INDEPENDENT STUDY: CLOUD NETWORKING

Prepared By: Sneha Rangari
Faculty Supervisor: Dr. Pu Wang

SEMESTER: FALL 2018 | CREDITS: 3
10th Dec 2018

Expected student learning outcomes/ objectives:
Distributed cloud networking, operating principles with SDN

Grading / evaluation criteria:
Survey of existing technologies and simulation verification.

Assignments:
Research articles, workshop discussions and practical simulation.

UNDERSTANDING PREVAILING RESEARCH:

Cloud networking, one of the emerging trend in industry which is a form of Software Defined Networking (SDN) technology, in which groups of networking switches and access devices can be deployed over the wide area as shared, virtual resources. There are lot of research papers proposing the solution on network management.

Based on the interested topic of operating principles of SDN, initial study started with understanding of four research papers of emerging solution on container networking. These papers include existing industry challenges like poor performance, poor portability, problems related to application deployment, etc. and proposed solutions on container networking, solving identity crises, measurement and evaluation for docker networking and networking in container.

Synopsis about research papers:

1. **FreeFlow - High Performance Container Networking**
   Problem: Current container networking solutions have either poor performance or poor portability, which undermines the advantages of containerization.
   Solution: FreeFlow, a container networking solution which achieves both high performance and good portability.
2. **FreeFlow: Software-based Virtual RDMA Networking for Containerized Clouds**
   Problem: Popular large-scale cloud applications are increasingly using containerization for high resource efficiency and lightweight isolation. Also data-intensive applications (e.g., deep learning frameworks) are adopting RDMA for high networking performance.
   Solution: FreeFlow fully satisfies the requirements from cloud environments, such as isolation for multi-tenancy, portability for container migrations, and controllability for control and data plane policies.
3. **AppSwitch: Resolving the Application Identity Crisis**
   Problem: The default application identity acquired from the host results in subtle and substantial problems related to application deployment, discovery and access, especially for modern distributed applications. A number of mechanisms and workarounds are used to address those problems but they only address them indirectly and incompletely.
   Solution: APPSWITCH, a novel transport layer network element that decouples applications from underlying network at the system call layer and enables them to be identified independently of the network.
4. **Measurement and Evaluation for Docker Container Networking**
   Evaluation: This paper investigates and analyzes the current status of the development of container network, mainly container network model and network solutions. It also measures and evaluates the performance of three mainstream network solutions (Flannel, Swarm Overlay, Calico).
   Result: Calico is of the highest performance but configuration is the most complicated; Flannel configuration is simple, the performance is not high; Swarm overlay configuration is simple with low performance.

   Looking at different concepts of research papers, we decided to have more focused understanding and simulation of FreeFlow.

PRESENTATIONS:

Next part of the independent study was to present thoughts based on understanding after going through two research papers namely FreeFlow and AppSwitch.

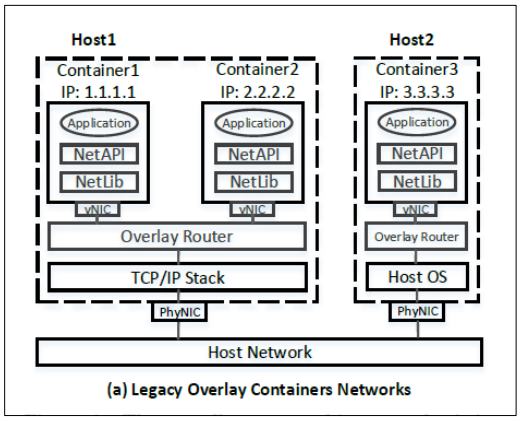
Important points from the presentation on topics are as follow:

- ## FREEFLOW:

To overcome the problem of poor performance and portability of existing networking solutions FreeFlow is designed which provides a virtual interface inside each container, and applications can use rdma via a virtual network on top of the virtual interface in an unmodified way. Ideally, the performance of the virtual network should be close to bare-metal rdma, and policies on both control and data path are flexible to be configured purely in software.

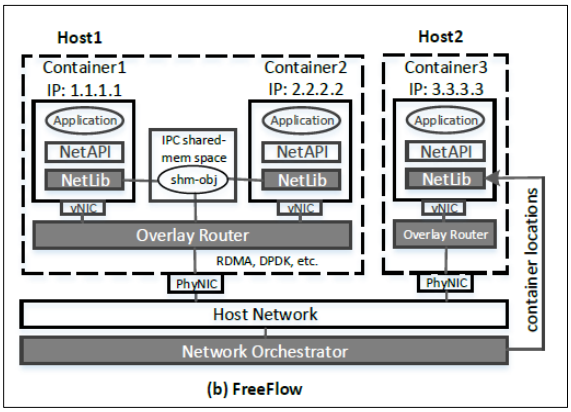
Architecture:

Existing Overlay Networking Solutions For Containers:



(a) Legacy Overlay Containers Networks

FreeFlow Container Solution:



(b) FreeFlow

Working:

Three key building blocks of FreeFlow:

1. **Customized Network Library**:
   - Decides which communication paradigm to use
   - Supports standard network programming APIs
   - Queries the network orchestrator for the location of the container it wishes to communicate with.
   - Uses shared memory to communicate with the other container, bypassing overlay router.
2. **Customized Overlay Routers**:
   - The traffic between routers and its local containers goes through shared-memory instead of software bridge; and
   - The traffic between different routers delivered via kernel bypassing techniques, e.g. RDMA or DPDK

3. **Customized Network orchestrator**
   - Keeps track of the realtime locations of each container in the cluster.
   - It also allows FreeFlow's network library to query for the physical deployment location of each container.

Overview: If the two containers are intra-host, shared-memory mechanism will be selected for data transfer, and if the two containers are inter-host, RDMA will be selected

- **APPSWITCH:**

A novel transport layer network element that decouples applications from underlying network at the system call layer and enables them to be identified independently of the network.

Architecture:

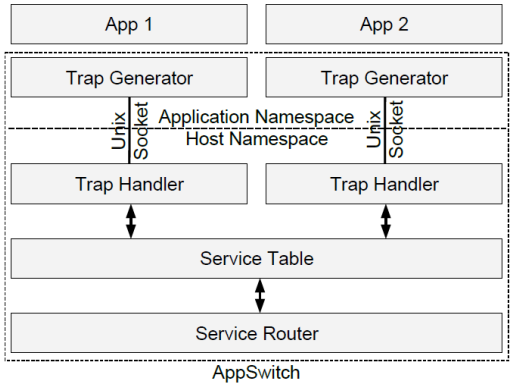


Figure 2: APPSWITCH architecture

Working:

Two key components

1. **Trap mechanism :**
   - Provides transparent application instrumentation
   - Redirects the network primitives of an application to a user space handler by interposing its network-related system call.

Has two components:

- Trap generator - intercepts network control plane system calls
- Trap handler - services them and returns result back to trap generator

2. **Service router**
   - Share a data structure called service table
   - Maintains a mapping between application identifiers and network level identifiers.
   - It propagates the contents of service table with other instances of APPSWITCH on other hosts over a gossip protocol

PRACTICAL SIMULATION OF FREEFLOW:

Next part was to actually perform practical simulation of FreeFlow which needs its source code including customized libraries (Netlib) and required specifications. To acquire source code and additional information I approached to the research team of FreeFlow.

FreeFlow Team: Hongqiang Harry Liu, Yibo Zhu, Daehyeok Kim and Tianlong Yu

Wherein they raised their helping hand and shared the practical implementation document as well as source code available on GitHub.

- ## SPECIFICATIONS OF DEVICES USED

  - To test the performance of FreeFlow we need actual bare-metal machines (Intel Xeon E5-2609 2.40GHz 4-cores CPU, 67 GB RAM, 40Gbps Mellanox CX3 NIC, CentOS 7) however it was not practically possible, so I have tested with virtual RDMA networking.
  - The OS is Ubuntu 14.04 with the kernel version 3.13.0-129-generic.
  - We run containers using Docker and set up a basic TCP/IP virtual network using Weave.
  - Mellanox OFED drivers

- ## OVERVIEW:

  FreeFlow works on top of popular overlay network solutions including flannel, weave, etc. The containers have their individual virtual network interfaces and IP addresses, and do not need direct access to the hardware NIC interface. A lightweight FreeFlow library inside containers intercepts RDMA and TCP socket APIs, and a FreeFlow router outside containers helps accelerate those APIs.
  FreeFlow is developed based on Linux RDMA project (https://github.com/linux-rdma/rdma-core), and released with MIT license.

- ## THREE WORKING MODES

  FreeFlow works in three modes: fully-isolated RDMA (master branch), semi-isolated RDMA, and TCP (TCP branch).

  I. Fully-isolated RDMA provides the best isolation between different containers and works the best in multi-tenant environment, e.g., cloud. While it offers

typical RDMA performance (40gbps throughput and 1 or 2 microsecond latency), this comes with some CPU overhead penalty.

II. The TCP mode accelerates the TCP socket performance to the same as host network. On a typical Linux server with a 40gbps NIC, it can achieve 25gbps throughput for a single TCP connection and less than 20 microsecond latency.

iii. Semi-isolated RDMA has the same CPU efficiency as host RDMA, while does not have full isolation on the data path. It works the best for single-tenant clusters, e.g., an internal cluster.

- **DEMO OF FREEFLOW**

Below are the steps of running Freeflow in fully-isolated RDMA mode:

PREPARATIONS:

Used Quickstart Terminal to launch virtual machine and set terminal environment to point to it, a default machine was automatically created.

Command: $ docker-machine create --driver virtualbox default

STEPS:

➢ Start Freeflow router (one instance per server)

Command: sudo docker run --name router1 --net host -e "FFR_NAME=router1" -e "LD_LIBRARY_PATH=/usr/lib/:/usr/local/lib/:/usr/lib64/" -v /sys/class/:/sys/class/ -v /freeflow:/freeflow -v /dev/:/dev/ --privileged -it -d ubuntu:14.04 /bin/bash

```
docker@default:/$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
e3f700ed21f6        ubuntu:14.04        "/bin/bash"         3 weeks ago
Up 3 weeks                              router1
```

Then log into the router container with

Command: sudo docker exec -it router1 bash

```
docker@default:/$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
e3f700ed21f6        ubuntu:14.04        "/bin/bash"         3 weeks ago
Up 3 weeks                              router1
docker@default:/$ sudo docker exec -it router1 bash
root@default:/#
```

Downloaded and installed RDMA libraries and drivers in router container.

Freeflow is developed and tested with "MLNX_OFED_LINUX-4.0-2.0.0.1-ubuntu14.04-x86_64.tgz"
from ,http://www.mellanox.com/page/products_dyn?product_family=26.

```
root@default:/freeflow# ls
Freeflow-master  master.zip                                  weave
Freeflow.git     mlnx_ofed_eula?mtag=linux_sw_drivers
root@default:/freeflow# sudo wget http://www.mellanox.com/page/mlnx_ofed_eula?mt
ag=linux_sw_drivers&mrequest=downloads&mtype=ofed&mver=MLNX_OFED-4.0-2.0.0.1&mna
me=MLNX_OFED_LINUX-4.0-2.0.0.1-ubuntu14.04-x86_64.tgz
[1] 19831
[2] 19832
[3] 19833
[4] 19834
root@default:/freeflow# --2018-12-08 04:56:01--  http://www.mellanox.com/page/ml
nx_ofed_eula?mtag=linux_sw_drivers
Resolving www.mellanox.com (www.mellanox.com)... 72.21.92.229
Connecting to www.mellanox.com (www.mellanox.com)|72.21.92.229|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: 'mlnx_ofed_eula?mtag=linux_sw_drivers.1'
```

```
exit
docker@default:/freeflow$ sudo docker exec -it router1 bash
root@default:/# cd freeflow
root@default:/freeflow# ls
Freeflow-master  mlnx_ofed_eula?mtag=linux_sw_drivers
Freeflow.git     mlnx_ofed_eula?mtag=linux_sw_drivers.1
master.zip       weave
root@default:/freeflow#
```

Then, build the code using the script build-router.sh. In ffrouter/, start the router by running "./router router1".

```
LICENSE    build-client.sh  ffrouter  libmempool  libraries-router
README.md  build-router.sh  images    libraries
root@default:/freeflow/Freeflow-master# ./build-router.sh
+ test -d ./config
+ autoreconf -ifv -I config
autoreconf: Entering directory `.'
autoreconf: configure.ac: not using Gettext
autoreconf: running: aclocal -I config --force -I config
autoreconf: configure.ac: tracing
autoreconf: running: libtoolize --copy --force
libtoolize: putting auxiliary files in AC_CONFIG_AUX_DIR, `config'.
```

Errors while running build-router.sh:

1) Libibverbs 1.1.8 requirement:

```
checking for stdint.h... yes
checking for unistd.h... yes
checking size of long... 8
checking for pthread_mutex_init in -lpthread... yes
checking for ibv_cmd_open_xrcd in -libverbs... no
configure: error: ibv_cmd_open_xrcd() not found.  librdmacm requires libibverbs
1.1.8 or later.
root@default:/freeflow/Freeflow-master/libraries-router/librdmacm-1.1.0mlnx#
```

2) Make error:

```
Processing triggers for libc-bin (2.19-0ubuntu6.14) ...
root@default:/freeflow/Freeflow-master/ffrouter# make
g++ -std=c++11 -O3   -c -o main.o main.cpp
In file included from ffrouter.h:9:0,
                 from main.cpp:4:
rdma_api.h:10:34: fatal error: infiniband/verbs_exp.h: No such file or directory
 #include <infiniband/verbs_exp.h>
                                  ^
compilation terminated.
make: *** [main.o] Error 1
root@default:/freeflow/Freeflow-master/ffrouter#
```

To solve this, libibverbs1.1.8 was installed.

https://rpmfind.net/linux/rpm2html/search.php?query=libibverbs-devel

```
root@default:/freeflow/Freeflow-master/libraries-router/librdmacm-1.1.0mlnx# ls
AUTHORS          config.h.in       libibverbs-1.2.1.tar
COPYING          config.h.in~      libibverbs-1.2.1.tar.gz
ChangeLog        config.log        libibverbs_1.1.8.orig.tar.gz
Makefile.am      configure         librdmacm
Makefile.in      configure.ac      librdmacm-master
NEWS             debian            librdmacm.spec.in
README           docs              man
aclocal.m4       examples          master.zip
autogen.sh       include           rdma-core
autom4te.cache   libibverbs-1.1.8  src
config           libibverbs-1.2.1
root@default:/freeflow/Freeflow-master/libraries-router/librdmacm-1.1.0mlnx# cd
libibverbs-1.1.8
root@default:/freeflow/Freeflow-master/libraries-router/librdmacm-1.1.0mlnx/libi
bverbs-1.1.8# ls
```

```
configure: creating ./config.status
config.status: creating Makefile
config.status: creating libibverbs.spec
config.status: creating config.h
config.status: config.h is unchanged
config.status: executing depfiles commands
config.status: executing libtool commands
root@default:/freeflow/Freeflow-master/libraries-router/librdmacm-1.1.0mlnx/libi
bverbs-1.1.8# make
make  all-am
make[1]: Entering directory `/freeflow/Freeflow-master/libraries-router/librdmac
m-1.1.0mlnx/libibverbs-1.1.8'
make[1]: Leaving directory `/freeflow/Freeflow-master/libraries-router/librdmacm
-1.1.0mlnx/libibverbs-1.1.8'
```

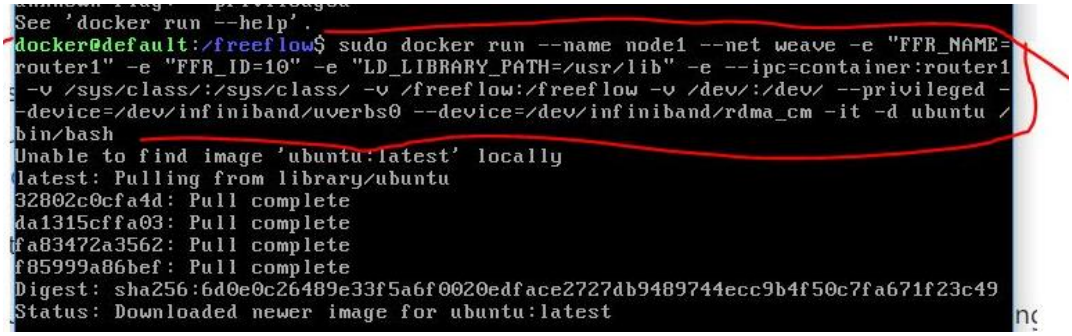
Figure: make command running successfully

Figure: make install command running successfully

- Repeat Step 1 to start the router in other hosts. You can capture a Docker image of router1 for avoiding repeating the installations and building.

- Start an application container on the same host as router1

  Command: sudo docker run --name node1 --net weave -e "FFR_NAME=router1" -e "FFR_ID=10" -e "LD_LIBRARY_PATH=/usr/lib" -e --ipc=container:router1 -v /sys/class/:/sys/class/ -v /freeflow:/freeflow -v /dev/:/dev/ --privileged --device=/dev/infiniband/uverbs0 --device=/dev/infiniband/rdma_cm -it -d ubuntu /bin/bash



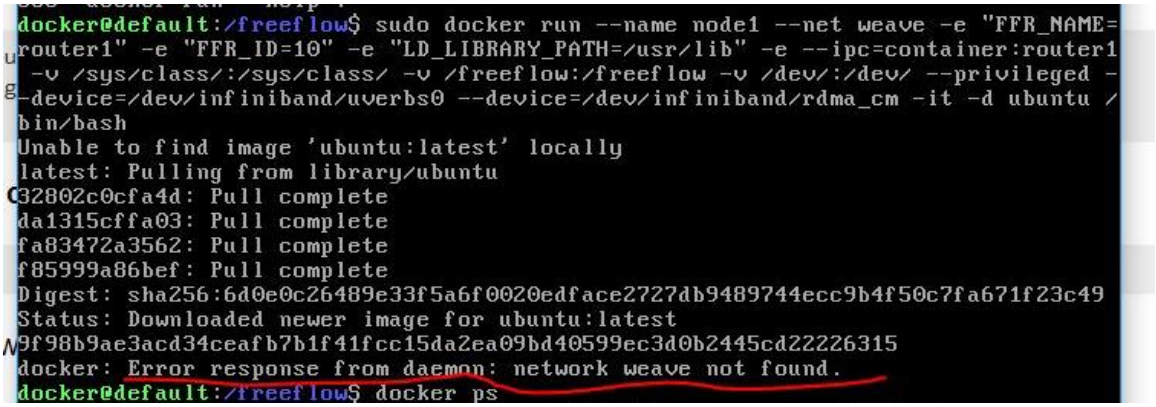
In this example, Weave (https://github.com/weaveworks/weave) is used.

Environment variable "FFR_NAME=router1" points to the container to the router (router1) on the same host; "FFR_ID=10" is the ID of the contaienr in FreeFlow. Each container on the same host should have a unique FFR_ID.

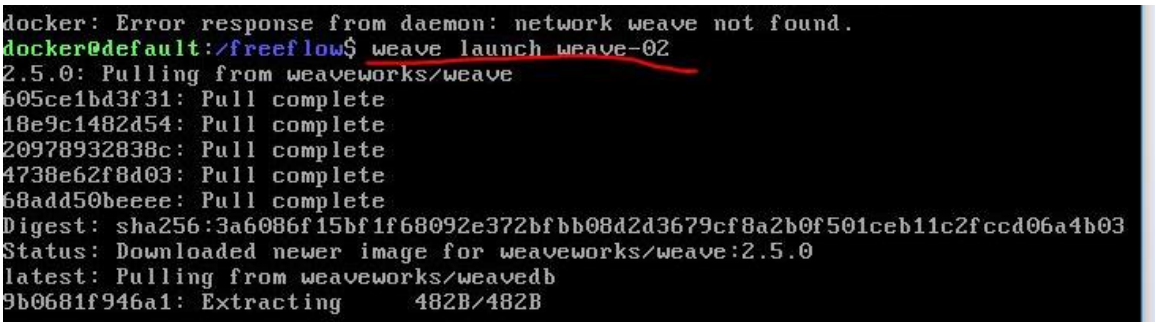
Download and install the same version of RDMA libraries and drivers as Step 1. Then build the the code of libraries/ and libmempool/ and install to /usr/lib/ (which is default).

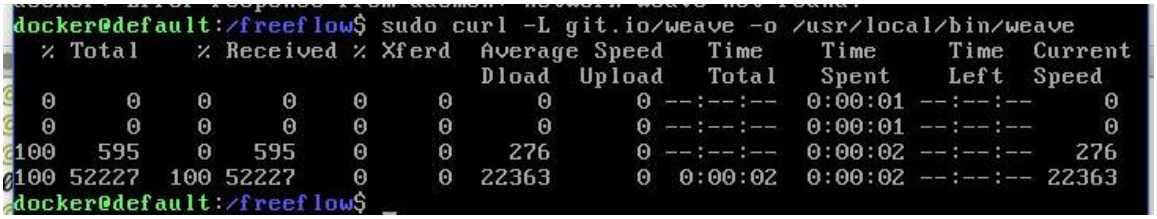
Errors:

1) Weave error:



After installing weave, error got solved.



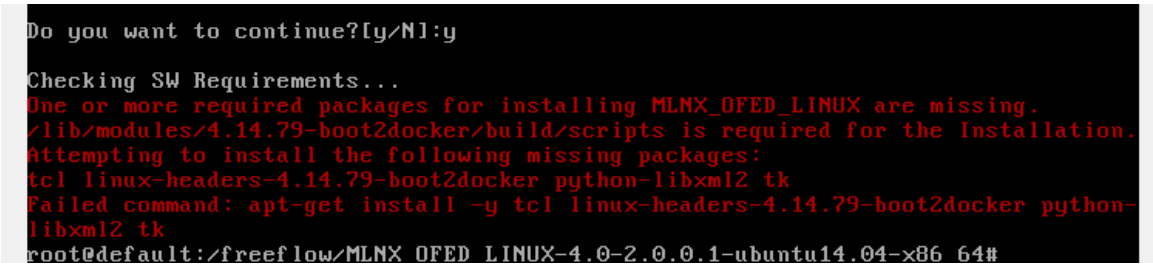


Step 4: Repeat Step 3 to start customer containers in more hosts.

- **TECHNICALLY LIMITATIONS:**

The issue was coming while installing the Mellanox OFED drivers as more than one packages were missing.

The solution for this issue was to install all the required packages to run the Mellanox Drivers. I started installing all the packages one by one which were necessary but remaining with two packages installation as below:

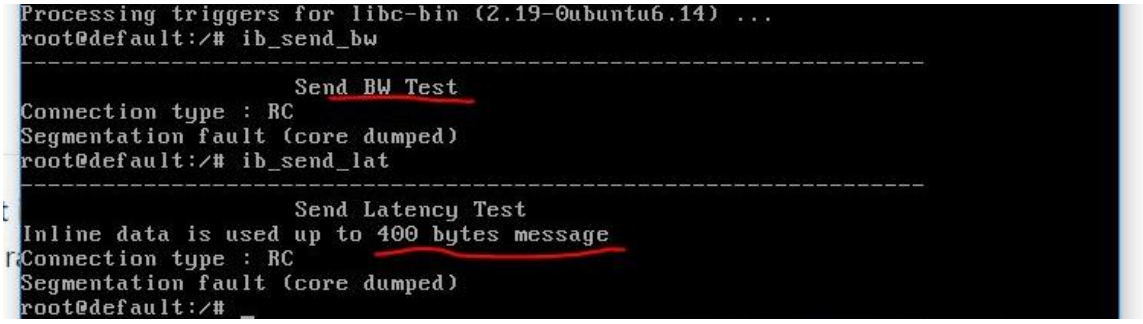


As packages namely linux-headers-4.14.79-boot2docker are not available for ubuntu 14, these are only issues faced to install the same.

Further I have contacted the support team of FreeFlow and as they are also new to this problem, they are working on Mellanox OFED installation issue.

VALIDATION:

In customer containers, install RDMA perftest tools with "sudo apt-get install perftest". Try "ib_send_bw" or "ib_send_lat".



Thus ib_send_lat performs a latency diagnostic and ib_send_bw performs a bandwidth test issued on the Linux InfiniBand host. Its not showing min and max timing for latency as drivers were not installed other customs containers are not created and not able to test bandwidth or latency between them.

CONCLUSION:

From this Independent research study, I have understood important concepts of cloud networking. After going through lot of research papers, workshop discussions and practical simulation I got expertise in following areas:

1) Container Networking
2) Virtual RDMA Networking
3) Existing technologies – FreeFlow, AppSwitch
4) Docker containerization
5) Linux concepts