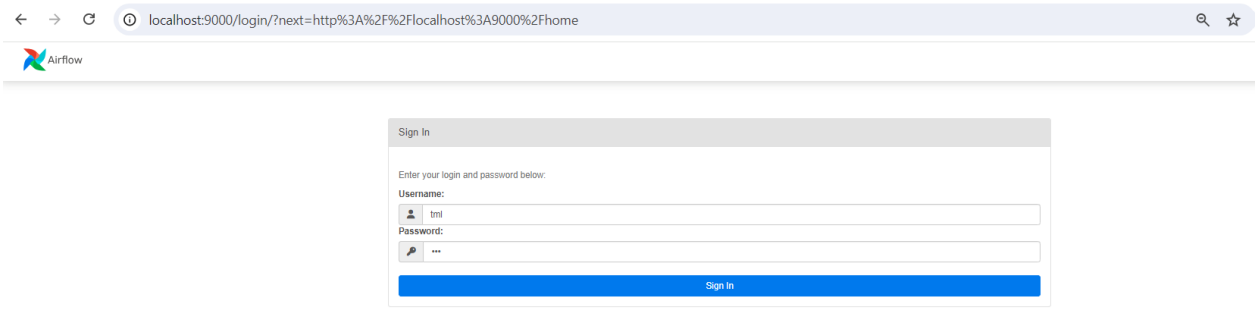


How To Use the TML Solution Container Test

> **Tip**
>
> Once you have the TML Solution container running you can go to your favourite browser and type the URL: http://localhost:9000

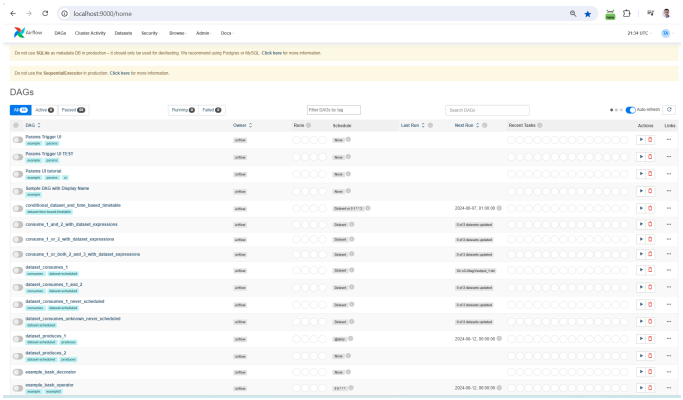
> **Note**
>
> The PORT number in the URL is what you specified in the Docker Run AIRFLOWPORT parameter i.e. **--env AIRFLOWPORT=9000**

After you enter the URL you will the following website:

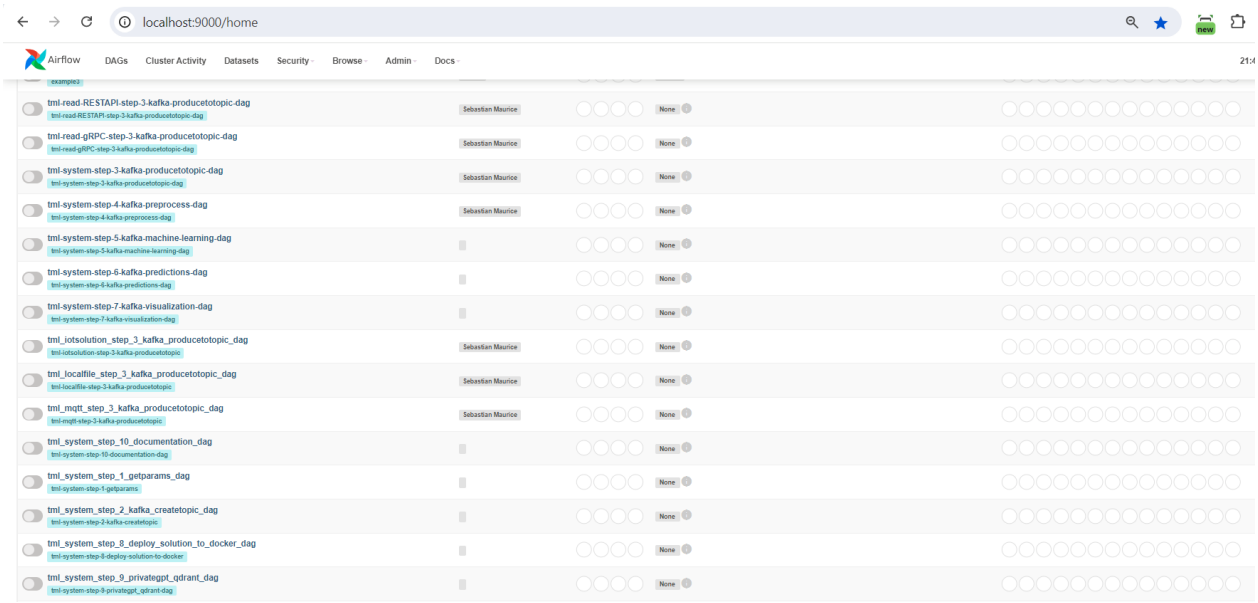


> **Tip**
>
> The username and password are both **tml**

After you have signed in successfully you will see the following screen with example DAGs:

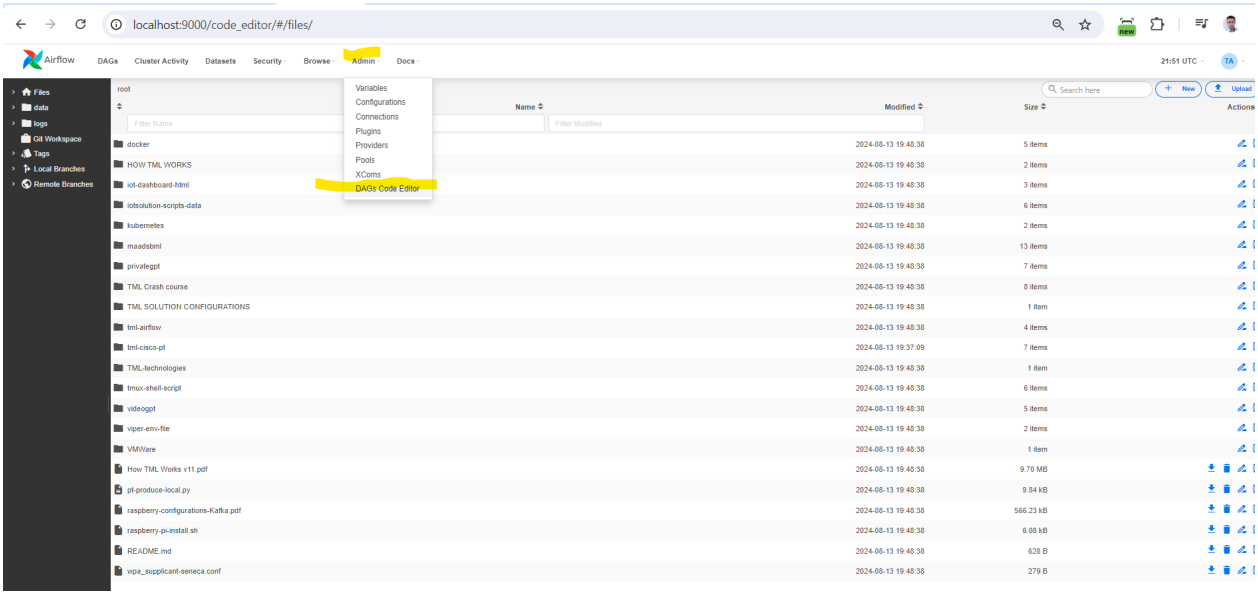


If you scroll down you will see the **TML DAGs** - as defined here: :ref:`DAG Table`. These are the DAGs you will use to build your TML Solutions:



Attention!

Next go into the DAG Code Editor: Select Drop-down menu **Admin --> DAGs Code Editor**. Most of your TML Solution building will be done here. Note the DAGs solution process flows defined here: :ref:`Apache Airflow DAGs`



Set Up Readthedocs

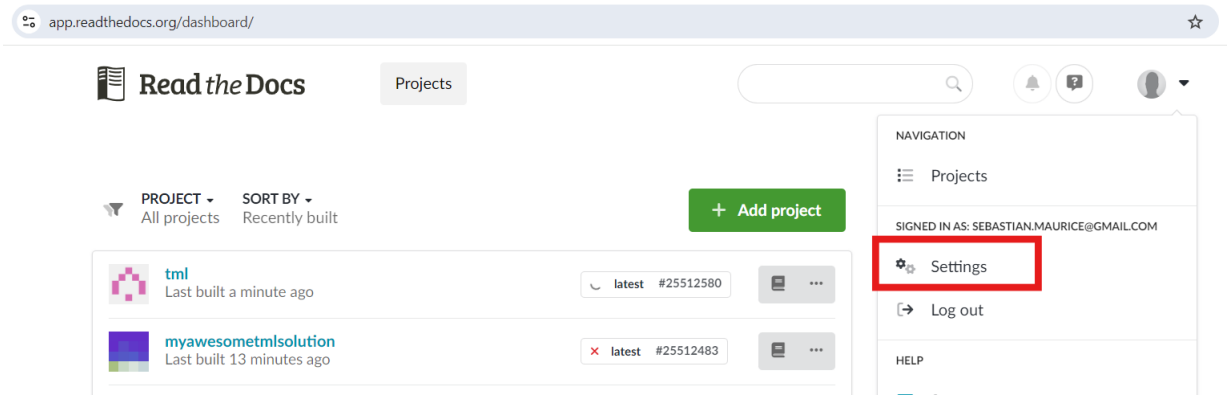
> Important
>
> TSS automatically creates a TML solution documentation as part of your solution. This is an important part of the process because it will give you instructions on how to operate your solution as well as the parameters used in the DAGs.

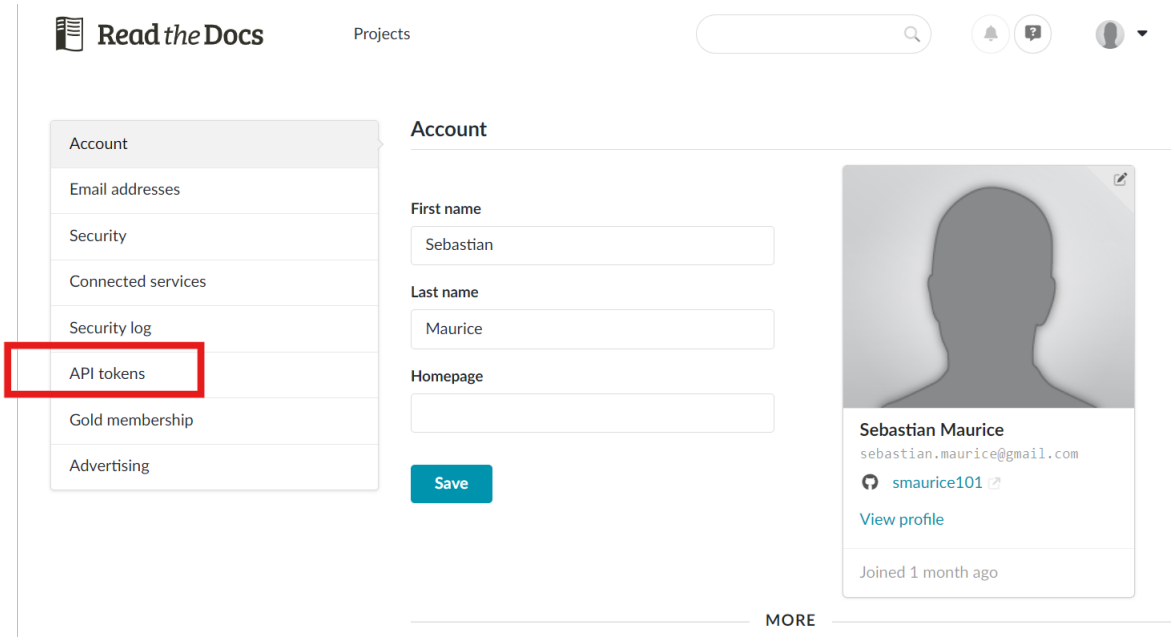
STEP 1: Create a Readthedocs Account

Create a free account at Readthedocs

STEP 2: Generate API Token

Once account created, go to your Profile settings and generate API Token - see figure below





STEP 3: Add Token in TSS Docker Run Command

Copy the Token and paste it in the TSS docker run command. Refer to :ref:`TSS Docker Run Command`

--env READTHEDOCS=<Token>

Real-Time Message Scoring (RTMS): How TML Maintains Past Memory of Events Using Sliding Time Windows in Real-Time

> **Tip**
>
> This capability is implemented in :ref:`STEP 4c: Preprocesing 3 Data: tml-system-step-4c-kafka-preprocess-dag`

> **Tip**
>
> Watch the RTMS Youtube Video

> **Note**
>
> While the RTMS is demonstrated for Cyber security, **it can be applied to any usecase in Retail, Finance, IoT, Energy, Manufacturing etc..** Anytime you want to analyse TEXT files and determine if events have occured in the past and quantify their importance (or lack of importance) then this is a powerful feature for you.
>
> Also, if you want to **cross-reference TML machine learning output of every entity to text files** i.e. log files, and "remember" their behaviour then this feature becomes very powerful for you. For example, you may be processing Entities in Step 4 and then want to determine if an entity is showing up in the logs or whether it is hacking in to your company using a slow and "occasional" attempt over time to EVADE detection algorithms, then RTMS can be very powerful to detect this complex behaviour.

Importance of RTMS For Cyber Crime Detection

- The growth of **real-time data** according to IDC Research **will reach 30% of global data in 2025 or roughly 90 ZB or 90 trillion gigabytes** mainly from IoT devices connected to the Internet
- This raises concerns and opportunities to process real-time data with Transactional Machine Learning (TML)
- The major concern with real-time streaming data from connected devices is the risk of Cybersecurity attacks
- Cyber crime is expected to cost the global economy $10.7 Trillion in 2025 and this number is growing
- This makes Cyber crime prevention and mitigation a Top Priority for global organizations regardless of size
- TML presents a powerful method of detecting, mitigating and preventing cyber attacks at the entity level by “remembering” past events in past sliding time windows and quantifies this by computing three scores:

- **Attack Score (AS):** Quantifies the attack vector. Higher number, more likely attack is occurring
- **Pattern Score (PS):** Quantifies the pattern vector. Higher number, more likely a pattern in the attack
- **Real-Time Memory Score (RTMS):** Combines both the Attack and Pattern Scores for an OVERALL score

The RTMS Method

1. User tells TML to keep a memory of past sliding time windows
2. User wants TML to search for malicious events from each entity (i.e. IP address, devices, etc..)
3. Malicious events are TEXT like: “authentication failures”, “unknown password”, “unknown users”
4. TML does a direct STRING search for these terms in the sliding time windows

> **Note**
>
> THIS METHOD DOES NOT NEED A VECTOR DB or PRIVATEGPT only TML Processing– this makes TML method very light weight and fast

5. The Data TML searches, in real-time, are text files that are most likely log files
6. The Log files can be files on the file system that TML reads OR logs that are directly streamed to Kafka with LogStash, Splunk, etc..

> **Note**
>
> Note: This data does NOT have to be any specific format – it can be ANY text file streamed in raw form.

7. As TML process these data in sliding time windows (for details on sliding time window go here: :ref:`TML Performs Entity Level Machine Learning and Processing`) it is computing in real-time the following Scores:
    1. **AttackScore** is computing the occurrence of malicious events in past windows and how likely this is an attack
    2. **PatternScore** uses a pattern threshold set by the users (we use 10 for demo) it counts the occurrence of events (that user is searching for) in past windows
    3. **RTMS Score** simply combines the Attack and Pattern scores for an overall score.

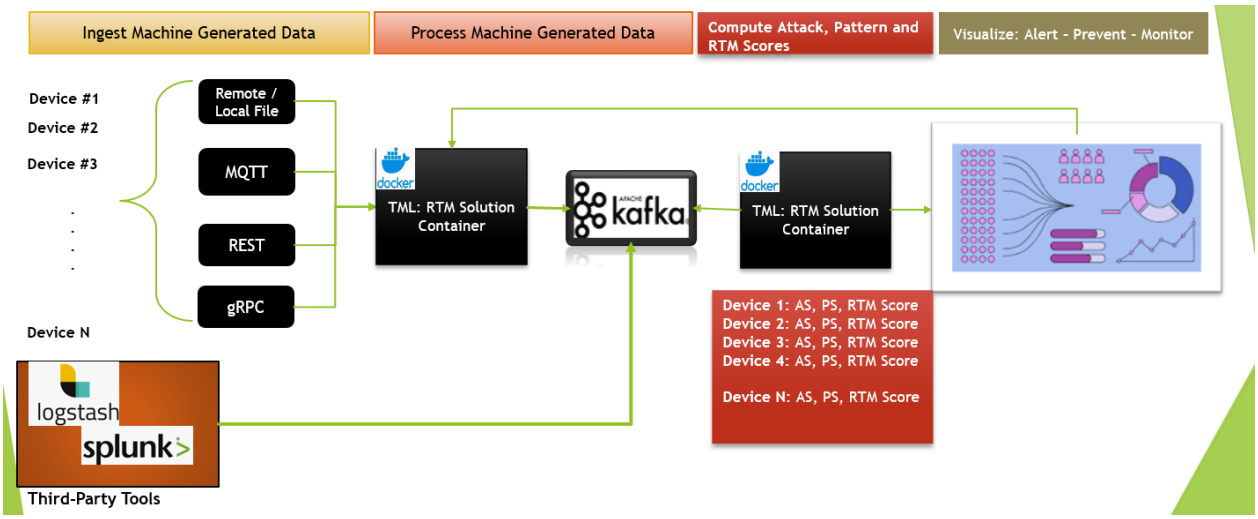
> **Note**
>
> NOTE: RTMS can exceed 1 because the pattern score can be greater than 1 i.e. events can exceed user pattern threshold. These score will obviously fluctuate in real-time time and alerts can be set up to trigger ALARMS of a cyber attack.

High-Level Reference Architecture

Important

Some important points to note about the architecture below:

1. The TML RTMS solution can analyse ANY log file and AS MANY as you like
2. You can use third-party tools like LogStash, Splunk etc. to stream directly to Apache Kafka
3. No format is needed for the log files - JUST STREAM IT TO KAFKA IN RAW FORM and tell TML in Step 4c what the Kafka Topic is in the **rtmsstream** JSON field.
4. You do NOT have to use Entities - you can immediately start analysing your log files for anomalies
5. If you are using entities - start processing in Step 4 and connect the entities by specifying the topic you stored entities (in Step 4) to **raw_data_topic** in Step 4c. Thats IT!
6. Build as many TML RTMS solutions you want with the TSS.



Enjoy the POWER of TML RTMS solution - that integrates real-time ML/AI entity level predictions with text files (like log files) to protect your global organizations - UNLIKE ANY OTHER TECHNOLOGY IN THE MARKET.

Maintaining Past Memory of Real-Time Data Without a Database: Demonstration

Important

It is important to note the following about the Attack and Pattern scores:

-
  - **Pattern Score** will look for all occurrences of search terms in each sliding time window. Meaning there may be MULTIPLE occurrences of search terms in the SAME sliding time window. This number can be greater than 1.

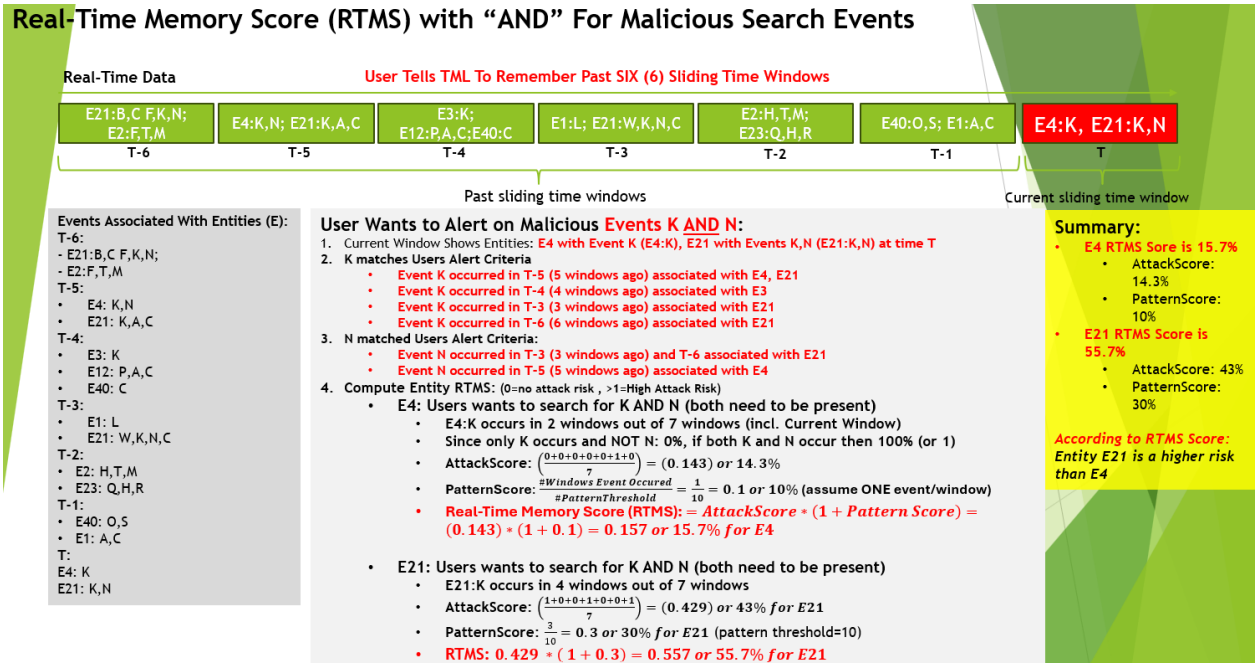
- **Pattern score** will check the number of windows GOING BACK as far as **RTMSMAXWINDOWS** parameter in STEP 1.

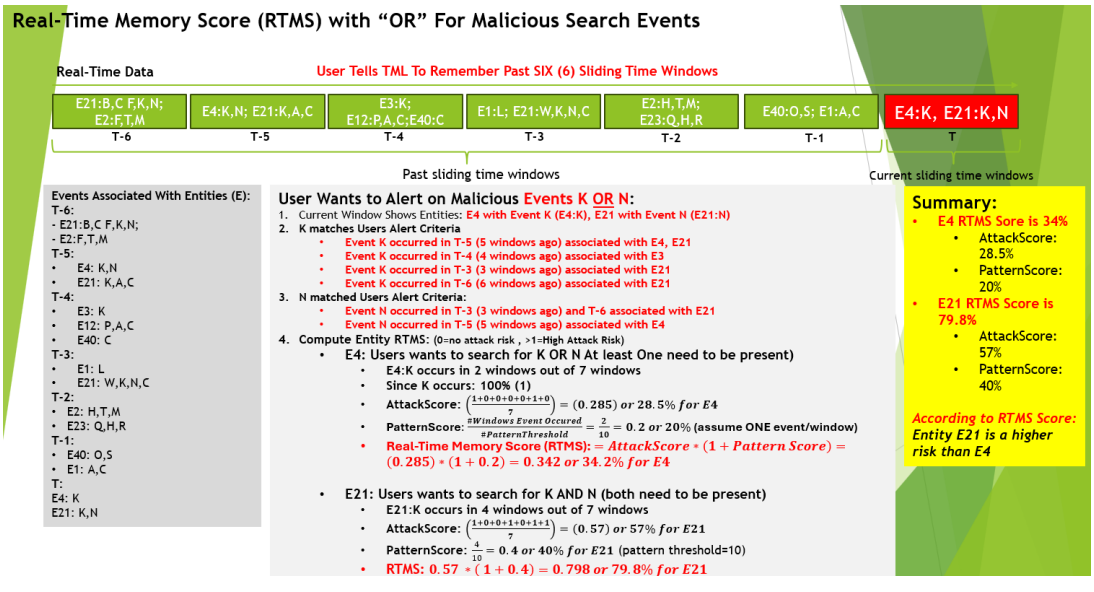
- So, if this number is 1000, TML will check all 1000 sliding time windows for the occurrence of the search terms.

- The **patternscorethreshold** can be set in Step 4c. This is the maximum occurrence of a pattern before raising an ALERT. This means the **Pattern Score MAY BE GREATER THAN 1**.

- **Attack Score** ONLY checks if window CONTAINS AN OCCURENCE of the search term. This number is either 1 or 0. The attack score is GOING BACK as far as the **rememberpastwindows** parameter in Step 4c.

- **User parameters:** rememberpastwindows and RTMSMAXWINDOWS are the core parameters that allows TML to **REMEMBER past events in real-time**.





TML Output of RTMS Scores

Output Explanation

Field	Explanation
hyperprediction	This is the RTMS Score
Entity	This is the entity being analysed. This can be anything you want.
GithubRemoteUrl	This is the GitHub Url for te RTMS solution output specific to your TML solution. All RTMS outputs are logged to Github automatically AND to Kafka topic. The log files are important for testing and validation.
Maintopic	This is the topic that holds the entity preprocessing from Step 4
Topicid	TML gives entity an internal integer ID. This entity (192.168.5.24) has an internal ID of 17. The format is the: **topicid<internal entity number>_<name of RTMS topic searched>**
Topic	The RTMS topic searched - containing TEXT
Type	Internal label
ProducerId	Internal label
TimeStamp	The time results were generated.
Unixtime	The Unixtime of TimeStamp
kafkakey	Unique key for this JSON in Kafka. If you want to audit these results these keys identify each message uniquely.
Preprocesstype	Type is **rtms**
UserSearchValues	These are the user search values. See tip below.
SearchTextFound	This is list of text that was found in the the Text files (log files) that contain your search terms. The list is truncated to 3000. But, this will give you a good indication of whats happening.
FinalAttackScore	The Final attack score
FinalPatternScore	The final pattern score
hash	Unique internal message hash
RTMSSCORE	The RTMS score.

NumAttackWindowsSearched	The number of attack windows that contain the search terms. This is upto **RememberPastWindows**
NumPatternWindowsSearched	This the number of windows that contain the search terms. Note: This in not restricted to RememberPastWindows, but upto **RTMSMAXWINDOWS** in Step 1 JSON field.
Filename	This is a file of these results saved to: **/rawdata/rtms** folder in the container.
TMLComment	This is the suggested auto-generated TML comment.
ActivityLevel	Based on the RTMS score this is what TML suggests. You can ofcourse use your own judgement.
RememberPastWindows	TML will remember the sliding windows upto this number.
PatternThreshold	This is a user threshold to alert when a pattern is equal to or greater than this number.
privateGPT_AI_response	This is the real-time response from the privateGPT container running LLM models from Deepseek or Mistral AI. See here for details.
prompt	The prompt provided by the user.
context	The context provided by the user.
pgptcontainer	The privateGPT container used from here.
pgpt_consumefrom	The kafka topic that Step 9 task will consume from.
pgpt_data_topic	The kafka topic Step 9 task will output results to.
contextwindowsize	The context window for the LLM. This is basically the maximum number of words LLM will process.
temperature	This is the LLM temperature parameter. Close to 0, the LLM will be more conservative in responses; close to 1, it will hallucinate.
pgptrollbackoffset	The amount of offsets to rollback the **pgpt_consumefrom** topic.

Tip

TML gives you are powerful capability to substiitute the **--entity--** placeholder with the **Entity** above. This makes it possible to search for each invidual entity in any log files.

> **Note**
>
> If you DO NOT want to use entities simply set the **'raw_data_topic'** to an empty string ('') in Step 4c. This will force TML to search ONLY the TEXT file topics for your search terms.

How TML Accomodates Evolving Threats

To detect evolving or changing cyber threats, TML can apply new user search terms in real-time by reading a local file containing search terms. For example, you can tell TML to read a file containing search terms that are updated every every 30 seconds, or every day, by user's internal process. TML can read this file, and update the search terms immediately to this list. This allows users to auto-update the threats that TML search for in real-time.

To update the search terms in real-time - you need to update two fields in Step 4c:

1. **localsearchtermfolder**: - Specify a folder of files containing search terms - each term must be on a new line - use comma to apply each folder to the rtmstream topic - Use @ =AND, | =OR to specify whether the terms in the file should be AND, OR

   For example, @mysearchfolder1,|mysearchfolder2, means all terms in mysearchfolder1 should be AND |mysearchfolder2, means all search terms should be OR'ed

   > **Important**
   >
   > **The search folders must exist in the local folder mapped to the /rawdata folder**. For example, if you specify mysearchfolder1, TML assumes the search files are in /rawdata/mysearchfolder1 (see here for details).

2. **localsearchtermfolderinterval**: - This is the number of seconds between reading the localsearchtermfolder.

   For example, if 30, the files will be read every 30 seconds - and searchterms will be updated

> **Tip**
>
> You can use RegEX statements in the search terms. This allows you to do build powerful RegEx expressions to filter log files.
>
> If using Regex expressions, you must prefix the expression by **rgx:**. For example, **rgx:p([a-z]+)ch**
>
> Regex expressions should be the only statement between ~, this is important if your Regex has a comma.
>
> With Regular expressions applied in real-time by TML RTMS, you have a MUCH WIDER search space to detect anomalous behaviours.

Regular Expressions Example

To check whether usernames DO NOT follow the proper format in the log files - you can use:

```
^[0-9A-Za-z]{6,16}$
```

- ^ indicates the start of a string, while $ indicates the end. Basically, this is ensuring that the entire string follows our rules, rather than only a subset of the string.
- [...] indicates a particular set of valid characters, otherwise called a character class; 0-9 allows numbers, A-Z allows uppercase letters, a-z allows lowercase. There are other indicators, and you can find a complete list in regex documentation.
- {6,16} indicates the allowed number of characters. If you just used {6}, you're testing for a length of exactly 6, while {6,} tests for minimum length.
- ^ denotes NOT or a negation of the results. For example, any characters NOT satisfying **[0-9A-Za-z]{6,16}**

To check whether passwords DO NOT follow the proper format (or any string) - you can use:

```
^(?=.*?[0-9])(?=.*?[A-Za-z]).{8,32}$
```

- (...) is a capture group. You can use them for capturing particular characters in specific orders.
- ?= is a positive lookahead. The search moves rightward through the string from the location in your regex you make this assertion in.
- . signifies any character is possible, while * means 'zero or more' of them.
- The extra question mark in ?=.*? makes the search lazy, which essentially means 'stop looking after the first time this requirement is met'.
- Translated into plain English, the first part of our statement ^(?=.*?[0-9]) means 'from the start of the string, find a number that is preceded by zero or more of any character'.
- Adding (?=.*?[A-Za-z]) means do the same for any letter, or 'from the start of the string, find a letter that is preceded by zero or more of any character'. This allows us to confirm the presence of a specified kind of character within the total set of what is allowed without regard to where it occurs in the string.
- The last part of our statement .{8,32}$ builds on our understanding of . usage. We don't want to limit what kinds of characters the actual password is allowed to be. In contrast, if limiting to letters and numbers only, you'd use [0-9A-Za-z]{8,32}$.

```
192\.168\.(224|225)\.\d{1,3}
```

- Values in yellow—192 and 168—are literal strings to be matched.
- Because the "." character is reserved in the regular expression language, to match a literal ".", you must escape it with a backslash . in your pattern definition.
- The 3rd octet needs to match either "224" or "225" and regex allows that with the "|" character. The OR pattern is bound in parentheses (). If there are more than two selections, | can be used to separate additional values: (224|225|230).
- The "d" represents a single digit (0-9). In the rex command example, above, I used a "+" to represent one or more of the preceding pattern. In this case, I am going to be more specific. Placing "1,3" in curly braces {1,3}, represents between 1 and 3 digits, since it was preceded by a "d".

```
(?<pass>[^&]+)
```

- ?<pass> specifies the name of the field that the captured value will be assigned to. In this case, the field name is "pass". This snippet in the regular expression matches anything that is not an ampersand.
- The square brackets [^&]+ signify a class, meaning anything within them will be matched; the carat symbol (in the context of a class) means negation. So, we're matching any single character that is not an ampersand.

- The plus sign extends that single character to one or more matches; this ensures that the expression stops when it gets to an ampersand, which would denote another value in the form_data.
- The parenthesis () signifies a capture group, while the value captured inside is assigned to the field name.

```
4[0-9]{15}
```

- This describes a string pattern starting with the digit 4 and having 15 digits in total that can have values from 0 to 9.

```
4[0-9]{12}(?:[0-9]{3})?
```

- This is relevant for strings that begin with the digit 4 and have 12 more digits with possible values from 0 to 9.
- After this sequence, a string can have or not have three more digits with values from 0 to 9. Thus, we can find not only credit card numbers with 16 digits but those with 13 digits as well.

```
\S+@\S+\.\S+
```

- A sequence of symbols without spaces before the @ symbol
- The @ symbol
- A sequence of symbols without spaces after the @ symbol
- A . symbol
- A sequence of symbols without spaces

```
(?:[a-z0-9!#$%&'*+\/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+\/=?^_`{|}~-]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-
\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|
[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])
```

This regular expression is also relevant for strings that have an email address format but includes additional bypasses, cycles, and filters. Here's a description of several constructions used in this RegEx:

- (?:) — Makes a grouping that cannot be referenced
- [a-z] — Sets possible options for characters
- ? — Makes the expression optional
- | — Sets alternation of two expressions on the left and right side of |
- * — Means that an expression matches zero or more of the preceding character

```
(([0-9]{1,4})\)([ .-]?)([0-9]{1,4})([ .-]?)([0-9]{1,4})
```

- It describes a line in the (####)%####%#### format, where #### could be a sequence from one to four digits, and the % symbol stands for one of three possible separation symbols: space, dot/period, or hyphen.

```
[a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z]+-?[a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z]*
```

- A RegEx to search for matches with one of the symbols listed in square brackets and divided by the | symbol that represents an alternative for matching the part to the left and the part to the right of the | symbol. An alternative will include options from a to z.

A RegEx that points directly to a capture group:

```
\w+-?\w*
```

- RegEx examples that work with capture groups — mechanisms that allow you to highlight and save matching text for further use. When a RegEx matches the text, any content within a capture group is saved in temporary variables. You can use those variables later in code.

Link to a capture group (marked as w):

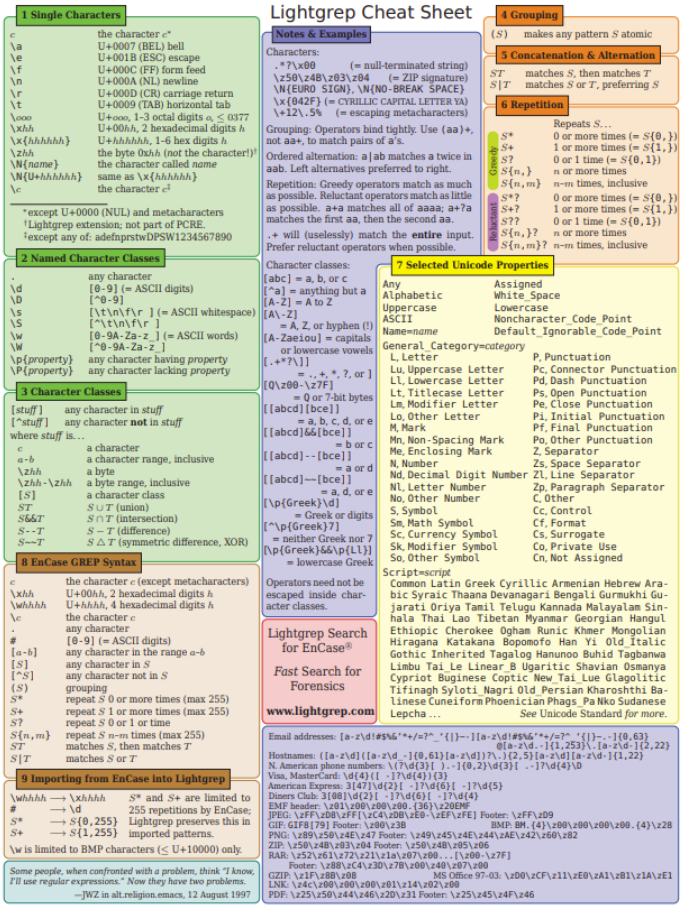
```
\w+-?1*
```

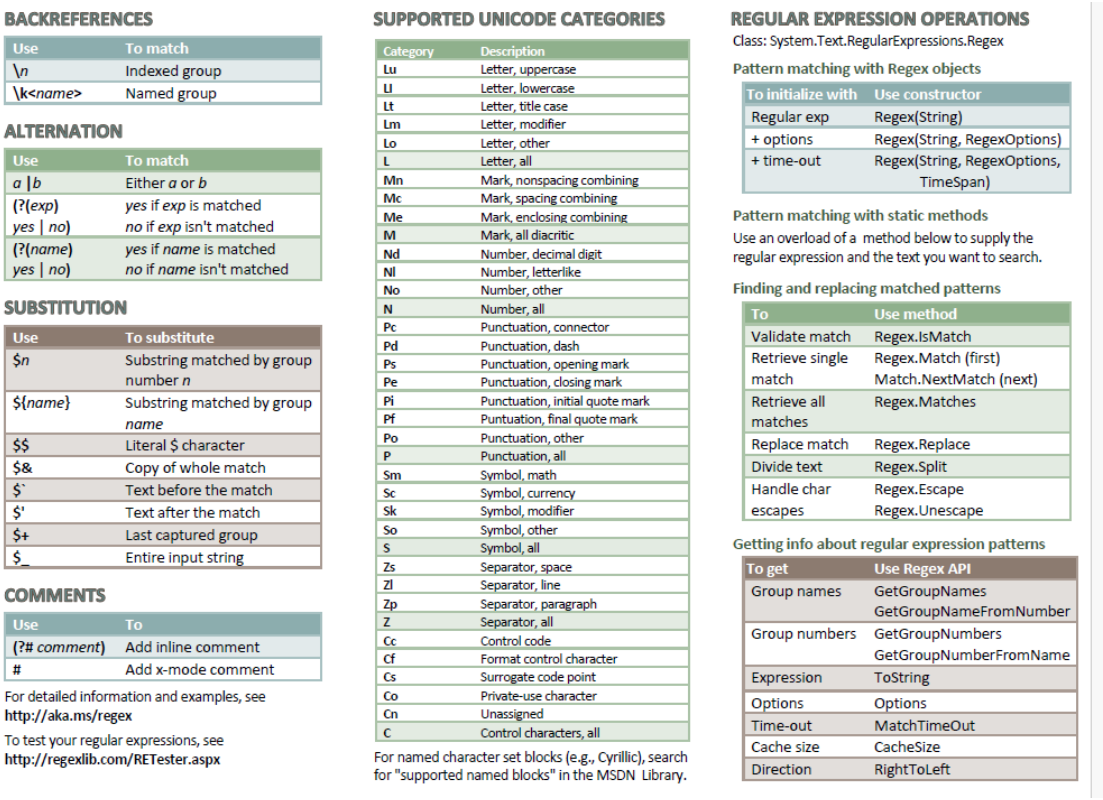
- A RegEx that includes a link to capture group ■1:

RegEx Cheat Sheets

Here are RegEx cheat sheets that may help to write more advanced regular for powerful searching of text files.

> ### Tip
>
> To test your RegEx you can use this onine tool: regexr





BACKREFERENCES

Use	To match
\n	Indexed group
\k<name>	Named group

ALTERNATION

Use	To match
a | b	Either *a* or *b*
(?(exp) yes | no)	*yes* if *exp* is matched *no* if *exp* isn't matched
(?(name) yes | no)	*yes* if *name* is matched *no* if *name* isn't matched

SUBSTITUTION

Use	To substitute
$n	Substring matched by group number *n*
${name}	Substring matched by group *name*
$$	Literal $ character
$&	Copy of whole match
$`	Text before the match
$'	Text after the match
$+	Last captured group
$_	Entire input string

COMMENTS

Use	To
(?# *comment*)	Add inline comment
#	Add x-mode comment

For detailed information and examples, see **http://aka.ms/regex**

To test your regular expressions, see **http://regexlib.com/RETester.aspx**

SUPPORTED UNICODE CATEGORIES

Category	Description
Lu	Letter, uppercase
Ll	Letter, lowercase
Lt	Letter, title case
Lm	Letter, modifier
Lo	Letter, other
L	Letter, all
Mn	Mark, nonspacing combining
Mc	Mark, spacing combining
Me	Mark, enclosing combining
M	Mark, all diacritic
Nd	Number, decimal digit
Nl	Number, letterlike
No	Number, other
N	Number, all
Pc	Punctuation, connector
Pd	Punctuation, dash
Ps	Punctuation, opening mark
Pe	Punctuation, closing mark
Pi	Punctuation, initial quote mark
Pf	Puntuation, final quote mark
Po	Punctuation, other
P	Punctuation, all
Sm	Symbol, math
Sc	Symbol, currency
Sk	Symbol, modifier
So	Symbol, other
S	Symbol, all
Zs	Separator, space
Zl	Separator, line
Zp	Separator, paragraph
Z	Separator, all
Cc	Control code
Cf	Format control character
Cs	Surrogate code point
Co	Private-use character
Cn	Unassigned
C	Control characters, all

For named character set blocks (e.g., Cyrillic), search for "supported named blocks" in the MSDN Library.

REGULAR EXPRESSION OPERATIONS

Class: System.Text.RegularExpressions.Regex

Pattern matching with Regex objects

To initialize with	Use constructor
Regular exp	Regex(String)
+ options	Regex(String, RegexOptions)
+ time-out	Regex(String, RegexOptions, TimeSpan)

Pattern matching with static methods

Use an overload of a method below to supply the regular expression and the text you want to search.

Finding and replacing matched patterns

To	Use method
Validate match	Regex.IsMatch
Retrieve single match	Regex.Match (first) Match.NextMatch (next)
Retrieve all matches	Regex.Matches
Replace match	Regex.Replace
Divide text	Regex.Split
Handle char escapes	Regex.Escape Regex.Unescape

Getting info about regular expression patterns

To get	Use Regex API
Group names	GetGroupNames GetGroupNameFromNumber
Group numbers	GetGroupNumbers GetGroupNumberFromName
Expression	ToString
Options	Options
Time-out	MatchTimeOut
Cache size	CacheSize
Direction	RightToLeft

TML Real-Time Message Scoring (RTMS) vs AI RAG

TML using real-time data is similar to RAG but different in other ways.

Attribute	TML RTMS	AI RAG
Speed	TML RTMS is much faster than RAG because TML RTMS does NOT use vector DB. All TML RTMS processing is real-time.	AI RAG require vector DB for search. Real-time is still difficult with RAG.
Prompting	TML users direct text based search	With RAG you can use prompt
Combining ML and AI in Real-Time	With TML you can combine TML output for each entity and cross-reference with TEXT files	This is not currently possible with RAG
Scalability	TML RTMS scales with Kubernetes to process unlimited documents at a very low cost	Scaling RAG models is difficult and can be costly

How RTMS Integrates with MITRE ATT&CK Framework

The MITRE ATT&CK framework for the Entreprise, Mobile and ICS (Industrial Control Systems) is used by 80% of global enterprises. **TML/RTMS is fully integrated with MITRE ATT&CK framework for Entreprise, Mobile and ICS** level threat detection and classification for improved threat insights to help in further fortifying organizations' threat and security technologies and processes.

Important

A key challenge by majority of organizations is the difficulty and inability to map events to specifc MITRE ATT&CK tactics and techniques. Specifically, in a research report (source)

> **"about 45 percent of survey respondents said their greatest challenge is the framework's inoperability with their security products and 43 percent said they find it difficult to map event-specific data to tactics and techniques."**

RTMS eliminates this challenge of mapping events to MITRE ATT&CK tactics and techniques by automating the classifications in real-time using our AI containers.

When RTMS searches logs for suspicious activity the messages it finds are sent to our privateGPT AI container, the AI determines a mitigation plan, and MITRE ATT&CK classification of the messages. The Attack, Pattern and RTMS scores are provided for the "grouped" MITRE ATT&CK tactics and techniques. See figure below of Mitre tactics and techniques.

The RTMS scoring and classification of messages, in accordance with MITRE ATT&CK framework, can offer organizations around the world invaluable insights into their organizations that can help them to:

- **determine gaps in deployed security solutions in their enterprise,**
- **for security policy implementation**
- **for threat modeling**

TML/RTMS with MITRE ATT&CK integration is a truly unique and powerful technological approach, in real-time, to give organizations **faster identifications of developing threats, but also offering invaluable guidance to fortify their security processes and technologies** that aligns with a global standard like MITRE ATT&CK.

> **Tip**
>
> The MITRE ATT&CK JSON used by RTMS for cross-referencing with AI output is here

> **Note**
>
> An example of this classification is here. Look at the last JSON fields:
>
> - **"tactic":** "Initial_Access",
> - **"technique":** "Phishing"
>
> **The above are MITRE ATT&CK tactic and technique, that are automatically classified by RTMS AI agent for all messages.** RTMS further groups on these tactic and techniques to compute the **grouped ATTACK, PATTERN and RTMS scores**. This is a powerful approach to further help organization's fortify their security processes and technologies to dramatically reduce the threat of cyber attacks.

Integrating RTMS with Real-Time AI Using PrivateGPT Containers and MITRE ATT&CK Classification

Below is output from Step 9 task, that takes messages in the "SearchTextFound", and send it to the PrivateGPT special containers

By using AI, users can prompt for any anomalies and resolutions suggested by AI. This is all done in real-time using local privateGPT containers, that makes this integration 100% FREE, SECURE and SCALABLE.

RTMS Grouped MITRE ATT&CK JSON

RTMS automatically classified the messages in accordance with MITRE ATT&CK classification matrix:

- **TACTIC**: Credential_Access-Initial_Access
- **TECHNIQUE**: Brute_Force

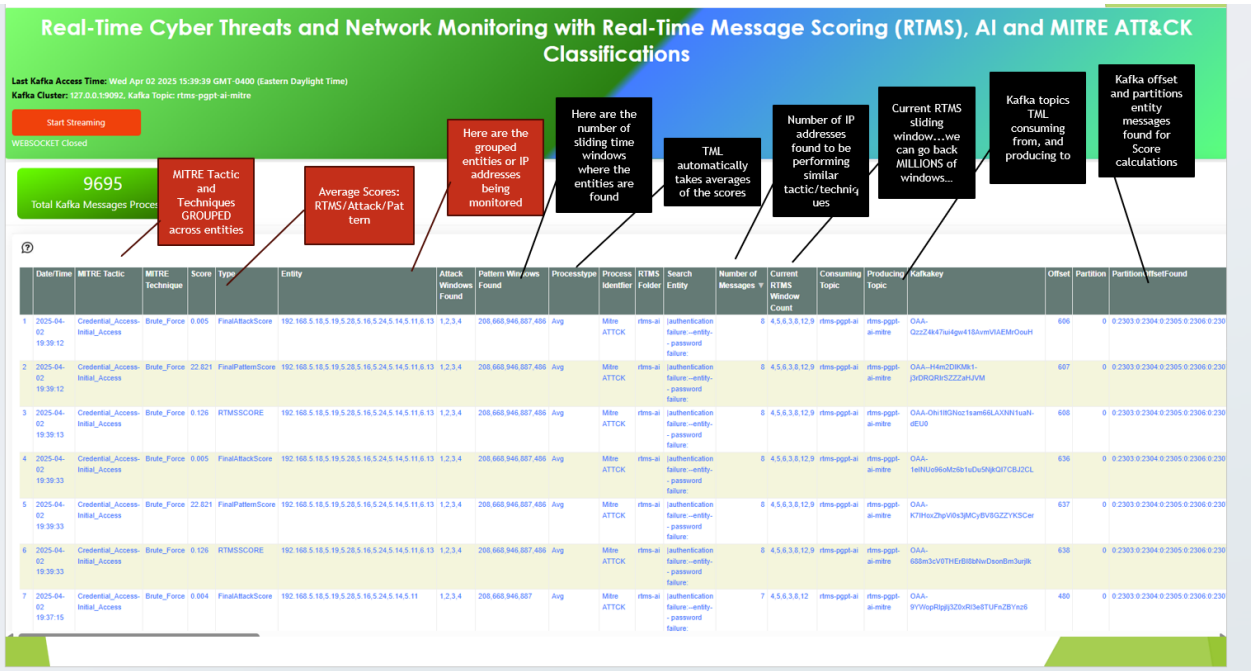
Note in the below JSON from RTMS:

1. **RTMS has found three IP addreses:** 5.10,5.14,6.100 (add prefix 192.168 in front of 5.10, 5.14, 6.100)
2. These three entities are using a **Brute Force** attack
3. The **Maintype=RTMSScore** and **Mainvalue=0.258** - this is the average score for the three entities. See here from more details.
4. The source Kafka topic that RTMS is consuming from is: **rtms-pgpt-ai**
5. The sink Kafka topic that RTMS produces the results to is: **rtms-pgpt-ai-mitre**
6. The messages that are found are in the **PartitionOffsetFound**
7. The **hyperprediction**: "0.258" is the same as Mainvalue
8. **NumAttackWindowsFound**: "5,4,3", are the number of sliding time windows RTMS is searching

9. **NumPatternWindowsFound**: "988,367", are the number of occurences of the messages that match the search terms
10. **SearchEntity**: "|authentication failure:--entity-- password failure:,Failed password for root:", these are the search terms
11. **rtmsfolder**: "rtms2", this is a local folder as well as Github folder and Kafka topic, where results are saved
12. **CurrentRTMSMAXWINDOW**: "14,13,18,19", this the current RTMS pattern window.

RTMS MITRE ATT&CK Dashboard

This is a simple out-of-the-box dashboard to give users a quick view in to RTMS output grouped entity classifications in accordance with ATT&CK.



RTMS Solution: Steps to Run It Yourself

Below shows the RTMS solutions architecture. We can now discuss in details how users and can build and run this RTMS solution.

> ### Note
>
> The entire RTMS solution is built using the TSS using the solution dag:
>
> - solution_preprocessing_ai_dag-cybersecurityrtms-3f10

> ### Tip
>
> There are TWO ways to run this RTMS solution for yourself:

1. Copy to your repo and run in your TSS environment. To do this you MUST:

    - Follow these Steps to copy projects from others repo and copy and paste the below in your TSS **CREATETMLPROJECT.txt:**

      ```
      https://github.com/smaurice101/raspberrypitss,cybersecurityrtms-3f10
      ```

2. Run the RTMS Docker container: This is the **EASIEST** and FASTEST way for anyone to run this solution in test or production. Note you can also scale this solution with Kubernetes - all YAML scripts are auto generated for you here. - Go here to het the Docker Run command for RTMS - **You MUST change the following environmental variables in the Docker Run Command:**

    1. Change **--env GITUSERNAME=** <Enter Github Username> (For quick testing use: **tsstmldemo**)
    2. Change **--env GITREPOURL=** <Enter Github Repo URL> (For quick testing use: **https://github.com/tsstmldemo/tsstmldemo**)
    3. Change **-v /your_localmachine/foldername:/rawdata:z** - create a folder in your local machine and mapped it here. For example, if you are in Linux, create a folder /rtms/rawdata, then the volume mapping is **-v /rtms/rawdata/foldername:/rawdata:z** - Create TWO folders and store your Log files in there: (Get sample log files from here if you like)

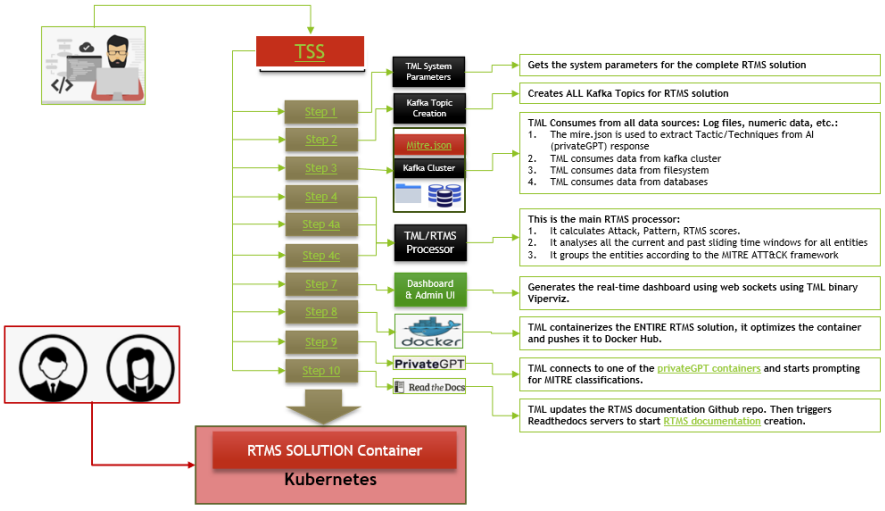
        - /rtms/rawdata/mylogs
        - /rtms/rawdata/mylogs2

        - Create TWO folders and store your search terms in there: (Get sample search files from here if you like) - /rtms/rawdata/mysearchfile1 - /rtms/rawdata/mysearchfile1
    4. Change **--env READTHEDOCS='<Enter Readthedocs token>'** ((For quick testing use: **'aefa71df39ad764ac2785b3167b77e8c1d7c553a'**)

        - This will create new solution documentation that can be found here: https://cybersecurityrtms-aefa-ai.readthedocs.io/en/latest/

Now that you know the basic setup to run the RTMS solution - just change it with your own files and folders...as you want!

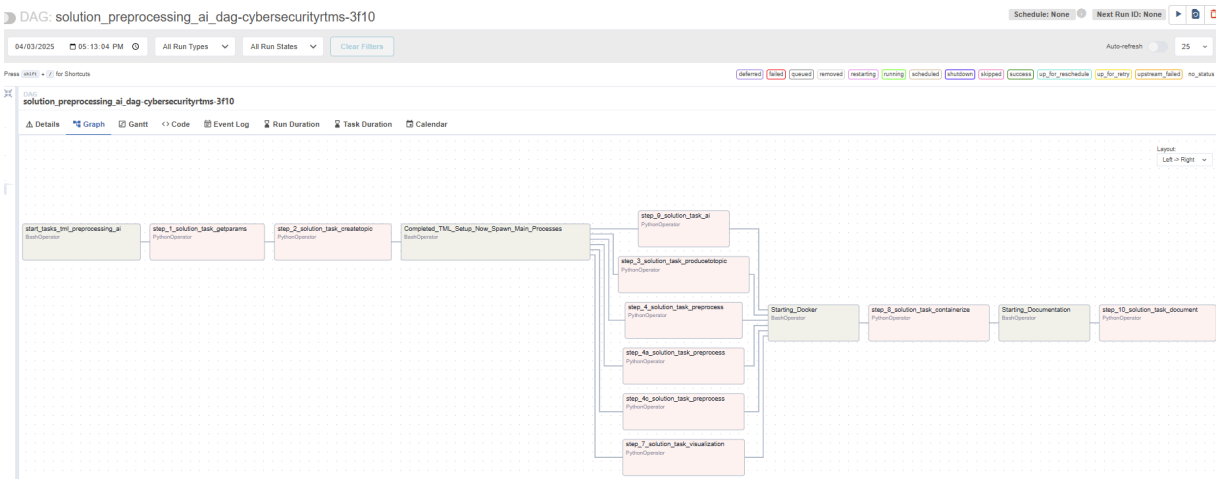
RTMS Solution Architecture



Here is a description of the solution architecture and RTMS solution development using the TSS:

TSS Steps	Description
Step 1	Gets all the RTMS TML system parameters and setups up the components of the entire solution.
Step 2	Creates all the Kafka topics needed for the solution to run.
Step 3	TML Consumes from all data sources: Log files, numeric data, etc.: - The mire.json is used to extract Tactic/Techniques from AI (privateGPT) response - TML consumes data from kafka cluster - TML consumes data from filesystem - TML consumes data from databases
Step 4	**This is the main Entity processor:** - It processes entity data that is then by the RTMS Step 4c
Step 4a	It groups the entities according to the MITRE ATT&CK framework
Step 4c	**This is the main RTMS processor:** - It calculates Attack, Pattern, RTMS scores - It analyses all the current and past sliding time windows for all entities
Step 7	This generates the dashboard and connects to the Viperviz binary.
Step 8	This step containerizes the entire RTMS solution, while optimizig it, and pushes it to Docker hub
Step 9	This connects to the privateGPT containers and RTMS sends the filtered messages to privateGPT for MITRE ATT&CK classifications. The mitre.json is used to extract the MITRE classification from the privateGPT responses. **Note, the mitre.json should be saved in your local folder mapped to /rawdata**
Step 10	This updates the RTMS documentation in the Github repo and triggers Readthedocs site to start generating the RTMS documentation. To find your documentation see here

RTMS in Graphical View of TSS Steps



Summary

- This has shown how TML implements real-time memory using sliding time windows for every entity
- For every entity: It quantified this memory in Three (3) scores:
    - **AttackScore (AS)**
    - **PatternScore (PS)**
    - **Real-Time Memory Score (RTMS)**

Important

The power of TML maintaining memory and computing the 3 scores is to capture **attacker behaviours that try to EVADE detection algorithms**. While the AttackScore may not indicate an attack, it may be picked up as a pattern in the PatternScore.

Also, TML/RTMS solution will automatically classify (using AI) messages in accordance with the Mitre Att&ck framework matrix for the Enterprise.

The Mitre Att&ck classifications could provide tremendous help for Enterprises to:

- determine gaps in deployed security solutions in their enterprise,
- improve security policy implementation
- improve threat modeling.

- Within Cyber security context: The power of this method using sliding time windows is the ability to detect hacking attempts that are deliberate in evading “detection algorithms” from common industry tools
- TML approach and method is a fast, low cost, method of maintaining memory of events as they occur or have occurred in the past that may be “occasional” events and VERY HARD TO DETECT from other commercial tools
- The simplicity of maintaining and incorporating memory by TML for EVERY ENTITY- without the need to vector DB – makes it lightweight, fast, and able to run WITHOUT the need for GPU (only CPU is needed)
- As attackers get more sophisticated in evading commercial algorithms’ detection methods – TML memory offers a continuous awareness of events that are current and have occurred in the past and correlates and quantifies these in a Score for triggering alerts and alarms immediately

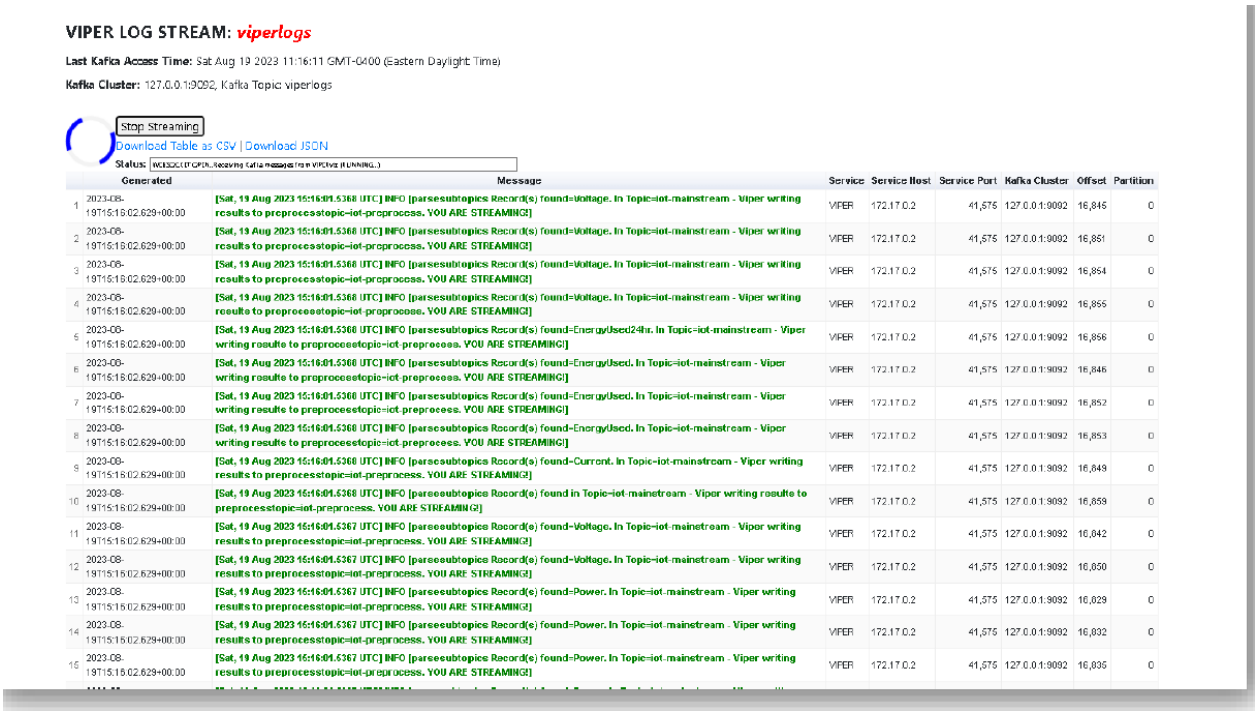
TML Real-Time Logs

Real-Time logs can be viewed in a browser by entering: http://localhost:9005/viperlogs.html?topic=viperlogs&append=0

> Note
>
> To view logs you must have Viperviz binary running. The Viperviz (:ref:`TML Solution Components`) binary will read the viperlogs Kafka topic and stream the log data to your browser.
>
> The above URL assumes Viperviz is listening on Port 9005. This port can be set in STEP 7 Visualization Dag: Refer to :ref:`STEP 7: Real-Time Visualization: tml-system-step-7-kafka-visualization-dag`

Log Example

Here is an example of logs from a TML Solution. GREEN items indicate Success, RED items indicate Errors.



VIPER LOG STREAM: *viperlogs*

Last Kafka Access Time: Sat Aug 19 2023 11:16:11 GMT-0400 (Eastern Daylight Time)

Kafka Cluster: 127.0.0.1:9092, Kafka Topic viperlogs

Stop Streaming

Download Table as CSV | Download JSON

Status: WEBSOCKET OPEN, Receiving Kafka messages from VIPERviz (RUNNING..)

	Generated	Message	Service	Service Host	Service Port	Kafka Cluster	Offset	Partition
1	2023-08-19T15:16:02.629+00:00	[Sat, 19 Aug 2023 15:16:01.5368 UTC] INFO [parsesubtopics Record(s) found=Voltage. In Topic=iot-mainstream - Viper writing results to preprocesstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,845	0
2	2023-08-19T15:16:02.629+00:00	[Sat, 19 Aug 2023 15:16:01.5368 UTC] INFO [parsesubtopics Record(s) found=Voltage. In Topic=iot-mainstream - Viper writing results to preprocesstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,851	0
3	2023-08-19T15:16:02.629+00:00	[Sat, 19 Aug 2023 15:16:01.5368 UTC] INFO [parsesubtopics Record(s) found=Voltage. In Topic=iot-mainstream - Viper writing results to preprocesstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,854	0
4	2023-08-19T15:16:02.629+00:00	[Sat, 19 Aug 2023 15:16:01.5368 UTC] INFO [parsesubtopics Record(s) found=Voltage. In Topic=iot-mainstream - Viper writing results to preprocesstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,855	0
5	2023-08-19T15:16:02.629+00:00	[Sat, 19 Aug 2023 15:16:01.5368 UTC] INFO [parsesubtopics Record(s) found=EnergyUsed24hr. In Topic=iot-mainstream - Viper writing results to preprocesstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,856	0
6	2023-08-19T15:16:02.629+00:00	[Sat, 19 Aug 2023 15:16:01.5368 UTC] INFO [parsesubtopics Record(s) found=EnergyUsed. In Topic=iot-mainstream - Viper writing results to preprocesstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,846	0
7	2023-08-19T15:16:02.629+00:00	[Sat, 19 Aug 2023 15:16:01.5368 UTC] INFO [parsesubtopics Record(s) found=EnergyUsed. In Topic=iot-mainstream - Viper writing results to preprocesstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,852	0
8	2023-08-19T15:16:02.629+00:00	[Sat, 19 Aug 2023 15:16:01.5368 UTC] INFO [parsesubtopics Record(s) found=EnergyUsed. In Topic=iot-mainstream - Viper writing results to preprocesstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,853	0
9	2023-08-19T15:16:02.629+00:00	[Sat, 19 Aug 2023 15:16:01.5368 UTC] INFO [parsesubtopics Record(s) found=Current. In Topic=iot-mainstream - Viper writing results to preprocesstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,849	0
10	2023-08-19T15:16:02.629+00:00	[Sat, 19 Aug 2023 15:16:01.5368 UTC] INFO [parsesubtopics Record(s) found in Topic=iot-mainstream - Viper writing results to preprocesstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,859	0
11	2023-08-19T15:16:02.629+00:00	[Sat, 19 Aug 2023 15:16:01.5367 UTC] INFO [parsesubtopics Record(s) found=Voltage. In Topic=iot-mainstream - Viper writing results to preprocesstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,842	0
12	2023-08-19T15:16:02.629+00:00	[Sat, 19 Aug 2023 15:16:01.5367 UTC] INFO [parsesubtopics Record(s) found=Voltage. In Topic=iot-mainstream - Viper writing results to preprocesstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,850	0
13	2023-08-19T15:16:02.629+00:00	[Sat, 19 Aug 2023 15:16:01.5367 UTC] INFO [parsesubtopics Record(s) found=Power. In Topic=iot-mainstream - Viper writing results to preprocesstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,829	0
14	2023-08-19T15:16:02.629+00:00	[Sat, 19 Aug 2023 15:16:01.5367 UTC] INFO [parsesubtopics Record(s) found=Power. In Topic=iot-mainstream - Viper writing results to preprocesstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,832	0
15	2023-08-19T15:16:02.629+00:00	[Sat, 19 Aug 2023 15:16:01.5367 UTC] INFO [parsesubtopics Record(s) found=Power. In Topic=iot-mainstream - Viper writing results to preprocesstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,836	0

TML Solution Components

Below describes the entire TML technology solution stack.

1. TML Components: Three Binaries

TML is comprised of 3 binaries written in Go: found on Github

a. *Viper* - source - sink binary for Apache Kafka - runs on MAC/Windows/Linux

b. *HPDE* - AutoML binary for real-time data - runs on MAC/Windows/Linux

c. *Viperviz* - Visualization binary for real-time dashboards - runs on MAC/Windows/Linux

> Important
>
> These 3 binaries perform all functions in every TML solution. These binaries can be seen as microservices that can be instantiated any number of times to scale your solution for **unlimited data processing**.
>
> **These binaries are critical for TML solutions and are the "secret sauce" inside TML.**

Binary	**Description**
Viper	This is the CORE binary that performs all the major TML functions of processing and machine learning. This binary acts like a microservice that can be instantiated any number of times to process large amounts of real-time data. This binary is compatible with REST API. TML solutions (using Python) connect to this binary and instruct it to stream data to Kafka, preprocess data, and develop training data sets for machine learning.
HPDE	Hyper-prediction technology performs all machine learning functions. Viper connects to HPDE, using REST, and instructs it to perform machine learning. By off-loading this function to HPDE, TML can perform very fast machine learning (in few seconds) for each entity and sliding time window. Refer to :ref:`TML Performs Entity Level Machine Learning and Processing` and :ref:`Entity Based Machine Learning By TML`

Viperviz	This binary performs real-time streaming visualization using websockets. This is a very powerful binary because it uses the underlying network to streaming data to a client browser for fast, and very cost-effective, visualization of real-time TML solution outputs. This means users do NOT need a third-party visualization tool like Tableau or PowerBI. Users can create amazing real-time dashboards quickly and cheaply. Refer to example dashboards here :ref:`TML Real-Time Dashboards`

2. TML Component: One Core Python Library

MAADSTML Python Library: https://pypi.org/project/maadstml/

MAADSTML Python library : API to build TML solutions that connect to the Viper binary. Refer to the MAADSTML API here :ref:`MAADSTML Python Library API`

> Important
>
> All TML solutions use the the MAADSTML python library. **This is a critical library, and controls the 3 binaries.**

3. TML Component: Apache Kafka

TML integrates with Apache Kafka - on-premise or in the cloud.

> Attention!
>
> TML binaries are integrated with Apache Kafka. TML solutions can be run On-PREMISE using Open Source Kafka or in the CLOUD using AWS MSK or Confluent Cloud.

4. TML Component: Docker Containers

All TML solutions are containerized with docker for production deployments.

5. TML Component: Kubernetes

All TML solution containers scale with Kubernetes. This allows companies to build fast, scalable, real-time solutions.

6. TML Component: PrivateGPT for Generate AI

TML uses PrivateGPT for fast, real-time, AI. The container is here Docker PrivateGPT

Refer to :ref:`TML and Generative AI` for more details.

7. TML Component: TML Solution Studio Container

For convenience, TML solution can be easily built using the TML Solution Studio container. Refer to :ref:`TML Solution Studio (TSS) Container` for further details.

How The TML Components Are Integrated

TML solutions are developed using the MAADSTML Python library that connects to the TML Binaries, using REST API, for streaming real-time data to Apache Kafka, processing data in Kafka, and performing machine learning. Once the TML solutions are built, they are containerized with Docker and scaled with Kubernetes.

> Attention!
>
> TML performs **in-memory processing** of real-time data and **does NOT require an external database** - ONLY KAFKA is needed. This results in dramatic cost- savings for storage, compute and network data transfers.
>
> TML **does NOT perform SQL queries**, it performs :ref:`JSON PROCESSING`. This results in much faster, and much cheaper processing of real-time data.

Copying TML Project(s) From Others Git Repo

With TSS you have the ability to copy TML projects from others Git repo. This is convient if you want to share your TML projects with others.

> Tip
>
> Also see here :ref:`Project Action Commands Summary` on projects in your own repo.

The process of copying from others Git repo is simple:

1. Goto the TSS and select from the top menu item: **Admin -> Dags Code Editor**
2. Navigate to the File: **root/tml-airflow/dags/tml-solutions/CREATETMLPROJECT.txt**

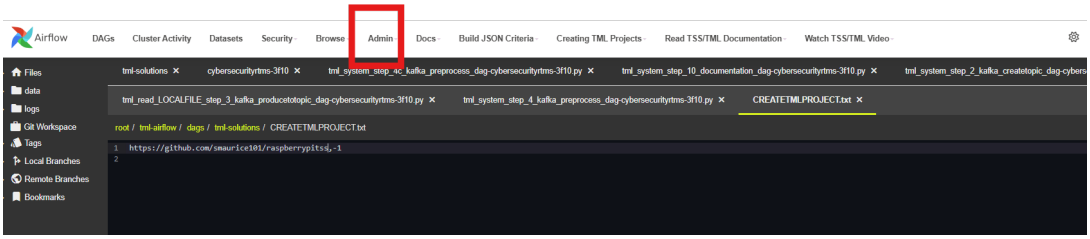
Once you are in the **CREATETMLPROJECT.txt** file - you can do the following scenarios:

You Want to Copy ALL TML Projects From Another Users Git Repo

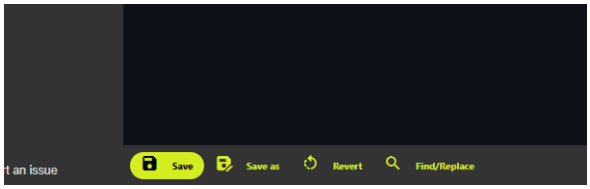
To do this:

- You need the other persons Git repo
- Enter **-1** after the repo, using comma as the separator.
- For example: - Say you want to copy all TML Project from: **https://github.com/smaurice101/raspberrypitss** - Then Enter:

```
https://github.com/smaurice101/raspberrypitss,-1
```



- Then press Save:



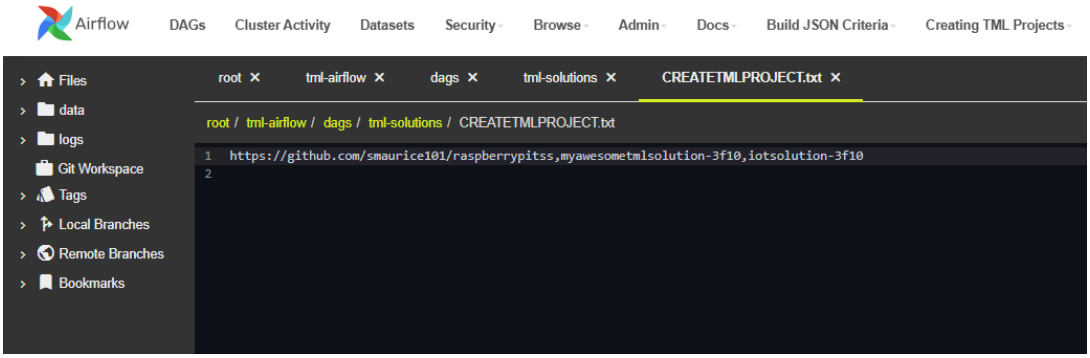
TSS will pull all TML projects from the other user's Git Repo and commit them to your repo. TSS will take care of file changes. Within few seconds, on your Git Repo you will see ALL of the other user's TML projects.

You Want to Copy SPECIFIC TML Projects From Another Users Git Repo

To do this:

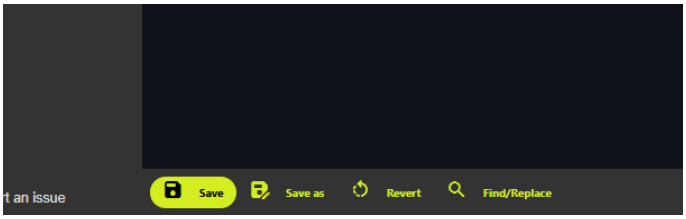
- You need the other persons Git repo
- Enter **the specific projects**, using comma as the separator.
- For example: - Say you want to copy ONLY **myawesometmlsolution-3f10** and **iotsolution-3f10** TML Projects from: **https://github.com/smaurice101/raspberrypitss** - Then Enter:

```
https://github.com/smaurice101/raspberrypitss,myawesometmlsolution-3f10,iotsolution-3f10
```



Here you are copying TWO projects: myawesometmlsolution-3f10,iotsolution-3f10

- Then press Save:



TSS will pull ONLY those TML projects from the other user's Git Repo and commit them to your repo. TSS will take care of file changes. Within few seconds, on your Git Repo you will see those specific projects of the other user's TML projects.

Thats it!

MAADS-VIPER Environmental Variable Configuration (Viper.env)

v.5.5.90+

> Note
>
> The setting of these environmental variables are automatically set for you in the TML Solution Studio DAGs. They are presented here for informational purposes.
>
> Refer to :ref:`Pre-Written 10 Apache Airflow DAGs To Speed Up TML Solution Builds`

This guide will provide common setup instructions for new users who want to run VIPER in their environment(s). For any questions, users are encouraged to email support@otics.ca .

1. **SETUP Instructions: [Watch the YouTube video](https://youtu.be/b1fuleC7d-8) or follow instructions below.**
    A. **For actual (non-Demo) use you will need:**
        1. **ADMIN.tok (available in ZIP for Viper on Github: https://github.com/smaurice101/transactionalmachinelearning)**
            1. This allows admin users to create topics, activate/deactivate topics, produce to topics
2. **Store all of the above files in the same directory you use to run VIPER**
    1. VIPER will automatically create necessary directories in that folder
    2. **Note:** For Linux users File/Folder permissions may need to be adjusted for VIPER 0644 is usually fine
3. **Start VIPER**
    1. By default, VIPER listens on “Localhost” port=8000
    2. You can easily adjust this to whatever host/port you want by typing: [Viper Executable] [host] [port]
4. **On Startup VIPER will check for:**
    1. Valid Tokens
    2. VIPER.ENV file

VIPER.ENV Configurations

1. **With SSL/TLS enabled**
    1. **If you have enabled SSL/TLS on Kafka brokers then you need to specify additional fields in the configuration file – for example purposes .PEM files are added to the configuration keys, but you can specify folder/file names as you wish:**
        1. SSL_CLIENT_CERT_FILE=client.cer.pem
        2. SSL_CLIENT_KEY_FILE=client.key.pem
        3. SSL_SERVER_CERT_FILE=server.cer.pem

1. **No SSL/TLS Security:**
    1. **Assuming you have Kafka/Zookeeper running on a broker, simply fill in the following information in the configuration file:**
        1. KAFKA_ADVERTISED_HOST_NAME=kafka
        2. KAFKA_ZOOKEEPER_CONNECT=
        3. KAFKA_CONNECT_BOOTSTRAP_SERVERS=localhost:9092
    2. **With HPDE:**
        1. HPDE_SERVER=
        2. HPDE_PORT=
    3. **With Confluent Cloud or AWS MSK Access (If NOT using Kafka Cloud these MUST be left blank):**
        1. CLOUD_USERNAME={API KEY}
        2. CLOUD_PASSWORD={API SECRET}
    4.
        a. SSL_CLIENT_CERT_FILE=
        b. SSL_CLIENT_KEY_FILE=
        c. SSL_SERVER_CERT_FILE=

Note: First time the plain text values need to be entered, on start VIPER will hide these values. You can update them with plain text again if you change the key/secret then simply restart VIPER to hide the updated values again.

Configuration Parameter	Description
BATCHTHREADS	This is used in batch functions like "viperpreprocessbatch" and indicates how many topicids to preprocess concurrently. For example, if BATCHTHREADS=5, and you are preprocessing 10 topicids in batch, then 5 will be preprocessed concurrently at a time.
CLOUD_USERNAME	SASL_PLAIN username to connect to Confluent Cloud
CLOUD_PASSWORD=	SASL_PLAIN password to connect to Confluent Cloud
COMPRESSIONTYPE	You can force the producer to compress data. You can set this to: NONE, SNAPPY, GZIP, LZ4, default is NONE.
COMPANYNAME	Specify company name. This is used when sending emails from AiMS dashboard.
DATARETENTIONINMINUTES	Specify how long you want to retain the data in Topics, in minutes. This is based on your data retention policy. For example, if DATARETENTIONINMINUTES=30, committed offsets will be deleted/compacted after 30 minutes. IF DATARETENTIONINMINUTES=0 or empty data is retained forever.

FROMADDR	From address to put in the emails.
HPDE_IP	HPDE (Hyper-Predictions for Edge Devices) is another product required for **Real-Time Machine Learning.** Specify the host where it is installed.
HPDE_PORT	HPDE listening port. Specify port. If you specifying port range use “startport:endport”, where start port and end port are numbers
KAFKA_ADVERTISED_HOST_NAME	Advertised host name in Kafka server properties
KAFKA_ZOOKEEPER_CONNECT	Zookeeper host name and port
KAFKA_CONNECT_BOOTSTRAP_SERVERS	Kafka bootstrap servers – separate multiple servers by comma
KAFKA_ROOT	Kafka root folder
KUBERNETES	If deploying to Kubernetes, set to 1 and VIPER will dynamically get IP address of Pod, and free port.
LOGSTREAMTOPIC	Enter the name of the topic that you want to write logs to. If this field is non-empty VIPER/HPDE/VIPERVIZ will all write logging information to this stream.
LOGSENDTOEMAILS	Viper will send log emails to these addresses: separate multiple addresses by comma.
LOGSENDTOEMAILSSUBJECT	You can add a custom subject for the email.
LOGSENDTOEMAILFOOTER	Specify additional text to be included in the footer of your email.
LOGSTREAMTOPICPARTITIONS	Enter number of partitions for LOGSTREAMTOPIC, i.e. 3
LOGSTREAMTOPICREPLICATIONFACTOR	Enter replication factor for LOGSTREAMTOPIC, i.e. 3
LOGSENDINTERVALMINUTES	Specify the minutes you want Viper to check the logs – it will email you a list of logs that have been created. This is convenient when you want a batch of logs to see what Viper is doing.
LOGSENDINTERVALONLYERROR	Set to 1 if you only want interval emails to check for ERROR or WARNING. If set to 0, all messages with ERROR, WARN, INFO will be checked, this is useful for debugging. For production set to 1.
MAADS_ALGORITHM_SERVER_PORT	MAADS algorithm server host PORT. This will require MAADS software installed in the host. This is needed to generate predictions from algorithms generated by MAADS.
MAILSERVER	SMTP mailserver host name for sending emails.
MAILPORT	SMTP mailserver port for sending emails.

MAXTRAININGROWS	Maximum number of rows for training dataset. Higher number will consumer more memory resources.
MAXOPENREQUESTS	How many outstanding requests a connection is allowed to have before sending on it blocks (default 5).
MAXPREDICTIONROWS	Maximum prediction batch size.
MINFORECASTACCURACY	Minimum forecast accuracy of trained TML model. Choose a number between 0-100, default is 0. A model is selected if it is greater than this value.
MYSQLDRIVERNAME	Enter MySQL driver name i.e. mysql
MYSQLDB	Enter MySQL DB name
MYSQLUSER	Enter MySQL username
MYSQLPASS	Enter MySQL password
MYSQLHOSTNAME	Enter MySQL hostname **If using MYSQL DOCKER set this to: host.docker.internal:3306**
MYSQLMAXLIFETIMEMINUTES	Enter max lifetime in minutes
MYSQLMAXCONN	Enter maximum connections
MYSQLMAXIDLE	Enter number of idle connections
MYSQL_ROOT_PASSWORD	MYSQL DOCKER Container: Set the Root password for MySQL
MYSQL_ROOT_HOST	MYSQL DOCKER Container: Set the Root host for MySQL ie. You can use % to accept connections from any host.
MYSQL_DATABASE	MYSQL DOCKER Container: Set the database name i.e. tmlids – **This should match MYSQLDB**
MYSQL_USER	MYSQL DOCKER Container: Set the username name i.e. tmluser, avoid “root” - **This should match MYSQLUSER**
MYSQL_PASSWORD	MYSQL DOCKER Container: Set the password **This should match MYSQLPASS**
MAXURLQUERYSTRINGBYTES	This is the size of the URL query string in bytes, if using viperhpdepredictprocess
MAXPREPROCESSMESSAGES	Number of message for preprocessing. Defaults to 2000. Higher number will consume more energy.
MAADS_ALGORITHM_SERVER	MAADS algorithm server host URL. This will require MAADS software installed in the host. This is needed to generate predictions from algorithms generated by MAADSBML.
MAXVIPERVIZROLLBACKOFFSET	Sets the maximum rollback offset in VIPERVIZ. This prevents memory heap issues.

MAXVIPERVIZCONNECTIONS	Total number of simultaneous connections to Viperviz. For example, MAXVIPERVIZCONNECTIONS=5
MAXPERCMESSAGES	Maximum messages when using Topicid to rollback stream. This is useful when even 1% rollbackback could result in millions of message if your total messages are in the billions. Setting MAXPERCMESSAGES=1000 for example, ensures message are 1000 messages from the last message.
MAXCONSUMEMESSAGES	The amount of message you want Viper to consume. Note consuming a large amount will impact memory and network.
MAADS_ALGORITHM_SERVER_MICROSERVICE	MAADS algorithm server microservice. This will require MAADS software installed in the host. If you use a reverse proxy to access the MAADS software then specify the name here.
MAADS_ALGORITHM_SERVER1	Additional MAADS algorithm server. You can list up to 10,000 MAADS algorithm servers. Just increment the “SERVER#”, where #=1,…,10000
MAADS_ALGORITHM_SERVER1_PORT	Additional MAADS algorithm server port.
MAADS_ALGORITHM_SERVER1_MICROSERVICE	Additional MAADS algorithm server microservice.
NOWINDOWOVERLAP	Set to 1, if you do NOT want sliding time windows to overlap.
NUMWINDOWSFORDUPLICATECHECK	This is an integer to specify how much data to retain to check for duplicates. For example, if NOWINDOWOVERLAP=0, then windows will overlap, but you do not want to re-process data which may result in duplicates, so this field will save data in MySQL and check if the Partition and Offset has already been processed, if so, it will not re-process it. If NUMWINDOWSFORDUPLICATECHECK=5, then the amount of data saved is 5 *(number of partitions) * (rollbackoffset) per topic and cluster.
ONPREM	Set to 1, if running VIPER on-premise, or 0 if using Cloud
POLLING_ALERTS	Polling for alerts in minutes. VIPER will poll for alerts and wait in minutes for the next poll.
SASLMECHANISM	Choose SASL mechanism. You can specify: PLAIN, SCRAM256, SCRAM512
SSL_CLIENT_CERT_FILE	SSL certificate file needed if Kafka is SSL/TLS enabled

SSL_CLIENT_KEY_FILE	SSL certificate key store file needed if Kafka is SSL/TLS enabled
SSL_SERVER_CERT_FILE	SSL certificate server key file needed if Kafka is SSL/TLS enabled
SMTP_USERNAME	SMTP username.
SMTP_PASSWORD	SMTP password.
SMTP_SSLTLS	Mailserver SSL/TLS enabled: true of false.
USEHTTP	Set to 1 if using HTTP to connect to VIPER. If SSL_CLIENT_CERT_FILE and SSL_CLIENT_KEY_FILE are specified then VIPER will automatically accept HTTPS connections. However, if USEHTTP=1, then regardless of certificates, HTTP will be used.
VIPER_IP	Specify IP for Viper, use * or leave empty for Viper to choose.
VIPER_PORT	Specify port. If you specifying port range use “startport:endport”, where start port and end port are numbers
VIPERVIZ_IP	Specify IP for Viperviz, use * or leave empty for Viper to choose.
VIPERVIZ_PORT	Specify port. If you specifying port range use “startport:endport”, where start port and end port are numbers
VIPERDEBUG	Set to 1, if you want additional screen logging, or 0. Set to 2, if you want additional screen **and** disk logging. Logs will be written to ./viperlogs/viperlogs.txt. This is helpful if you want to see details when building TML solutions. However, for production deployments, VIPERDEBUG should be set to 1 for optimized performance.
WRITETOVIPERDB	Set to 1, if you want to write Egress and Ingress bytes. Set to 0 if you do not want to write to viper.db. By setting to 0 this will speed up VIPER.
WRITELASTCOMMIT	Set to 1 if you want to record the last offset in the partition for each topic, or 0 if not. This is convenient if you do NOT want to RE-PROCESS data that has already been processed.

RTMSMAXWINDOWS	This number determines how far back you want TML to rememeber sliding time windows. For example, if RTMSMAXWINDOWS=10000, TML will remember 10000 past sliding time windows to compute the PatternScore. This is a very powerful TML feature. See :ref:`How TML Maintains Past Memory of Events Using Sliding Time Windows`

1. **You are done! Start VIPER.**
2. **Additional Documentation for Accessing VIPER Functionality**
3. **VIPER is accessed by two methods:**
    1. **MAADSTML python library: <https://pypi.org/project/maadstml/>**
        1. Scroll down to: **MAADS-VIPER Connector to Manage Apache KAFKA:**
    2. **REST API:**
        1. When starting VIPER type “Help” to see all the REST endpoints
        2. The endpoints can be called from ANY programming language.
4. Users can send an email to support@otics.ca for additional help with any of the functions.
5. OTICS provides up to **2 hours free virtual training** on an as-needed basis for clients or groups of clients.