



# How Transactional Machine Learning (TML) Processing and Machine Learning Works? And Other Details

Dr. Sebastian Maurice  
October 2023

# Table of Contents

[This File on Github](#)

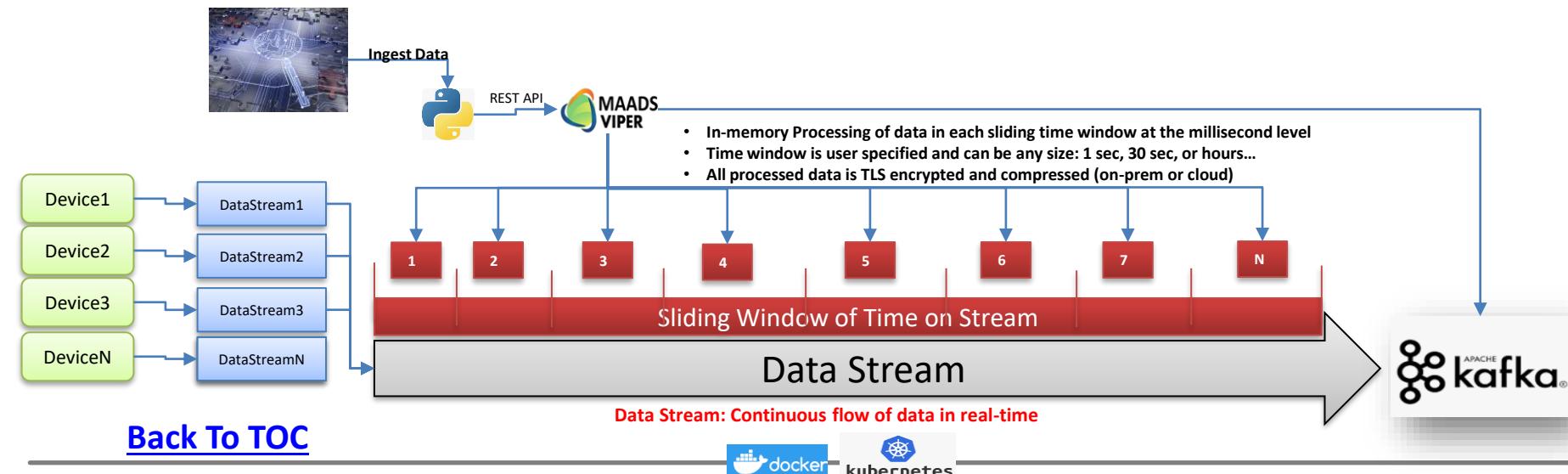
1. [TML Overview](#)
2. [How TML Preprocessing works?](#)
3. [How TML Machine Learning works?](#)
4. [How TML Processes JSON data in real-time?](#)
5. [How TML Processes JSON data in real-time? Example](#)
6. [TML Solution Container](#)
7. [TML IOT Dashboard](#)
8. [TML LOG STREAMING](#)
9. [STEPS TO RE-CREATING TML IOT SOLUTION FOR STUDENTS on WSL \(Windows Subsystem for Linux\)](#)
10. [TML SOLUTION COMPONENTS](#)
11. [APPENDIX A: Vmware Setup](#)
12. [APPENDIX B: Setting Up Internet on VM Ware](#)
13. [APPENDIX C: Vmware: Pulling/Running TML Docker Container and TML Streaming Dashboard](#)
14. [APPENDIX D: Vmware: Building Your Own Docker Container](#)
15. [APPENDIX E: Going Inside the TML Container](#)
16. [APPENDIX F: Kubernetes Setup](#)
17. [APPENDIX G: TML Folder Structure in Container](#)
18. [APPENDIX H: TML-CISCO Cybersecurity and Network Monitoring Solution](#)
19. [APPENDIX I: Additional sources](#)
20. [APPENDIX J: Transactional Machine Learning Book](#)
21. [APPENDIX K: Setting up Confluent Kafka Cloud](#)

# TML Overview

- Transactional Machine Learning (TML) is a platform technology that performs high-speed processing and machine learning on **real-time data streams**
- TML is comprised of 3 Binaries (for Linux/Mac/Windows on AMD/ARM/PPC chipsets 32 and 64-bit) found on GitHub: <https://github.com/smaurice101/transactionalmachinelearning>
  1. MAADS-Viper (intelligent source/sink connector for Apache Kafka)
  2. MAADS-HPDE (Auto Machine Learning Technology)
  3. MAADS-Viperviz (Visualization Streaming over Websockets)
- Binaries are developed using Go programming language
- TML Solutions are developed using the MAADSTML Python Library: <https://pypi.org/project/maadstml/>
- TML Binaries are integrated with Apache Kafka (the platform for storing and managing real-time data): <https://kafka.apache.org>
- All TML Processing is performed In-Memory and TML does not use SQL queries for processing and machine learning – **it processes devices at the ENTITY LEVEL – meaning each device (data) is processed individually.**
  - It is currently the only technology that performs, entity level, in-memory processing, AutoML, no SQL on data streams with Apache Kafka
  - TML solutions scale with Docker and Kubernetes
- Transactional Machine Learning Book can be found on Amazon
- TML is taught at Seneca in select courses

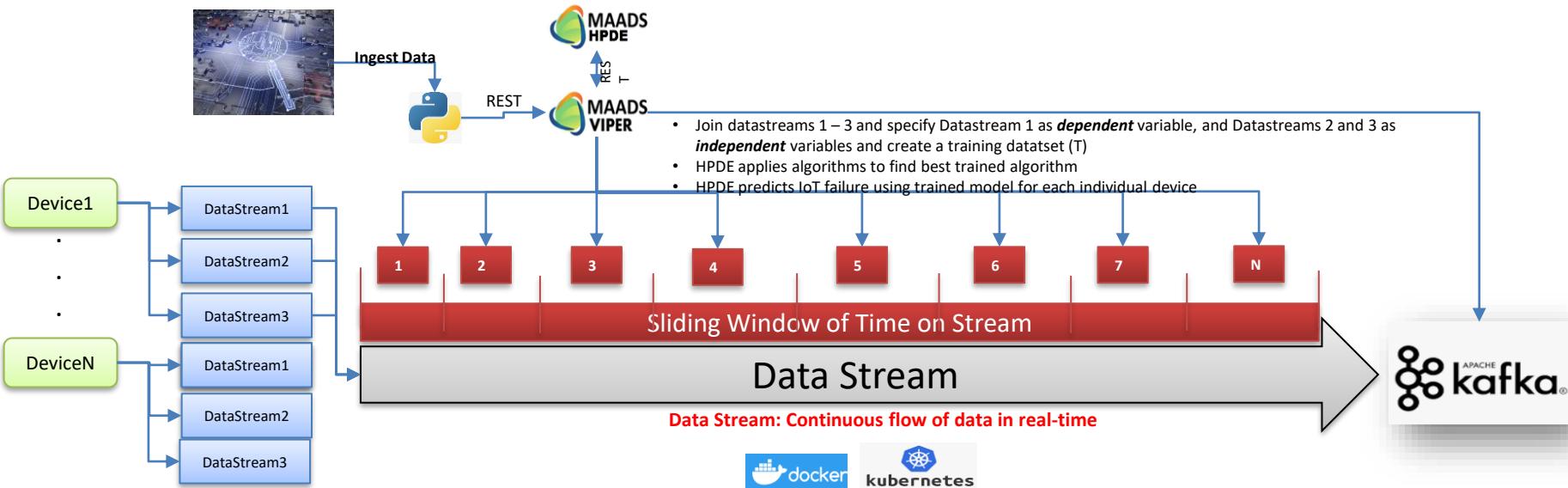
# How TML Preprocessing works?

- All data streams from devices flow into Apache Kafka to a Kafka Topic
- TML performs **in-memory** processing of data in the Kafka Topic using TWO components across all **sliding time windows**
  - Python Script that uses the [MAADSTML python library](#) functions
  - [MAADS-VIPER binary](#) that can run in Linux, Mac, Windows (or other operating systems) on any Chip (32 or 64 bit) architecture (AMD, ARM, PPC, S390x, etc.)
- 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***
- ***Our technology can process unlimited number of devices (billions at high speed)***
- All TML solutions are containerized with Docker and scale with Kubernetes



# How TML Machine Learning works?

- All data streams from devices flow into Apache Kafka to a Kafka Topic
- TML performs **in-memory** machine learning of data in the Kafka Topic by **joining data streams** using THREE components across all **sliding time windows**:
  - Python Script that uses the [MAADSTML python library](#) functions
  - [MAADS-VIPER binary](#) that can run in Linux, Mac, Windows (or other operating systems) on any Chip (32 or 64 bit) architecture (AMD, ARM, PPC, S390x, etc.)
  - [MAADS-HPDE binary](#) that can run in Linux, Mac, Windows (or other operating systems) on any Chip (32 or 64 bit) architecture (AMD, ARM, PPC, S390x, etc.)
- 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**
- **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**
- **Our technology can build unlimited machine learning models (billions at high speeds) for unlimited number of devices (billions at high speed)**
- All TML solutions are containerized with Docker and scale with Kubernetes



[Back To TOC](#)

# How TML Processes JSON data in real-time?

- TML uses json paths (fields) to extract data from JSONs
- It processeses a group of JSONs in a sliding time window by using a field called **Jsoncriteria** – **which requires a user to indicate how they want to extract data from a grouped or aggregate json messages in sliding time window**
- A **jsoncriteria** has **7 fields**:
  - **jsoncriteria=**
  - **'uid=**,filter:allrecords~\ \ ← **uid**: This is the json field to group by for example DSN or Device SerialNumber
  - **subtopics=~\ \ ← subtopics**: This is the json field to the name of the field you want to process
  - **values=~\ \ ← values**: This is the json field containing the value of the subtopic
  - **identifiers=~\ \ ← identifier**: This is the json field containing any label or identifier for the values
  - **datetime=~\ \ ← datetime**: This is the json field containing datetime, must be in UTC format – i.e. 2006-01-02T15:04:05
  - **msgid=~\ \ ← msgid**: this is the json field containing further details about the values
  - **latlong='** ← This is json field contain latitude and longitude. You can use a ":" to combine lat:long

[Back To TOC](#)

# How TML Processes JSON data in real-time? Example

- If I have:

```
{"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}  
{"metadata":{"oem_id":"32795e59","oem_model":"SQR141U1XXW","dsn":"AC000W016399396","property_name":"Current","display_name":"Current (mA)","base_type":"integer","event_type":"datapoint"},"datapoint":{"id":"de422f10-7faa-11ec-3925-f218ec2b4e1d","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": 28.22, "long": -140.22}  
{"metadata":{"oem_id":"32795e59","oem_model":"SQR441U1XXW","dsn":"AC000W016399127","property_name":"EnergyUsed","display_name":"Energy Used (mWh)","base_type":"integer","event_type":"datapoint"},"datapoint":{"id":"de3f833c-7faa-11ec-b4ba-126e4b986056","updated_at":"2022-01-27T19:53:59Z","created_at":"2022-01-27T19:53:59Z","echo":false,"closed":false,"value":2668340,"metadata":{},"created_at_from_device":"2022-01-27T19:51:31Z","user_uuid":"c4d88504-64b4-11eb-902d-0a580ae9bff0","discarded":false,"scope":"user","direction":"output"}, "lat": 24.22, "long": -149.22}  
{"metadata":{"oem_id":"32795e59","oem_model":"SQR441U1XXW","dsn":"AC000W016399127","property_name":"EnergyUsed24hr","display_name":"Energy Used 24hr (mWh)","base_type":"integer","event_type":"datapoint"},"datapoint":{"id":"de475850-7faa-11ec-dfce-f2bfc16ef579","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":"2022-01-27T19:51:31Z","user_uuid":"c4d88504-64b4-11eb-902d-0a580ae9bff0","discarded":false,"scope":"user","direction":"output"}, "lat": 23.22, "long": -143.22}
```

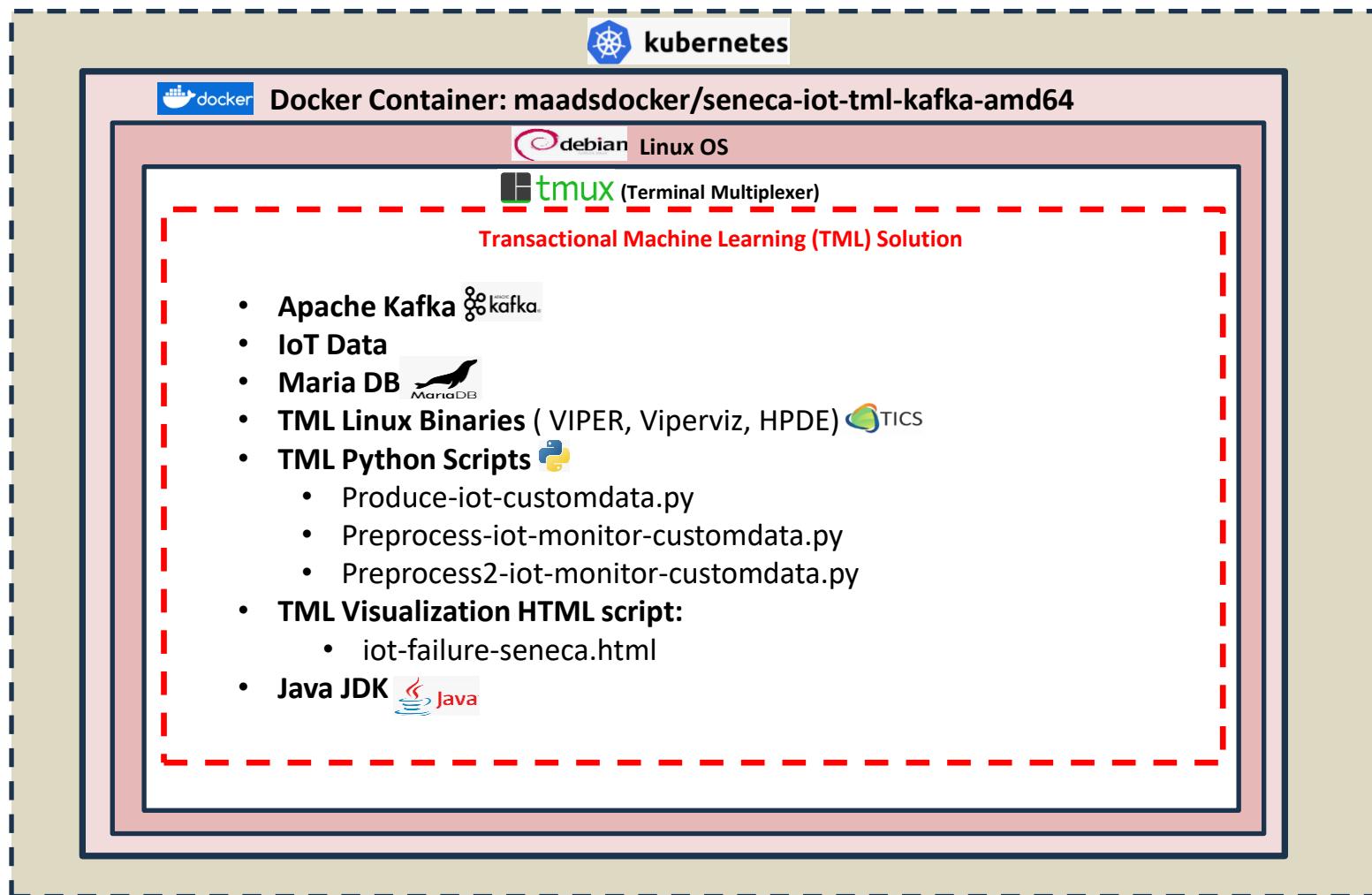
- I can extract, group and process them by specifying the following JSON criteria:

- jsoncriteria=

- **uid= metadata.dsn**,filter:allrecords~\ ← **uid**: This is the json field to group by for example DSN or Device SerialNumber
- **subtopics= metadata.property\_name**~\ ← **subtopics**: This is the json field to the name of the field you want to process
- **values= datapoint.value**~\ ← **values**: This is the json field containing the value of the subtopic
- **identifiers= metadata.display\_name**~\ ← **identifier**: This is the json field containing any label or identifier for the values
- **datetime= datapoint.updated\_at**~\ ← **datetime**: This is the json field in UTC format – i.e. 2006-01-02T15:04:05
- **msgid= datapoint.id** ~\ ← **msgid**: this is the json field containing further details about the values
- **latlong=lat:long**' ← This is json field contain latitude and longitude. You can use a ":" to combine lat:long

[Back To TOC](#)

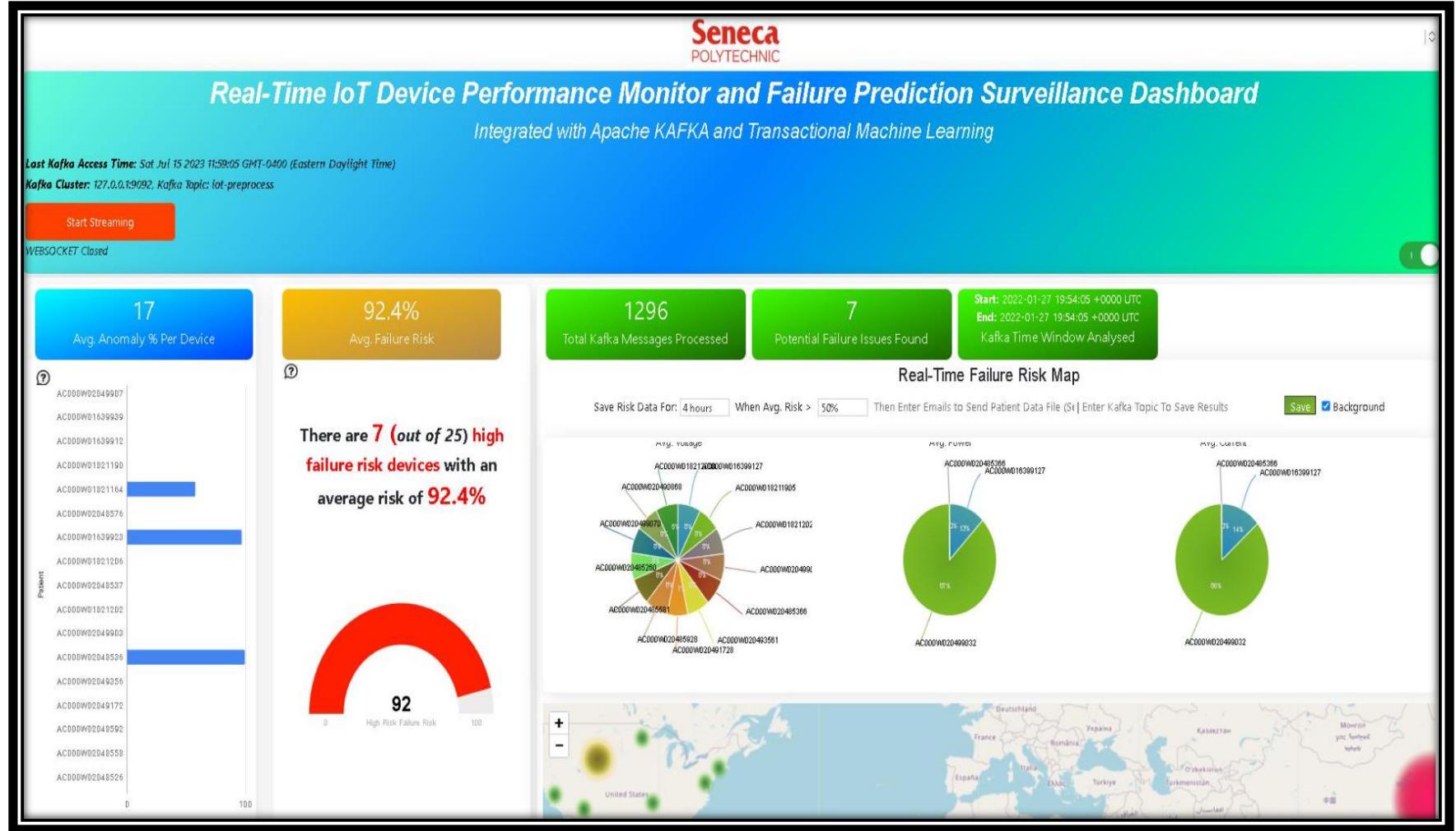
# TML Solution Container (Linux/Mac)



Mac user: maadsdocker/seneca-iot-tml-kafka-mac

[Back To TOC](#)

# TML IOT Dashboard



[Back To TOC](#)

# TML LOG STREAMING



## 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

The screenshot shows a log streaming interface with the following details:

- Stop Streaming** button
- Download Table as CSV | Download JSON** links
- Status:** [WEBSOCKET OPEN. Receiving Kafka messages from VIPERDEV (RUNNING...)]
- Generated** column (Timestamp)
- Message** column (Log entries)
- Service**, **Service Host**, **Service Port**, **Kafka Cluster**, **Offset**, and **Partition** columns for each log entry.

Generated	Message	Service	Service Host	Service Port	Kafka Cluster	Offset	Partition
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 preprocessstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,845	0
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 preprocessstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,851	0
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 preprocessstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,854	0
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 preprocessstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,855	0
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 preprocessstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,856	0
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 preprocessstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,846	0
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 preprocessstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,852	0
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 preprocessstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,853	0
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 preprocessstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,849	0
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 preprocessstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,859	0
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 preprocessstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,842	0
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 preprocessstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,850	0
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 preprocessstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,829	0
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 preprocessstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,832	0
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 preprocessstopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	16,835	0

[Back To TOC](#)

# STEPS TO RE-CREATING TML IOT SOLUTION FOR STUDENTS on WSL (Windows Subsystem for Linux)

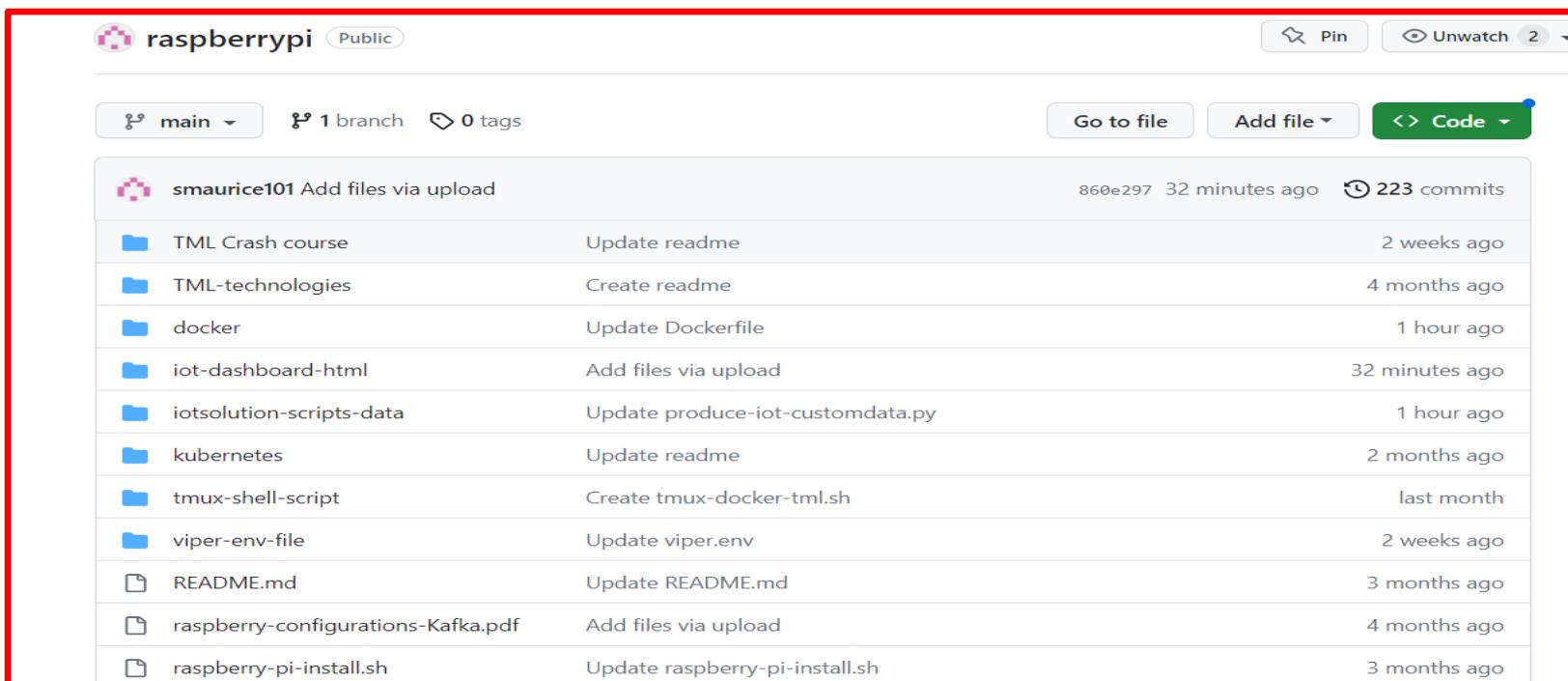
[Back To TOC](#)

# TML Student Solution: Re-Creating TML Solution

- Students can build their own streaming solution
- Before building your own solution – students re-create the solution **in Slide 6 and 7 to learn the components**

## STEPS TO TAKE TO RE-CREATE IOT SOLUTION:

1. Create your own Git Repository (**DO NOT ADD readme.md**) by cloning:
  - a) <https://github.com/smaurice101/raspberrypi.git> (you should see image below in YOUR OWN Github Account)



The screenshot shows a GitHub repository page for 'raspberrypi'. At the top, it says 'raspberrypi' (Public). Below that, there are buttons for 'main' (branch), '1 branch' (branch count), '0 tags' (tag count), 'Go to file', 'Add file', and 'Code'. The main area displays a list of commits from user 'smaurice101'. The commits are as follows:

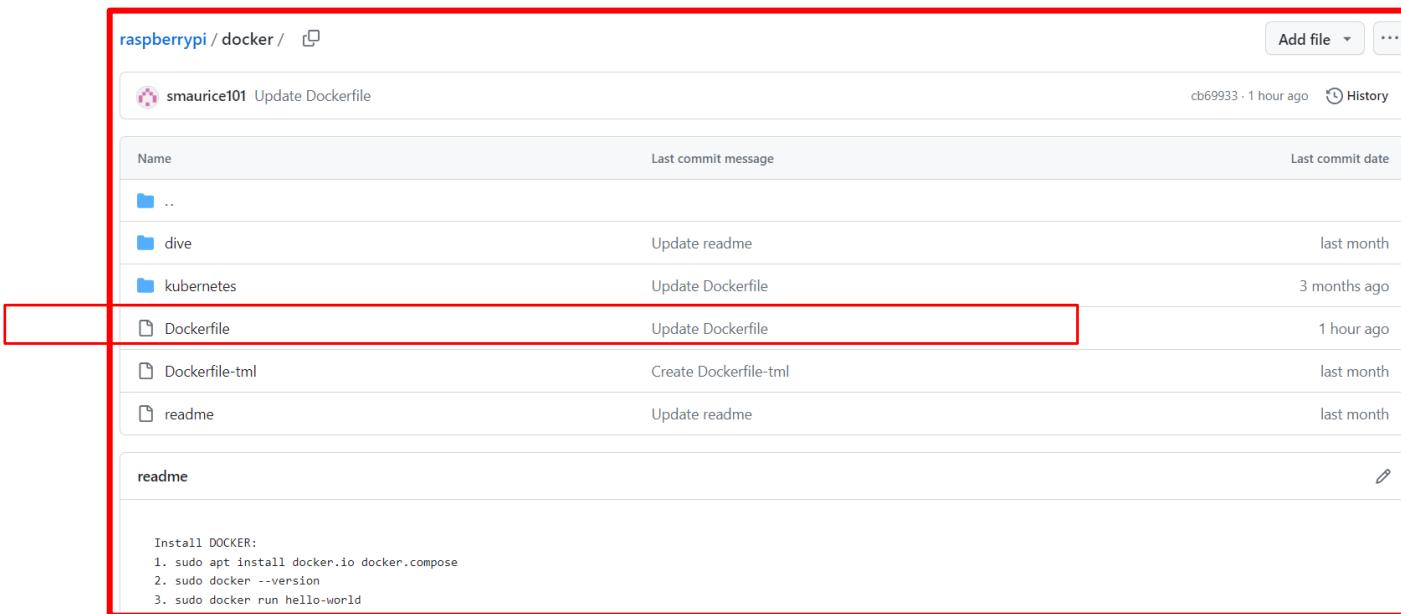
Commit	Message	Date
860e297	Add files via upload	32 minutes ago
Update readme		2 weeks ago
5f33a2d	Create readme	4 months ago
Update Dockerfile		1 hour ago
9a2a2c1	Add files via upload	32 minutes ago
Update produce-iot-customdata.py		1 hour ago
7a2a2c1	Update readme	2 months ago
Update tmux-docker-tml.sh		last month
5f33a2d	Create tmux-shell-script	
Update viper.env		2 weeks ago
Update README.md		3 months ago
9a2a2c1	Add files via upload	4 months ago
Update raspberry-pi-install.sh		3 months ago

[Back To TOC](#)

# TML Student Solution: Re-Creating TML Solution

## STEPS TO TAKE TO RE-CREATE IOT SOLUTION:

2. **Install docker in Linux VM or WSL (WSL is recommended):**
  1. Run: sudo apt install docker.io docker.compose
3. In your raspberry pi repo – GOTO **docker** folder
  1. **Copy the Dockerfile to your LOCAL computer (NOTE: File name **MUST** be exactly Dockerfile – no file extensions)**



[Back To TOC](#)

# TML Student Solution: Re-Creating TML Solution

## **STEPS TO TAKE TO RE-CREATE IOT SOLUTION:**

4. Go to the location where you stored Dockerfile on your LOCAL computer
  1. **Confirm Dockerfile exists**

```
smaurice@DESKTOP-H0DIAMM: /mnt/c/MAADS/DOCKER/TML-Solution/docker/seneca
smaurice@DESKTOP-H0DIAMM: /mnt/c/MAADS/DOCKER/TML-Solution/docker/seneca$ ls
Dockerfile
```

[Back To TOC](#)

# TML Student Solution: Re-Creating TML Solution

## STEPS TO TAKE TO RE-CREATE IOT SOLUTION:

5. Create a Dockerhub Account: <https://hub.docker.com/>
  - My account is: **maadsdocker** (REPLACE WITH YOUR OWN DOCKER HUB ACCOUNT)
6. RUN docker build in the SAME folder where Dockerfile is saved:
  1. Run: **docker build -t maadsdocker/seneca-iot-tml-kafka-amd64 --build-arg CHIP=AMD64 --network=host .**
  2. **NOTE: The “ . ” at the end – this must be there**
  3. **NOTE: DO NOT USE YOUR GITHUB Account in docker build command**
  4. **NOTE: IF YOU ARE MAC USER – REPLACE CHIP=AMD64 with CHIP=MAC**
  5. You can choose any container name you wish

```
smaurice@DESKTOP-H0DIAMM:/mnt/c/MAADS/DOCKER/TML-Solution/docker/seneca$ docker build -t maadsdocker/seneca-iot-tml-kafka-amd64 --build-arg CHIP=AMD64 --network=host .
```

**TIP:** After you did your normal build – you can use the FASTER Docker Build command:

```
docker build -t maadsdocker/seneca-iot-tml-kafka-amd64 --build-arg CHIP=AMD64 --build-arg CACHEBUST=$(date +%-s) --network=host .
```

**OR FOR MAC USERS:**

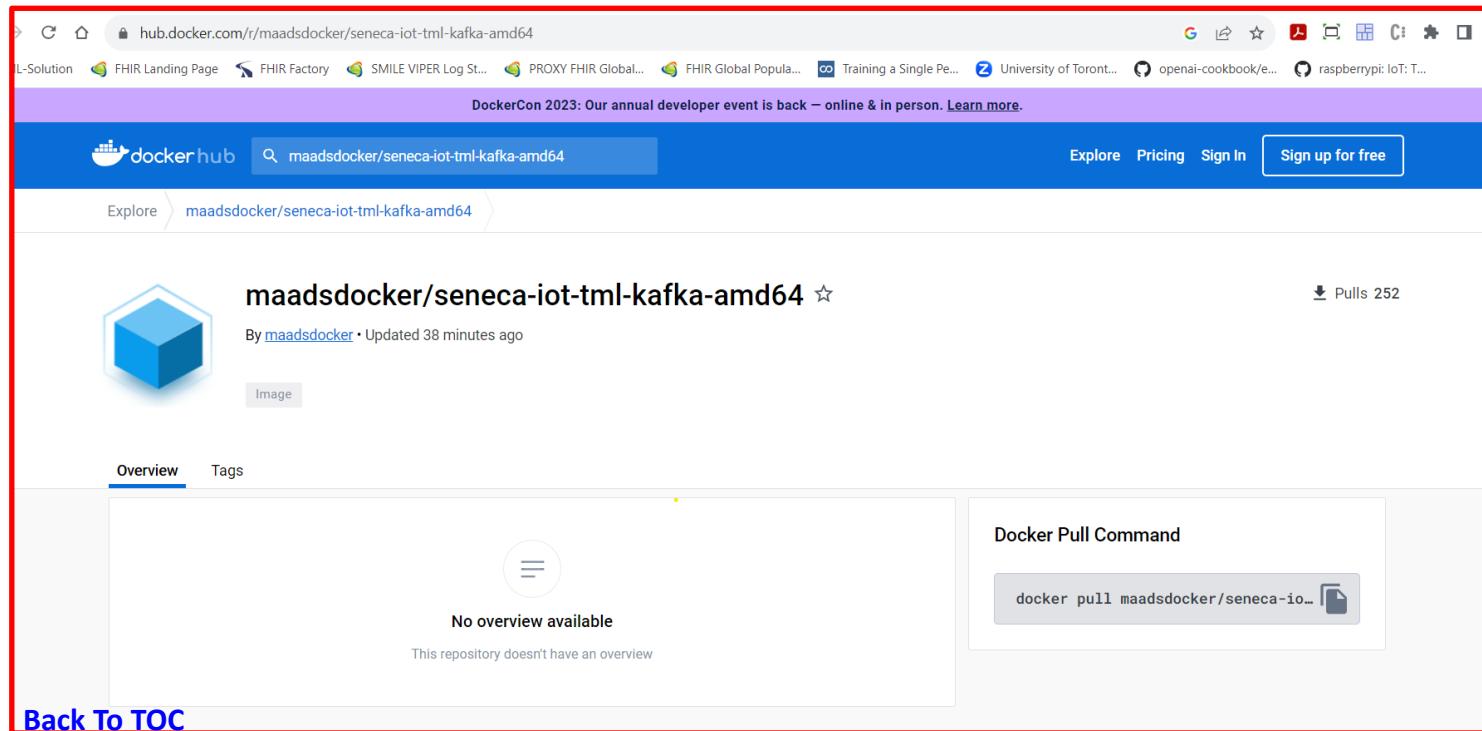
```
docker build -t maadsdocker/seneca-iot-tml-kafka-amd64 --build-arg CHIP=MAC --build-arg CACHEBUST=$(date +%-s) --network=host .
```

[Back To TOC](#)

# TML Student Solution: Re-Creating TML Solution

## **STEPS TO TAKE TO RE-CREATE IOT SOLUTION:**

7. If your Docker build is successful you now have a Docker Container called: **seneca-iot-tml-kafka-amd64**
8. **You can now PUSH your container to your Docker Hub account:**
  - Run: **docker push maadsdocker/seneca-iot-tml-kafka-amd64**
9. If your Push is successful you will see your container in Docker Hub under your account



[Back To TOC](#)

# TML Student Solution: Re-Creating TML Solution

## **STEPS TO TAKE TO RE-CREATE IOT SOLUTION:**

### **10. Run your container:**

- 1. Run: docker run -p 9005:9005 maadsdocker/seneca-iot-tml-kafka-amd64**
- 2. NOTE: The “ -p“ this will FORWARD Port 9005 and map HOST Port 9005 to CONTAINER Port 9005**
- 3. You MUST port forward for TML Dashboard to work**

```
smaurice@DESKTOP-HODIANN: /mnt/c/MAADS/DOCKER/TML-Solution/docker/seneca$ docker run -p 9005:9005 maadsdocker/seneca-iot-tml-kafka-amd64
```

[Back To TOC](#)

# TML Student Solution: Re-Creating TML Solution

## **STEPS TO TAKE TO RE-CREATE IOT SOLUTION:**

### **11. RAW DATA FOR SOLUTION:**

1. <https://docs.google.com/uc?export=download&id=1yRgDYrWnHu74NYX9GMAVDjR10ZyfoZvh>



IoTData - Google Drive

**Students can change this path to their own data.**

- Insert your file ID into this URL (<https://drive.google.com/uc?export=download&id=>), then surround the URL with quotes so that Bash doesn't misinterpret the &, like so:
- Get file ID by going to share -> copy link -> then get id from C0Py link:  
[https://drive.google.com/file/d/1mGcHQC7IxiTFYeUSFof3fDppVSC4rq3v/view?usp=drive\\_link](https://drive.google.com/file/d/1mGcHQC7IxiTFYeUSFof3fDppVSC4rq3v/view?usp=drive_link)
- Specifically, you will need to use this URL: <https://drive.google.com/uc?export=download&id=>
- YOU WILL NEED TO ADD THE id FOR YOUR FILE - THIS CAN BE FOUND BY RIGH-CLICKING ON YOUR FILE IN GOOGLE DRIVE - CHOOSE SHARE -> THEN COPY LINK -THEN COPY THE TEXt BETWEEN /d and /view.
- For example, here is a similar link: [https://drive.google.com/file/d/1mGcHQC7IxiTFYeUSFof3fDppVSC4rq3v/view?usp=drive\\_link](https://drive.google.com/file/d/1mGcHQC7IxiTFYeUSFof3fDppVSC4rq3v/view?usp=drive_link)
- The id is 1mGcHQC7IxiTFYeUSFof3fDppVSC4rq3v. The download url will be:  
<https://drive.google.com/uc?export=download&id=1mGcHQC7IxiTFYeUSFof3fDppVSC4rq3v>

**This url will need to be replaced in Dockerfile to download your IoTData.zip**

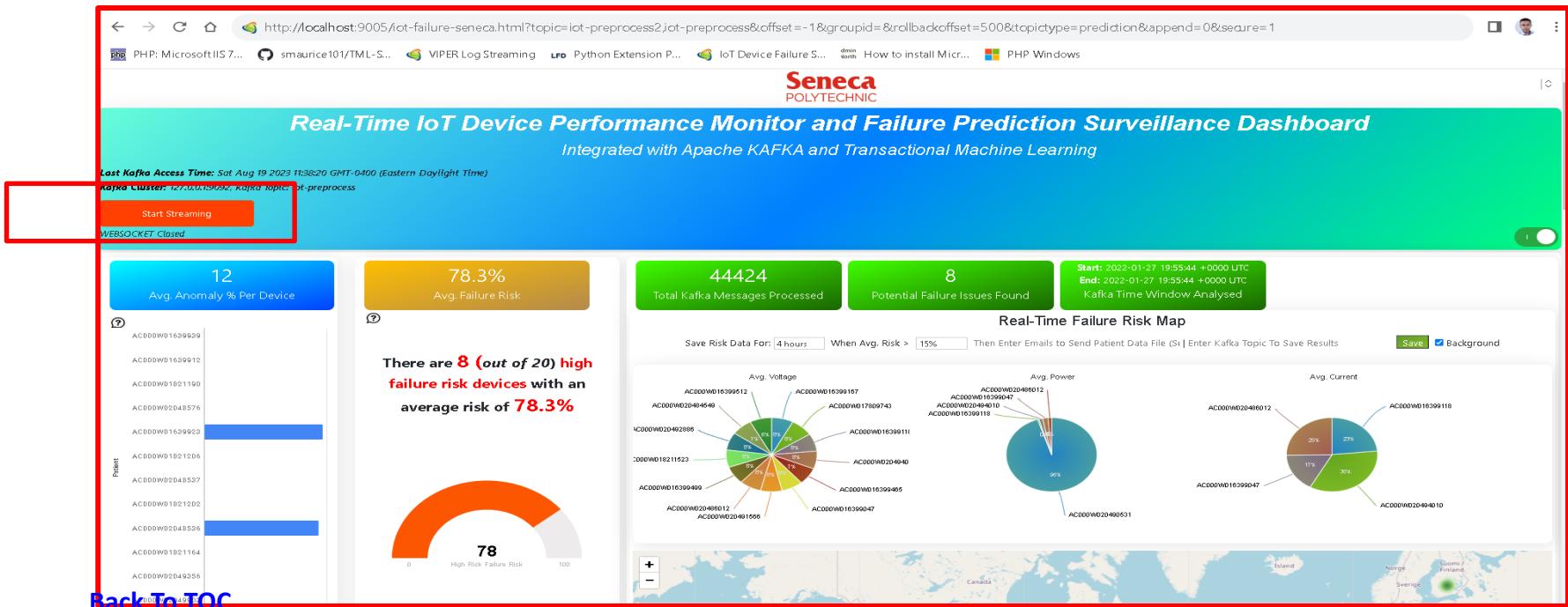
[Back To TOC](#)

# TML Student Solution: Re-Creating TML Solution

## STEPS TO TAKE TO RE-CREATE IOT SOLUTION:

### 12. RUN TML DASHBOARD:

1. Open a Browser on the machine running the container
2. **PASTE This URL in your browser:** <http://localhost:9005/iot-failure-seneca.html?topic=iot-preprocess2,iot-preprocess&offset=-1&groupid=&rollbackoffset=500&topicstype=prediction&append=0&secure=1>
3. **CLICK START STREAMING BUTTON**



# TML Student Solution: Re-Creating TML Solution

## STEPS TO TAKE TO RE-CREATE IOT SOLUTION:

### 13. RUN TML LOG STREAMING:

1. Open a Browser on the machine running the container
2. PASTE This URL in your browser: <http://localhost:9005/viperlogs.html?topic=viperlogs&append=0>
3. Click **Start Streaming** button

VIPER LOG STREAM: **viperlogs**

Last Kafka Access Time: Sat Aug 19 2023 11:38:21 GMT-0400 (Eastern Daylight Time)

Kafka Cluster: 127.0.0.1:9092 Kafka Topic: viperlogs

Status:

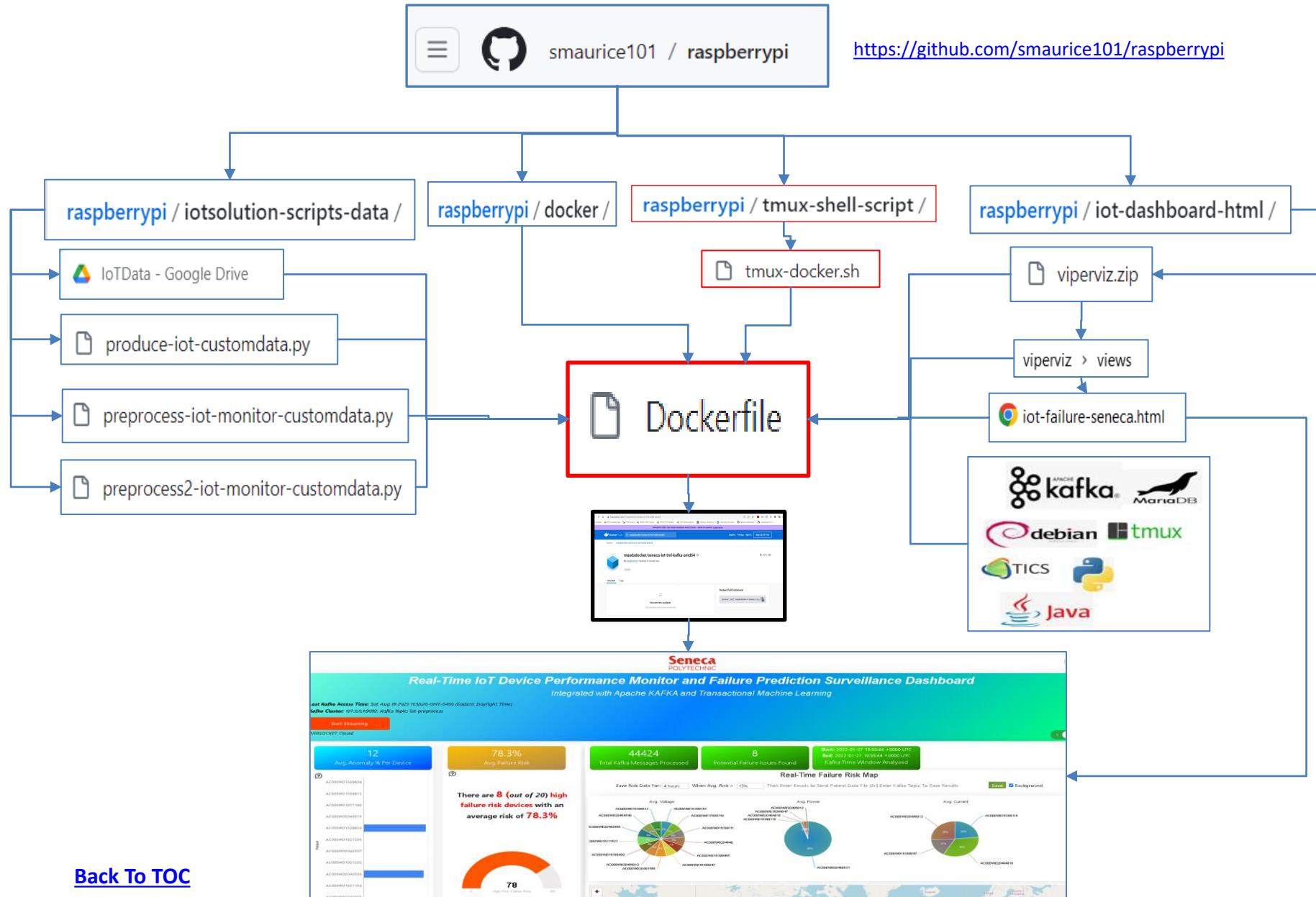
Generated	Message	Service	Service Host	Service Port	Kafka Cluster	Offset	Partition
1 2023-08-19T15:38:12.436+00:00	[Sat, 19 Aug 2023 15:38:10.7230 UTC] INFO [parsesubtopics Record(s) found=EnergyUsed24hr. In Topic=iot-mainstream - Viper writing results to preprocessTopic=iot-preprocess. YOU ARE STREAMING!]	VIPER	172.17.0.2	41,575	127.0.0.1:9092	419,690	0
2 2023-08-19T15:38:12.436+00:00	[Sat, 19 Aug 2023 15:38:10.7230 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	419,693	0
3 2023-08-19T15:38:12.436+00:00	[Sat, 19 Aug 2023 15:38:10.7230 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	419,694	0
4 2023-08-19T15:38:12.436+00:00	[Sat, 19 Aug 2023 15:38:10.7230 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	419,695	0
5 2023-08-19T15:38:12.436+00:00	[Sat, 19 Aug 2023 15:38:10.7229 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	419,672	0
6 2023-08-19T15:38:12.436+00:00	[Sat, 19 Aug 2023 15:38:10.7229 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	419,676	0
7 2023-08-19T15:38:12.436+00:00	[Sat, 19 Aug 2023 15:38:10.7229 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	419,679	0
8 2023-08-19T15:38:12.436+00:00	[Sat, 19 Aug 2023 15:38:10.7229 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	419,689	0
9 2023-08-19T15:38:12.436+00:00	[Sat, 19 Aug 2023 15:38:10.7229 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	419,691	0
10 2023-08-19T15:38:12.436+00:00	[Sat, 19 Aug 2023 15:38:10.7229 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	419,669	0
11 2023-08-19T15:38:12.436+00:00	[Sat, 19 Aug 2023 15:38:10.7229 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	419,674	0

[Back To TOC](#)

# YOU ARE NOW STREAMING!

[Back To TOC](#)

# TML SOLUTION COMPONENTS



[Back To TOC](#)

# APPENDIX A

## VMWare Setup

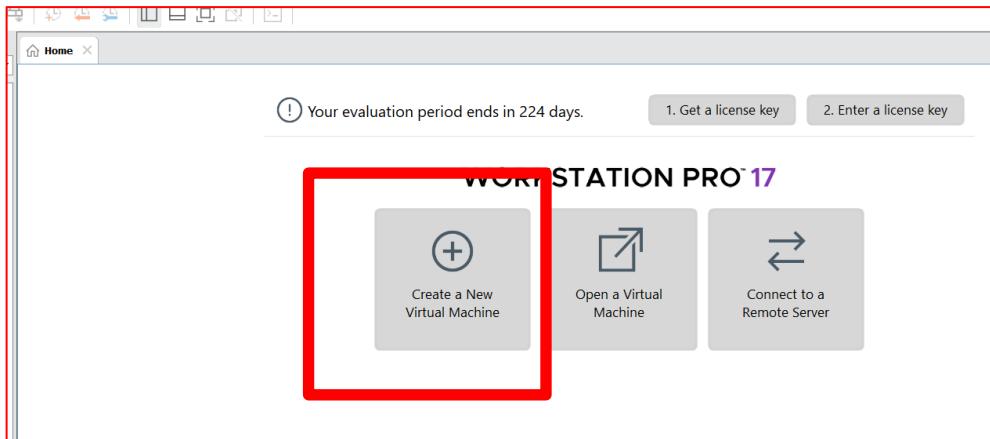
[Back To TOC](#)

# VMWare VM Setup Recommendations

- Students can use WSL (Windows subsystem for Linux) or VMWare (from myapps)
  - We will discuss the VMWare Setup
- Students should run their VM off a USB 3.0 (fast USB) drive preferably 250G or more
- VM OS should be Linux Ubuntu – the ISO can be downloaded from  
<https://ubuntu.com/download/desktop>

## Steps to Creating a VM:

1. Plug in your USB drive – lets say it is assigned DRIVE E: (your drive may differ – if so replace E: with your drive)
2. Start VM – and CLICK **Create a New Virtual Machine**:



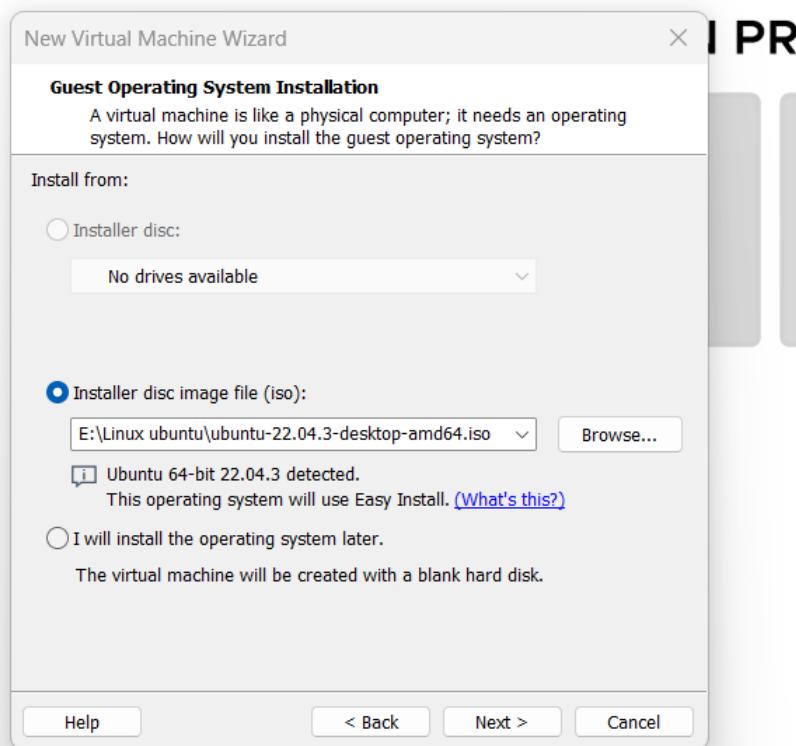
[Back To TOC](#)

# VMWare VM Setup Recommendations

## Steps to Creating a VM:

3. Choose Typical (Recommended) Setup and Click Next

4. In the next screen **Browse to your Downloaded Linux ISO on Your USB Drive and Click Next**

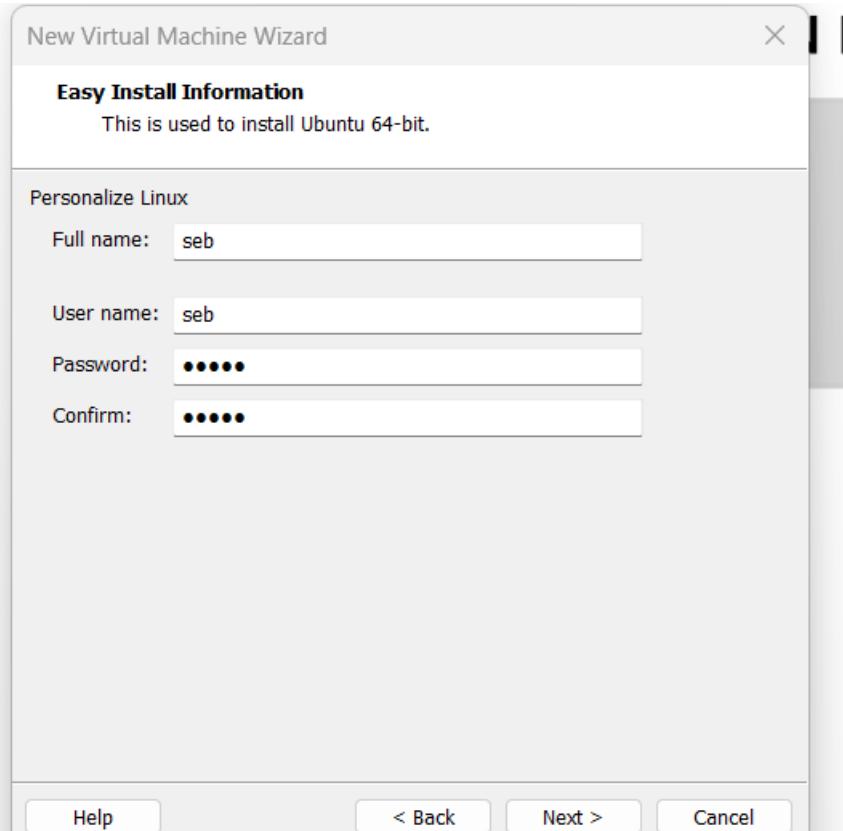


[Back To TOC](#)

# VMWare VM Setup Recommendations

## Steps to Creating a VM:

5. Enter your Name, and username/password (choose a simple username and password so you **do not forget it**)

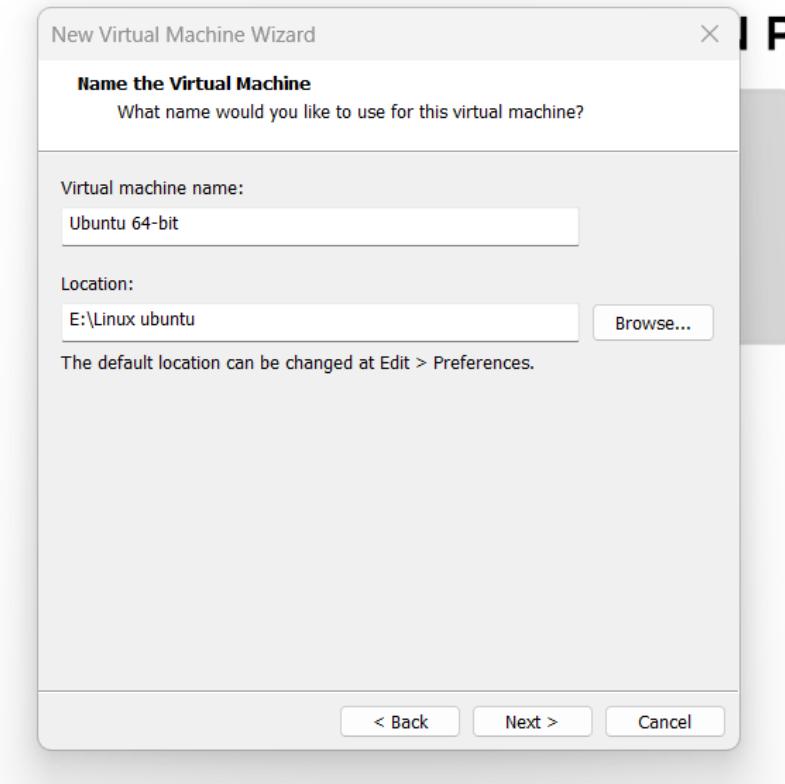


[Back To TOC](#)

# VMWare VM Setup Recommendations

## Steps to Creating a VM:

6. Change the Default Location of Your VM to your USB drive (Drive E) click NEXT – (*you may want to create a Linux Ubuntu Folder on Drive E to keep your VM files separate from other files on your USB*)

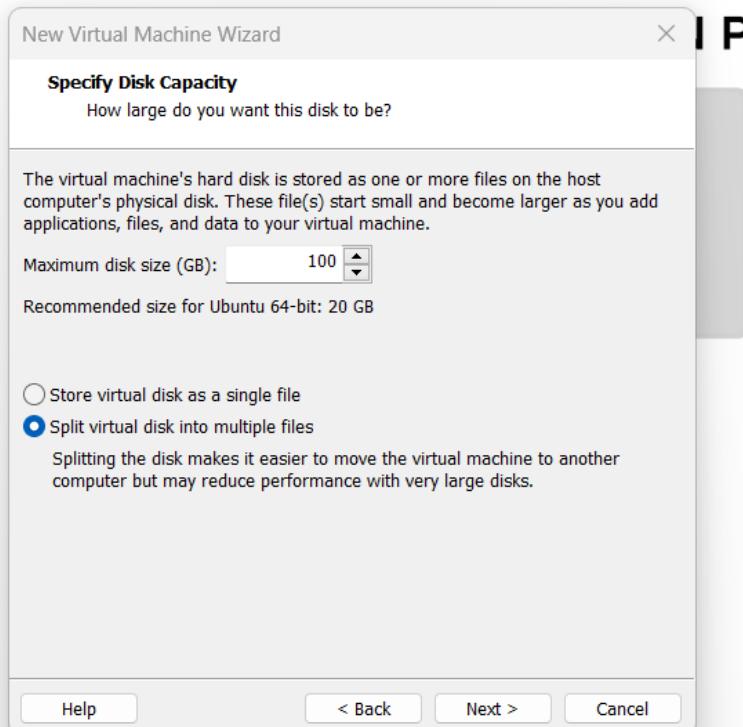


[Back To TOC](#)

# VMWare VM Setup Recommendations

## Steps to Creating a VM:

7. Choose 100GB for Maximum disk space AND Split virtual disk into multiple files click NEXT

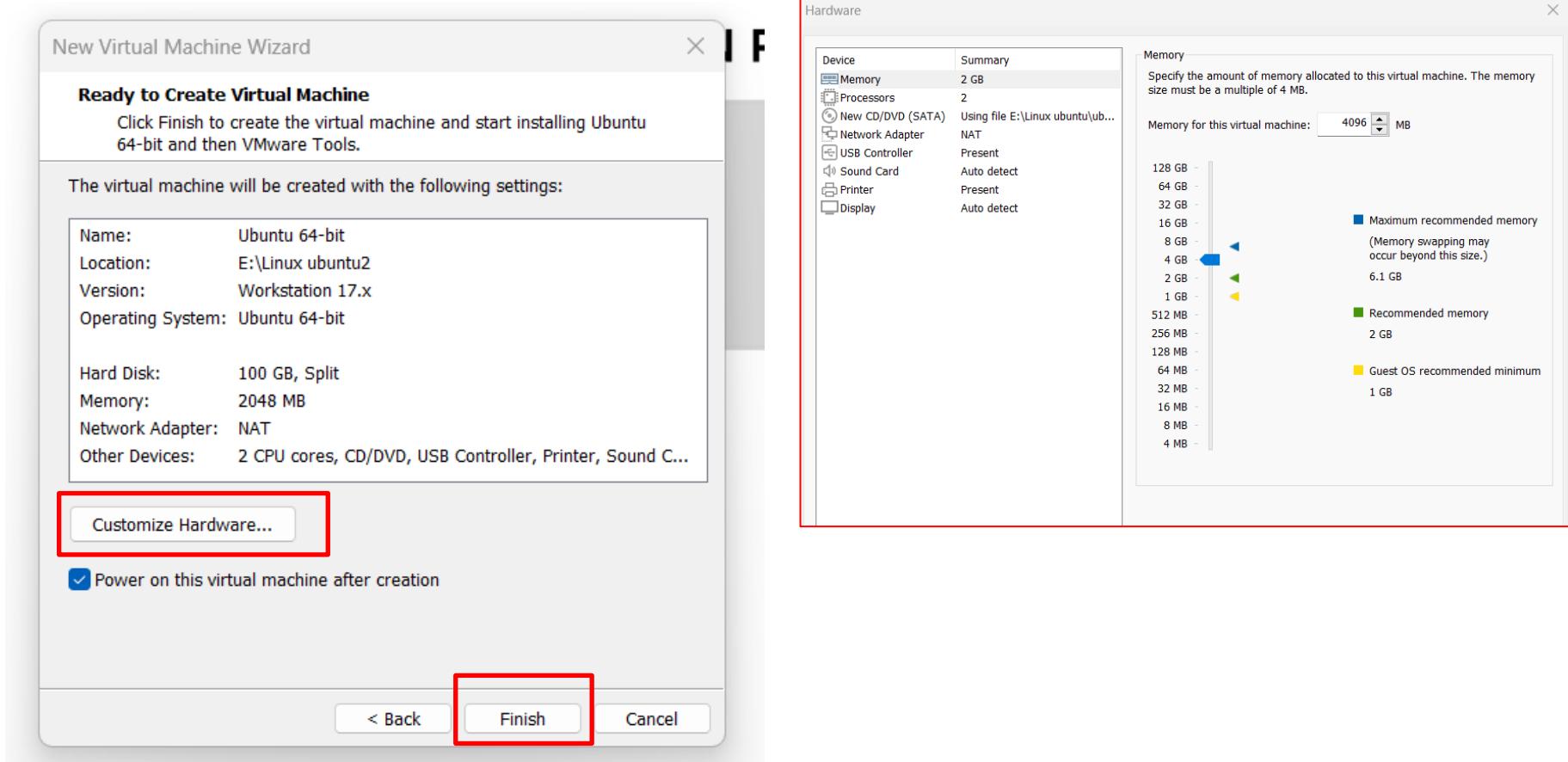


[Back To TOC](#)

# VMWare VM Setup Recommendations

## Steps to Creating a VM:

8. Click Customize Hardware and choose minimum of 4GB for memory then Click Finish

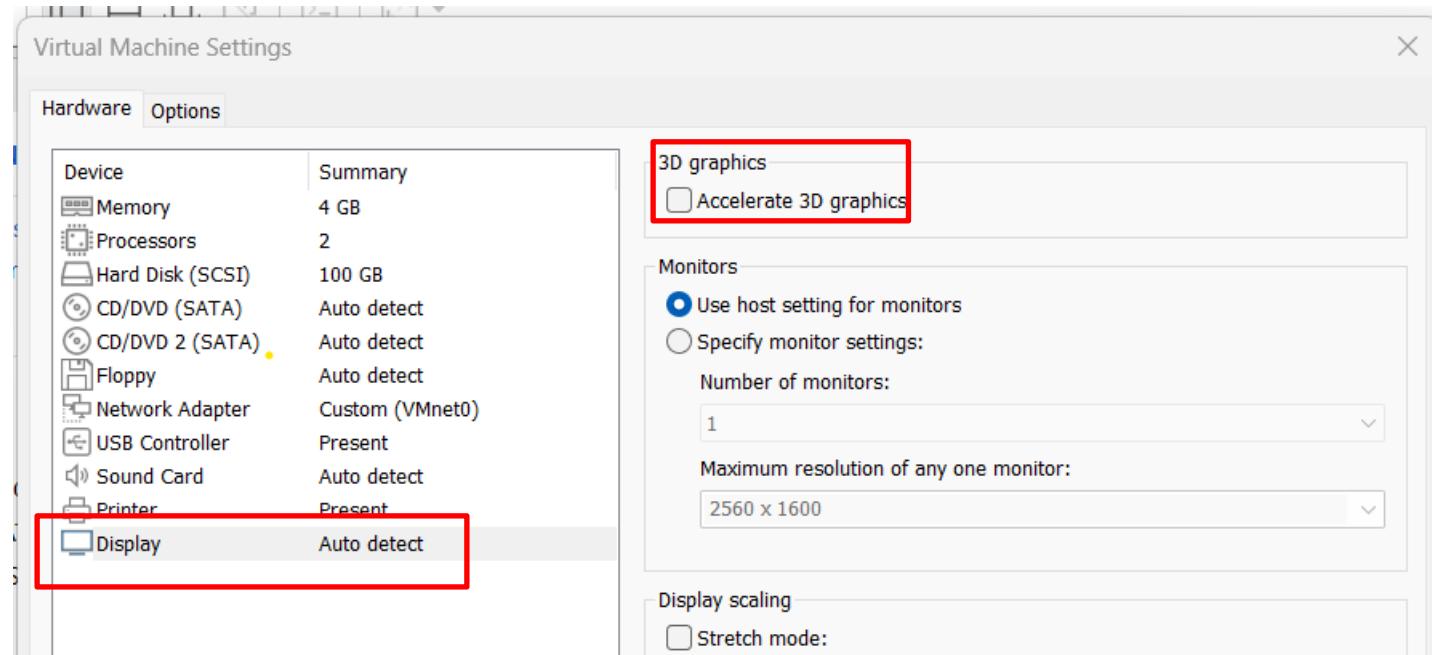


[Back To TOC](#)

# VMWare VM Setup Recommendations

## Steps to Creating a VM:

### 8b. Make sure to Uncheck 3D Graphics

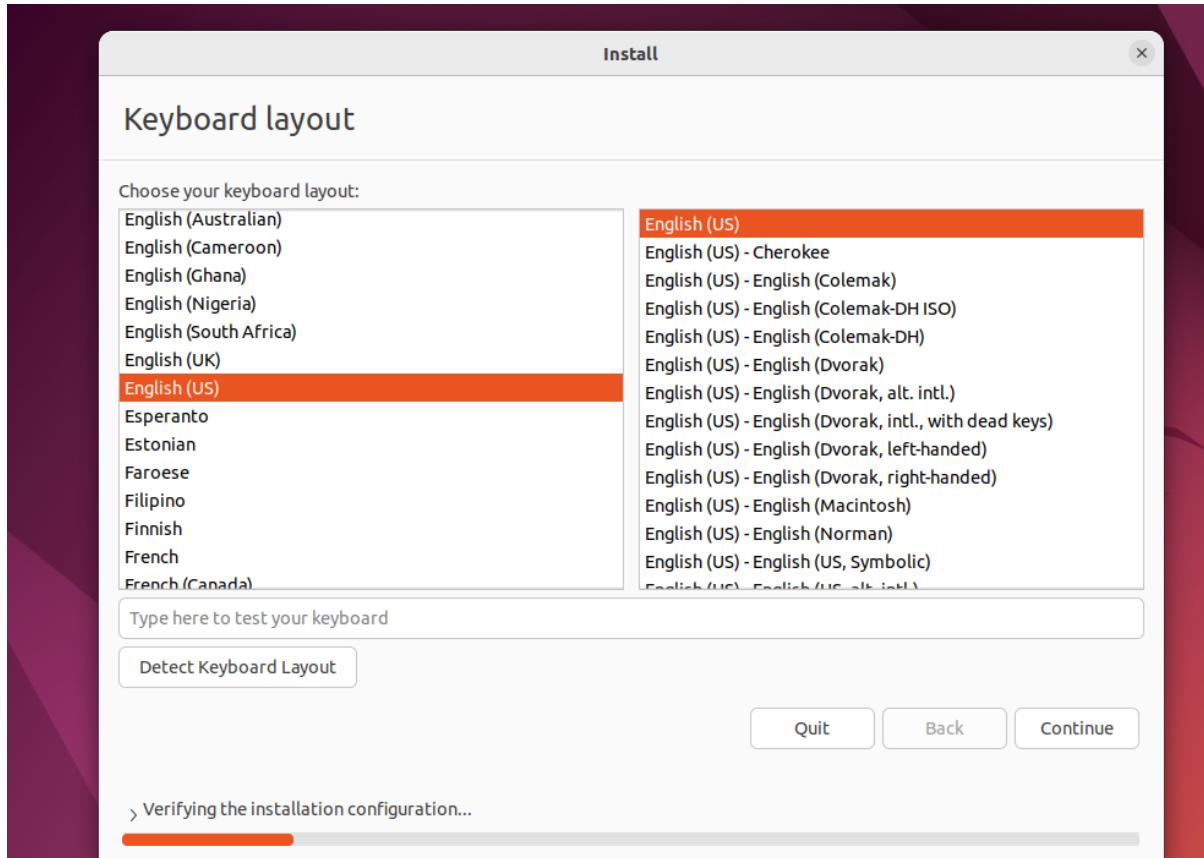


[Back To TOC](#)

# VMWare VM Setup Recommendations

## Steps to Creating a VM:

9. Linux Ubuntu will Start Installing – Click Continue



[Back To TOC](#)

# VMWare VM Setup Recommendations

## Steps to Creating a VM:

9. Linux Ubuntu will Start Installing - Click Continue

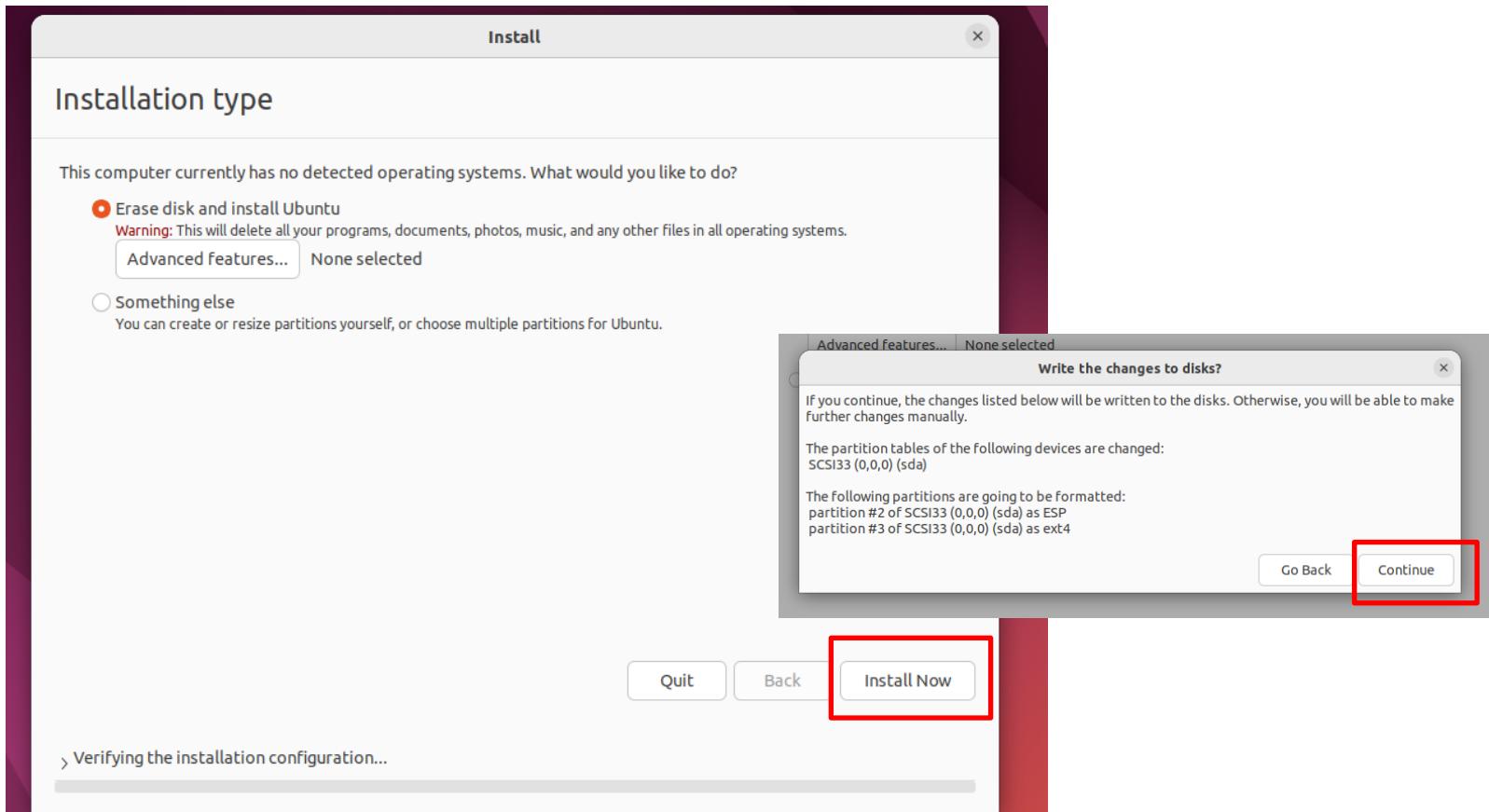


[Back To TOC](#)

# VMWare VM Setup Recommendations

## Steps to Creating a VM:

9. Linux Ubuntu will Start Installing – **Click Install Now and Click Continue for rest of screens**

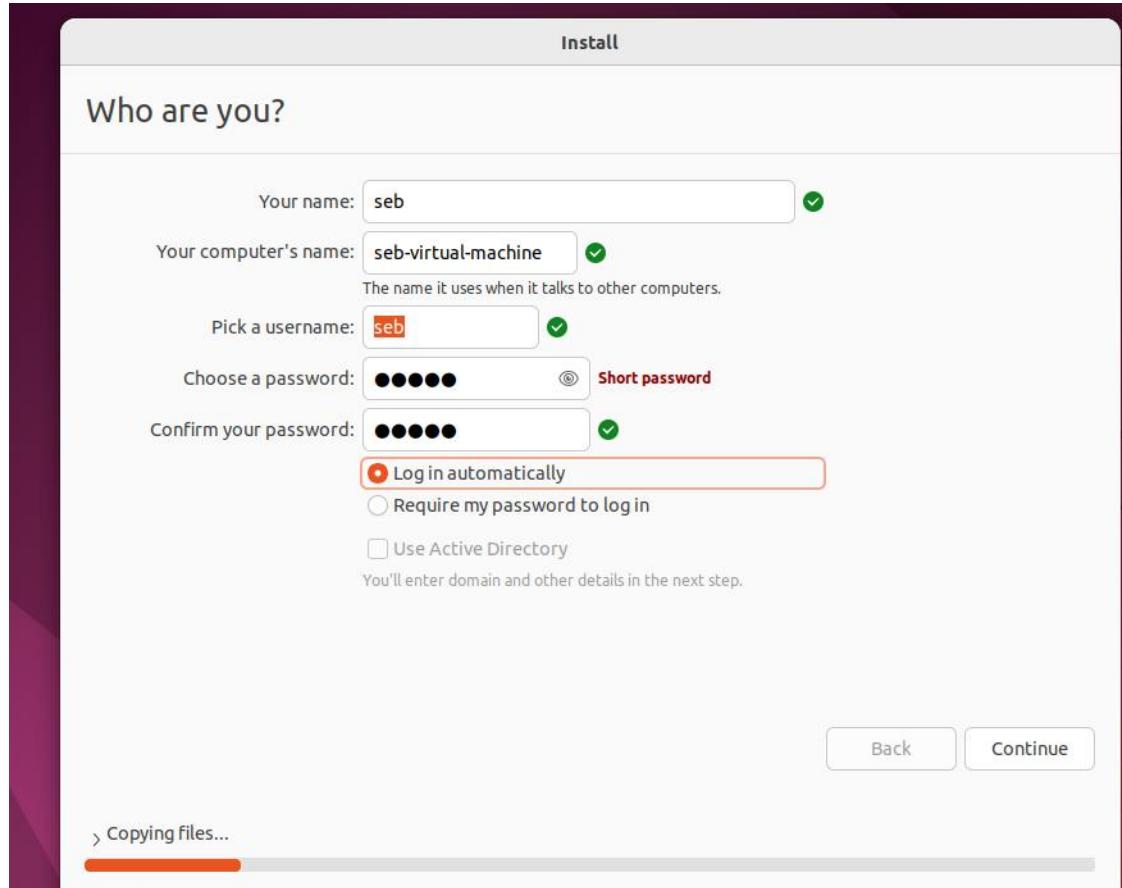


[Back To TOC](#)

# VMWare VM Setup Recommendations

## Steps to Creating a VM:

10. Who are you? Enter information and click Continue

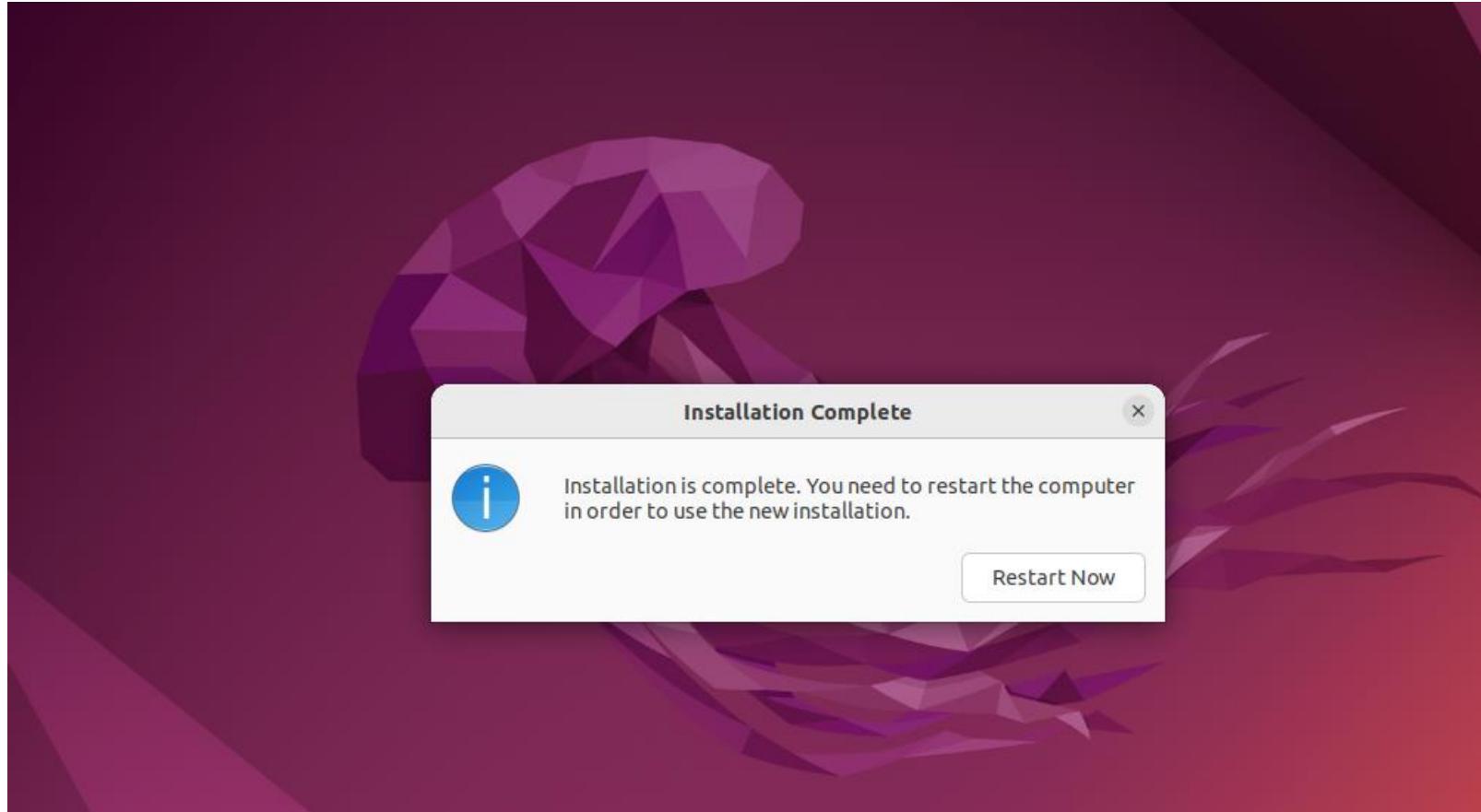


[Back To TOC](#)

# VMWare VM Setup Recommendations

## Steps to Creating a VM:

11. Restart VM to complete installation



[Back To TOC](#)

# VMWare VM Setup Recommendations

## Steps to Creating a VM:

12. Click NEXT

The screenshot shows a step in the VMWare VM setup process titled "Enable Ubuntu Pro". At the top, there are "Previous" and "Next" buttons. The "Ubuntu Pro" section contains an orange icon with the Ubuntu logo and the text "Ubuntu Pro". It describes the benefits of upgrading to Ubuntu Pro, mentioning security updates until 2032, compliance with FedRAMP, FIPS, STIG, and HIPAA, and the ability to manage up to 5 machines. A link to "ubuntu.com/pro" is provided for more information. Below this, a message says "Enable Ubuntu Pro for this installation or skip this step." There are two options: "Enable Ubuntu Pro" (with a warning icon indicating an internet connection is required) and "Skip for now" (selected, with a note that it can be enabled later via the 'pro attach' command).

Enable Ubuntu Pro

 Ubuntu Pro

Upgrade this machine to Ubuntu Pro for security updates on a much wider range of packages, until 2032. Fulfill FedRAMP, FIPS, STIG and HIPAA and other compliance and hardening requirements with certified tooling and crypto-modules. Free up to 5 machines.

More information on [ubuntu.com/pro](http://ubuntu.com/pro).

Enable Ubuntu Pro for this installation or skip this step.

Enable Ubuntu Pro     ! An internet connection is required to enable Ubuntu Pro

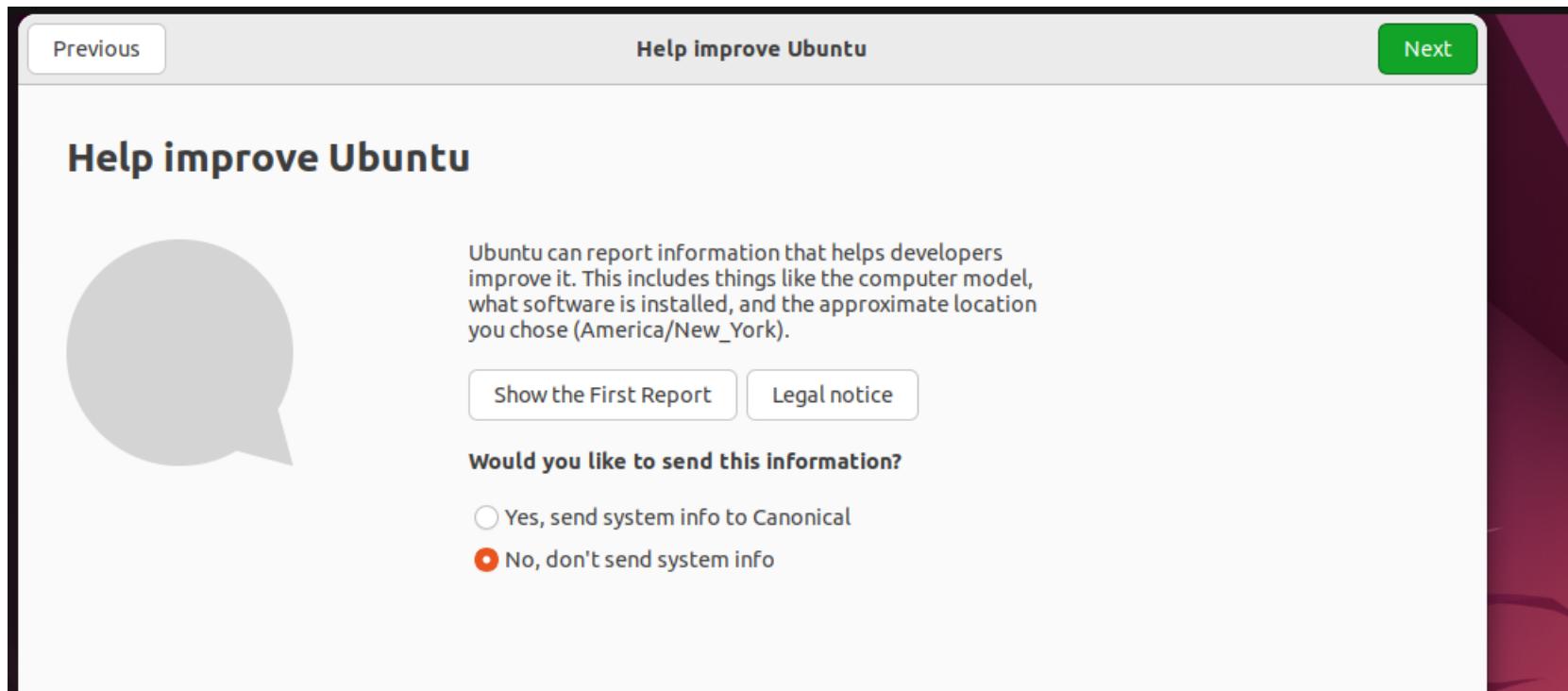
Skip for now  
You can always enable Ubuntu Pro later via the 'pro attach' command

[Back To TOC](#)

# VMWare VM Setup Recommendations

## Steps to Creating a VM:

13. Click NEXT



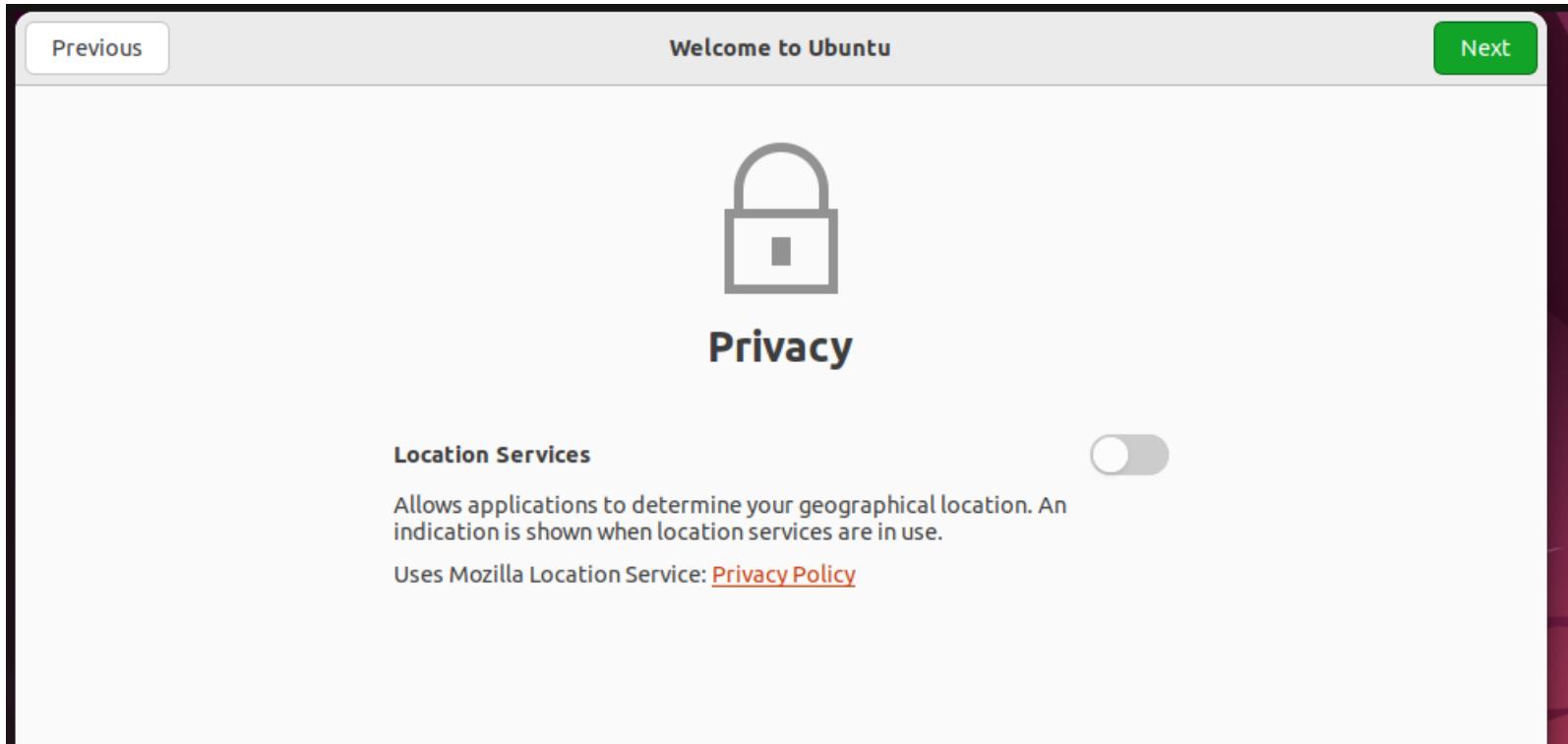
The screenshot shows a step in the Ubuntu VM setup process titled "Help improve Ubuntu". At the top, there are "Previous" and "Next" buttons. The "Help improve Ubuntu" section contains a large gray speech bubble icon. It explains that Ubuntu can report information to help developers improve the software, mentioning computer model, installed software, and location. Below this are two buttons: "Show the First Report" and "Legal notice". A question "Would you like to send this information?" is followed by two radio button options: "Yes, send system info to Canonical" (unchecked) and "No, don't send system info" (checked).

[Back To TOC](#)

# VMWare VM Setup Recommendations

## Steps to Creating a VM:

14. Click NEXT

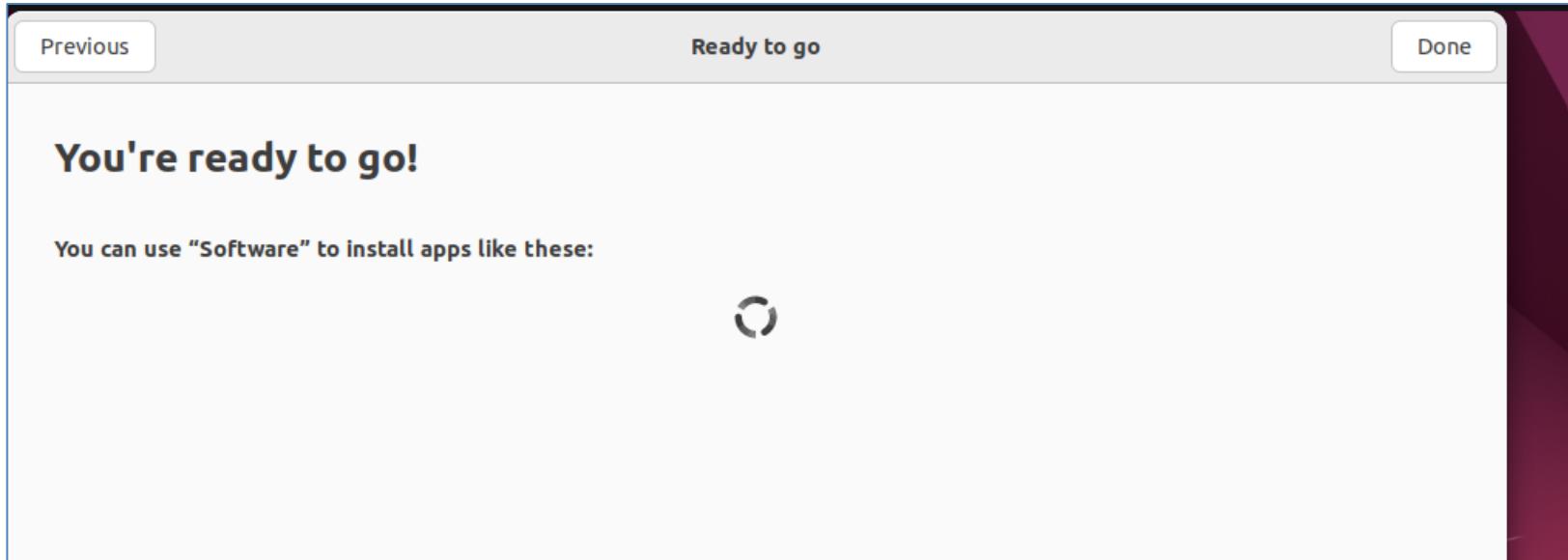


[Back To TOC](#)

# VMWare VM Setup Recommendations

## Steps to Creating a VM:

15. Click **DONE!**



[Back To TOC](#)

# APPENDIX B

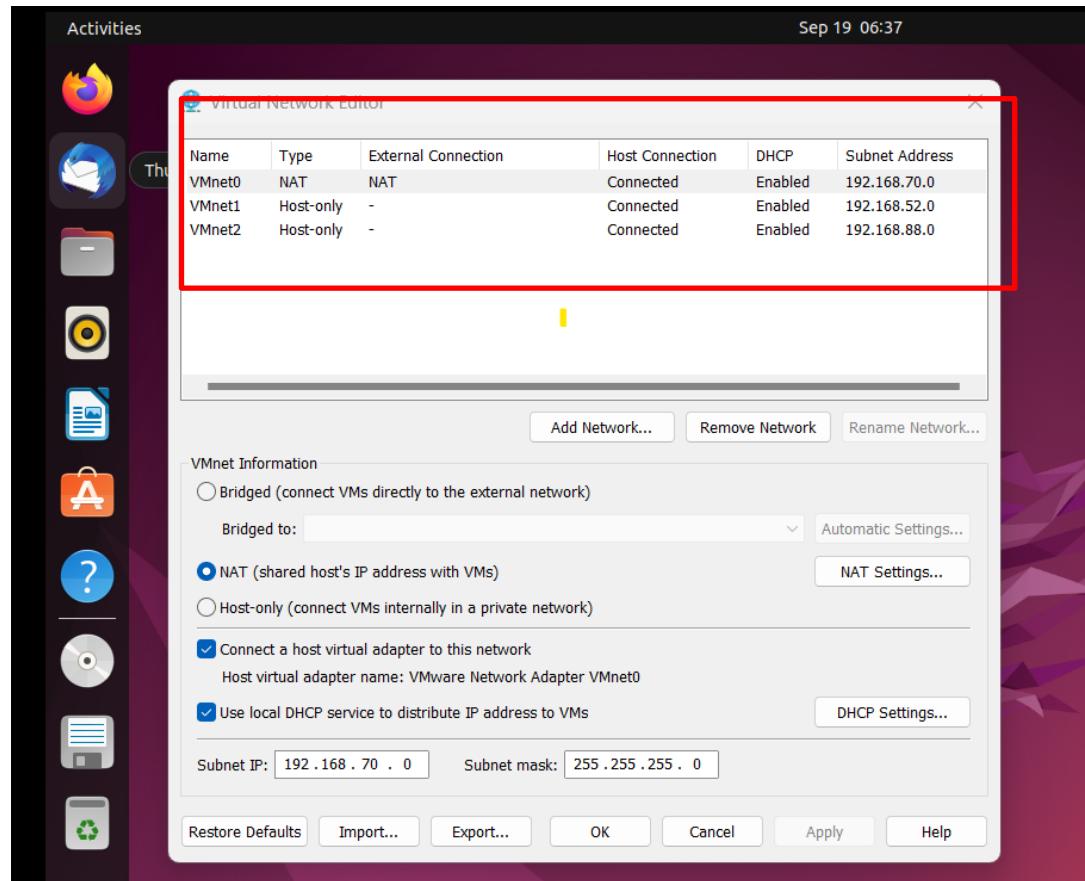
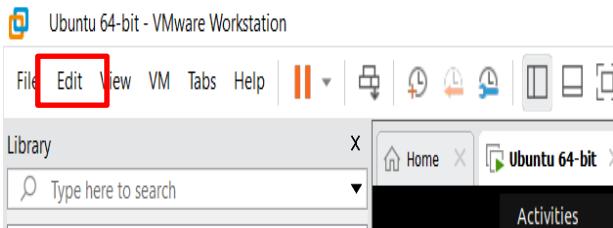
## Setting Up Internet on VM Ware

[Back To TOC](#)

# Setting Up Internet

- If you do not have internet connection in your VM then you can do the following:

## Step 1: Choose EDIT from Main Menu then Choose Virtual Network Editor



### Make sure you have:

- 3 networks:
  - VMnet0 (Set to NAT)
  - VMnet1 (set to Host Only)
  - VMnet2 (Set to Host Only)
- All should be **Connected**
- Click Apply**

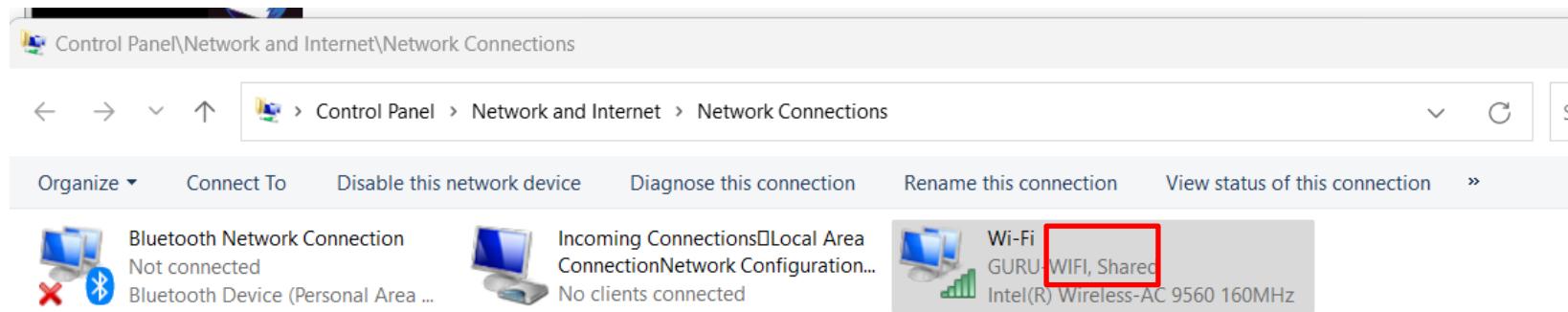
[Back To TOC](#)

# Setting Up Internet

- If you do not have internet connection in your VM then you can do the following:

**Step 2: Go to Control Panel -> Network and Internet -> Network Connections**

- If using Wi-Fi – make sure Wi-Fi is **Sharing the Connection**



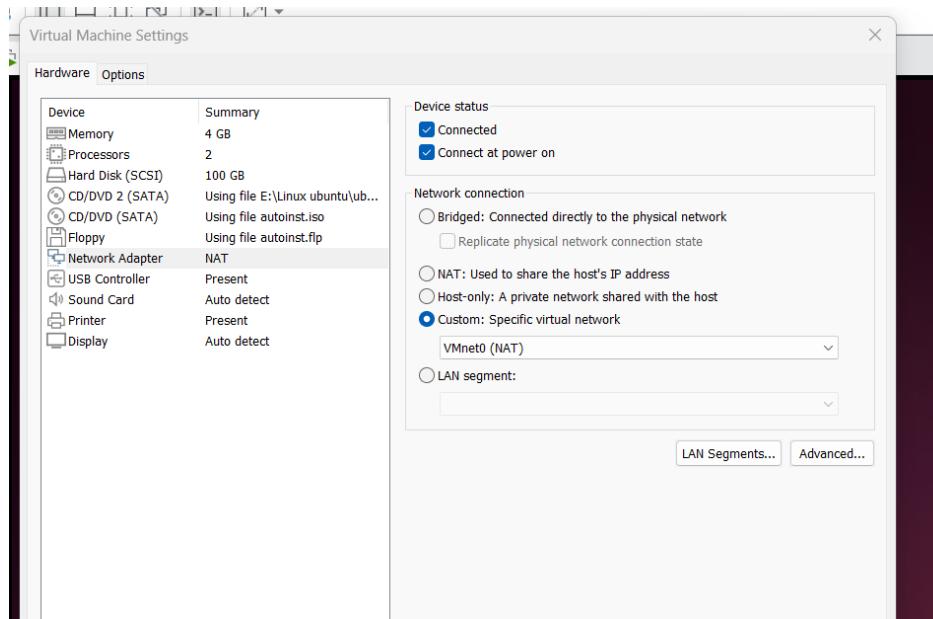
[Back To TOC](#)

# Setting Up Internet

- If you do not have internet connection in your VM then you can do the following:

**Step 3: Go to VM -> Settings from Main Menu and Choose Network Adapter**

- **Select Custom -> Choose VMNet0 -> Click OK**

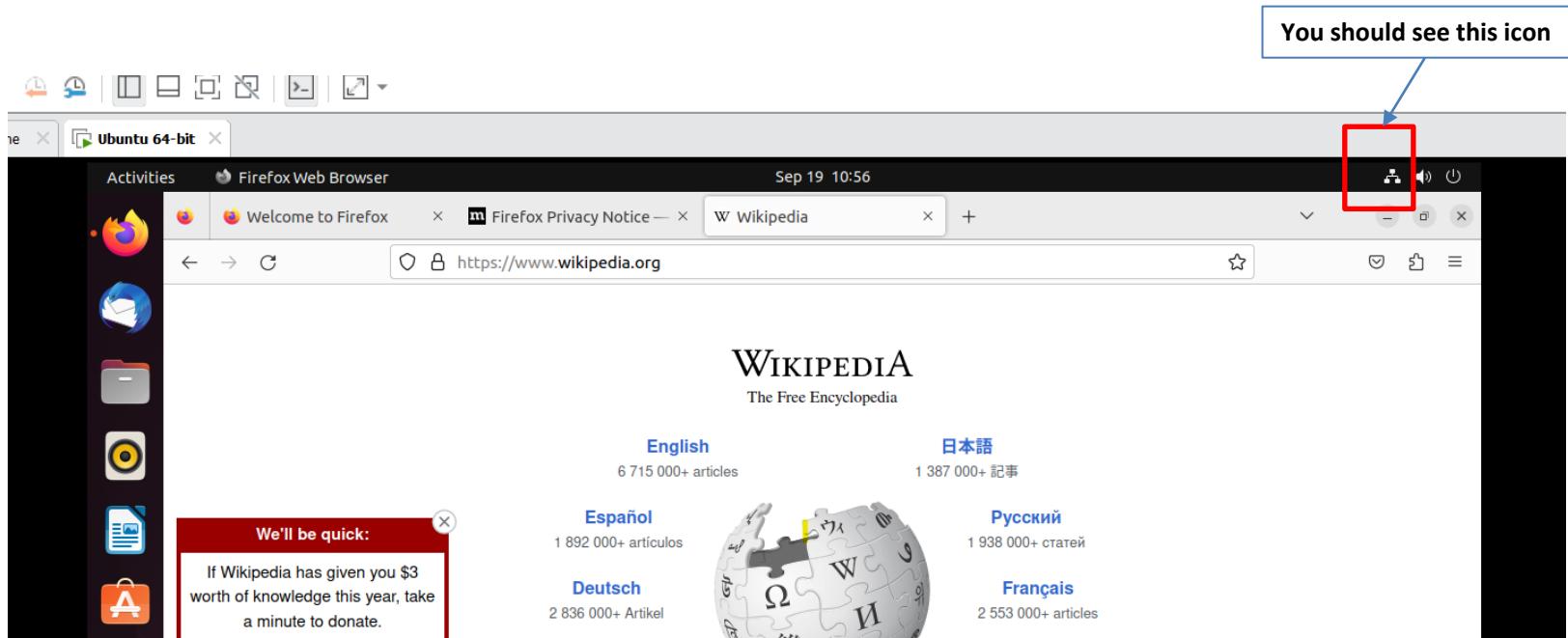


[Back To TOC](#)

# Setting Up Internet

- If you do not have internet connection in your VM then you can do the following:

## Step 4: You Should Now Have Internet!



[Back To TOC](#)

# APPENDIX C

Vmware: Pulling/Running TML Docker Container  
and TML Streaming Dashboard

[Back To TOC](#)

# Running TML Docker Container and Streaming Dashboard

## Step 1: Start Terminal in your VM



[Back To TOC](#)

# Running TML Docker Container and Streaming Dashboard

## Step 2: Install docker

- **FIRST RUN:** sudo apt-get update
- **THEN RUN:** sudo apt-get install docker.io docker.compose

```
seb@seb-virtual-machine:~$ sudo apt-get update
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:2 http://us.archive.ubuntu.com/ubuntu jammy InRelease [270 kB]
Get:3 http://us.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:4 http://us.archive.ubuntu.com/ubuntu jammy-backports InRelease [109 kB]
Get:5 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [802 kB]
Get:6 http://security.ubuntu.com/ubuntu jammy-security/main i386 Packages [327 kB]
Get:7 http://security.ubuntu.com/ubuntu jammy-security/main Translation-en [168 kB]
```



[Back To TOC](#)

# Running TML Docker Container and Streaming Dashboard

## Step 3: Change permissions on docker.sock

- **RUN:** sudo chmod 666 /var/run/docker.sock
- **THEN RUN:** docker ps

A screenshot of a terminal window with a dark background and purple geometric patterns. The terminal shows two commands being run: 'sudo chmod 666 /var/run/docker.sock' and 'docker ps'. The output of 'docker ps' is shown, listing columns for CONTAINER ID, IMAGE, COMMAND, CREATED, STATUS, PORTS, and NAMES. The status column shows one container as 'Up 10 seconds'.

```
seb@seb-virtual-machine:~$ sudo chmod 666 /var/run/docker.sock
seb@seb-virtual-machine:~$ docker ps
CONTAINER ID        IMAGE         COMMAND      CREATED          STATUS          PORTS     NAMES
seb@seb-virtual-machine:~$
```

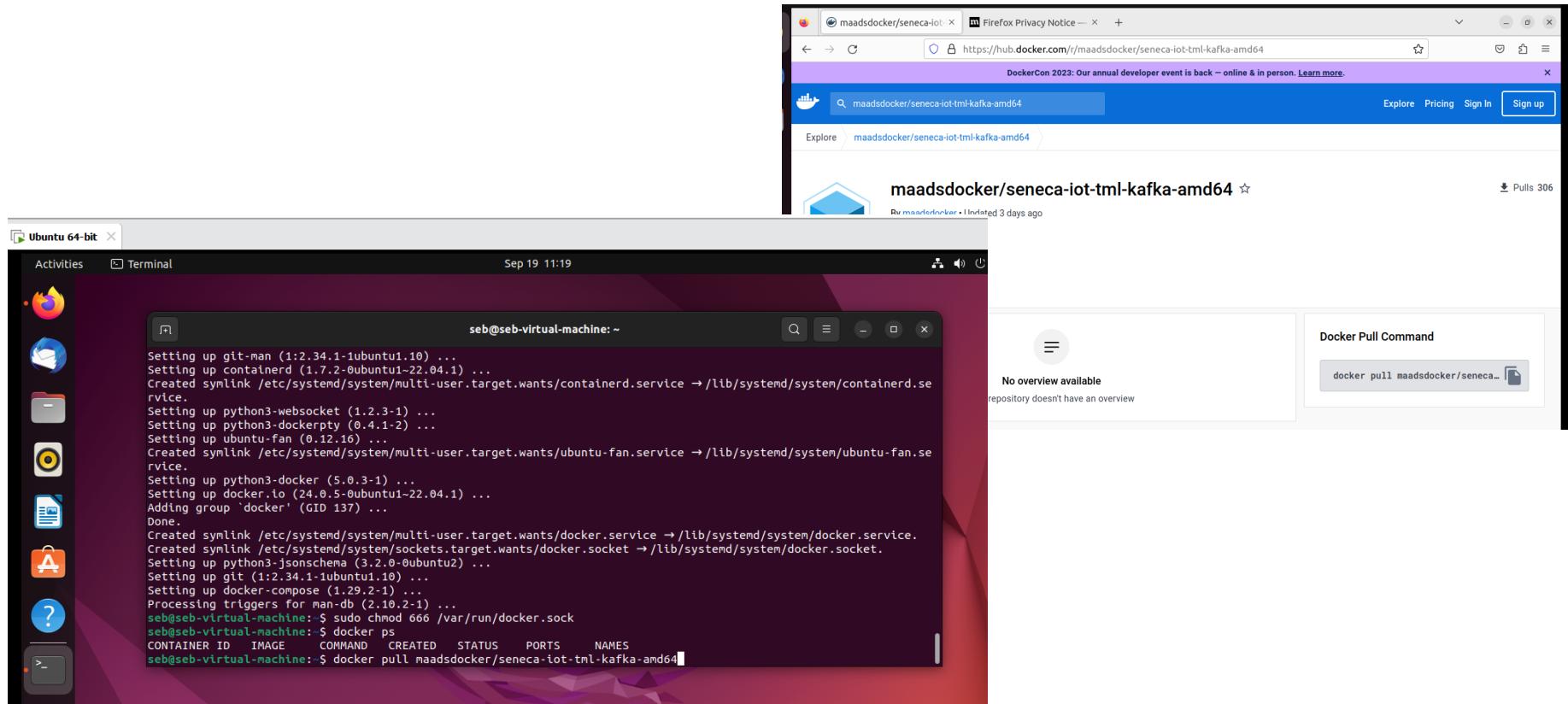
[Back To TOC](#)

# Running TML Docker Container and Streaming Dashboard

Step 4: You can now pull the TML Container on Docker Hub:

<https://hub.docker.com/maadsdocker/seneca-iot-tml-kafka-amd64>

- **RUN:** docker pull maadsdocker/seneca-iot-tml-kafka-amd64



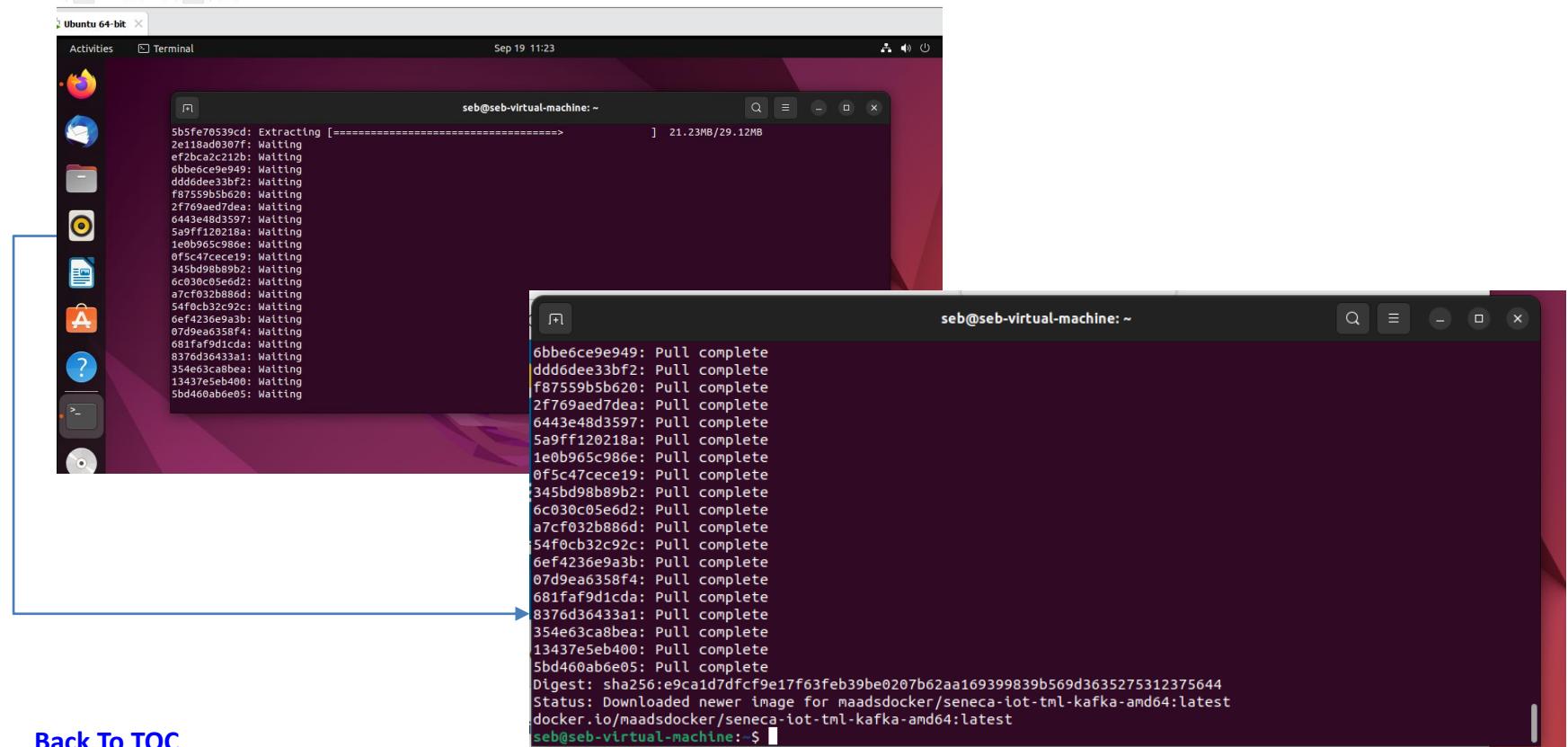
[Back To TOC](#)

# Running TML Docker Container and Streaming Dashboard

Step 4: You can now pull the TML Container on Docker Hub:

<https://hub.docker.com/maadsdocker/seneca-iot-tml-kafka-amd64>

You are now pulling the container (this can take about ~10 minutes):



```
5b5fe70539cd: Extracting [=====] 21.23MB/29.12MB
2e118ad0307f: Waiting
ef2bc2a2c212b: Waiting
6bbe6ce9e949: Waiting
ddd6dee33bf2: Waiting
f87559b5b620: Waiting
2f769aed7dea: Waiting
6443e48d3597: Waiting
5a9ff120218a: Waiting
1e0b965c986e: Waiting
0fc5c47cece19: Waiting
345bd98bb89b2: Waiting
6c03c05e6d2: Waiting
a7cf032b886d: Waiting
54f0cb32c92c: Waiting
6ef4236e9a3b: Waiting
07d9ea6358f4: Waiting
681faf9d1cda: Waiting
8376d3e433a1: Waiting
354e63ca8bea: Waiting
13437e5eb400: Waiting
5bd460ab6e05: Waiting

6bbe6ce9e949: Pull complete
ddd6dee33bf2: Pull complete
f87559b5b620: Pull complete
2f769aed7dea: Pull complete
6443e48d3597: Pull complete
5a9ff120218a: Pull complete
1e0b965c986e: Pull complete
0fc5c47cece19: Pull complete
345bd98bb89b2: Pull complete
6c03c05e6d2: Pull complete
a7cf032b886d: Pull complete
54f0cb32c92c: Pull complete
6ef4236e9a3b: Pull complete
07d9ea6358f4: Pull complete
681faf9d1cda: Pull complete
8376d3e433a1: Pull complete
354e63ca8bea: Pull complete
13437e5eb400: Pull complete
5bd460ab6e05: Pull complete
Digest: sha256:e9ca1d7dfcf9e17f63feb39be0207b62aa169399839b569d3635275312375644
Status: Downloaded newer image for maadsdocker/seneca-iot-tml-kafka-amd64:latest
docker.io/maadsdocker/seneca-iot-tml-kafka-amd64:latest
```

[Back To TOC](#)

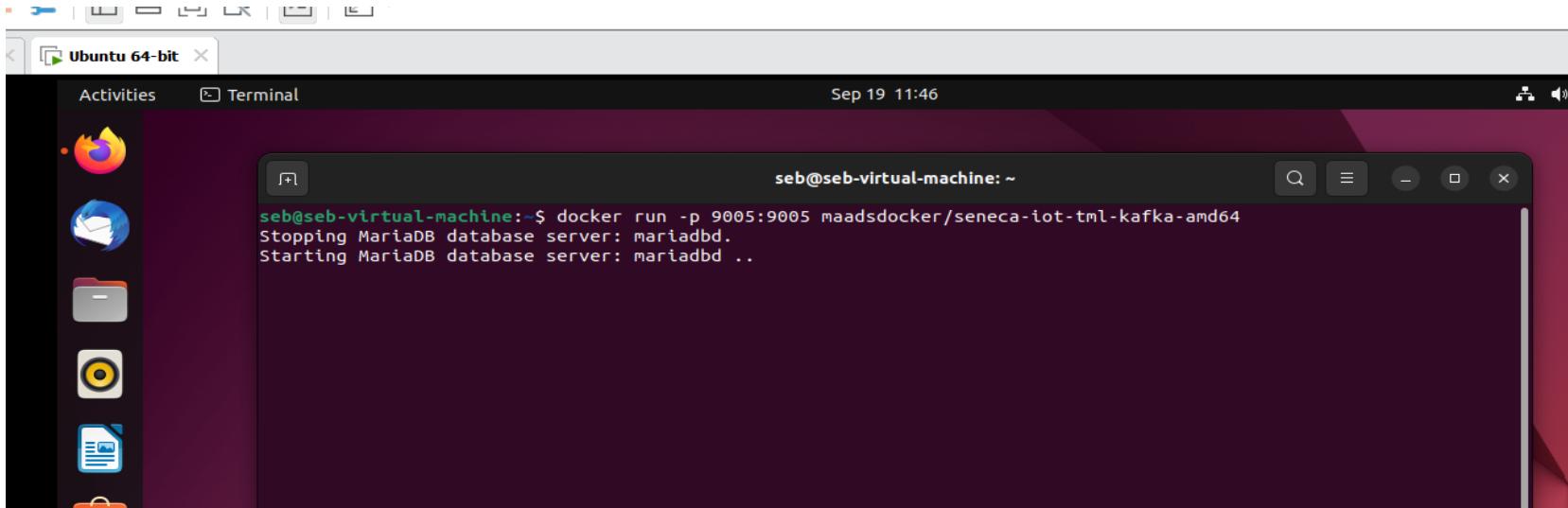
# Running TML Docker Container

## Step 5: You can now Run the container

**RUN:** docker run -p 9005:9005 maadsdocker/seneca-iot-tml-kafka-amd64

**NOTE:** -p is needed for port forwarding: **-p 9005:9005** will tell Docker that connections on the host port 9005 will be forwarded to the Container port 9005.

- **MAADS-Viperviz is listening on Port 9005 inside the container**



[Back To TOC](#)

# Running TML Streaming Dashboard

## Step 6: You can now Run the Dashboard

- Open Firefox Browser and Enter the URL: <http://localhost:9005/iot-failure-seneca.html?topic=iot-preprocess2,iot-preprocess&offset=-1&groupid=&rollbackoffset=500&topicstype=prediction&append=0&secure=1>
- You should see the TML streaming dashboard

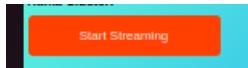


- The TML Dashboard is a live streaming dashboard running from your Docker Container
- It is analysing REAL IoT device data for failures by analysing their VOLTAGE, CURRENT and POWER data in real-time
- It shows RED, YELLOW, GREEN bubbles to indicate which devices are likely to fail (RED)
- It used Preprocessing type: AnomProb, Trend, and AVG to process streaming data from EACH device (at the entity level)

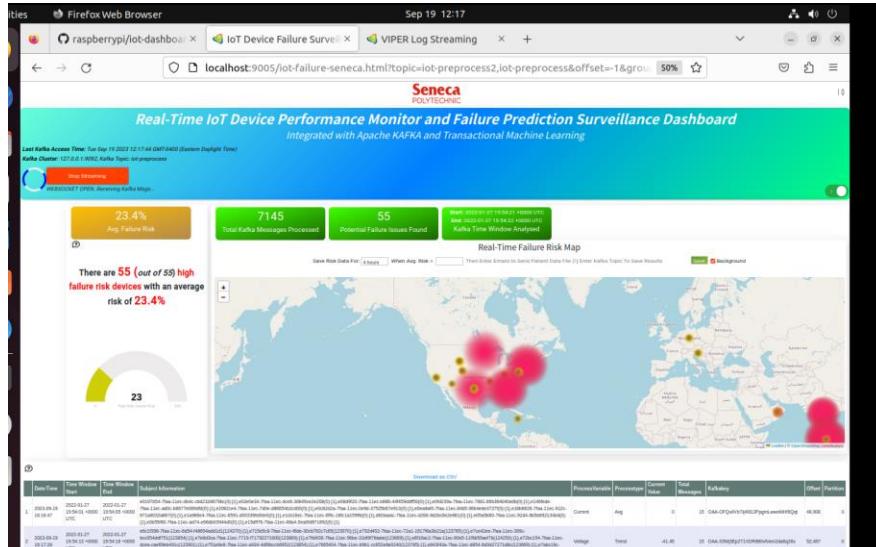
[Back To TOC](#)

# Running TML Streaming Dashboard

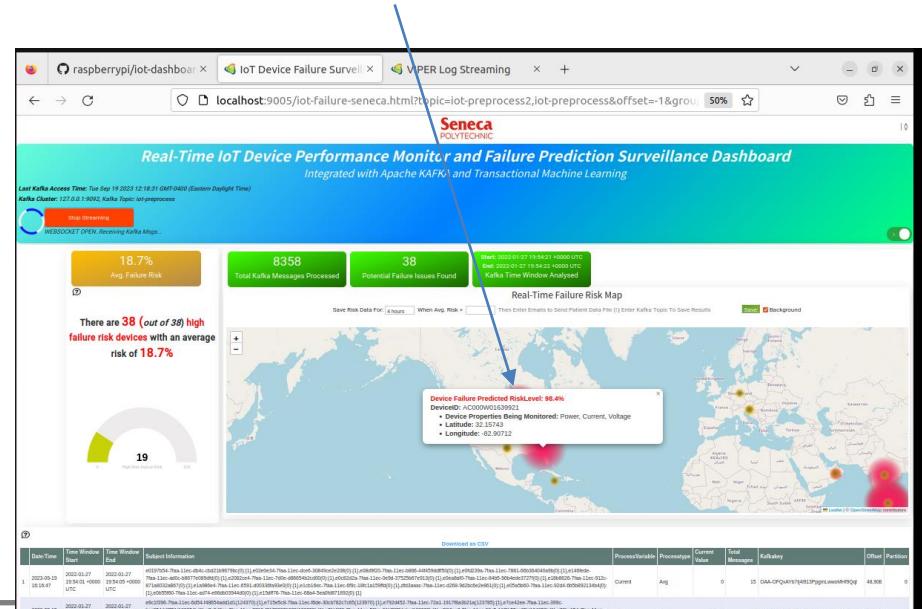
**Step 7: Click the Button: START STREAMING and wait few seconds**



**NOTE: If you don't see enough data – just be patient – you can also Press STOP and START STREAMING the MAP will start to populate**



## Hover over the map



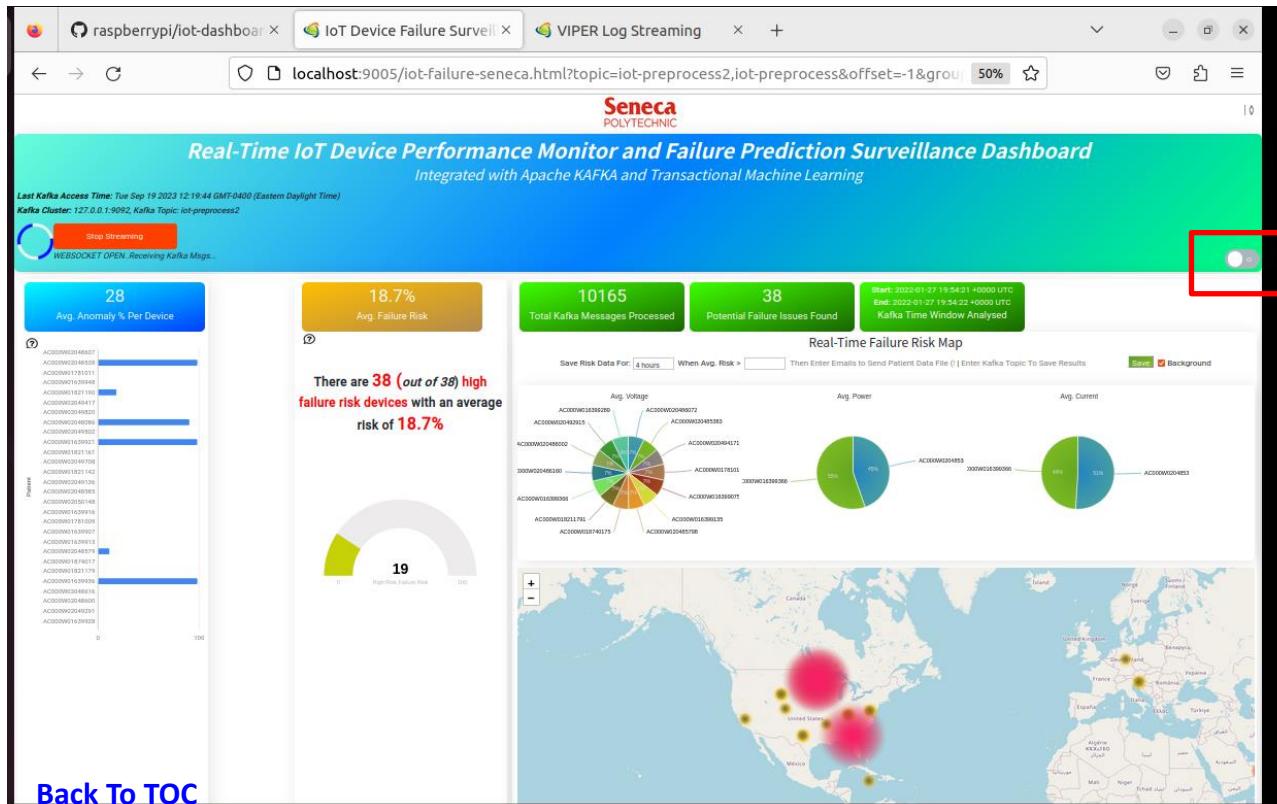
[Back To TOC](#)

# Running TML Streaming Dashboard

Step 7: Click the Button: START STREAMING and wait few seconds



**NOTE: If you don't see enough data – just be patient – you can also Press STOP and START STREAMING - the MAP will start to populate**



# Running TML Streaming Dashboard

- **Scroll down to see the table**
  - **You can also DOWNLOAD the table data to CSV**

# processstypes

The screenshot shows a Firefox browser window with two tabs open: "raspberrypi/iot-dashboard" and "IoT Device Failure Surveillance". The main content area displays a table of log entries from a Kafka topic. The table has columns for Date/Time, Time Window Start, Time Window End, Subject Information, ProcessVariables, ProcessType, Current Value, Total Messages, KafkaKey, Offset, and Partition. A red box highlights a specific row in the table, and a blue arrow points towards it from the top right. The background features a map of Africa and a status bar at the bottom.

2020.R2

# Running TML Streaming Dashboard

You also view TML Log Data – Open a New Tab in Firefox and enter:

<http://localhost:9005/viperlogs.html?topic=viperlogs&append=0>

Service	Service Host	Service Port	Kafka Cluster	Offset	Partition
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.181	0
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.180	0
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.179	0
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.178	0
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.177	0
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.176	0
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.175	0
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.174	0
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.173	0
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.172	0
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.171	0
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.170	0
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.169	0
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.168	0
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.167	0
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.166	0
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.165	0
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.164	0
VIPER	172.17.0.2	38.573	127.0.0.1:9092	1.163	0

[Back To TOC](#)

# APPENDIX D

## Vmware: Building Your Own Container

[Back To TOC](#)

# Building Your Own Container

Step 1: Create Docker directory and copy Dockerfile

Create Docker folder in your VM

**RUN:** `mkdir docker`

```
seb@seb-virtual-machine:~$ mkdir docker
seb@seb-virtual-machine:~$ ls
Desktop  docker  Documents  Downloads  kubernetes  Music  Pictures  Public  snap  Templates  Videos
seb@seb-virtual-machine:~$
```

Go to: <https://github.com/smaurice101/raspberrypi/blob/main/docker/Dockerfile>

1. **COPY** Dockerfile contents from GitHub to your Local machine

1. **RUN: cd docker**
2. **RUN: nano Dockerfile**
3. Right – click and **PASTE** contents
4. **Press these keys in nano: Ctlr + O THEN Ctlr + X <choose YES both times>**

You should see in the Docker folder your “Dockerfile” and ready for Build:

```
seb@seb-virtual-machine:~$ cd docker
seb@seb-virtual-machine:~/docker$ ls
Dockerfile
seb@seb-virtual-machine:~/docker$
```

[Back To TOC](#)

# Building Your Own Container

## Step 2: To build your own container

**RUN: docker build -t <container name> --build-arg CHIP=<chip arch> --no-cache --network=host .**

**For example RUN: docker build -t maadsdocker/seneca-iot-tml-test --build-arg CHIP=AMD64 --no-cache --network=host .**

**MAC USERS: docker build -t maadsdocker/seneca-iot-tml-test --build-arg CHIP=MAC --no-cache --network=host .**

```
seb@seb-virtual-machine:~/docker$ docker build -t maadsdocker/seneca-iot-tml-test --build-arg CHIP=AMD64 --no-cache  
--network=host .
```

- **maadsdocker** is my Dockerhub username (replace it with your Docker hub username)
- **seneca-iot-tml-test** is the name of my container – you can choose any name like:
  - **docker build -t maadsdocker/daffyduck --build-arg CHIP=AMD64 --no-cache --network=host .**

```
NOTE: You MUST have Docker installed to run this command.  
seb@seb-virtual-machine:~/docker$ docker build -t maadsdocker/seneca-iot-tml-test --build-arg CHIP=AMD64 --no-cache  
--network=host .  
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.  
Install the buildx component to build images with BuildKit:  
https://docs.docker.com/go/buildx/  
  
Sending build context to Docker daemon 10.24kB  
Step 1/19 : FROM python:3.9-slim  
3.9-slim: Pulling from library/python  
a803e7c4b030: Extracting [>  
bf3336e84c8e: Download complete  
3614ca5053cf: Download complete  
7f93433c11f3: Download complete  
2fd2c896255c: Download complete
```

--network=host

this command

[Back To TOC](#)

# Building Your Own Container

Step 3: To build your own container faster

AFTER You Build your container you can use a FASTER build – this is useful if you are making frequent changes to your container. Fast builds will take ~5 minutes.

**RUN: docker build -t <container name> --build-arg CHIP=<chip arch> --build-arg CACHEBUST=\$(date +%s) --network=host .**

Form example: **docker build -t maadsdocker/seneca-iot-tml-test --build-arg CHIP=AMD64 --build-arg CACHEBUST=\$(date +%s) --network=host .**

**FOR MAC USERS:** **docker build -t maadsdocker/seneca-iot-tml-test --build-arg CHIP=MAC --build-arg CACHEBUST=\$(date +%s) --network=host .**

[Back To TOC](#)

# Building Your Own Container

## Step 4: Push Your Container to Docker Hub

### FIRST LOGIN TO YOUR DOCKERHUB ACCOUNT FROM YOUR VM:

**RUN: docker login**

```
seb@seb-virtual-machine:~/docker$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: maadsdocker
Password:
WARNING! Your password will be stored unencrypted in /home/seb/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded
```

**NOW RUN: docker push <container name>**

**For example: docker push maadsdocker/seneca-iot-tml-test:latest**

**(Note: Do NOT use sudo)**

```
Successfully built aad47e5f7ec8
Successfully tagged maadsdocker/seneca-iot-tml-test:latest
seb@seb-virtual-machine:~/docker$ 
seb@seb-virtual-machine:~/docker$ docker push maadsdocker/seneca-iot-tml-test:latest
```

```
seb@seb-virtual-machine: ~/docker
Q - X
7522013e3863: Pushing [=====] 25.52MB/36.28MB
bc78a03b550f: Waiting
Back To TOC
```

# Building Your Own Container

## Step 4b: View Your Container in Docker Hub

The screenshot shows the Docker Hub interface for a private repository named `maadsdocker/seneca-iot-tml-test`. The repository has one private repository. The General tab is selected. A callout box highlights the Docker commands section, which contains the command `docker push maadsdocker/seneca-iot-tml-test:tagname`. Other tabs visible include Tags, Builds, Collaborators, Webhooks, and Settings.

maadsdocker / seneca-iot-tml-test

Description

This repository does not have a description

Last pushed: 2 minutes ago

Docker commands

To push a new tag to this repository:

```
docker push maadsdocker/seneca-iot-tml-test:tagname
```

Tags

Automated Builds

[Back To TOC](#)

# Building Your Own Container

Step 5: Run Your Container

**CONGRATULATIONS! You just built your OWN Docker Container!**

**Follow Steps in APPENDIX C using your NEW container name!**

[Back To TOC](#)

# APPENDIX E

- Going Inside the Container

[Back To TOC](#)

# Go Inside the Container

**STEP 1: Open a new terminal window**

**RUN: docker ps**

A screenshot of an Ubuntu 64-bit desktop environment. On the left is a dock with icons for the Dash, Home, Applications, and Files. A terminal window titled "Terminal" is open, showing the command "seb@seb-virtual-machine:~\$ docker ps". The output of the command is displayed in a table:

CONTAINER ID	IMAGE	NAMES	COMMAND	CREATED	STATUS
59f1ac48f086	maadsdocker/seneca-iot-tml-kafka-amd64		"/bin/bash -c 'while..."	35 minutes ago	Up 35 minutes
0.0.0.0:9005->9005/tcp, :::9005->9005/tcp		happy_gauss			

seb@seb-virtual-machine:~\$

[Back To TOC](#)

# Go Inside the Container

## STEP 2: Go inside the container

**RUN: docker exec -it <enter CONTAINER ID>**

- **For example: docker exec -it 59f1ac48f086 bash**
- **You are now inside the container!**
- **RUN: ls** (to see folders inside the container)

```
seb@seb-virtual-machine:~$ docker exec -it 59f1ac48f086 bash
root@59f1ac48f086:/# █
```

```
root@59f1ac48f086:/# ls
Hpde      Viper-preprocess  Viperviz  deploy  home   lib64    mnt    root    srv    tmux
IotSolution  Viper-preprocess2  bin       dev     lib     libx32   opt     run     sys    usr
Kafka      Viper-produce      boot     etc     lib32   media   proc    sbin   tmp    var
root@59f1ac48f086:/#
```

[Back To TOC](#)

# Go Inside the Container

## STEP 3: List TMUX windows inside the container

**RUN: tmux ls**

- **These windows are your TML Solution running in real-time!**

```
root@59f1ac48f086:/# tmux ls
kafka: 1 windows (created Tue Sep 19 15:46:24 2023)
preprocess-data-python-8001: 1 windows (created Tue Sep 19 15:46:41 2023)
preprocess-data-viper-8001: 1 windows (created Tue Sep 19 15:46:34 2023)
preprocess2-data-python-8002: 1 windows (created Tue Sep 19 15:46:41 2023)
preprocess2-data-viper-8002: 1 windows (created Tue Sep 19 15:46:34 2023)
produce-iot-data-python-8000: 1 windows (created Tue Sep 19 15:46:41 2023)
produce-iot-data-viper-8000: 1 windows (created Tue Sep 19 15:46:34 2023)
visualization-viperviz-9005: 1 windows (created Tue Sep 19 15:46:41 2023)
zookeeper: 1 windows (created Tue Sep 19 15:46:20 2023)
root@59f1ac48f086:/# █
```

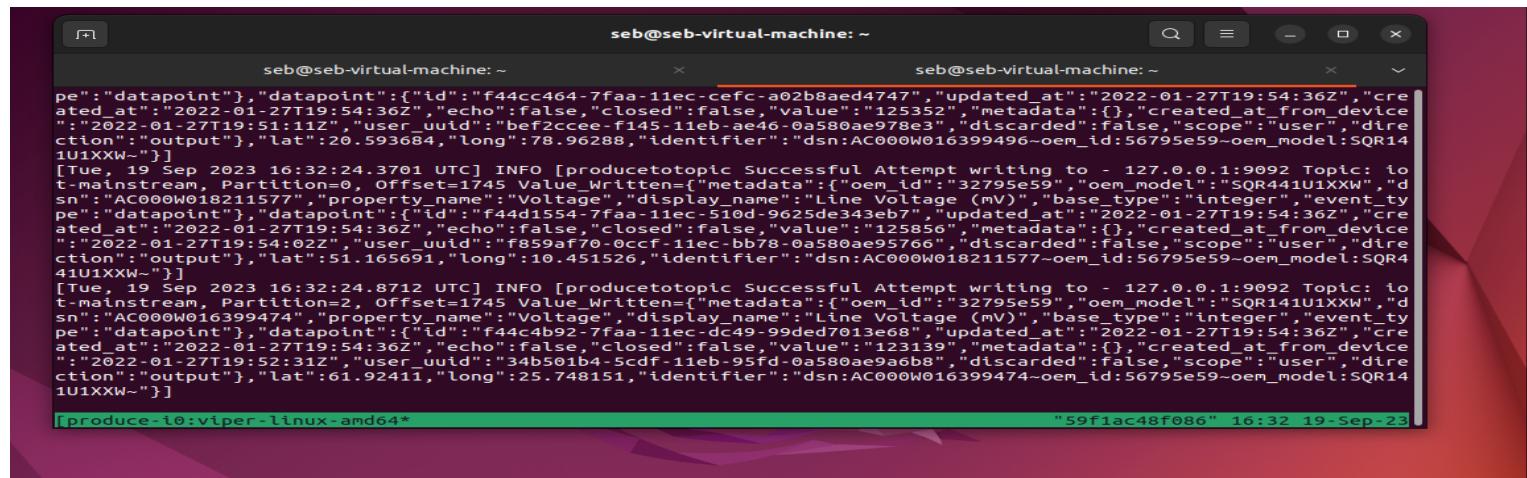
[Back To TOC](#)

# Go Inside the Container

## STEP 4: Go inside a TMUX window inside the container

**RUN: tmux a -t produce-iot-data-viper-8000**

- This is the RAW IoT data that is streaming to Kafka all running inside your container.
- To Detach from the TMUX window press at the same time: **Ctrl+B** (only press once and then let go of these buttons), **THEN** just press **D**



The screenshot shows a tmux session with two windows. The left window displays raw IoT data being written to Kafka topics. The right window shows the command used to run the producer. A blue arrow points from the bottom terminal window to the tmux session.

```
seb@seb-virtual-machine: ~
[Thu, 21 Sep 2023 16:32:24.3701 UTC] INFO [producetotopic] Successful Attempt writing to - 127.0.0.1:9092 Topic: iot-mainstream, Partition=0, Offset=1745 Value_Written={"metadata":{"oem_id": "32795e59", "oem_model": "SQR441u1XXW", "dsn": "AC000W018211577"}, "property_name": "Voltage", "display_name": "Line Voltage (mV)", "base_type": "integer", "event_type": "datapoint"}, "datapoint": {"id": "f44cc464-7faa-11ec-a02b8aed4747", "updated_at": "2022-01-27T19:54:36Z", "created_at": "2022-01-27T19:51:11Z", "user_uuid": "bef2ccee-f145-11eb-a4e6-0a580ae978e3", "discarded": false, "scope": "user", "direction": "output"}, "lat": 26.593684, "long": 78.96288, "identifier": "dsn:AC000W016399496-oem_id:56795e59-oem_model:SQR141u1XXW~"}
[Tue, 19 Sep 2023 16:32:24.3701 UTC] INFO [producetotopic] Successful Attempt writing to - 127.0.0.1:9092 Topic: iot-mainstream, Partition=0, Offset=1745 Value_Written={"metadata":{"oem_id": "32795e59", "oem_model": "SQR441u1XXW", "dsn": "AC000W018211577"}, "property_name": "Voltage", "display_name": "Line Voltage (mV)", "base_type": "integer", "event_type": "datapoint"}, "datapoint": {"id": "f44d1554-7faa-11ec-510d-9625de343eb7", "updated_at": "2022-01-27T19:54:36Z", "created_at": "2022-01-27T19:54:02Z", "user_uuid": "f859af70-0acf-11ec-bb78-0a580ae95766", "discarded": false, "scope": "user", "direction": "output"}, "lat": 51.165691, "long": 10.451526, "identifier": "dsn:AC000W018211577-oem_id:56795e59-oem_model:SQR441u1XXW~"}
[Tue, 19 Sep 2023 16:32:24.8712 UTC] INFO [producetotopic] Successful Attempt writing to - 127.0.0.1:9092 Topic: iot-mainstream, Partition=2, Offset=1745 Value_Written={"metadata":{"oem_id": "32795e59", "oem_model": "SQR141u1XXW", "dsn": "AC000W016399474"}, "property_name": "Voltage", "display_name": "Line Voltage (mV)", "base_type": "integer", "event_type": "datapoint"}, "datapoint": {"id": "f44c4b92-7faa-11ec-dc49-99ded7013e68", "updated_at": "2022-01-27T19:54:36Z", "created_at": "2022-01-27T19:52:31Z", "user_uuid": "34b501b4-5cdf-11eb-95fd-0a580ae9a6b8", "discarded": false, "scope": "user", "direction": "output"}, "lat": 61.92411, "long": 25.748151, "identifier": "dsn:AC000W016399474-oem_id:56795e59-oem_model:SQR141u1XXW~"}]
[produce-10:viper-linux-amd64* 59f1ac48f086" 16:32 19-Sep-23]
```

```
root@59f1ac48f086:/# tmux a -t produce-iot-data-viper-8000
[detached (from session produce-iot-data-viper-8000)]
root@59f1ac48f086:/#
```

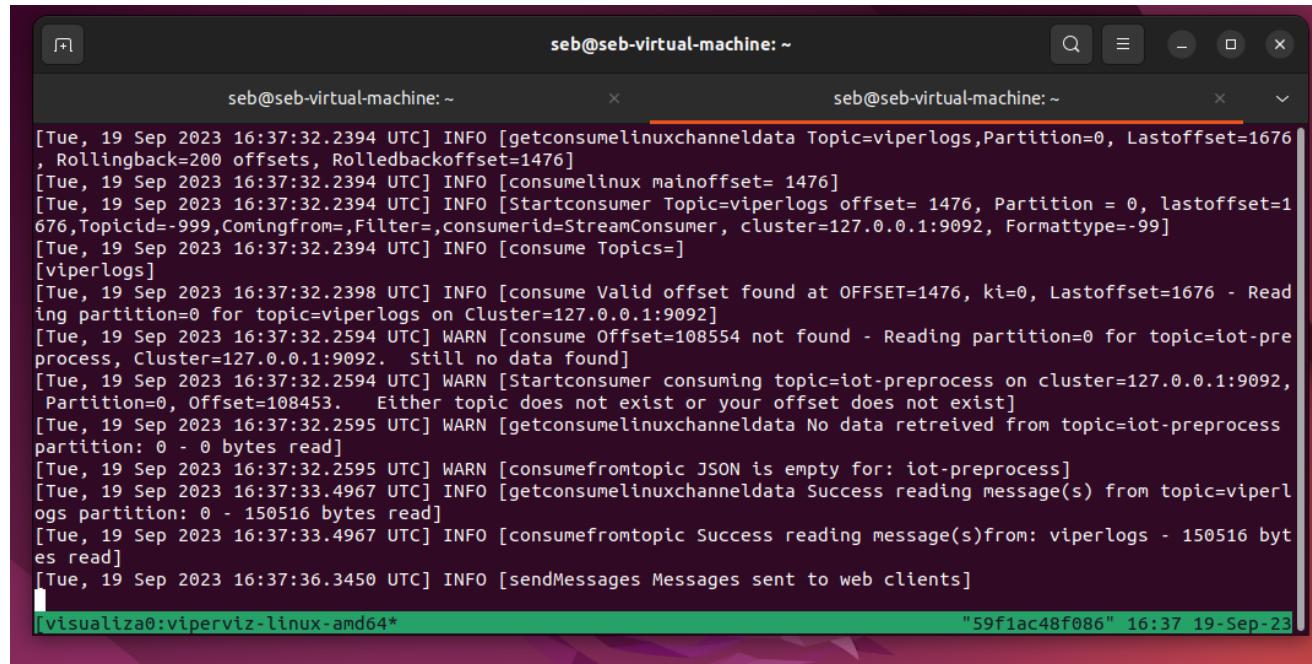
[Back To TOC](#)

# Go Inside the Container

**STEP 5: Go inside another TMUX window inside the container**

**RUN: tmux a -t visualization-viperviz-9005**

- This is the MAADS-Viperviz binary that is powering your Dashboard.**



The screenshot shows a tmux session with two panes. The left pane displays log messages from the MAADS-Viperviz binary, which is consuming data from a Kafka topic named 'viperlogs'. The right pane shows a terminal prompt for 'seb@seb-virtual-machine'. The bottom status bar indicates the session ID is '59f1ac48f086' and the current time is '16:37 19-Sep-23'.

```
[Tue, 19 Sep 2023 16:37:32.2394 UTC] INFO [getconsumelinuxchanneldata Topic=viperlogs,Partition=0, Lastoffset=1676, Rollingback=200 offsets, Rolledbackoffset=1476]
[Tue, 19 Sep 2023 16:37:32.2394 UTC] INFO [consumelinux mainoffset= 1476]
[Tue, 19 Sep 2023 16:37:32.2394 UTC] INFO [Startconsumer Topic=viperlogs offset= 1476, Partition = 0, lastoffset=1676,Topicid=-999,Comingfrom=,Filter=,consumerid=StreamConsumer, cluster=127.0.0.1:9092, Formattype=-99]
[Tue, 19 Sep 2023 16:37:32.2394 UTC] INFO [consume Topics=]
[viperlogs]
[Tue, 19 Sep 2023 16:37:32.2398 UTC] INFO [consume Valid offset found at OFFSET=1476, ki=0, Lastoffset=1676 - Reading partition=0 for topic=viperlogs on Cluster=127.0.0.1:9092]
[Tue, 19 Sep 2023 16:37:32.2594 UTC] WARN [consume Offset=108554 not found - Reading partition=0 for topic=iot-preprocess, Cluster=127.0.0.1:9092. Still no data found]
[Tue, 19 Sep 2023 16:37:32.2594 UTC] WARN [Startconsumer consuming topic=iot-preprocess on cluster=127.0.0.1:9092, Partition=0, Offset=108453. Either topic does not exist or your offset does not exist]
[Tue, 19 Sep 2023 16:37:32.2595 UTC] WARN [getconsumelinuxchanneldata No data retrieved from topic=iot-preprocess partition: 0 - 0 bytes read]
[Tue, 19 Sep 2023 16:37:32.2595 UTC] WARN [consumefromtopic JSON is empty for: iot-preprocess]
[Tue, 19 Sep 2023 16:37:33.4967 UTC] INFO [getconsumelinuxchanneldata Success reading message(s) from topic=viperlogs partition: 0 - 150516 bytes read]
[Tue, 19 Sep 2023 16:37:33.4967 UTC] INFO [consumefromtopic Success reading message(s)from: viperlogs - 150516 bytes read]
[Tue, 19 Sep 2023 16:37:36.3450 UTC] INFO [sendMessages Messages sent to web clients]
```

```
zookeeper:~# windows (created Tue Sep 19 15:47:20 2023)
root@59f1ac48f086:/# tmux a -t produce-iot-data-viper-8000
[detached (from session produce-iot-data-viper-8000)]
root@59f1ac48f086:/# tmux a -t visualization-viperviz-9005
[detached (from session visualization-viperviz-9005)]
root@59f1ac48f086:/#
```

[Back To TOC](#)

# Go Inside the Container

**STEP 6: Exit from your container**

**RUN: exit**



```
[root@59f1ac48f086:/] exit
exit
seb@seb-virtual-machine:~$
```

A screenshot of a terminal window with a dark background and purple/red geometric patterns at the bottom. The terminal shows a root shell on a VM, with the command 'exit' being typed and then run. The prompt changes from 'root@...' to 'seb@...'.

The terminal window displays the following text:

```
[root@59f1ac48f086:/] exit
exit
seb@seb-virtual-machine:~$
```

[Back To TOC](#)

# TML - TMUX Window Explanation

- TML solution is controlled by 6 separate TMUX windows running in their own Linux instance:
- The Python windows control the Viper windows
- For example:
  - **produce-iot-data-python-8000** uses REST API to control **produce-iot-data-viper-8000**
  - **preprocess-data-python-8001** uses REST API to control **preprocess-data-viper-8001**
  - **preprocess2-data-python-8002** uses REST API to control **preprocess2-data-viper-8002**
- The Dashboard is controlled by the window:
  - **visualization-viperviz-9005**
- The Viper windows have the MAADS-Viper binary running
- The Python windows have the TML python scripts running

```
root@59f1ac48f086:/# tmux ls
kafka: 1 windows (created Tue Sep 19 15:46:24 2023)
preprocess-data-python-8001: 1 windows (created Tue Sep 19 15:46:41 2023)
preprocess-data-viper-8001: 1 windows (created Tue Sep 19 15:46:34 2023)
preprocess2-data-python-8002: 1 windows (created Tue Sep 19 15:46:41 2023)
preprocess2-data-viper-8002: 1 windows (created Tue Sep 19 15:46:34 2023)
produce-iot-data-python-8000: 1 windows (created Tue Sep 19 15:46:41 2023)
produce-iot-data-viper-8000: 1 windows (created Tue Sep 19 15:46:34 2023)
visualization-viperviz-9005: 1 windows (created Tue Sep 19 15:46:41 2023)
zookeeper: 1 windows (created Tue Sep 19 15:46:20 2023)
```

[Back To TOC](#)

# APPENDIX F

- Kubernetes setup with Docker Container

[Back To TOC](#)

# Kubernetes Setup

## STEP 1: Create kubernetes directory

First RUN: **mkdir kubernetes**

Then RUN: **ls**

Then RUN: **cd kubernetes**

```
seb@seb-virtual-machine:~$ mkdir kubernetes
```

```
seb@seb-virtual-machine:~$ ls
Desktop  Documents  Downloads  kubernetes  Music  Pictures  Public  snap  Templates  Videos
seb@seb-virtual-machine:~$
```

```
seb@seb-virtual-machine:~/kubernetes$
```

[Back To TOC](#)

# Kubernetes Setup

## STEP 2: install Kubernetes (minikube)

**Note:** minikube is a One node Kubernetes cluster – it is meant for development and testing – it is the SAME as a production grade Kubernetes clusters with the same functionality.

**DOWNLOAD MINIKUBE in Kubernetes folder:**

**RUN:** `wget https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64`

**Install minikube:**

**RUN:** `sudo install minikube-linux-amd64 minikube`

**You should now see:**

```
seb@seb-virtual-machine:~/kubernetes$ ls  
minikube  minikube-linux-amd64
```

[Back To TOC](#)

# Kubernetes Setup

## STEP 3: install kubectl

**Note:** kubectl is a command line tool that is very popular and controls all functionality of Kubernetes (minikube)

**DOWNLOAD Kubectl in Kubernetes folder:**

**RUN:** curl -LO <https://storage.googleapis.com/kubernetes-release/release/> curl -s <https://storage.googleapis.com/kubernetes-release/release/stable.txt> `bin/linux/amd64/kubectl

**NOTE IF YOU DO NOT HAVE curl you can install it like this:**

- a. sudo apt update && sudo apt upgrade
- b. sudo apt install curl

**RUN: chmod +x kubectl**

**RUN: sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl**

**You should now see:**

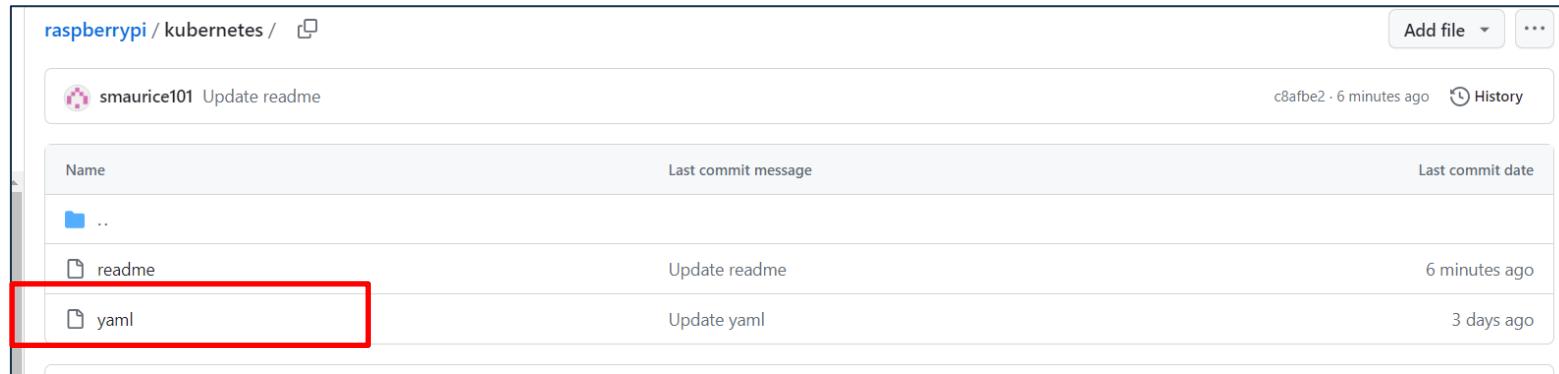
```
seb@seb-virtual-machine:~/kubernetes$ ls  
kubectl  minikube  minikube-linux-amd64
```

[Back To TOC](#)

# Kubernetes Setup

## STEP 4: Download YAML file – Kubernetes (minikube) script

- Go to: <https://github.com/smaurice101/raspberrypi/tree/main/kubernetes>



A screenshot of a GitHub repository page for 'raspberrypi / kubernetes'. The 'yaml' file is highlighted with a red box. The table shows the following data:

Name	Last commit message	Last commit date
..		
readme	Update readme	6 minutes ago
<b>yaml</b>	Update yaml	3 days ago

- Click “yaml”
- Download yaml file to your local kubernetes folder
  - **RENAME** this file: yaml → **senecaiot.yml**

You should now see:

```
seb@seb-virtual-machine:~/kubernetes$ ls
kubectl  minikube  minikube-linux-amd64  senecaiot.yml
```

[Back To TOC](#)

# Kubernetes Setup

## STEP 5: Start Kubernetes (minikube)

**RUN: minikube start --driver=docker**

(make sure you are in your Kubernetes folder)

**Note: You must have docker installed – if not go to APPENDIX C and follow those instructions.**

**You should see this:**

```
seb@seb-virtual-machine:~$ minikube start driver=docker
😄 minikube v1.31.2 on Ubuntu 22.04
💡 Using the docker driver based on existing profile
Starting control plane node minikube in cluster minikube
Pulling base image ...
Restarting existing docker container for "minikube" ...
Preparing Kubernetes v1.27.4 on Docker 24.0.4 ...
Configuring bridge CNI (Container Networking Interface) ...
Verifying Kubernetes components...
■ Using image gcr.io/k8s-minikube/storage-provisioner:v5
💡 Enabled addons: default-storageclass, storage-provisioner
🌟 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
seb@seb-virtual-machine:~$
```

**RUN: minikube status**

```
seb@seb-virtual-machine:~/kubernetes$ minikube status
minikube
  type: Control Plane
  host: Running
  kubelet: Running
  apiserver: Running
  kubeconfig: Configured
```

[Back To TOC](#)

# Kubernetes Setup

## STEP 6: Create Kubernetes POD (this is your docker container)

Note: Kubernetes will PULL your docker image and run it. If you open the **senecaiot.yml** file, the Docker image path is located in the “image” field.

**RUN: kubectl apply -f senecaiot.yml**

(make sure you are in your kubernetes folder)

**RUN: kubectl get pods**

You should see this (your Docker container is now running in Kubernetes):

```
seb@seb-virtual-machine:~/kubernetes$ kubectl get pods
NAME                  READY   STATUS    RESTARTS   AGE
seneca-iot-deployment-78757d978d-czht5  1/1     Running   3 (6m9s ago)  2d20h
seb@seb-virtual-machine:~/kubernetes$ █
```

[Back To TOC](#)

# Kubernetes Setup

## STEP 7: Port Forwarding to Run Dashboard

Open a new terminal window

**RUN: kubectl get pods**

Note: the pod NAME in the example it is: **seneca-iot-deployment-78757d978d-czhts** – your pod NAME will be different

**RUN: kubectl port-forward seneca-iot-deployment-78757d978d-czhts 9005:9005**

You should see this:

```
seb@seb-virtual-machine:~/kubernetes$ kubectl port-forward seneca-iot-deployment-78757d978d-czhts 9005:9005
Forwarding from 127.0.0.1:9005 -> 9005
Forwarding from [::1]:9005 -> 9005
```

[Back To TOC](#)

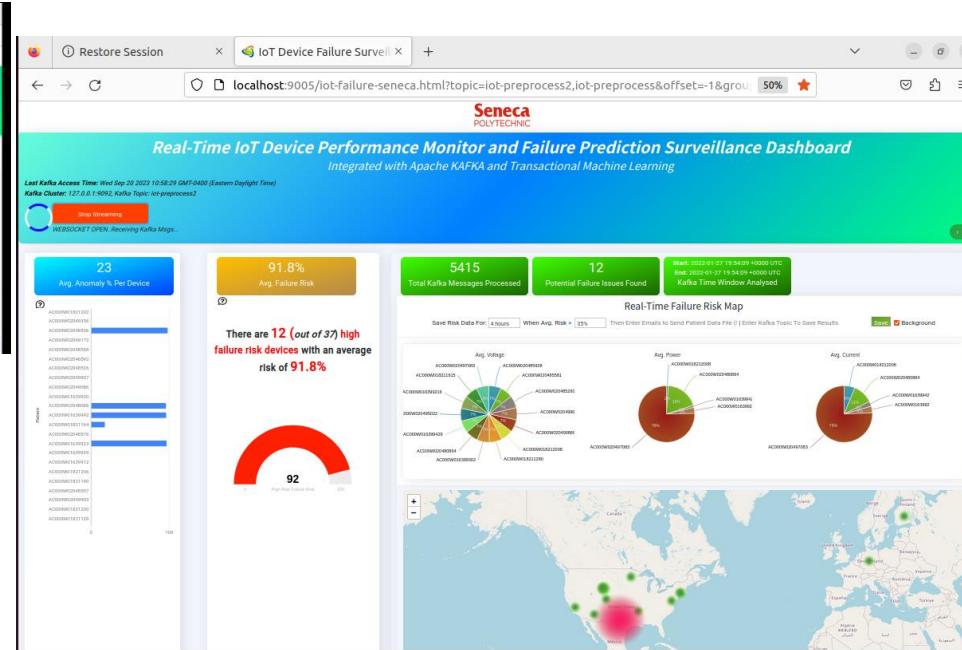
# Kubernetes Setup

## STEP 8: Start Your TML Dashboard

Open a Web Browser in your VM (i.e. Firefox)

**ENTER URL:** <http://localhost:9005/iot-failure-seneca.html?topic=iot-preprocess2,iot-preprocess&offset=-1&groupid=&rollbackoffset=500&topicType=prediction&append=0&secure=1>

You should see this: (As shown in APPENDIX C)



[Back To TOC](#)

# Kubernetes Setup

## STEP 9: Go inside Kubernetes Pod

Open a new terminal window

**RUN: kubectl get pods**

Note: the pod NAME in the example it is: **seneca-iot-deployment-78757d978d-czhts** – your pod NAME will be different

**RUN: kubectl exec -it seneca-iot-deployment-78757d978d-czhts bash**

You should see this: (You are now inside the container running in Kubernetes)

```
seb@seb-virtual-machine:~/kubernetes$ kubectl get pods
NAME                  READY   STATUS    RESTARTS   AGE
seneca-iot-deployment-78757d978d-czhts   1/1     Running   3 (21m ago)   2d20h
seb@seb-virtual-machine:~/kubernetes$ kubectl exec -it seneca-iot-deployment-78757d978d-czhts bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
root@minikube:/# ls
Hpde      Viper-preprocess  Viperviz  deploy  home   lib64    mnt    root  srv  tmux
IotSolution  Viper-preprocess2 bin       dev     lib    libx32   opt    run   sys  usr
Kafka      Viper-produce    boot      etc     lib32   media   proc   sbin  tmp  var
root@minikube:/# 
```

[Back To TOC](#)

# Kubernetes Setup

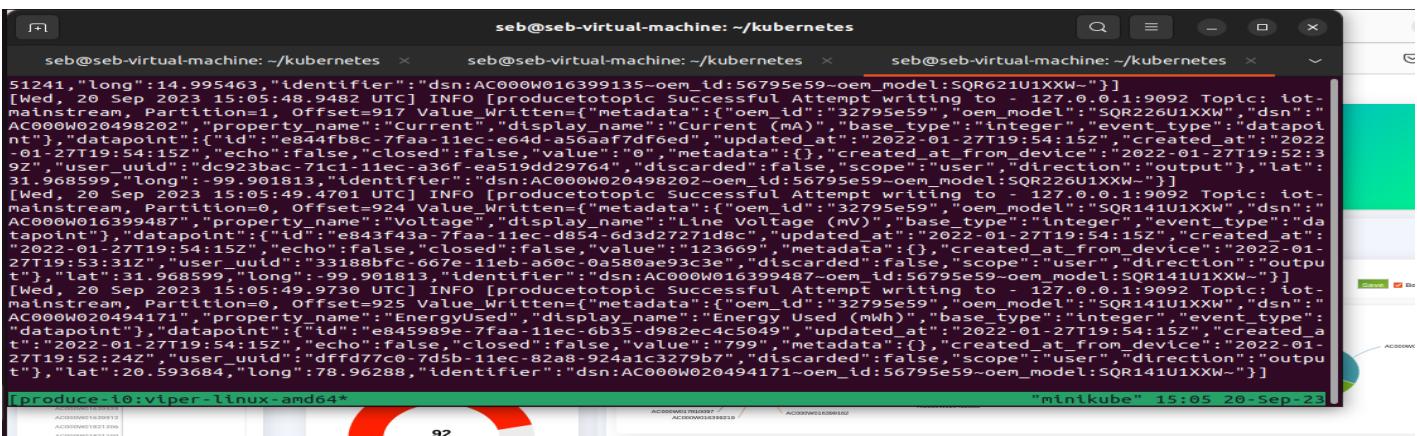
## STEP 10: TMUX into windows (See APPENDIX D and run those same commands)

RUN: tmux ls

RUN: tmux a -t produce-iot-data-viper-8000

You should see this: (You are now inside the container running in Kubernetes)

```
root@minikube:~# tmux ls
kafka: 1 windows (created Wed Sep 20 14:40:21 2023)
preprocess-data-python-8001: 1 windows (created Wed Sep 20 14:40:39 2023)
preprocess-data-viper-8001: 1 windows (created Wed Sep 20 14:40:31 2023)
preprocess2-data-python-8002: 1 windows (created Wed Sep 20 14:40:39 2023)
preprocess2-data-viper-8002: 1 windows (created Wed Sep 20 14:40:32 2023)
produce-iot-data-python-8000: 1 windows (created Wed Sep 20 14:40:39 2023)
produce-iot-data-viper-8000: 1 windows (created Wed Sep 20 14:40:31 2023)
visualization-viperviz-9005: 1 windows (created Wed Sep 20 14:40:39 2023)
zookeeper: 1 windows (created Wed Sep 20 14:40:17 2023)
root@minikube:/#
```



```
seb@seb-virtual-machine:~/kubernetes
seb@seb-virtual-machine:~/kubernetes
seb@seb-virtual-machine:~/kubernetes
seb@seb-virtual-machine:~/kubernetes
[Wed, 20 Sep 2023 15:05:48.9482 UTC] INFO [producetotopic] Successful Attempt writing to - 127.0.0.1:9092 Topic: iot-mainstream, Partition=1, Offset=917 Value_Written={"metadata": {"oem_id": "32795e59", "oem_model": "SQR226U1XXW", "dsn": "AC000W020498202", "property_name": "Current (mA)", "display_name": "Current (mA)", "base_type": "integer", "event_type": "datapoint"}, "datapoint": {"id": "e844fb8c-7faa-11ec-e64d-a56aaf7df0ed", "updated_at": "2022-01-27T19:54:15Z", "created_at": "2022-01-27T19:54:15Z", "echo": false, "closed": false, "value": "0", "metadata": {}}, "created_at_from_device": "2022-01-27T19:52:39Z", "user_uuid": "dc923bac-71c1-11ec-a36f-ea519dd29764", "discarded": false, "scope": "user", "direction": "output"}, "lat": 31.968599, "long": -99.901813, "identifier": "dsn:AC000W020498202-oem_id:56795e59-oem_model:SQR226U1XXW-"}]
[Wed, 20 Sep 2023 15:05:49.4701 UTC] INFO [producetotopic] Successful Attempt writing to - 127.0.0.1:9092 Topic: iot-mainstream, Partition=0, Offset=924 Value_Written={"metadata": {"oem_id": "32795e59", "oem_model": "SQR141U1XXW", "dsn": "AC000W016399487", "property_name": "Voltage", "display_name": "Line Voltage (mV)", "base_type": "integer", "event_type": "datapoint"}, "datapoint": {"id": "e844fb8c-7faa-11ec-e64d-d854-6d3d2721d8c", "updated_at": "2022-01-27T19:54:15Z", "created_at": "2022-01-27T19:54:15Z", "echo": false, "closed": false, "value": "123669", "metadata": {}}, "created_at_from_device": "2022-01-27T19:53:31Z", "user_uuid": "33188bfc-667e-11eb-a60c-0a580ae93c3e", "discarded": false, "scope": "user", "direction": "output"}, "lat": 31.968599, "long": -99.901813, "identifier": "dsn:AC000W016399487-oem_id:56795e59-oem_model:SQR141U1XXW-"}]
[Wed, 20 Sep 2023 15:05:49.9730 UTC] INFO [producetotopic] Successful Attempt writing to - 127.0.0.1:9092 Topic: iot-mainstream, Partition=0, Offset=925 Value_Written={"metadata": {"oem_id": "32795e59", "oem_model": "SQR141U1XXW", "dsn": "AC000W020494171", "property_name": "Energy_Used (mWh)", "display_name": "Energy_Used (mWh)", "base_type": "integer", "event_type": "datapoint"}, "datapoint": {"id": "e845989e-7faa-11ec-0b35-d982ec4c5049", "updated_at": "2022-01-27T19:54:15Z", "created_at": "2022-01-27T19:54:15Z", "echo": false, "closed": false, "value": "799", "metadata": {}}, "created_at_from_device": "2022-01-27T19:52:24Z", "user_uuid": "dffff77c0-7d5b-11ec-82a8-924a1c3279b7", "discarded": false, "scope": "user", "direction": "output"}, "lat": 20.593684, "long": 78.96288, "identifier": "dsn:AC000W020494171-oem_id:56795e59-oem_model:SQR141U1XXW-"}]
```

[Back To TOC](#)

# Kubernetes Setup

## STEP 11: Exit

**EXIT OUT OF TMUX**

**RUN: Ctlr+B, D**

**EXIT OUT OF Container**

**RUN: exit**

**You should see this: (You are now inside the container running in Kubernetes)**

```
root@minikube:/# tmux a -t produce-iot-data-viper-8000
[detached (from session produce-iot-data-viper-8000)]
root@minikube:/#
```

```
root@minikube:/# exit
exit
seb@seb-virtual-machine:~/kubernetes$
```

[Back To TOC](#)

# Kubernetes Setup

## STEP 12: Enable Kubernetes Dashboard

**RUN:** minikube addons enable dashboard

You should see this:

```
seb@seb-virtual-machine:~/kubernetes$ minikube addons enable dashboard
💡 dashboard is an addon maintained by Kubernetes. For any concerns contact minikube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernetes/minikube/blob/master/OWNERS
  ■ Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
  ■ Using image docker.io/kubernetesui/dashboard:v2.7.0
💡 Some dashboard features require the metrics-server addon. To enable all features please run:

  minikube addons enable metrics-server

🌟 The 'dashboard' addon is enabled
```

Run the Kubernetes Dashboard:

**RUN:** minikube dashboard

(to exit dashboard press Ctrl + C in your terminal)

Name	Images	Labels	Pods
seneca-iot-deployment	maadsdocker/seneca-iot-tml-kafka-amd:64	-	1/1

[Back To TOC](#)

# Kubernetes (minikube) Popular Commands

## Dashboard:

- enable dashboard: minikube addons enable dashboard
- run dashboard: minikube dashboard

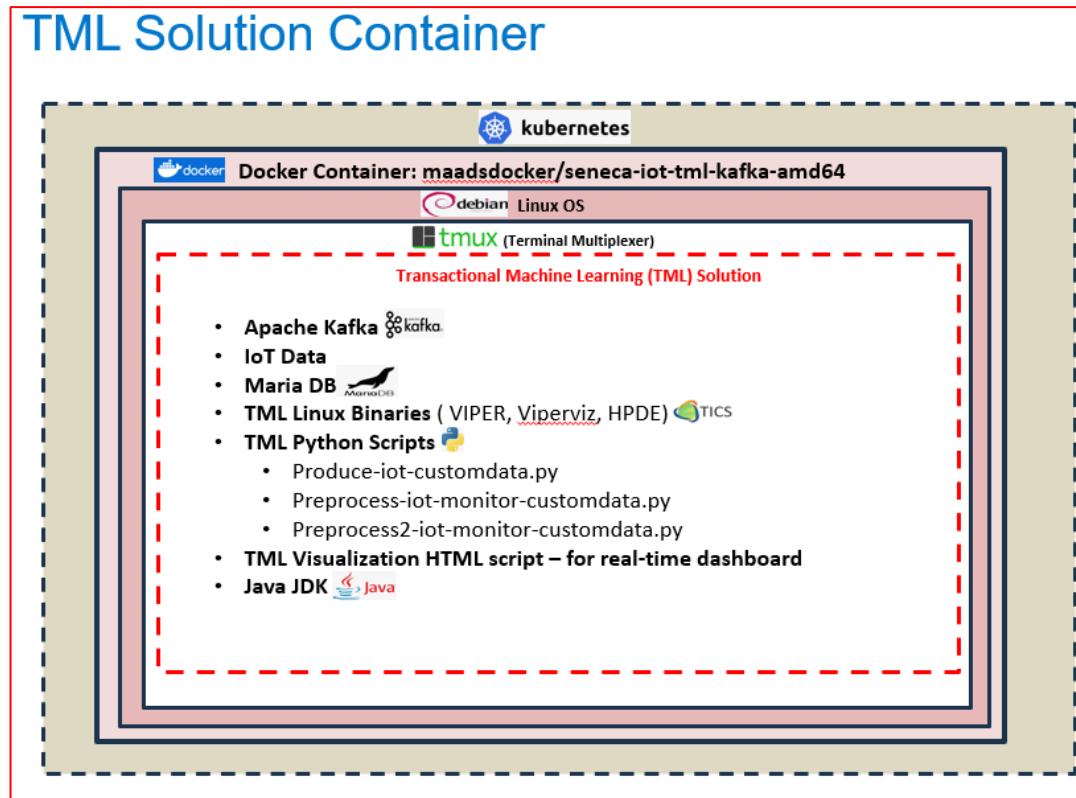
## COMMAND kubectl commands:

- 1. create pod: kubectl apply -f senecaiot.yml
- 2. kubectl delete pods seneca-iot-pod
- 3. kubectl get pods
- 4. kubectl exec -it container\_name bash
- 5. kubectl describe pods pod\_name
- 6. kubectl delete all --all --all-namespaces
- 7. kubectl expose deployment seneca-iot-deployment --port=9005 --target-port=9005 --name=seneca-iot --type=LoadBalancer
- 8. kubectl port-forward <pod name> 9005:9005
- ERRORS:
  - if error or kubelet and aapiserver is STOPPED run:
    - a. minikube stop
    - b. minikube delete
    - c. minikube start
    - d. minikube status

[Back To TOC](#)

# CONGRATULATIONS!

You just built, pulled, ran, streamed, and analysed a TML Solution Docker container in Kubernetes and completed the deployment of the ENTIRE TML SOLUTION CONTAINER!



[Back To TOC](#)

# APPENDIX G

- TML Folder Structure in Container

[Back To TOC](#)

# TML Solution Folder Structure in Container

- There are 6 parts to the TML solution that run in the Docker Container:

  1. Producing data to Kafka
  2. Preprocess data that is streaming to Kafka
  3. Preprocess2 data to determine the probability of anomalies in the data
  4. Visualization to stream preprocess and preprocess2 to dashboard
  5. TMUX – Runs all the TML solution windows
  6. Kafka – storage for the RAW data

The TML Solution Container Structure is as follows: (Note your docker container id will be different from the id: effa956fa33a in the example below – simple **RUN: docker ps** to get your CONTAINER ID)

```
seb@seb-virtual-machine:~$ docker exec -it effa956fa33a bash
root@effa956fa33a:/# ls
Hpde Kafka Viper-preprocess2 Viperviz boot dev home lib32 libx32 mnt proc run srv tmp usr
IotSolution Viper-preprocess Viper-produce bin deploy etc lib lib64 media opt root sbin sys tmux var
root@effa956fa33a:/#
```

[Back To TOC](#)

# TML Solution Folder Structure in Container

1. **Producing data to Kafka:** this produces Raw data to Apache Kafka that is running inside the container – this RAW data is used for the entire solution
- a) The name of the RAW Data file called **IoTData.txt** it's location in the **IotSolution** folder

```
seb@seb-virtual-machine:~$ docker exec -it effa956fa33a bash
root@effa956fa33a:/# ls
Hdpe      Kafka      Viper-preprocess2  Viperviz  boot   dev   home  lib32  libx32  mnt   proc   run   srv   tmp   usr
IotSolution  Viper-preprocess  Viper-produce    bin     deploy  etc   lib    lib64  media  opt    root   sbin  sys   tmux  var
root@effa956fa33a:/#
```

```
root@effa956fa33a:/IotSolution# ls
IoTData.txt  iot-ml-predictions_topicid_logistics.py  preprocess-iot-monitor-customdata.py  produce-iot-customdata.py
dsntmlidmain.csv  iot-ml-training_topicid_logistics.py  preprocess2-iot-monitor-customdata.py  readme
```

- b) The Python Script: **produce-iot-customdata.py** reads each line of IoTData.txt and streams the data to Kafka.
  - the script calls the MAADS-Viper binary called **viper-linux-amd64** that is running in the **Viper-produce** folder which produces the Raw data to **Kafka topic: iot-mainstream**

```
root@effa956fa33a:/Viper-produce# ls
'MAADSViper Installation Guide.pdf'  admin.tok  client.key.pem  token.tok  viper.db  viper.env.bak  viperlogs
'MAADSViper-Product Brief.pdf'       client.cer.pem  server.cer.pem  viper-linux-amd64  viper.env  viper.txt
```

[Back To TOC](#)

# TML Solution Folder Structure in Container

1. **Preprocess data to Kafka:** this preprocesses the Raw data by reading a Kafka Topic: **iot-mainstream**, and writes the output to another Kafka topic: **iot-preprocess**

```
seb@seb-virtual-machine:~$ docker exec -it effa956fa33a bash
root@effa956fa33a:/# ls
Hdde      Kafka      Viper-preprocess2  Viperviz  boot   dev   home  lib32  libx32  mnt   proc   run   srv   tmp   usr
IotSolution  Viper-preprocess  Viper-produce    bin     deploy  etc   lib    lib64  media  opt   root   sbin  sys   tmux  var
root@effa956fa33a:/#
```

```
root@effa956fa33a:/IotSolution# ls
IoTData.txt      iot-ml-predictions_topicid_logistics.py  preprocess-iot-monitor-customdata.py  produce-iot-customdata.py
dsntmlidmain.csv  iot-ml-training_topicid_logistics.py    preprocess2-iot-monitor-customdata.py  readme
```

- a) The Python Script: **preprocess-iot-monitor-customdata.py** consumes (reads) from the Kafka Topic: **iot-mainstream** and produces (writes) it to another Kafka topic: **iot-preprocess**
- this python script calls the MAADS-Viper binary called **viper-linux-amd64** that is running in the **Viper-preprocess** folder and does all the reading/writing to Kafka topics.

```
root@effa956fa33a:/Viper-preprocess# ls
'MAADSViper Installation Guide.pdf'  admin.tok      client.key.pem  token.tok        viper.db    viper.env.bak  viperlogs
'MAADSViper-Product Brief.pdf'       client.cer.pem server.cer.pem  viper-linux-amd64  viper.env    viper.txt
```

[Back To TOC](#)

# TML Solution Folder Structure in Container

1. **Preprocess2 data to Kafka:** this preprocesses2 the Raw data by reading a Kafka Topic: preprocess, and writes the output to another Kafka topic: iot-preprocess2

```
seb@seb-virtual-machine:~$ docker exec -it effa956fa33a bash
root@effa956fa33a:/# ls
Hdde      Kafka      Viper-preprocess2  Viperviz  boot   dev   home  lib32  libx32  mnt   proc   run   srv   tmp   usr
IotSolution  Viper-preprocess  Viper-produce    bin     deploy  etc   lib    lib64  media  opt   root   sbin  sys   tmux  var
root@effa956fa33a:/#
```

```
root@effa956fa33a:/IotSolution# ls
IoTData.txt      iot-ml-predictions_topicid_logistics.py  preprocess-iot-monitor-customdata.py  produce-iot-customdata.py
dsntmlidmain.csv  iot-ml-training_topicid_logistics.py  preprocess2-iot-monitor-customdata.py  readme
```

- a) The Python Script: **preprocess2-iot-monitor-customdata.py** consumes (reads) from the Kafka Topic: **iot-preprocess** and produces (writes) it to another Kafka topic: **iot-preprocess2**
- this python script calls the MAADS-Viper binary called **viper-linux-amd64** that is running in the **Viper-preprocess2** folder and does all the reading/writing to Kafka topics.

```
root@effa956fa33a:/Viper-preprocess2# ls
'MAADSViper Installation Guide.pdf'  admin.tok      client.key.pem  token.tok          viper.db  viper.env.bak  viperlogs
'MAADSViper-Product Brief.pdf'       client.cer.pem  server.cer.pem  viper-linux-amd64  viper.env  viper.txt
```

[Back To TOC](#)

# TML Solution Folder Structure in Container

## 1. Visualization Folder is in Viperviz

```
seb@seb-virtual-machine:~$ docker exec -it effa956fa33a bash
root@effa956fa33a:/# ls
Hdpe      Kafka          Viper-preprocess2  Viperviz  boot   dev   home  lib32  libx32  mnt   proc   run   srv   tmp   usr
IotSolution Viper-preprocess  Viper-produce    bin     deploy  etc   lib    lib64  media  opt    root   sbin  sys   tmux  var
root@effa956fa33a:/#
```

## 2. The TML binary that streams data from Kafka topics: **iot-preprocess** and **iot-preprocess2** to the client browser is **viperviz-linux-amd64**

```
root@7d125f62d0b1:/Viperviz# ls
'MAADDS-Viper Installation Guide.pdf'  client.key.pem  viper.db  viperviz-linux-amd64
admin.tok                                server.cer.pem  viper.env  viperviz.txt
client.cer.pem                            token.tok      viperviz   viperviz.zip
```

## 3. The TML IoT Dashboard is in the folder: Viperviz -> viperviz -> views called: **iot-failure-seneca.html**

```
root@7d125f62d0b1:/Viperviz/viperviz/views# ls
OTICS-New-Logo-Black.png           externaldashboardpasswords_for_nginx  medication-fraud.html
aims.html                           fg_fhir_landing.html                medicationfrauddash.png
anomaly.html                        fhirhealthmonitor-json            nft-priceprediction-Blockchain.html
apex                                fhirpopulationhealth.html        optimization.html
backgroundscripts                   fhirpopulationhealthmap-chatgpt.html  oticscico.png
banner.png                          fhirpopulationhealthmap.html       prediction.html
cluster.jpg                         firstgenesislogo.png            preprocess.html
cluster.png                         generictopics.html              searchanomalies.html
clusteranalysis.html                help.png                      senecalogo.png
clusteranalysisotics.html          img                           smilelogo.png
crypto-marketrisk-Blockchain.html  iot-failure-Blockchain.html    testheatmap.html
crypto1.png                         iot-failure-machinetraining.html  testheatmap2.html
crypto2.png                         iot-failure-seneca.html         testheatmap3.html
crypto3.png
```

This **iot-failure-seneca.html** makes a websocket connection to **viperviz-linux-amd64**, which directly reads Kafka topics: **iot-preprocess** and **iot-preprocess2**: **viperviz-linux-amd64** is listening on port 9005.

[Back To TOC](#)

# TML Dashboard

**The TML Dashboard runs by entering this URL in the browser:**

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

**NOTE: The URL [topic=iot-preprocess2,iot-preprocess](#) – these are precisely the topics where the preprocessed data are being produced in Kafka.**

# TMUX: Terminal Multiplexer

- When Docker starts up – it runs a **tmux-docker.sh** at the **ENTRYPOINT INSIDE Dockerfile:**

```
325    RUN tmx -RT raspberrypi  
326  
327    ENTRYPOINT ["/bin/bash", "-c", "while true; do sleep 1; done | ./tmux/tmux-docker.sh"]
```

- This file is located in **tmux** folder:

```
seb@seb-virtual-machine:~$ docker exec -it effa956fa33a bash  
root@effa956fa33a:/# ls  
Hdpe      Kafka          Viper-preprocess2  Viperviz  boot   dev   home  lib32  libx32  mnt   proc   run   srv  tmp   usr  
IotSolution  Viper-preprocess  Viper-produce     bin      deploy  etc   lib    lib64  media  opt   root   sbin  sys  tmux  var  
root@effa956fa33a:/#
```

```
root@7d125f62d0b1:/tmux# ls  
tmux-docker.sh
```

- tmux-docker.sh** automatically creates and runs all the TML solution files and applications running in their own Linux instances.

```
root@7d125f62d0b1:/# tmux ls  
kafka: 1 windows (created Fri Sep 22 19:09:24 2023)  
preprocess-data-python-8001: 1 windows (created Fri Sep 22 19:09:41 2023)  
preprocess-data-viper-8001: 1 windows (created Fri Sep 22 19:09:34 2023)  
preprocess2-data-python-8002: 1 windows (created Fri Sep 22 19:09:41 2023)  
preprocess2-data-viper-8002: 1 windows (created Fri Sep 22 19:09:34 2023)  
produce-iot-data-python-8000: 1 windows (created Fri Sep 22 19:09:41 2023)  
produce-iot-data-viper-8000: 1 windows (created Fri Sep 22 19:09:34 2023)  
visualization-viperviz-9005: 1 windows (created Fri Sep 22 19:09:41 2023)  
zookeeper: 1 windows (created Fri Sep 22 19:09:19 2023)
```

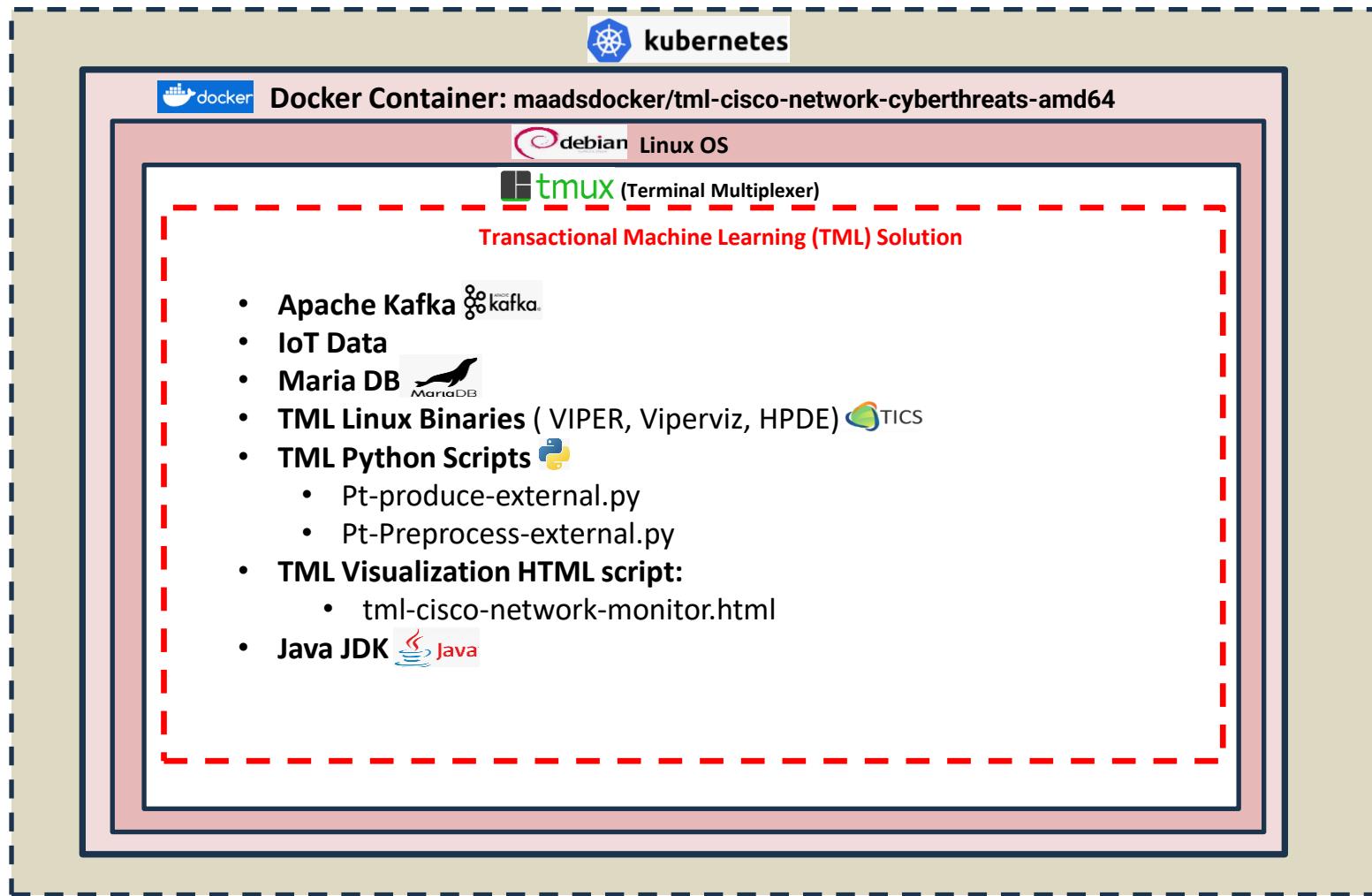
[Back To TOC](#)

# APPENDIX H

- TML-CISCO Cybersecurity and Network Monitoring Solution

[Back To TOC](#)

# TML – Cisco Solution Container (Linux/Mac)



Mac user: `maadsdocker/tml-cisco-network-cyberthreats-mac`

[Back To TOC](#)

# TML-CISCO Cybersecurity and Network Monitoring Solution

- This solution is an interactive solution between Students and Instructor
- **The goal of this solution is the real-time detection of network anomalies and teaching students how data is analysed and interpreted for Real-Time data driven decision-making**
- The Instructor plays the role of a hacker – who tries to hack into one of the end devices connected to a Cisco Switch
- The students TML solution should, in real-time, be able to detect which machine(s) the instructor is hacking into
- In addition – the instructor can also take a machine offline – this should also be detected by the TML solution
- ***STUDENTS WILL BUILD THEIR OWN TML Solution using the scripts provided – this solution is similar to the one shown here in the container and dashboard***

```
smaurice@DESKTOP-H0DIAMM:/mnt/c/MAADS/DOCKER/TML-Solution/docker/tml-cisco$ docker exec -it 85f2f2f6df56 bash
root@85f2f2f6df56:/# ls
[Hpde] [Viper-preprocess] [Viperviz] boot dev home lib32 libx32 mnt proc run srv tmp usr
[Kafka] [Viper-produce] bin deploy etc lib lib64 media opt root sbin sys tmux var
```

[Back To TOC](#)

# Setting Up Confluent Kafka Cloud

- Instructors will need to setup a FREE 30 day Confluent Kafka Cloud Account – for instructions see:
  - How to setup Kafka Cloud with TML Binary can be found in the video here:
  - <https://github.com/smaurice101/raspberryPi/tree/main/TML%20Crash%20course/Videos>
  - Kafka Cloud accounts are FREE for 30 days on Confluent.io (***no credit card needed***)

# TML-CISCO: Container RUN Command

- In [Dockerhub](#) instructors and students can find:
  - Linux Container: maadsdocker/tml-cisco-network-cyberthreats-amd64
  - MAC Container: maadsdocker/tml-cisco-network-cyberthreats-mac
  - These containers can be pulled using: **docker pull maadsdocker/tml-cisco-network-cyberthreats-amd64** (MAC users use the MAC container)
- These containers can be run in several ways using **docker run** by changing the value of the **RUNTYPE** variable:

EXAMPLE: `docker run -d --net="host" --env RUNTYPE=-1 --env BROKERHOSTPORT=pkclzvrd.us-west4.gcp.confluent.cloud:9092 --env KAFKAPRODUCETOPIC=cisco-network-mainstream --env HACKEDHOSTS=5.100-i,6.18-i,5.18-i --env CLOUDUSERNAME=Z6ZIK7OALQZXYYC2 --env CLOUDPASSWORD=uBGI02SFiZ69BBJhjE7vjE7mf5S8XqaMTpAv93doab5Gvn6szZ+Vbue4saX8CGo maadsdocker/tml-cisco-network-cyberthreats-amd64`

**NOTE: Give data 30 – 60 seconds to populate and preprocessed on Dashboard.**

- The next slide defines the **RUNTYPE** and other parameter settings in Docker Run.

[Back To TOC](#)

# TML-CISCO: Docker Parameter Settings

Scenario	RUNTYPE	BROKERHOSTPORT	KAFKAPRODUCETOPICT	HACKEDHOSTS	CLOUDUSERNAME	CLOUDPASSWORD
1*	-1 (instructors enter this)	This is the Confluent Kafka Cloud Broker Host and Port. <b>it will be provided by the Instructor.</b> It will look similar to this: pkc-lzvrd.us-west4.gcp.confluent.cloud:9092 <a href="https://www.confluent.io/">https://www.confluent.io/</a> 30 Day FREE Kafka Cloud accounts can be setup without credit card.	Instructors will set this to a string value for example: <b>cisco-network-mainstream</b> . This is the topic that raw data will be streamed to. <b>Instructors can specify any topic name.</b>	Instructors specific which hosts they want to hack into, the format is: [subnet].[hostid]-[i or d], where i=increasing packets, d=decreasing packets. For example:5.100-i,6.18-i,5.18-i – means hosts: 5.100, 6.18 and 5.18 are being hacked with increasing packets. <b>SOLUTION IS SIMULATING PACKETS.</b>	Instructor downloads the API Key from Confluent Kafka Cloud. <b>This is the KEY.</b>	Instructor downloads the API Key from Confluent Kafka Cloud. <b>This is the SECRET.</b>
2*	-2 (students)	Students will enter the Kafka Cloud Broker in 1*: <b>This will be given to them by instructor.</b>	Student enter the same topic name in 1*. <b>This will be given to them by instructor.</b>	<b>STUDENTS' TML SOLUTION MUST IDENTIFY THE HACKED HOSTS</b>	Students receive this from their Instructor.	Students receive this from their instructor.
3	0	<b>STUDENT TESTING:</b> Using CLOUD Kafka Enter kafka cloud broker host/port. Student MUST have Cisco Packet Tracer file running. See 1* below.	Students specify any Kafka topic. Or leave blank – then default topic will be used: <b>cisco-network-mainstream</b>	Students specify any hacked hosts.	Student downloads the API Key from Confluent Kafka Cloud. <b>This is the KEY.</b>	Student downloads the API Key from Confluent Kafka Cloud. <b>This is the SECRET.</b>
4	1	<b>STUDENT TESTING:</b> Using LOCAL Kafka running in the container. <b>Enter: 127.0.0.1:9092. Cisco data is read from a LOCAL FILE: cisco_network_data.txt</b>	Students specify any Kafka topic. Or leave blank – then default topic will be used: <b>cisco-network-mainstream</b>	Students specify any hacked hosts.	n/a – leave empty	n/a – leave empty
5	2	<b>STUDENT TESTING:</b> Using CLOUD Kafka Enter Kafka Cloud broker host and port. <b>LOCAL FILE: cisco_network_data.txt is read and streamed to Kafka cloud</b>	Students specify any Kafka topic. Or leave blank – then default topic will be used: <b>cisco-network-mainstream</b>	Students specify any hacked hosts.	Student downloads the API Key from Confluent Kafka Cloud. <b>This is the KEY.</b>	Student downloads the API Key from Confluent Kafka Cloud. <b>This is the SECRET.</b>
6	3	<b>VISUALIZATION ONLY:</b> Using CLOUD Kafka or On-prem use: <b>127.0.0.1:9092</b>	n/a	n/a	<b>For Cloud: key or leave empty for on-prem</b>	<b>For Cloud: secret or leave empty for on-prem</b>

- Scenarios 1\* and 2\* are the MAIN Instructor and Student Presentation scenarios:

1\* - Instructors MUST run the Docker container with RUNTYPE = -1Instructors MUST have Cisco Packet Tracer running : [tml-hacker-networksecurity-setup v3.pkt](#) – this file can be downloaded from: <https://github.com/smaurice101/raspberryipi/tree/main/tml-cisco-pt/pfile>

2\* Students MUST run their Docker container with RUNTYPE = -2

[Back To TOC](#)

# Valid Hosts to Hack

- This is based on the Cisco Packer Tracer file: [tml-hacker-networksecurity-setup v3.pkt](#)
- Found here: <https://github.com/smaurice101/raspberryPi/tree/main/tm/cisco-pt/ptfile>
- Cisco Packer Tracer can be installed from [MyApps](#)

## Valid hosts on interface: 192.168.5.1

- 5.100
- 5.101
- 5.10
- 5.11
- 5.12
- 5.13
- 5.14
- 5.15
- 5.16
- 5.17
- 5.18
- 5.19
- 5.21
- 5.22
- 5.23
- 5.24
- 5.25
- 5.26
- 5.27
- 5.28
- 5.29
- 5.30
- 5.31

## Valid hosts on interface: 192.168.6.1

- 6.100
- 6.101
- 6.10
- 6.11
- 6.12
- 6.13
- 6.14
- 6.15
- 6.16
- 6.17
- 6.18
- 6.19
- 6.20
- 6.21
- 6.22
- 6.23
- 6.24
- 6.25
- 6.26

[Back To TOC](#)

# Docker Run Commands For Instructor and Students

## A. INSTRUCTOR Docker Run Command Must be:

```
docker run -d --net="host" --env RUNTYPE=1 --env BROKERHOSTPORT=<replace> --env  
KAFKAPRODUCETOPICTopic=<replace> --env HACKEDHOSTS=<replace> --env  
CLOUDUSERNAME=<replace> --env CLOUDPASSWORD=<replace> maadsdocker/tm-cisco-network-  
cyberthreats-amd64
```

- *instructor MUST enter values for <replace> and NOTIFY STUDENTS*

## B. STUDENTS Docker Run Command Must be:

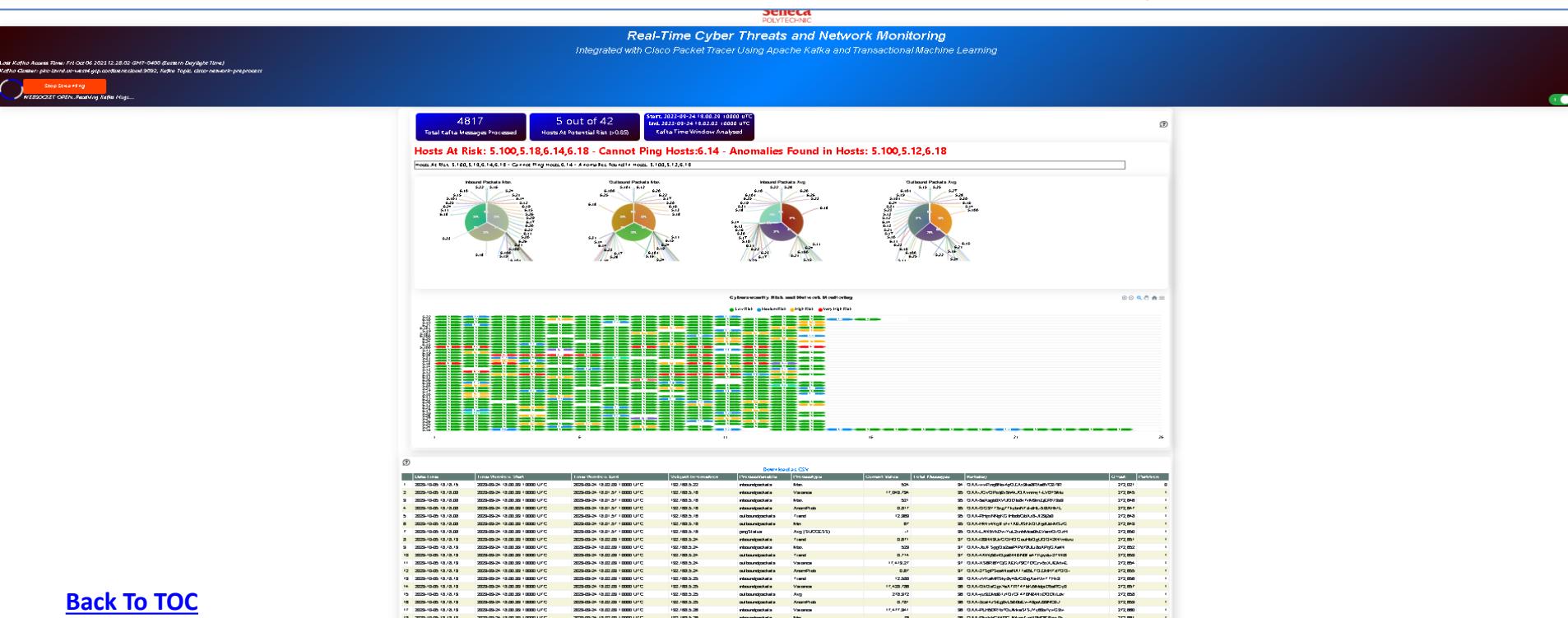
```
docker run -p VIPERVIZPORT : VIPERVIZPORT --env RUNTYPE=2 --env VIPERVIZPORT=<student  
chosen> --env BROKERHOSTPORT=<replace> --env KAFKAPRODUCETOPICTopic=<replace> --env  
CLOUDUSERNAME=<replace> --env CLOUDPASSWORD=<replace> maadsdocker/tm-cisco-network-  
cyberthreats-amd64
```

- *Students MUST enter values for <replace> RECEIVED FROM instructor values*

# Cyber Threats/Network Monitoring Dashboard

- Once you have properly run the docker container you can visualize the dashboard by entering this URL in the browser:
  - STUDENTS USE: (IF VIPERVIZPORT=9000)**
    - <http://localhost:9000/ml-cisco-network-monitor.html?topic=cisco-network-preprocess&offset=1&groupid=&rollbackoffset=200&topicstype=prediction&append=0&secure=1>
  - INSTRUCTORS USE: (IF VIPERVIZPORT=10000)**
    - <http://localhost:10000/ml-cisco-network-monitor.html?topic=cisco-network-preprocess&offset=1&groupid=&rollbackoffset=200&topicstype=prediction&append=0&secure=1>
  - Press **Start Streaming** Button: You will see

**NOTE: Visualization requires -p (port-forwarding) – DO NOT use –net="host" in your Docker Run command, Visualization can be run as a standalone container using RUNTYPE=3**

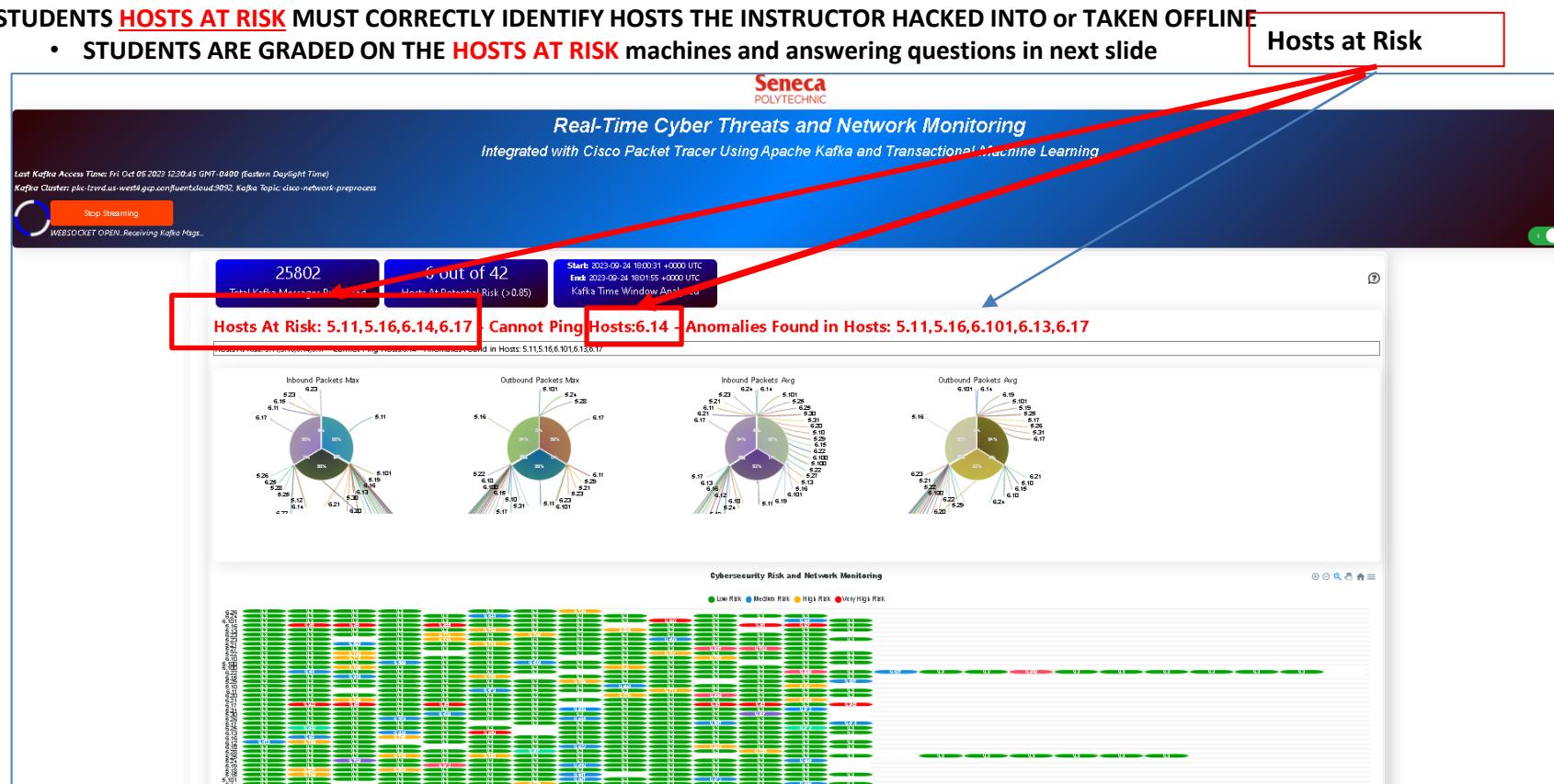


[Back To TOC](#)

# Explaining The Dashboard: EXAMPLE

Using this Docker Run Command: `docker run -p 9000:9000 --env VIPERVIZPORT=9000 --env RUNTYPE=2 --env BROKERHOSTPORT=pkczvrd.us-west4.gcp.confluent.cloud:9092 --env KAFKAPRODUCETOPIC=cisco-network-mainstream --env HACKEDHOSTS=5.11-i,5.16-i,6.17-i --env CLOUDUSERNAME=Z6ZIK7OALQZXYYC2 --env CLOUDPASSWORD=uBGI02SFiZ69BBJhjE7vjE7mfd5S8XqaMTpAv93doab5Gvn6szZ+Vbue4saX8CGomaadsdocker/tm/cisco-network-cyberthreats-amd64`

- The solution first simulates the hacking into 3 machines: **5.11, 5.16 and 6.17** (as specified by instructor)
- Web Browser connects to Container Port for **VIPERVIZ** on Port **9000**
- The **STUDENT DASHBOARD CORRECTLY** identified these 3 machines as "**HOSTS AT RISK**" in **REAL-TIME!**
- Note:** Machine **6.14** was taken offline – the Dashboard again **CORRECTLY** identified that **6.14** cannot be pinged
- Solution also performs further algorithmic checks for anomalies on all hosts
- STUDENTS HOSTS AT RISK** MUST CORRECTLY IDENTIFY HOSTS THE INSTRUCTOR HACKED INTO or TAKEN OFFLINE
  - STUDENTS ARE GRADED ON THE **HOSTS AT RISK** machines and answering questions in next slide



[Back To TOC](#)

# Student Will Process Real-Time Data with TML and Answer the Following Questions in the Dashboard

**Note: Student will receive a template Dashboard to Work from.**

1. Which machines are HOSTS AT RISK?
2. Which HOSTS AT RISK cannot be pinged?
3. What is the avg outbound packets for HOSTS AT RISK?
4. What is the avg inbound packets for HOSTS AT RISK?
5. What is the maximum outbound packets for HOSTS AT RISK?
6. What is the maximum inbound packets for HOSTS AT RISK?
7. Which HOSTS AT RISK have increasing outbound packets in bytes?
8. Which HOSTS AT RISK have decreasing outbound packets in bytes?
9. Which HOSTS AT RISK have increasing inbound packets in bytes?
10. Which HOSTS AT RISK have decreasing inbound packets?
11. Which machines are down? on switch 1.
12. Which machines are down? on switch 2.
13. Which machine has the highest variance in inbound packets?
14. Which machine has the highest variance in outbound packets?
15. Which machine has an anomaly probability in outbound packets above 80%
16. Which machine has an anomaly probability in inbound packets above 80%

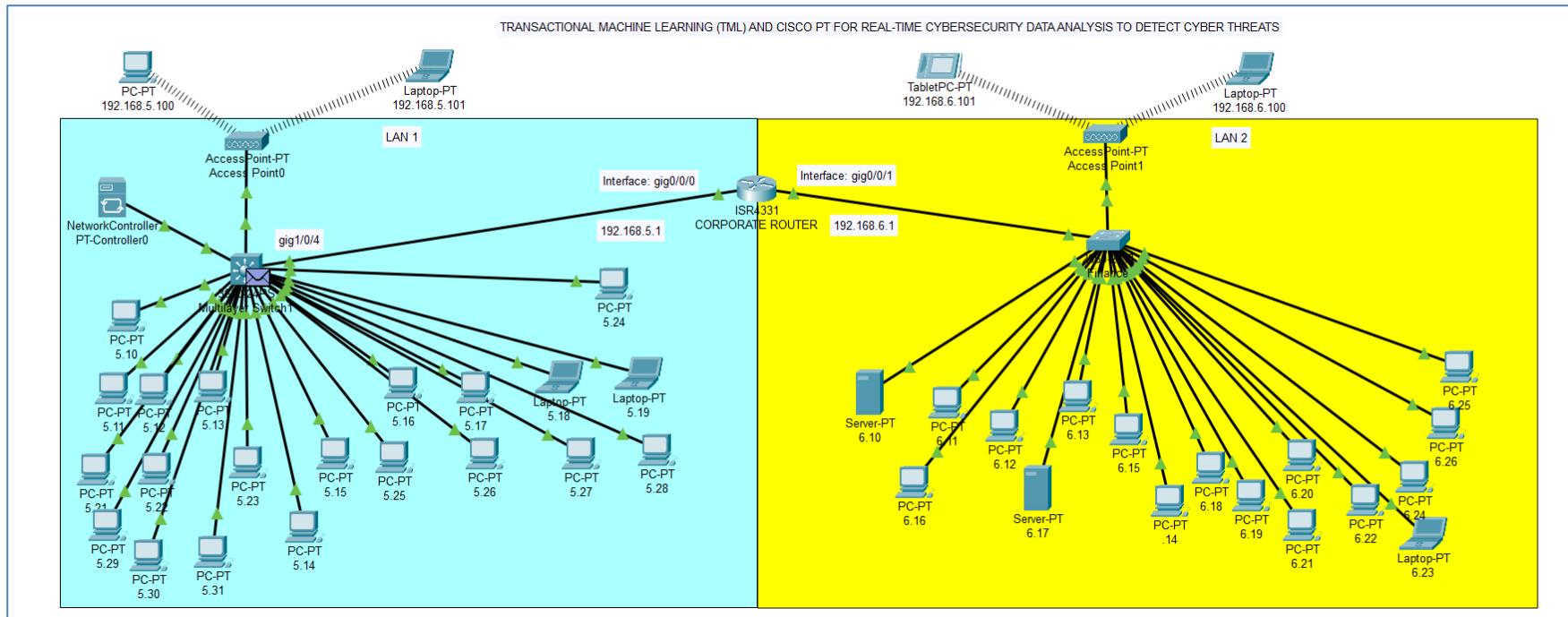
[Back To TOC](#)

# Solution: Cisco Network in Packet Tracer

Below diagram shows a network in Packet Tracer which will be used as part of the solution:  
(Packet Tracer file can be downloaded from here: <https://github.com/smaurice101/raspberrypi/tree/main/tml-cisco-pt/ptfile>)

1. One Router
2. Two Switches connected to the Router
3. 2 WIFI devices connected to the access points

This network can be simulated and packet data can be extracted for analysis using REST API



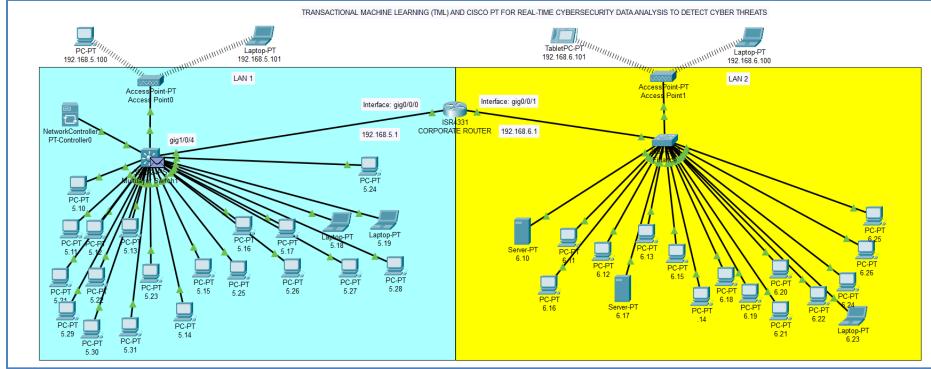
[Back To TOC](#)

# Solution Architecture

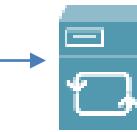
Instructors will receive a VM containing the entire solution

Instructor VM

**Instructor Tells The TML Solution Container at Run Time Which Machine They are hacking into**



Cisco Packet Tracer .pkt file



SDN Running in PT  
on Port 58000

Network data is pulled  
From PT using REST API



TML Solution  
Container

A Free Confluent Kafka Cloud account will be needed – it is valid for 30 days.



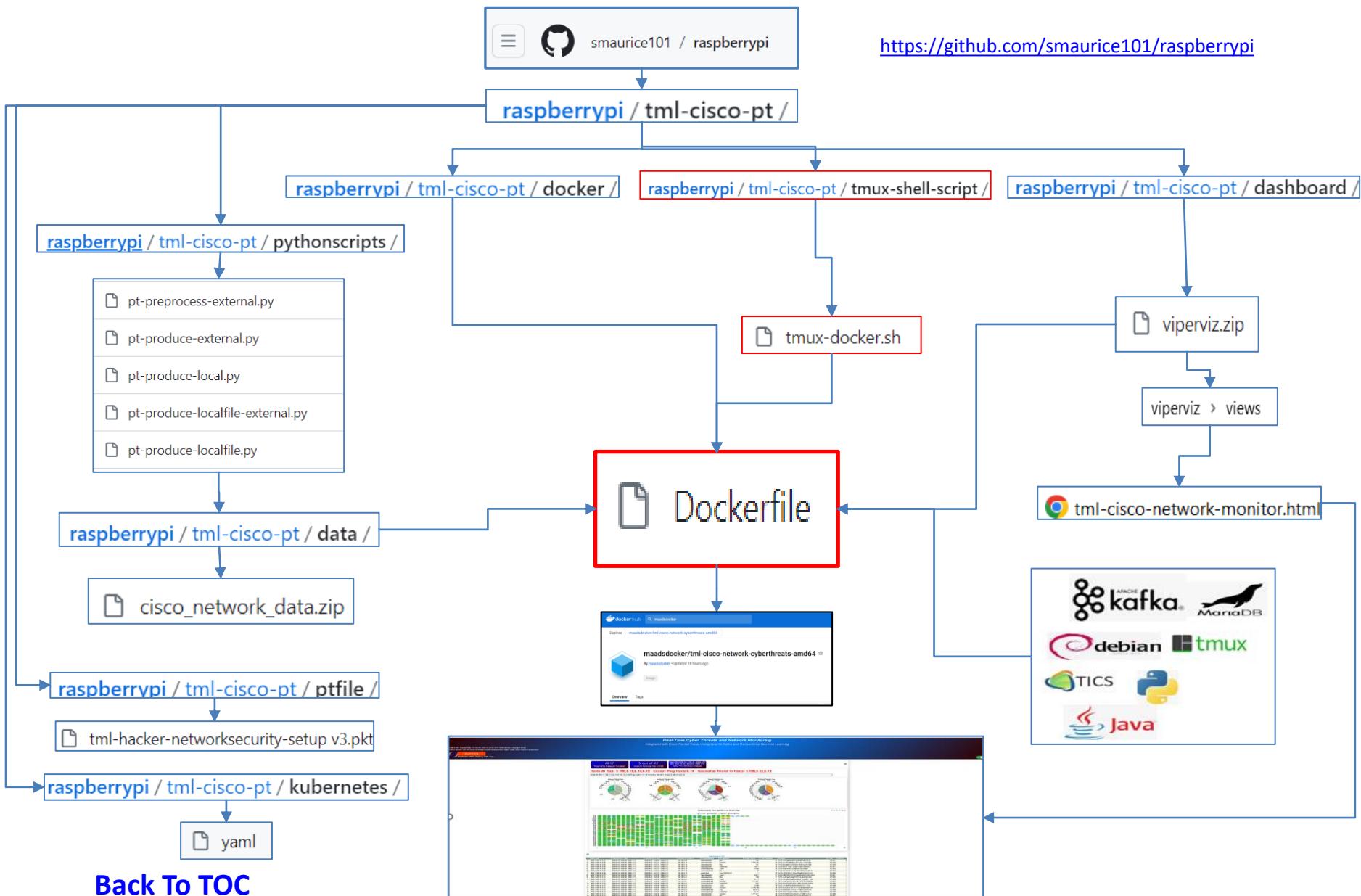
- Student teams build their TML solution container using a Template provided to them by instructor
- Student teams present their dashboard to class
- Team must answer any of the 16 questions in **real-time** from instructor
- **For each team Instructor can in real-time change the machine he is hacking into**



STUDENT: TML  
Solution  
Container

Student VM

# TML –CISCO SOLUTION COMPONENTS



# Technologies

- Cisco packet tracer is part of myApps
- Docker
- Students run their solution in VMWare VM using Docker Container
- TML technologies are downloaded from GitHub
- TML is already part of CSP 450 and CYT 160 and was taught in CSP 400 in the summer and very successful with students wanting to learn more TML

[Back To TOC](#)

# APPENDIX I

- Additional sources

# Additional TML Resources:

## 1. TML Crash Course Videos:

<https://github.com/smaurice101/raspberrypi/tree/main/TML%20Crash%20course/Videos>

•

## 1. TML Binaries:

a) <https://github.com/smaurice101/transactionalmachinelearning>

•

## 1. MAADSTML Python Library:

a) <https://pypi.org/project/maadstml/>

•

## 1. TML Blogs:

a) [Stream Processing/Analytics Tools Like Apache Flink is NOT Transactional Machine Learning](#)

b) [Data Quality Checking in Data Streams](#)

c) [A Fast and Simple Way To Migrate Data Streams Between Kafka Clusters: An Alternative to MirrorMaker2](#)

d) [NFT \(Ethereum\) Price Prediction with Transactional Machine Learning, Kafka \(or Redpanda\) and Blockchain](#)

e) [Detecting Medication Fraud at Scale with Transactional Machine Learning and Blockchain](#)

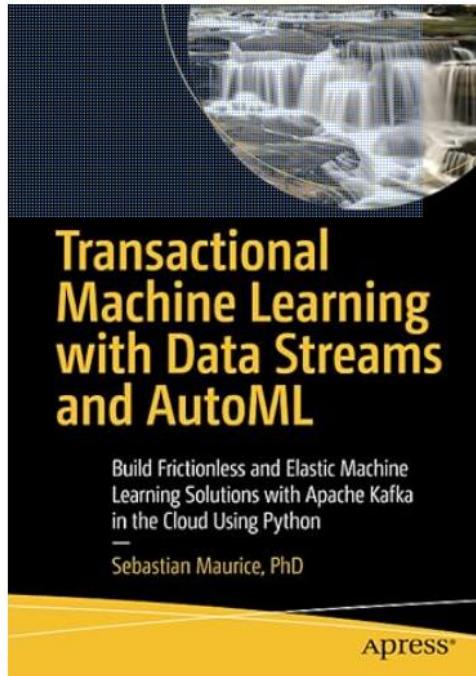
f) [TML and Cybersecurity](#)

g) [Contextualizing ChatGPT with Healthcare Data Streams](#)

[Back To TOC](#)

# APPENDIX J

- Transactional Machine Learning Book
- Amazon.ca



## Introduction: Big Data, Auto Machine Learning, and Data Streams

Data streams are a class of data that is continuously updated and captured and grows in volume and is largely unbounded [Aggarwal, 2007; Wrench et al., 2016]. Consider how our everyday lives contribute to data streams. Every time we purchase something with a credit card, the purchasing event information about your name, purchase amount, product purchased, time and date purchased, location where it was purchased, quantity, product code, and so on are all captured in real time and stored in a data storage platform capable of storing large amounts of data. Browsing the Web also results in enormous amounts of data flowing through IP networks that are being captured by your Internet service providers (ISPs). Even the cars we drive are becoming more connected to the Internet. The car manufacturers are capturing and storing all of the telemetry and GPS data.

Data continues to seep into all facets of our lives. Everyday items that we use today such as refrigerators, cars, washing machines, TVs, and so on create massive amounts of data each day. By some estimates, we create 2.5 quintillion bytes of data each day. And, most of the world's data was created in just the past few years. This is impressive in terms of scale and shows that data is flooding our world in ways we never imagined 10 or 15 years ago. Most of us are familiar with data that exists in database tables, flat files, and dataframes, but a new category of data that is creating new challenges for data engineers, scientists, and analysts is massive, fast-moving streams of data, driven by a digitally connected world. We are all aware of the growth of data and its value for organizations [Read et al., 2019; Read et al., 2020; Guzy and Wozniak, 2020; Lang et al., 2020], but we are still in the early stages of managing and analyzing fast-moving streams of data,

© Sebastian Maurice 2021

1

Intermediate-Advanced

ISBN-10



1484270223

ISBN-13



978-1484270226

Edition



1st ed.

Publication date



May 20 2021

Language



English



[Back To TOC](#)

# APPENDIX K

- Setting Up Confluent Kafka Cloud
- Watch Video:

<https://github.com/smaurice101/raspberrypi/tree/main/TML%20Crash%20course/Videos/Part%201>

# HAPPY STREAMING WITH TML!