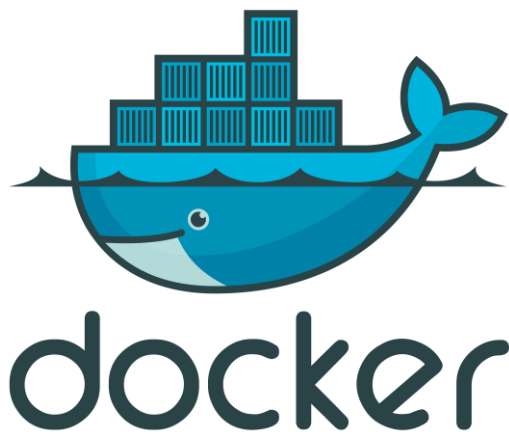


REPORT- TASK 3

ML Inference model on Docker and Firecracker: Comparative analysis



Firecracker

Engr. Yasir Hussain
Head of Department (Computer Science)
University College of Dera Murad Jamali (LUAWMS)- Pakistan

Contents

Docker	3
Introduction:	3
The Docker platform	3
Docker architecture:	3
The Docker daemon	4
The Docker client	4
Docker registries	4
Docker objects	4
Images	4
Containers	4
Installation:	5
Firecracker	7
Introduction:	7
Characteristics:.....	7
Secure.....	7
High Performance	7
Battle-Tested.....	7
Low Overhead	7
Open Source.....	7
Firecracker Security:	8
Simple Guest Model.....	8
Process Jail	8
Static Linking	8
Building Firecracker:	8
Running Firecracker:	9
First shell terminal:	9
Second shell terminal:.....	9
Guest Virtual Machine:	12
Firecracker network setup:	12
On the host:	12
In Guest Machine:	13
Machine Learning Inference Model:.....	15

Introduction:	15
Working of ML Inference Model.....	15
Support Vector Machine:.....	15
Distributed Denial of Service Attack:	16
Dataset:.....	16
ML Model in Python:.....	17
ML Inference code:	19
Deployment on Docker :.....	21
Deployment on Firecracker.....	25
Evaluation	27
Accuracy:.....	27
Precision:.....	27
Recall:.....	27
F1-score:.....	27
Testing Time:.....	27
CPU Utilization:	28
Memory Consumption:	28
Summary	30
Purpose:	30
Architecture:	30
Efficiency:.....	30
Security	30
Which is Better?	30
References	32
Figure 1 Docker Architecture	4
Figure 2 Firecracker MicroVm.....	7
Figure 3 Machine Learning Inference Model.....	15
Figure 4 Evaluation Metrics	28
Figure 5 Testing Time	28
Figure 6 CPU Utilization	28
Figure 7 Memory (RAM) Utilization	29

Docker

Introduction:

Docker is an open platform for developing, shipping, and running applications. Docker provides services to run software on a separate environment, Infrastructure could be managed in the same as application with Docker.

The Docker platform

Docker have isolated environment called container to run applications and packages. Because of isolation user can run multiple containers within single host. Containers are lightweight and contain everything needed to run the application [1]. Containers are easy to share within for working pattern.

Docker have following features:

- Containers used to develop applications with its components,
- You can use container to distribute and test applications.
- After completing the build of application, can deploy in the production environment.

Docker architecture:

Docker is built on a client server model the docker client communicates with the docker demon which overseas building operating and transferring your data containers. [2] Docker client and demon can run on the same system or a docker client can connect to a remote docker daemon the docker client and demon interacts using a rest API over UNIX sockets are a network interface. The docker compose is another docker client that allows you to work with application made-up of a collection of containers.

The Docker daemon

Docker daemon manages the images and containers. It also manages the volumes and network for docker images. Docker daemon perform tasks by listening for Docker API requests.

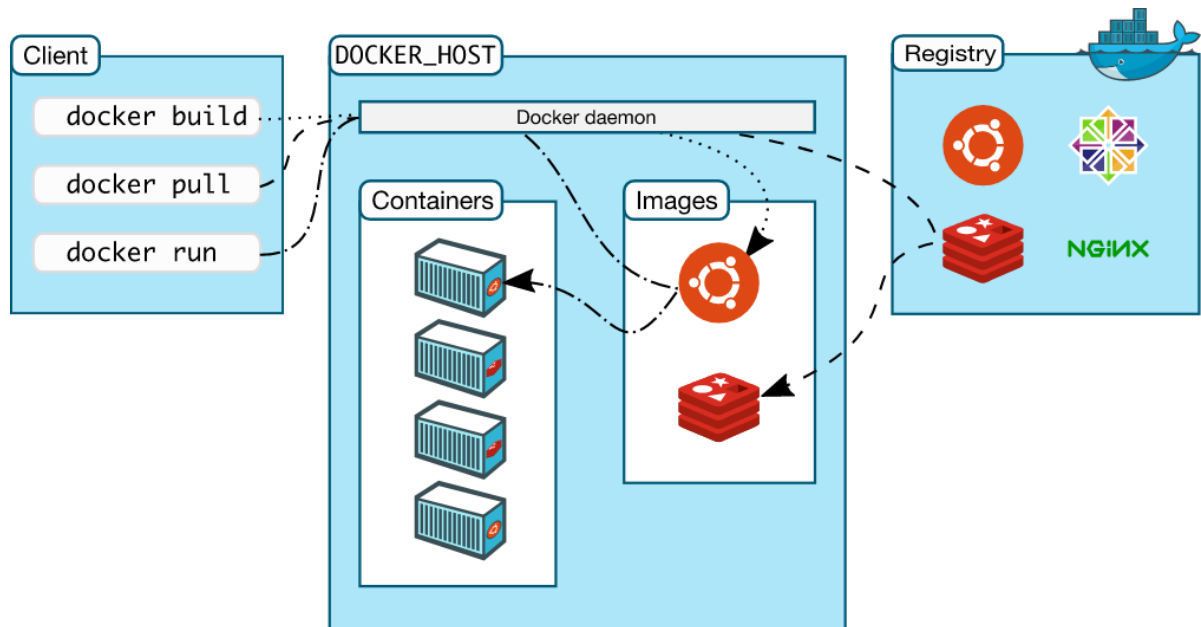


Figure 1 Docker Architecture

The Docker client

The Docker client (docker) is the main way of interaction between user and Docker. Dockerd run the command given by user like, run command. It uses the Docker API. More than one daemon can be used by user.

Docker registries

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry [3].

Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

Images

An image is a read-only framework containing Docker container creation instructions. An image is frequently based on another image, with some additional customization.

Containers

A container is an executable instance of an image. Container can be created, modified and removed from the docker hub and CLI mode. At run time, consumption by container can also be monitored.

Installation:

- 1) Before installation download docker from docker hub by this [link](#)
- 2) Double Click the Docker Desktop Installer in Downloaded folder.



- 3) In Initial configuration Setup will ask to check some configuration and click next.

Installing Docker Desktop 4.15.0 (93002)

Configuration

- ☒ Use WSL 2 instead of Hyper-V (recommended)
- ☒ Add shortcut to desktop

- 4) Then Setup will start unpacking files for installation.

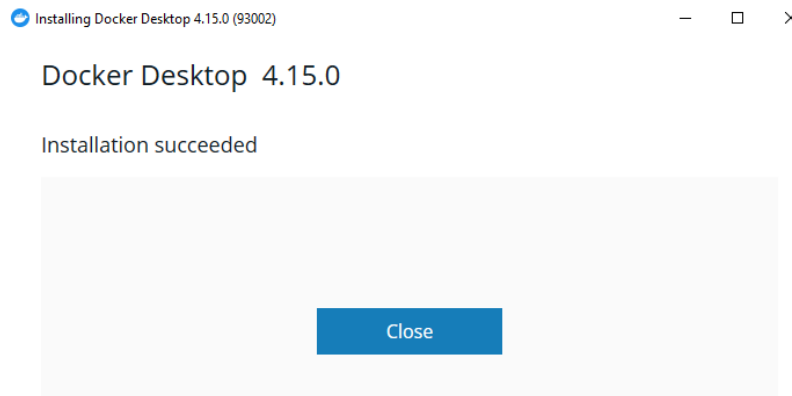
Installing Docker Desktop 4.15.0 (93002)

Docker Desktop 4.15.0

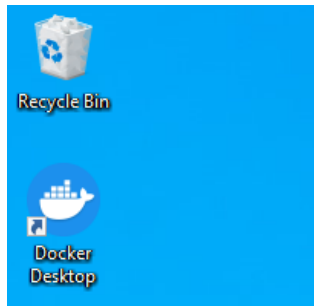
Unpacking files...

```
Unpacking file: frontend/resources/app.asar
Unpacking file: frontend/resources.pak
Unpacking file: frontend/locales/zh-TW.pak
Unpacking file: frontend/locales/zh-CN.pak
Unpacking file: frontend/locales/vi.pak
Unpacking file: frontend/locales/ur.pak
Unpacking file: frontend/locales/uk.pak
Unpacking file: frontend/locales/tr.pak
Unpacking file: frontend/locales/th.pak
Unpacking file: frontend/locales/te.pak
Unpacking file: frontend/locales/ta.pak
Unpacking file: frontend/locales/sw.pak
```

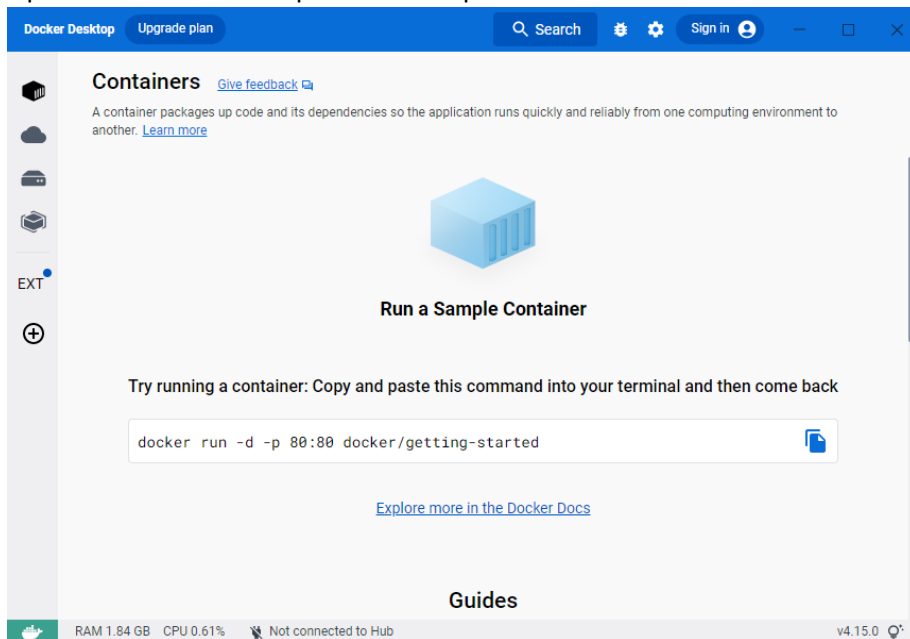
- 5) Once installation complete, close the setup window.



6) After successful installation you will find a shortcut icon of docker on your desktop.



7) Open the Docker desktop from desktop.



The Docker desktop is ready to use on Windows Operating system.

Firecracker

Introduction:

Firecracker is an open-source virtualization technology that is purpose-built for creating and managing secure, multi-tenant container and function-based services that provide serverless operational models. [4]

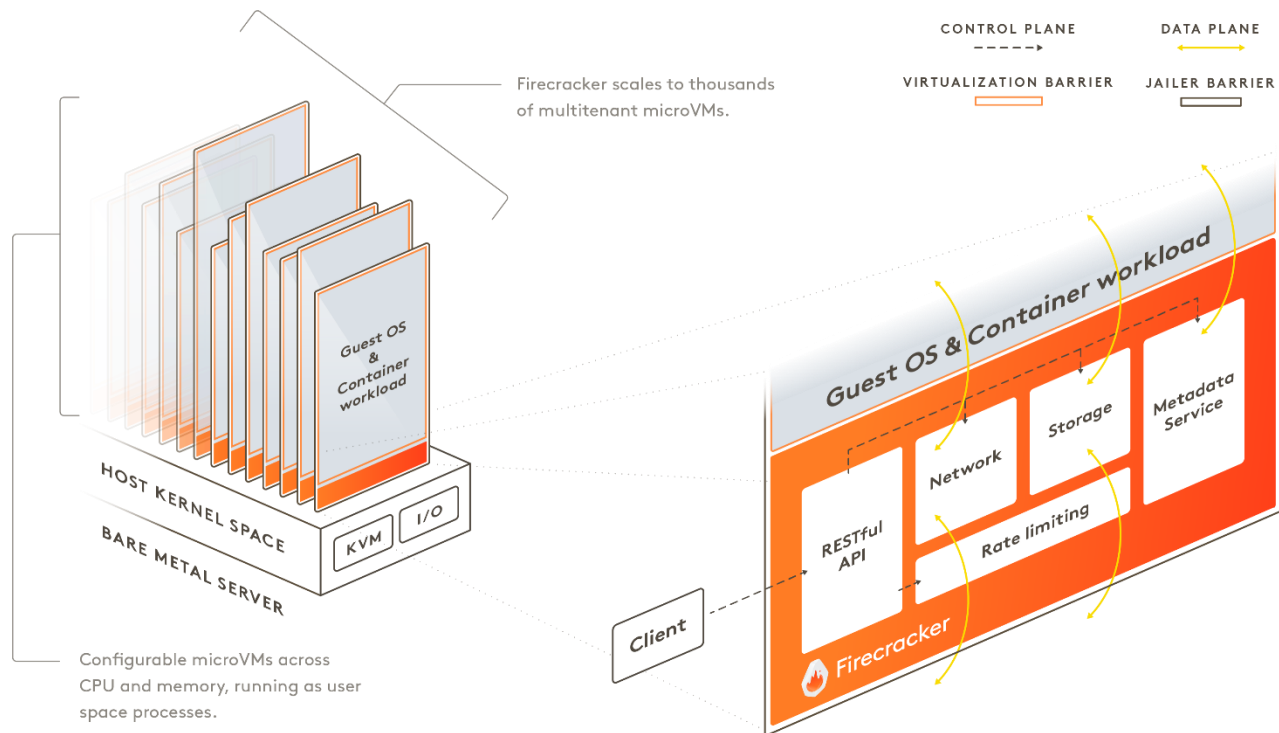


Figure 2 Firecracker MicroVm

Characteristics:

Secure – Because of Multiple level of isolation, Firecracker is less exposed for the attacks.

High Performance – A microVM could be launch in about 125 milliseconds.

Battle-Tested – Firecracker has already been deployed with AWS services including Lambda and AWS Fargate. [5]

Low Overhead – Little amount of memory about 5MiB is used per MicroVm. Can also configure own resources per VM.

Open Source – Firecracker is open source and available for the pull and contributions.

Firecracker Security:

Firecracker provides multiple security features.

Simple Guest Model – Firecracker guests are presented with a very simple virtualized device model in order to minimize the attack surface: a network device, a block I/O device, a Programmable Interval Timer, the KVM clock, a serial console, and a partial keyboard (just enough to allow the VM to be reset).

Process Jail – The Firecracker process is jailed using cgroups and seccomp BPF, and has access to a small, tightly controlled list of system calls.

Static Linking – Jailer is used to launch, the statically linked processes of Firecracker to ensure safety of environment.

Building Firecracker:

- 1) By using git on Linux, clone the source code of firecracker on local folder.

```
root@Anonymous:~# git clone https://github.com/firecracker-microvm/firecracker
Cloning into 'firecracker'...
remote: Enumerating objects: 39483, done.
remote: Counting objects: 100% (219/219), done.
remote: Compressing objects: 100% (144/144), done.
remote: Total 39483 (delta 101), reused 139 (delta 73), pack-reused 39264
Receiving objects: 100% (39483/39483), 23.69 MiB | 156.00 KiB/s, done.
Resolving deltas: 100% (25002/25002), done.
root@Anonymous:~#
```

- 2) Open the firecracker repository root directory.

```
root@Anonymous:~/firecracker# ls
build.rs      CREDITS.md      PGP-KEY.asc      src
Cargo.lock    deny.toml        pre-commit        tests
Cargo.toml     docs             README.md         THIRD-PARTY
CHANGELOG.md  FAQ.md           resources         tools
CHARTER.md    LICENSE          rusty-hook.toml
CODE_OF_CONDUCT.md MAINTAINERS.md SECURITY.md
CONTRIBUTING.md NOTICE          SPECIFICATION.md
```

- 3) Develop by the following command

```
root@Anonymous:~# cd firecracker/
root@Anonymous:~/firecracker# tools/devtool build
[Firecracker devtool] Starting build (debug, musl) ...
warning: patch for `kvm-bindings` uses the features mechanism. default-features and
features will not take effect because the patch dependency does not support this m
echanism
    Updating git repository `https://github.com/firecracker-microvm/kvm-bindings`
    Updating git submodule `https://github.com/rust-vmm/rust-vmm-ci.git`
    Updating git repository `https://github.com/firecracker-microvm/micro-http`
    Updating git submodule `https://github.com/rust-vmm/rust-vmm-ci.git`
```

```
root@Anonymous:~/firecracker# toolchain="$(uname -m)-unknown-linux-musl"
```

4) Two binaries will be created on following path in root folder of firecracker [6].

- Build / cargo_target / \${toolchain} / debug / firecracker
- build / cargo_target / \${toolchain} / debug / jailer

```
root@Anonymous:~/firecracker/build/cargo_target/x86_64-unknown-linux-musl/debug# ls
build deps examples firecracker firecracker.d incremental jailer jailer.d
```

Running Firecracker:

First, make sure you have the firecracker binary available

Open Two Shell prompt from the same folder where Firecracker binary and these above-mentioned Kernel and file system image resides

First shell terminal:

- 1) make firecracker to create its API socket.

```
sl/debug# rm -f /tmp/firecracker.socket
```

- 2) start firecracker:

```
sl/debug# ./firecracker --api-sock /tmp/firecracker.socket
```

Second shell terminal:

- 1) Download the Kernel and rootfs . to download the kernel, we make a bash script first.

```
ug# nano download.sh
```

- 2) Write script in file to download Kernel and rootfs.

```
GNU nano 7.0 download.sh
#!/usr/bin/env sh
arch=$(uname -m)
dest_kernel="kernel"
dest_rootfs="rootfs"
image_bucket_url="https://s3.amazonaws.com/spec.ccfc.min/img/quickstart_guide/$arch"

if [ ${arch} = "x86_64" ]; then
    kernel="${image_bucket_url}/kernels/vmlinux.bin"
    rootfs="${image_bucket_url}/rootfs/bionic.rootfs.ext4"
elif [ ${arch} = "aarch64" ]; then
    kernel="${image_bucket_url}/kernels/vmlinux.bin"
    rootfs="${image_bucket_url}/rootfs/bionic.rootfs.ext4"
else
    echo "Cannot run firecracker on $arch architecture!"
    exit 1
fi

echo "Downloading $kernel..."
curl -fsSL -o $dest_kernel $kernel

echo "Downloading $rootfs..."
curl -fsSL -o $dest_rootfs $rootfs

echo "Saved kernel file to $dest_kernel and root block device to $dest_rootfs."
```

- 3) After running the bash script, the files will start downloading.

```
root@Anonymous:~/firecracker/build/cargo_target/x86_64-unknown-linux-musl/debug# bash download.sh
Downloading https://s3.amazonaws.com/spec.cfc.min/img/quickstart_guide/x86_64/kernels/vmlinux.bin...
```

- 4) After downloading the kernel and rootfs file, time to tell firecracker to set the kernel and rootfs.
5) To set the Kernel, you will make a bash script which contain commands to set Guest Kernel.

```
#!/usr/bin/env sh

arch=`uname -m`
kernel_path=$(pwd)/debian-vm.bin

if [ ${arch} = "x86_64" ]; then
    curl --unix-socket /tmp/firecracker.socket -i \
        -X PUT 'http://localhost/boot-source' \
        -H 'Accept: application/json' \
        -H 'Content-Type: application/json' \
        -d "{
            \"kernel_image_path\": \"${kernel_path}\",
            \"boot_args\": \"console=ttyS0 reboot=k panic=1 pci=off\"
        }"
elif [ ${arch} = "aarch64" ]; then
    curl --unix-socket /tmp/firecracker.socket -i \
        -X PUT 'http://localhost/boot-source' \
        -H 'Accept: application/json' \
        -H 'Content-Type: application/json' \
        -d "{
            \"kernel_image_path\": \"${kernel_path}\",
            \"boot_args\": \"keep_bootcon console=ttyS0 reboot=k panic=1 pci=off\"
        }"
else
    echo "Cannot run firecracker on ${arch} architecture!"
    exit 1
fi
```

- 6) Run the set_kernel.sh script. The following output make sure that kernel is set up.

```
root@Anonymous:~/Firecracker/firecracker/build/cargo_target/x86_64-unknown-linux-musl/debug# ./set_kernel.sh
HTTP/1.1 204
Server: Firecracker API
Connection: keep-alive
```

- 7) Now, set the rootfs by running following script.

```
#!/usr/bin/env sh

rootfs_path=$(pwd)/debianrootfs.ext4"
curl --unix-socket /tmp/firecracker.socket -i \
-X PUT 'http://localhost/drives/rootfs' \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
-d "{
  \"drive_id\": \"rootfs\",
  \"path_on_host\": \"${rootfs_path}\",
  \"is_root_device\": true,
  \"is_read_only\": false
}"
```

- 8) Run the script- set_rootfs.sh

```
root@Anonymous:~/Firecracker/firecracker/build/cargo_target/x86_64-unknown-linux-musl/debug# bash set_rootfs.sh
HTTP/1.1 204
Server: Firecracker API
Connection: keep-alive
```

- 9) start the guest machine: I created a file with name StartVm.sh and written script to run VM.

```
/debug# nano StartVm.sh
```

Save the following script in file created above.

```
#!/usr/bin/env sh
curl --unix-socket /tmp/firecracker.socket -i \
-X PUT 'http://localhost/actions' \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
  "action_type": "InstanceStart"
}'
```

- 10) Run the script to start guest VM on firecracker.

```
root@Anonymous:~/Firecracker/firecracker/build/cargo_target/x86_64-unknown-linux-musl/debug# bash StartVM.sh
HTTP/1.1 204
Server: Firecracker API
Connection: keep-alive
```

Guest Virtual Machine:

The first shell will prompt ne guest machine. The default user and password for machine is 'root'.

```
test-machine login: root
Password:
Last login: Wed Dec 14 12:30:00 UTC 2022 on ttyS0
Linux test-machine 5.2.1 #3 SMP Thu Jul 18 15:17:15 EEST 2019 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@test-machine:~#
```

As you can notice that our guest virtual machine is running on firecracker with root privileges

```
root@test-machine:~# whoami
root
root@test-machine:~#
```

You can also check the network ports connected to the Virtual machine.

```
root@test-machine:~# ip addr
-bash: ip: command not found
root@test-machine:~# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
root@test-machine:~#
```

As you can see by default there no network configuration.

Firecracker network setup:

Currently firecracker supports only TUN/TAP network backend with no multi queue support. The simple steps followed here will assume that your internet-facing interface is wlan0, you have nothing else using tap0 and no other iptables rules.

On the host:

- 1) The first step is to create Tap device on host.
- 2) Then you have a few options for routing traffic out of the tap device, through your host's network interface. One option is NAT
- 3) All commands are written in a bash script.

```

#!/usr/bin/env sh
sudo ip tuntap add tap0 mode tap
sudo ip addr add 172.16.0.1/24 dev tap0
sudo ip link set tap0 up
sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
sudo iptables -A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
sudo iptables -A FORWARD -i tap0 -o eth0 -j ACCEPT
sudo iptables-save > iptables.rules.old

```

Run the script netconfig.sh

```

l/debug# bash netconfig.sh

```

- 4) Before starting the guest, configure the network interface using Firecracker's API:
Create a bash script that contain commands to configure network.

```

/debug# nano network.sh

```

```

#!/usr/bin/env sh
curl --unix-socket /tmp/firecracker.socket -i \
-X PUT 'http://localhost/network-interfaces/wlan0' \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
  "iface_id": "wlan0",
  "guest_mac": "AA:FC:00:00:00:01",
  "host_dev_name": "tap0"
}'

```

Run the file network.sh before starting virtual machine.

```

root@Anonymous:~/Firecracker/firecracker/build/cargo_target/x86_64-unknown-linux-musl/debug# bash network.sh
HTTP/1.1 204
Server: Firecracker API
Connection: keep-alive

```

After running the configuration and starting guest Virtual machine again you can notice that now you have a network interference on the guest machine.

In Guest Machine:

In Guest machine now you can check the network interferences.

```

root@test-machine:~# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether aa:fc:00:00:00:01 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::a8fc:ff:fe00:1/64 scope link
        valid_lft forever preferred_lft forever
root@test-machine:~#

```

As you can see now there is an interference with name of eth0 and activated on guest machine.

Run the following command on Guest machine to assign the Ip on ethernet.

```
root@test-machine:~# ip addr add 172.16.0.2/24 dev eth0
root@test-machine:~# ip link set eth0 up
root@test-machine:~# ip route add default via 172.16.0.1 dev eth0
root@test-machine:~#
```

you can add a public DNS server to /etc/resolv.conf by adding a line like this:

```
root@test-machine:~# cat > /etc/resolv.conf
nameserver 8.8.8.8
root@test-machine:~#
```

Now you can check the network address of interference

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_f
default qlen 1000
    link/ether aa:fc:00:00:00:01 brd ff:ff:ff:ff:ff:ff
    inet 172.16.0.2/24 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::a8fc:ff:fe00:1/64 scope link
        valid_lft forever preferred_lft forever
root@test-machine:~#
```

After completing the network setup, you can check the connectivity by Pinging the Host machine.

```
root@test-machine:~# ping 192.168.100.206
PING 192.168.100.206 (192.168.100.206) 56(84) bytes of data.
64 bytes from 192.168.100.206: icmp_seq=1 ttl=64 time=0.308 ms
64 bytes from 192.168.100.206: icmp_seq=2 ttl=64 time=1.23 ms
64 bytes from 192.168.100.206: icmp_seq=3 ttl=64 time=1.02 ms
64 bytes from 192.168.100.206: icmp_seq=4 ttl=64 time=0.979 ms
```

Now the firecracker and VM is completely ready to run for further tasks.

Machine Learning Inference Model:

Introduction:

Machine learning (ML) inference is the process in which it takes live data and feed the data to ML model. Machine learning model then calculate output as a single value based on input values. There are features by which machine learning inference model give the output.

Working of ML Inference Model.

There are three main components needed to deploy machine learning inference model.

- Data source could be one or more
- Machine learning model on system host.
- Output destination of data could be one or more.

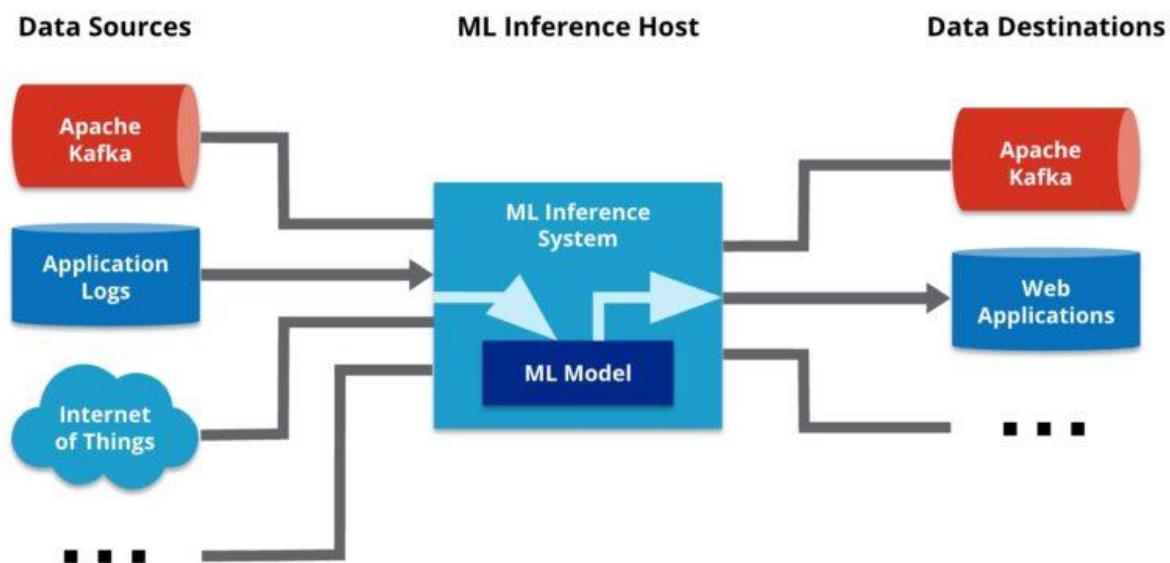


Figure 3 Machine Learning Inference Model

Support Vector Machine:

Support vector machines (SVMs) is learning method used for classification and regression. It a supervised learning method.

The advantages of support vector machines are:

- better for high dimensional datasets.
- Not effected by large difference between number of samples and dimensions.
- Uses a part of training points in the decision function, it makes it more memory efficient.
- Can use different kernel functions.
- I have used Support Vector Machine (SVM) algorithm to train my Model.

Distributed Denial of Service Attack:

A distributed denial of service attack is a type of cyber-attack that aims to disturb the availability of a targeted service by overwhelming it with traffic from multiple sources. Details of attacks are launched by a group of computers or devices that are compromised by malware and controlled by a single attacker.

There have been several examples of distributed denial of service attack on edge computing systems. Here are a few examples.

- 1) In 2019 Akamai Technologies published a report on a DDoS attack that targeted an edge computing system used by a gaming company. The attack generated over 100 Gbps of traffic and lasted for several hours, causing disruptions to the gaming company's services. [7]
- 2) In 2020, a DDoS attack targeted a cloud-based edge computing platform used by a financial services company. The attack generated over 1 TBPS of traffic in last year for several days, causing disruptions to the financial services company's services. [8]
- 3) In 2021 DDoS attacks targeted a content delivery network (CDN) that was used to deliver content to edge computing systems. The attack generated over 2.3 TBPS of traffic and lasted for several hours, causing disruption to the CDN's services. [9]

These examples demonstrate their significant impact that DDoS attacks can have on edge computing systems and the importance of implementing measures to protect against such attacks.

Dataset:

Dataset used for this model is "DDoS attack SDN Dataset" [10]. The Mininet emulator is used to create dataset. A single Ryu controller is used for this project. Then ten topologies are created in the Mininet.

The number of features taken in this dataset are 13. The features are:

- pktcount – Number of packets
- bytecount – number of bytes
- Port- Port number
- dur – duration in seconds
- tx_byte- the number of bytes transferred from the switch port
- rx_byte- the number of bytes received on the switch port
- pktperflow- it is the packet count during a single flow
- byteperflow – Is the byte count during a single flow
- tx_kbps - data transfer rate
- rx_kbps – data receiving rate
- tot_kbps – total sum of tx_kbps and rx_kbps
- flow: Number of flows monitored on interval of 30 seconds
- label – 0 for normal traffic and 1 for malicious traffic.

And the total number of records in dataset is 104345.

ML Model in Python:

Python language is used to create or model and to train our model so it can predict the real-life attacks.

The IDE used to program the model is Anaconda.

Step 1: Load the dataset already available in Local machine.

```
@author: Engr-Yasir  
"""
```

```
import pandas as pd  
dataset = pd.read_csv('C:/Users/ab12/Documents/ml-Model/dataset_sdn1.csv')
```

Output Variables:

dataset	DataFrame	(104345, 13)	Column names: pktcount, bytecount, dur, flows, pktperflow, byteperfl ...
---------	-----------	--------------	---

Step2: Split the data in x and y. 'x' is the feature of dataset and 'y' is the label of records.

```
X = dataset.iloc[:,0:12]  
y = dataset.iloc[:,12:13]
```

Output Variables.

y	DataFrame	(104345, 1)	Column names: label
X	DataFrame	(104345, 12)	Column names: pktcount, bytecount, dur, flows, pktperflow, byte ...

Step3: Split the dataset in training and testing sets. Here we are using 75% of dataset as training and 25% as testing purpose

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

Output variables:

X_test	DataFrame	(26087, 12)	Column names: pktcou ...
X_train	DataFrame	(78258, 12)	Column names: pktcou ...
y_test	DataFrame	(26087, 1)	Column names: label
y_train	DataFrame	(78258, 1)	Column names: label

Step4: Use Standard Scaler library of Sklearn to standardize our dataset.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Output variables:

X_test	float64	(26087, 12)	[[0.50968912 0.95094804 -0.101091686 ... -0 ...
X_train	float64	(78258, 12)	[[-1.01091686 -0.78274816 -0.101091686 ... 0 ...

Step5: Import the Support vector machine (SVM) library and name it as Classifier.

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
```

Step6: Now time to train our model on training dataset. We fit the data in classifier.

```
classifier.fit(X_train, y_train)
```

Output:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=False, random_state=0,
    shrinking=True, tol=0.001, verbose=False)
```

Step7: After running the above command model will start training, once its complete training, save the trained model.

```
import pickle
pickle.dump(classifier,open('C:/Users/ab12/Documents/ml-Model/ml.pkl','wb'))
```

A file will be created on given path with name 'ml.pkl'. we can use this file to predict the data.

Step8: As our model is trained, now test the model on testing dataset. To test the model, we provide just feature without label. Model will predict the labels for each record.

```
y_pred = classifier.predict(X_test)
```

Output variables:

y_pred	int64	(26087,)	[0 1 1 ... 0 1 0]
--------	-------	----------	-------------------

Step9: calculate the evaluation metrics. here I have taken Accuracy, Precision, Recall, and F1-score as evaluation metrics.

```

from sklearn.metrics import accuracy_score ,precision_score,recall_score ,f1_score
print('\n      Accuracy is %f Percent' %(accuracy_score(y_test,y_pred)*100))
print('\n      Precision score is %f Percent' %(precision_score(y_test,y_pred)*100))
print('\n      Recall Score is %f Percent' %(recall_score(y_test,y_pred)*100))
print('\n      F1-score is %f Percent' %(f1_score(y_test, y_pred)*100))

```

Output:

```

Accuracy is 93.161345 Percent

Precision score is 88.720539 Percent

Recall Score is 94.256757 Percent

F1-score is 91.404895 Percent

```

ML Inference code:

There are two modes in inference model. One is, inference model will take a complete file of stored records captured from network traffic and predict the traffic is normal or malicious and second is code will ask user to input the required parameters and the predict the output of traffic. Inference model is command line base.

Mode 1: This code will take a sheet from user and predict the values by trained model we already saved. As output, it will label each packet as normal or malicious and display the evaluation metrics.

```

file_path = input('\n \n Enter the path of file: ')
if os.path.exists(file_path):
    data = pd.read_csv(file_path)
    X_test= data.iloc[:, 0:12].values
    Y_test= data.iloc[:, 12:13].values

    from sklearn.preprocessing import StandardScaler
    sc = StandardScaler()
    X_test = sc.fit_transform(X_test)

    classifier = joblib.load('ml.pkl')

    t1= datetime.datetime.now()
    y_pred= classifier.predict(X_test)

    t2= datetime.datetime.now()

    counter = 1
    for i in y_pred:
        if i == 0:
            print("\n The Network packet number " + str(counter) + " is Normal.")
            counter += 1
        else:
            print("\n The Network packet number " + str(counter) + " is Malicious")
            counter += 1

    from sklearn.metrics import accuracy_score,precision_score,recall_score ,f1_score
    print('\n      Accuracy is %f' %(accuracy_score(Y_test,y_pred)*100))
    print('\n      Precision score is %f' %(precision_score(Y_test,y_pred)*100))
    print('\n      Recall Score is %f' %(recall_score(Y_test,y_pred)*100))
    print('\n      F1-score is %f' %(f1_score(Y_test, y_pred)*100))
    delta= t2-t1
    testingtime=int(delta.total_seconds()*1000)
    print("\n Testing time of model is ", testingtime, "milliseconds")

```

Myode2: In second mode code will ask user to input value of each parameter manually, on given values the model will predict the traffic is normal or malicious.

```

classifier = joblib.load('ml.pkl')
print("Extract the following feature from network traffic and input values")
pktcount=int(input('Total number of packets:'))
bytecount=int(input('Total number of bytes:'))
dur=int(input('Total duration in seconds:'))
flows=int(input('Number of flow Entries in switch:'))
pktperflow=int(input('Packet count during a single flow:'))
byteperflow=int(input('Byte count during a single flow:'))
port_no=int(input('Port number:'))
tx_bytes=int(input('Number of bytes transfer from switch port:'))
rx_bytes=int(input('Number of bytes received on switch port:'))
tx_kbps=int(input('data transfer rate:'))
rx_kbps=int(input('data receiving rate:'))
tot_kbps=int(input('Total bandwidth:'))

X_test=[[pktcount,bytecount,dur,flows,pktperflow,byteperflow,port_no,tx_bytes,rx_bytes,tx_k
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_test = sc.fit_transform(X_test)

t1= datetime.datetime.now()
y_pred= classifier.predict(X_test)
t2= datetime.datetime.now()

for i in y_pred:
    if i == 0:
        print("\n The Network packet number " + str(counter) + " is Normal.")
    else:
        print("\n The Network packet number " + str(counter) + " is Malicious")
delta= t2-t1
testingtime=int(delta.total_seconds()*1000)
print("\n\nTesting time of model is ", testingtime, "milliseconds")

```

Output: After execution the code, Model will ask to select mode of prediction.

```

(base) C:\Users\ab12\Documents\ml-Model>python Exc.py
DDOS- DETECTION
Developed by: Yasir Hussain

A Machine Learning Inference Model to detect DDOS attack by analysing Network traffic.
This Model is Designed to work in two modes.
1: Predict Traffic from Network traffic file
2: Predict Traffic by Manual input values
Choose the mode of Prediction:

```

On selecting First mode it will ask the path of file.

```

Choose the mode of Prediction: 1

Enter the path of file: _

```

As soon path is provided, model will start prediction the output and give results.

```

The Network packet number 1000 is Normal.

    Accuracy is 92.400000 Percent

    Precision score is 91.948052 Percent

    Recall Score is 88.721805  Percent

    F1-score is 90.306122 Percent

Testing time of model is  424 milliseconds

would you like to analyse more Network traffic?(y/n)

```

Then model will ask to continue or quit.

```

would you like to analyse more Network traffic?(y/n)

```

On selecting mode 2 the model will ask value for each parameter separately and predict the output.

```

Choose the mode of Prediciton: 2
Extract the following feature from network traffic and input values
Total number of packets:2742790
Total number of bytes:456546
Total duration in seconds:322343
Number of flow Entries in switch:5646747
Packet count during a single flow:34535
Byte coubt during a single flow:354656
Port number:4534
Number of bytes transfer from switch port:454
Number of bytes received on switch port:4534
data transfer rate:56
data receiving rate:346
Total bandwidth:4636

The Network packet is Malicious

Testing time of model is  15 milliseconds

would you like to analyse more Network traffic?(y/n)

```

Choosing to quit will display a message and quit the model.

```

would you like to analyse more Network traffic?(y/n)  n

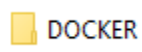
Good Bye!

```

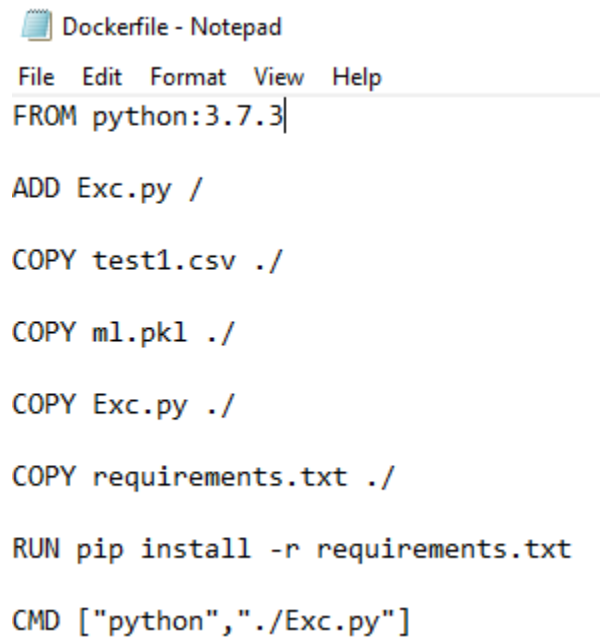
Deployment on Docker :

To deploy our machine learning inference model in docker, Initial step is to build a docker.

Step1: Create a separate folder for the Files of Docker. Here I named folder as DOCKER.



Step2: To build a docker, there must be a docker file. Create docker file notepad and save it without any extension. Docker file contains the necessary command to run during building docker image.



```
File Edit Format View Help
FROM python:3.7.3|

ADD Exc.py /

COPY test1.csv ./

COPY ml.pkl ./

COPY Exc.py ./

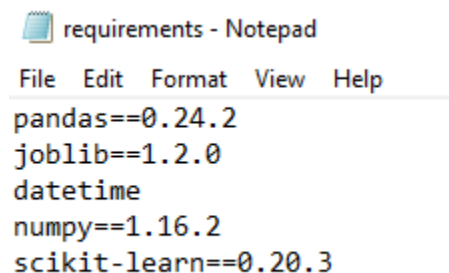
COPY requirements.txt ./

RUN pip install -r requirements.txt

CMD ["python","./Exc.py"]
```

For docker image we are using Python as base image with version of 3.7.3. Then we copy required files from our host machine to docker image. Run command install the libraries and packages required for running code from the text file. At end CMD command will initialize terminal to run the executable file.

Step3: create the requirement.txt file which contains the required packages to run the model.



```
File Edit Format View Help
pandas==0.24.2
joblib==1.2.0
datetime
numpy==1.16.2
scikit-learn==0.20.3
```

NOTE: install the same version of packages that were installed on host machine during the developing model.

Step4: Keep all required files in the main folder created in first step.

Name	Date modified	Type	Size
Dockerfile	12/19/2022 3:00 AM	File	1 KB
Exc	12/18/2022 12:12 AM	PY File	5 KB
ml.pkl	12/17/2022 11:16 PM	PKL File	1,695 KB
requirements	12/19/2022 3:04 AM	Text Document	1 KB
test1	12/17/2022 2:13 PM	Microsoft Excel C...	59 KB

Step5: Open terminal on same path of Docker folder. For building docker, write the following command in terminal.

```

Anaconda Prompt
(base) C:\Users\ab12\Documents\ml-Model\DOCKER>docker build -t ml-model -f Dockerfile .

```

Command have following parameters:

Build: Build is used to create a new docker image from docker file.

-t: create terminal assigned to image.

-f: it assigns the docker file to build

Step6: Run the command. It will start collection required environment, packages and files for docker image.

```

(base) C:\Users\ab12\Documents\ml-Model\DOCKER>docker build -t ml-model -f Dockerfile .
[+] Building 6.8s (11/12)
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 32B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/python:3.7.3
=> [1/7] FROM docker.io/library/python:3.7.3@sha256:9e0b4f32487ca1863b45383420b8db7799
=> [internal] load build context
=> => transferring context: 120B
=> CACHED [2/7] ADD Exc.py /
=> CACHED [3/7] COPY test1.csv ./
=> CACHED [4/7] COPY ml.pkl ./
=> CACHED [5/7] COPY Exc.py ./
=> CACHED [6/7] COPY requirements.txt ./
=> CACHED [7/7] RUN pip install -r requirements.txt
=> exporting to image
=> => exporting layers
=> => writing image sha256:5053250b9a1cf234078209hef98d4f20b7ce8fc7e320fe44a7ec2a10e1f

```

Step7: after completion of above process, check the images build for docker.

```

(base) C:\Users\ab12\Documents\ml-Model\DOCKER>docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
ml-model      latest    5053250b9a1c   42 minutes ago 1.28GB

```

Step 8: Run the built docker.

```

(base) C:\Users\ab12\Documents\ml-Model\DOCKER>docker run --cpus 2 --memory 2g -it ml-model

```

The defined flags in this command are:

Run: this flag is to run the image on docker as container.

--cpus: it is used to limit the CPU for specific container. Here 2 virtual CPU has assigned to the container.

--memory: This flag assigns the RAM to the container. 2GB of ram is assigned to the container.

-it: is used to start a Docker container in interactive mode.

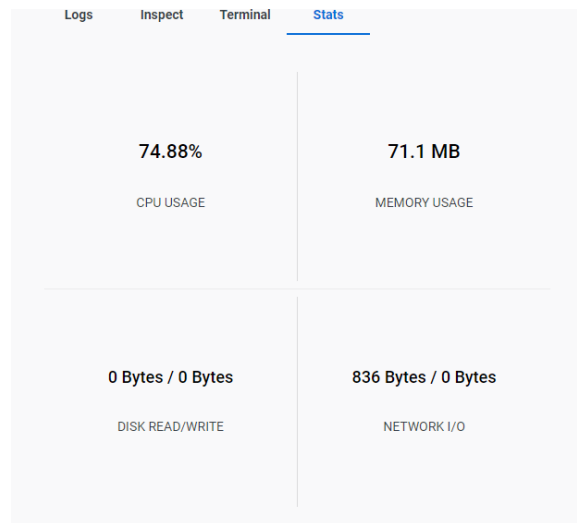
```
base) C:\Users\ab12\Documents\ml-Model\DOCKER>docker run --cpus 2 --memory 2g -it ml-model
DDOS-DETECTION
Developed by: Yasir Hussain
A Machine Learning Inference Model to detect DDOS attack by analysing Network traffic.
This Model is Designed to work in two modes.
1: Predict Traffic from Network traffic file
```

Machine learning inference model will start as docker image in container.

After giving the required parameter the model will analyses the data and give output.

```
The Network packet number 999 is Normal.
The Network packet number 1000 is Normal.
Accuracy is 92.400000 Percent
Precision score is 91.948052 Percent
Recall Score is 88.721805 Percent
F1-score is 90.306122 Percent
Testing time of model is 1171 milliseconds
would you like to analyse more Network traffic?(y/n)
```

To check the total CPU utilized in this process, analyses the stats given for each container in docker desktop portal. We will consider the top usage by the container.



Deployment on Firecracker


Step1: Copy All the required files to the Firecracker VM. You can use ssh service on the Guest VM to transfer files from Host machine.

```
root@test-machine:/home/ml-model# ls
Exc.py  ml.pkl  test1.csv
root@test-machine:/home/ml-model#
```

Step2: Install the required packages in the Machine by APT.

Step3: Execute the Script of Machine learning Inference Model.

```
root@test-machine:/home/ml-model# python3 Exc.py
```



```
Developed by: Yasir Hussain

A Machine Learning Inference Model to detect DDOS attack by analysing Network traffic.
```

This Model is Designed to work in two modes.

- 1: Predict Traffic from Network traffic file
- 2: Predict Traffic by Manual input values

Step4: Give the file to analyze the network traffic.

```
This Model is Designed to work in two modes.

1: Predict Traffic from Network traffic file

2: Predict Traffic by Manual input values

Choose the mode of Prediciton: 1

Enter the path of file: test1.csv
/usr/lib/python3/dist-packages/sklearn/utils/validation.py:429: DataConversionWarni
```

Step5: You can see the output of analyses.

```

The Network packet number 1000 is Normal.

    Accuracy is 92.400000 Percent

    Precision score is 91.948052 Percent

    Recall Score is 88.721805 Percent

    F1-score is 90.306122 Percent

Testing time of model is 532 milliseconds

would you like to analyse more Network traffic?(y/n)

```

```

2. Predict Malware by Manual Input values

Choose the mode of Prediciton: 2
Extract the following feature from network traffic and input values
Total number of packets:212323
Total number of bytes:2312312
Total duration in seconds:334
Number of flow Entries in switch:31
Packet count during a single flow:23
Byte coubt during a single flow:47447
Port number:2
Number of bytes transfer from switch port:78
Number of bytes received on switch port:21
data transfer rate:4447
data receiving rate:2123
Total bandwidth:447

The Network packet is Malicious

Testing time of model is 1 milliseconds

would you like to analyse more Network traffic?(y/n)

```

Step6: You can monitor the CPU usage by tools like ps, top, htop et

```

top - 17:18:26 up 5:05, 1 user, load average: 0.70, 0.54, 0.64
Tasks: 1 total, 0 running, 1 sleeping, 0 stopped, 0 zombie
%Cpu(s): 11.2 us, 5.2 sy, 0.0 ni, 83.0 id, 0.6 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 3712.6 total, 723.2 free, 1815.8 used, 1173.7 buff/cache
MiB Swap: 3876.0 total, 3637.7 free, 238.3 used, 1190.9 avail Mem

  PID USER   PR  NI  VIRT  RES  SHR S  %CPU  %MEM    TIME+  COMMAND
 6641 root    20   0 2109416 241764 4540 S   42.7   6.4   0:13.30 firecracker

```

Evaluation

The evaluation of Machine learning inference model is done by some Performance metrics.

Accuracy:

Accuracy is a measure of how well a model can predict the correct outcome it is defined as a number of correct predictions made by the model divided by the total number of predictions made in other words accuracy is the proportion of correct predictions made by the model.

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

Precision:

Precision is a measure of the proportion of positive predictions that are actually correct it is calculated by dividing the number of true positive predictions by the sum of the true positive and false positive predictions.

$$\text{Precision} = \frac{TP}{(TP + FP)}$$

Recall:

Recall is a measure of the proportion of actual positive examples that were correctly classified by the model it is calculated by dividing the number of true positive predictions by the sum of the true positive and false negative predictions.

$$\text{Recall} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalseNegative}}$$

F1-score:

the F1 score is a metric that combines precision and recall it is calculated by taking the harmonic mean of precision and recall.

$$F1 \text{ Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Testing Time:

The testing time of a model refers to the amount of time it takes we will read the model on a test data set.

	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	Testing Time(ms)
Docker	92.4	91.94	88.72	90.3	1171
Firecracker	92.4	91.94	88.72	90.3	532

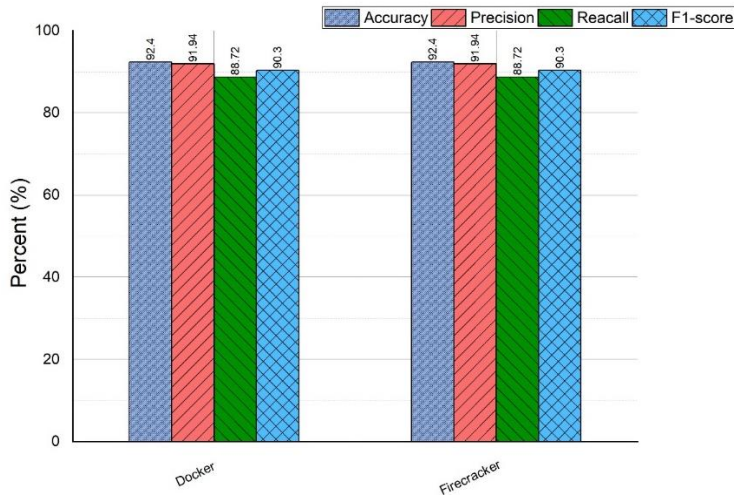


Figure 4 Evaluation Metrics

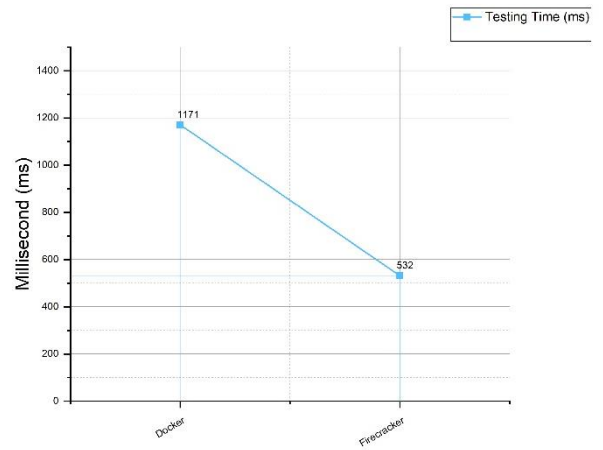


Figure 5 Testing Time

CPU Utilization:

CPU utilization refers to the amount of processing resources that are being used by a computer or device at a given time it is typically measured as a percentage of the total available processing power of the CPU.

	Percent (%)
Docker	74.88
Firecracker	42.7

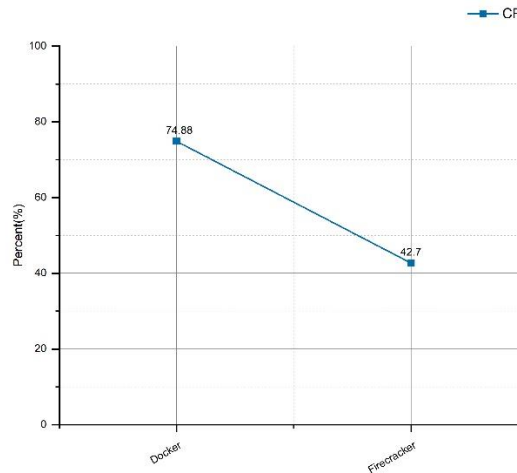


Figure 6 CPU Utilization

Memory Consumption:

Memory utilization refers to the amount of memory also known as ram that is being used by a computer or device at a given time.

	Megabytes (Mb)
Docker	71.1
Firecracker	131.02

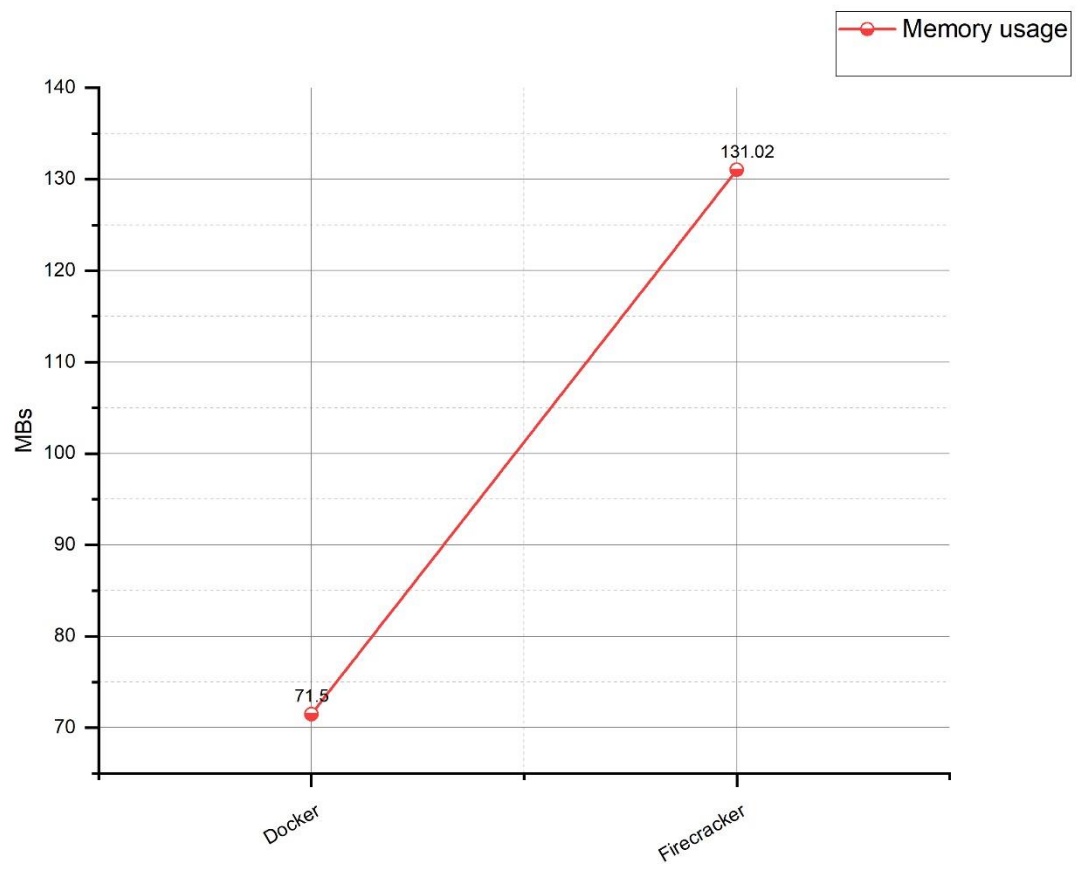


Figure 7 Memory (RAM) Utilization

Summary

Docker and Firecracker are both technologies that can be used to run application in lightweight virtual environments known as containers however they differ in several key aspects.

Purpose:

Docker is a general-purpose containerization platform over firecracker is specifically designed for running micro-VMS in a secure and efficient manner micro-VMS are lightweight variant of virtual machines that are optimized for running a single process or application

Architecture:

Docker uses a client server architecture where the docker demon server manages the containers and communicates with the docker client, firecracker on the other hand uses a unikernel-based architecture where a lightweight microkernel is used to run the application directly on the host operating system without the need for a full fresh virtual machine

Efficiency:

Firecracker is designed to be more efficient than docker as it uses a minimal return time and does not require the overhead of a full virtual machine. This makes it well suited for any multiple instances of workload that have low CPU memory and IO requirements such as serverless functions

Security:

Firecracker is designed to be more secure than docker as it uses a minimalist design and runs the application directly on the host operating system rather than instead of full virtual machine this reduces the attack surface and make it more difficult for an attacker to compromise the system

In summary docker is a general-purpose containerization platform that is suitable for running a wide variety of application while cracker is specialized platform that is optimized for running micro-VMS in a secure and efficient manner

Which is Better?

It is difficult to say which is “better” between Docker and Firecracker, as they are both useful tools with different purposes. Docker is a containerization platform that allows you to package application and their dependencies into self-contained units called containers, which can be easily deployed and run on the host machine with docker installed. Firecracker it's a lightweight virtualization technology that allows you to run multiple isolated environments called micro-VMS and on a single host machine.

Docker is generally easier to use and is more widely adopted as it has been around for longer and has a larger user base. It is good choice for developers who want to package and deploy their application in a portable and consistent way.

On other hand firecrackers is designed for a different use case it is intended to be used for running multiple workloads on cloud infrastructure in a fast and secure way. it is optimized for high performance and low overhead making it an attractive choice for running workloads that need to be isolated from each other but still need to run on the same host machine.

In general, whether you should use docker or fire content on your specific needs and use cases if you are a developer looking to a package and deploy your application in a portable way docker might be a good choice if you are running workloads on cloud infrastructure and need fast and secure isolation firecracker might be a better option.

References

- [1] "Docker Overview," [Online]. Available: <https://docs.docker.com/get-started/overview/index.html>.
- [2] J. Patil, "Docker Introduction, Architecture, and Command Details," oct 2022. [Online]. Available: <https://dzone.com/articles/docker-introduction-and-architecture?fromrel=true>.
- [3] P. pushkarna, "docker for java Developers," [Online]. Available: <https://slides.com/pulkitpushkarna/docker-for-java-developers/fullscreen>.
- [4] A. A. a. M. B. a. A. F. a. A. I. a. A. L. a. R. N. a. P. P. a. D.-M. Popa, "Firecracker: Lightweight virtualization for serverless applications," *NSDI 2020*, 2020.
- [5] J. Barr, "Firecracker – Lightweight Virtualization for Serverless Computing," 26 NOV 2018. [Online]. Available: <https://aws.amazon.com/blogs/aws/firecracker-lightweight-virtualization-for-serverless-computing/>.
- [6] "Getting Started," AWS, [Online]. Available: <https://github.com/firecracker-microvm/firecracker/blob/main/docs/getting-started.md>.
- [7] A. technologies, "Akamai observes record breaking 100 gbps ddos attack," 2019. [Online]. Available: <http://www.akamai.com/us/en/about/news/press/2019-press/akamai-observes-record-breaking-100-gbps-ddos-attack.jsp>.
- [8] J. Vijayan, "Financial Services Firms Targeted in Ransom DDoS Attacks in 2020," 2021. [Online]. Available: <https://www.darkreading.com/attacks-breaches/100-financial-services-firms-targeted-in-ransom-ddos-attacks-in-2020>.
- [9] P. Arntz, 2021. [Online]. Available: <https://www.malwarebytes.com/blog/news/2021/08/largest-ddos-attack-ever-reported-gets-hoovered-up-by-cloudflare>.
- [10] N. Ahuja, G. Singal and D. Mukhopadhyay, "DDOS attack SDN Dataset," *doi: 10.17632/jxpfjc64kr.1*, 2020.