# DevOps Notes: Vagrant and Linux

## Vagrant Architecture:

- Vagrant providers is like adapter for hypervisor and vagrant cli tool.
- **arm64** is for Mac architecture for host machine - similarly adm64 is for intel 5,6, 7 etc
- Always check the architecture and the vagrant provider supported to download.

```
➜ ~ sudo vagrant box list
sloopstash/amazon-linux-2 (virtualbox, 1.1.1, (amd64))
sloopstash/ubuntu-linux-18-04 (virtualbox, 1.1.1, (amd64))
sloopstash/ubuntu-linux-18-04 (vmware_desktop, 1.1.1, (amd64))



sudo vagrant box add sloopstash/amazon-linux-2 --provider
vmware_desktop -a amd64

Below error bcz registry not available for vmaware_desktop

➜ ~ sudo vagrant box add sloopstash/amazon-linux-2 --provider
vmware_desktop -a amd64
==> box: Loading metadata for box 'sloopstash/amazon-linux-2'
box: URL: https://vagrantcloud.com/api/v2/vagrant/sloopstash/amazon-
linux-2
The box you're attempting to add doesn't support the provider
you requested. Please find an alternate box or use an alternate
provider. Double-check your requested provider to verify you didn't
simply misspell it.

If you're adding a box from HashiCorp's Vagrant Cloud, make sure the
box is
released.

Name: sloopstash/amazon-linux-2
Address: https://vagrantcloud.com/api/v2/vagrant/sloopstash/amazon-
linux-2
Requested provider: vmware_desktop (amd64)
```

Ubuntu linux 22.04:

-------------------

```
sudo vagrant box add sloopstash/ubuntu-linux-22-04 --provider
vmware_desktop -a amd64
Will succeed - bcz amd64 architecture is supported from registry

sudo vagrant box add sloopstash/ubuntu-linux-22-04 --provider
vmware_desktop -a amd64 --box-version 2.1.1

Will Fail - bcz amd64 architecture is not supported from registry
The box you're attempting to add has no available version that
matches the constraints you requested. Please double-check your
settings. Also verify that if you specified version constraints,
that the provider you wish to use is available for these constraints.

Box: sloopstash/ubuntu-linux-22-04
Address: https://vagrantcloud.com/api/v2/vagrant/sloopstash/ubuntu-
linux-22-04
Constraints: 2.1.1
Available versions: 1.1.1, 2.1.1


sudo vagrant box add sloopstash/ubuntu-linux-22-04 --provider
vmware_desktop -a arm64 --box-version 2.1.1

arm64 is for Mac architecture
```

**Vagrant Box:**
Disk files required to build virtual machine
processor architecture (of host machine - means laptop or platform where VM being used ex: ARM64 or AMD64)
Ex: AWS EC2 is vagrant provider

Vagrant boxes are built using tool called Packer for ex: and put in registry (hashicorp)
https://portal.cloud.hashicorp.com/vagrant
Check local directory ~/.vagrant.d/

```
This directory contains boxes and etc ex:
/Users/rakshithtb/.vagrant.d/boxes/sloopstash-VAGRANTSLASH-ubuntu-
linux-18-04/1.1.1

.
```

```
└── amd64
├── virtualbox
|  ├── Vagrantfile
|  ├── box.ovf
|  ├── metadata.json
|  ├── ubuntu-bionic-18.04-cloudimg-configdrive.vmdk
|  ├── ubuntu-bionic-18.04-cloudimg.mf
|  └── ubuntu-bionic-18.04-cloudimg.vmdk
└── vmware_desktop
├── Vagrantfile
├── disk-s001.vmdk
├── disk-s002.vmdk
├── disk-s003.vmdk
├── disk-s004.vmdk
├── disk-s005.vmdk
├── disk-s006.vmdk
├── disk-s007.vmdk
├── disk-s008.vmdk
├── disk-s009.vmdk
├── disk-s010.vmdk
├── disk-s011.vmdk
├── disk-s012.vmdk
├── disk-s013.vmdk
├── disk-s014.vmdk
├── disk-s015.vmdk
├── disk-s016.vmdk
├── disk-s017.vmdk
├── disk.vmdk
├── metadata.json
├── ubuntu-18.04-amd64.nvram
├── ubuntu-18.04-amd64.vmsd
├── ubuntu-18.04-amd64.vmx
└── ubuntu-18.04-amd64.vmxf
```

Vagrant Conf file:
VagrantFile is what makes it a Iac component for vagrant"

- IaC automation format in vagrant to automate and run virtual machine.
- It uses Ruby syntax
- Any additional tool can be added using vagrant provision:
  - ex: Ansible
  - shell
  - chef etc
-

Very useful for MAC:
https://gist.github.com/sbailliez/f22db6434ac84eccb6d3c8833c85ad92

**Building on premise Virtual machines for Kubernetes cluster:**
chose server edition (without GUI) for vagrant boxes alternate is desktop version
vagrant multi machine configuration - multiple machines can be defined in VM  config file
and config for each machine

/opt/kickstart-kubernetes
https://github.com/sloopstash/kickstart-kubernetes/wiki

```
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-k8s-mtr-1

sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant up
sloopstash-k8s-wkr-1

sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
ssh sloopstash-k8s-wkr-1

to halt:
sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
halt sloopstash-k8s-mtr-1

To destroy:
sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
destroy sloopstash-k8s-mtr-1

reason why we use vagrant_cwd (vagrant current working directory)
format - bcz vagrant files resides in nested directories - we should
always use root path for the vagrant up or any command for that sake
(best practice)

Mac
----
$ sudo git clone https://github.com/sloopstash/kickstart-docker.git
/opt/kickstart-docker
$ sudo chown -R $USER /opt/kickstart-docker
$ cd /opt/kickstart-docker
```

**Docker Swarm:**

```
sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant up
sloopstash-dkr-swm-mgr-1


sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
ssh sloopstash-dkr-swm-mgr-1

To halt:
sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
halt sloopstash-dkr-swm-mgr-1

To destroy:
sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
destroy sloopstash-dkr-swm-mgr-1
```

Linux:

```
Technology: 1990's (middle) -> Unix -> Linux (operating system)
Last years (nealy 6 years) => People havent understood the [core] of
the technology that is the OS.
DevOps (Automation, Infra, Platf) + Cybersecurity + Cloud + Fullstack
+ Testing + AI/ML engineer + Big Data (High-level) => Changing +/-
Linux (Docker+Kubernet+Containerd+CRIO) / Unix + Hardware (storage+
compute+network) (core) => dont change much

$ sudo VAGRANT_CWD=./vagrant/ubuntu-linux-18-04/vmware/amd64/server
vagrant up
above is for amd64

$ sudo VAGRANT_CWD=./vagrant/ubuntu-linux-22-04/vmware/arm64/server
vagrant up
above is for arm64

cat /etc/os-release
it will have all details like packager like debian, redhat for
/vagrant/alma-linux-9/vmware/arm64/server

NAME="AlmaLinux" => distro
VERSION="9.5 (Teal Serval)"
ID="almalinux"
ID_LIKE="rhel centos fedora" => red hat enterprise linux
VERSION_ID="9.5"
```

```
PLATFORM_ID="platform:el9"
PRETTY_NAME="AlmaLinux 9.5 (Teal Serval)"
ANSI_COLOR="0;34"
LOGO="fedora-logo-icon"
CPE_NAME="cpe:/o:almalinux:almalinux:9::baseos"
HOME_URL="https://almalinux.org/"
DOCUMENTATION_URL="https://wiki.almalinux.org/"
BUG_REPORT_URL="https://bugs.almalinux.org/"

ALMALINUX_MANTISBT_PROJECT="AlmaLinux-9"
ALMALINUX_MANTISBT_PROJECT_VERSION="9.5"
REDHAT_SUPPORT_PRODUCT="AlmaLinux"
REDHAT_SUPPORT_PRODUCT_VERSION="9.5"
SUPPORT_END=2032-06-01
```

```
uname -a
```

gives us linux kernal name version and all details

```
# Find Linux kernel name.
$ uname -s
Linux

# Find Linux kernel release.
$ uname -r
5.14.0-503.23.2.el9_5.aarch64

# Find Linux kernel version.
$ uname -v
#1 SMP PREEMPT_DYNAMIC Wed Feb 12 10:27:25 UTC 2025

# Find Linux kernel architecture.
$ uname -m
aarch64

# Find Linux kernel processor type.
$ uname -p
aarch64
```

task for me:
Checky why ubuntu 22-04 is not getting up and
How to build an automated Amazon Linux 2 server-based VM with Vagrant?
(task)

```
https://github.com/sloopstash/kickstart-linux/wiki/Automate-and-
deploy-Linux-VMs
```

```
sloopstash-dkr-swm-mgr-1: /tmp/vagrant-shell: line 7: python: command
not found
sloopstash-dkr-swm-mgr-1: mkdir: cannot create directory
‘/etc/supervisord.d’: File exists
```

**Linux text editors: (Terminal we can say - Text user interface)**

```
Vim
Edit: i
Copy: CTRL+SHIFT+C
Paste: CTRL+SHIFT+V
Command: ESC
Exit without saving: ESC - :q! - ENTER
Save and exit: ESC - :wq - ENTER

Nano
Copy: CTRL+SHIFT+C
Paste: CTRL+SHIFT+V
Exit without saving: CTRL+X - SHIFT+N - ENTER
Save and exit: CTRL+X - SHIFT+Y - ENTER
```

**Linux bootstrap:**

```
# Get Linux kernel info including host info.
$ hostnamectl status
# List Linux kernel modules.
$ lsmod - list of all modules
$ less /proc/modules - list all the live modules loaded for the kernel

# Get Linux kernel module info.
$ modinfo iwlwifi => Wireless WiFi driver for Linux
filename: /lib/modules/5.14.0-
503.26.1.el9_5.aarch64/kernel/drivers/net/wireless/intel/iwlwifi/iwlwi
fi.ko.xz
license: GPL
description: Intel(R) Wireless WiFi driver for Linux

$ modinfo e1000 => ethernet driver
```

```
filename: /lib/modules/5.14.0-
503.26.1.el9_5.aarch64/kernel/drivers/net/ethernet/intel/e1000/e1000.k
o.xz
license: GPL v2
description: Intel(R) PRO/1000 Network Driver

# List Linux kernels - it will have fallback kernels. if any kernel
fails to bootstrap
$ ls /boot/vmlinuz*
$ ls /boot/initrd.img*
$ ls /boot/initramfs*
# View GRUB bootloader configuration file.
$ cat /boot/grub/grub.cfg
$ cat /boot/grub2/grub.cfg -- boot loader configuration which decides
which kernel to load or bootstap

ps -ef => system process list
kthreadd is the first process in the list and it will start or handle
all other kthread modules
UID PID PPID C STIME TTY TIME CMD
root 1 0 0 02:14 ? 00:00:00 /usr/lib/systemd/systemd --switched-root -
-system --deserialize 31 no_timer_check
root 2 0 0 02:14 ? 00:00:00 [kthreadd]
root 3 2 0 02:14 ? 00:00:00 [pool_workqueue_]
root 4 2 0 02:14 ? 00:00:00 [kworker/R-rcu_g]
root 5 2 0 02:14 ? 00:00:00 [kworker/R-rcu_p]
root 6 2 0 02:14 ? 00:00:00 [kworker/R-slub_]
root 7 2 0 02:14 ? 00:00:00 [kworker/R-netns]
root 8 2 0 02:14 ? 00:00:01 [kworker/0:0-cgroup_destroy]


In the above we can see PPID which is process parent ID which is point
to 2 which is kthreadd
```

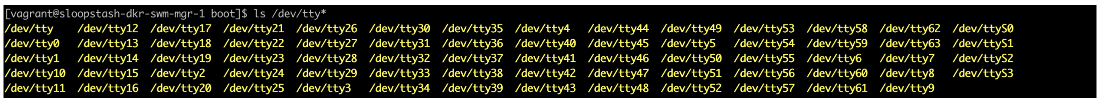
**User space:**

```
TTY - tele type terminal - are the input device interfaces with Linux
kernel
Linux supports 64 tty - ex: 64 keyboards can be connected



to overcome the 64 limit - we have concept called PTY => pseudo or
virtual terminal and no limitation for ex: power adpater socket to one
switch
```

```
ls /dev/tty* - list all 64 tty devices
```



```
[vagrant@sloopstash-dkr-swm-mgr-1 boot]$ ls /dev/tty*
/dev/tty    /dev/tty12  /dev/tty17  /dev/tty21  /dev/tty26  /dev/tty30  /dev/tty35  /dev/tty4   /dev/tty44  /dev/tty49  /dev/tty53  /dev/tty58  /dev/tty62  /dev/ttyS0
/dev/tty0   /dev/tty13  /dev/tty18  /dev/tty22  /dev/tty27  /dev/tty31  /dev/tty36  /dev/tty40  /dev/tty45  /dev/tty5   /dev/tty54  /dev/tty59  /dev/tty63  /dev/ttyS1
/dev/tty1   /dev/tty14  /dev/tty19  /dev/tty23  /dev/tty28  /dev/tty32  /dev/tty37  /dev/tty41  /dev/tty46  /dev/tty50  /dev/tty55  /dev/tty6   /dev/tty7   /dev/ttyS2
/dev/tty10  /dev/tty15  /dev/tty2   /dev/tty24  /dev/tty29  /dev/tty33  /dev/tty38  /dev/tty42  /dev/tty47  /dev/tty51  /dev/tty56  /dev/tty60  /dev/tty8   /dev/ttyS3
/dev/tty11  /dev/tty16  /dev/tty20  /dev/tty25  /dev/tty3   /dev/tty34  /dev/tty39  /dev/tty43  /dev/tty48  /dev/tty52  /dev/tty57  /dev/tty61  /dev/tty9
```

```
ls /dev/pts - List pyt master ex:
0 ptmx => here 0 means pty salve count (actually 1) and ptmx => master
pty
0 1 ptmx => I logged in via ssh in another terminal - now the pty
slaves count is total 2
0 1 2 ptmx => I logged in via ssh in another terminal - now the pty
slaves count is total 3



shell is needed to execute the command
ps -ef | grep bash

vagrant 5603 5602 0 02:18 pts/0 00:00:00 -bash
vagrant 5798 5603 0 03:28 pts/0 00:00:00 grep --color=auto bash

Now do ssh in another terminal and run above command:
vagrant 5603 5602 0 02:18 pts/0 00:00:00 -bash
vagrant 5803 5802 0 03:29 pts/1 00:00:00 -bash
vagrant 5826 5603 0 03:29 pts/0 00:00:00 grep --color=auto bash

pts above is salve pty and above shell is associated with it.
```

**Bash variables:**
https://github.com/sloopstash/kickstart-linux/wiki/Automate-and-deploy-Linux-VMs

```
Configure environment variables
Supported environment variables
# Allowed values for $OS_NAME variable.
* centos-linux-7
* ubuntu-linux-18-04
* ubuntu-linux-22-04
* amazon-linux-2
* alma-linux-8
* alma-linux-9
* rocky-linux-8
* rocky-linux-9

# Allowed values for $OS_ARCHITECTURE variable.
* amd64
* arm64

# Allowed values for $OS_EDITION variable.
* server
* desktop

# Allowed values for $HYPERVISOR variable.
* virtualbox
* vmware
Set environment variables
# Store environment variables.
$ export OS_NAME=ubuntu-linux-22-04
$ export OS_ARCHITECTURE=amd64
$ export OS_EDITION=server
$ export HYPERVISOR=virtualbox


to check if variale exists: echo $OS_NAME;

# Switch to Linux starter-kit directory.
$ cd /opt/kickstart-linux

# Boot Linux VM using Vagrant.
$ sudo
VAGRANT_CWD=./vagrant/$OS_NAME/$HYPERVISOR/$OS_ARCHITECTURE/$OS_EDITIO
N vagrant up

# SSH to Linux VM using Vagrant.
$ sudo
VAGRANT_CWD=./vagrant/$OS_NAME/$HYPERVISOR/$OS_ARCHITECTURE/$OS_EDITIO
N vagrant ssh

# Exit from Linux VM.
$ exit

# Halt Linux VM using Vagrant.
$ sudo
VAGRANT_CWD=./vagrant/$OS_NAME/$HYPERVISOR/$OS_ARCHITECTURE/$OS_EDITIO
```

```
N vagrant halt

# Provision Linux VM using Vagrant.
$ sudo
VAGRANT_CWD=./vagrant/$OS_NAME/$HYPERVISOR/$OS_ARCHITECTURE/$OS_EDITIO
N vagrant provision

# Destroy Linux VM using Vagrant.
$ sudo
VAGRANT_CWD=./vagrant/$OS_NAME/$HYPERVISOR/$OS_ARCHITECTURE/$OS_EDITIO
N vagrant destroy
```

Variables across child shell:

```
# Make an environment variable available to both parent and child
shells and its processes.
$ export DB_PASSWORD
# Make an environment variable unavailable from child shell and its
processes (run in parent shell).
$ export -n DB_PASSWORD
```

Which Shell on MAC:

```
echo $SHELL;
/bin/zsh

echo $0;
-zsh
```

show current shell

While both Zsh and Bash are Unix shells, Zsh is generally considered more customizable and user-friendly for interactive use with advanced features like auto-completion and plugin support, while Bash is preferred for its simplicity and reliability when writing scripts due to its strict POSIX compliance and cross-platform compatibility

Major shells -
Bourne
Bourne-Again (Bash)
C
Korn
Z

Do vagrant ssh and install any shells based on need

```
# Install C shell.
$ sudo apt install csh
```

```
# Install Korn shell.
$ sudo apt install ksh
# Install Z shell.
$ sudo apt install zsh
```

**User:**

```
sudo adduser tuto

sudo passwd tuto => test123

sudo userdel tuto

cat /etc/passwd - password directory will be created for tuto
cat /etc/group - group directory will be created for tuto

/home -> tuto directory will be created inside it

Switch to diff user
su - tuto
Password: test123

[tuto@sloopstash-dkr-swm-mgr-1 ~]$


If you're using Amazon Linux it's CentOS-based, which is RedHat-based.
RH-based installs use yum not apt-get. Something like yum search httpd
should show you the available Apache packages - you likely want yum
install httpd24.

Run below command as tuto user:
yum install nginx

Error: This command has to be run with superuser privileges (under the
root user on most systems).

sudo yum install nginx
[sudo] password for tuto:
tuto is not in the sudoers file. This incident will be reported.


now add tuto to sudoer admm file (restrictive sudoer)
sudo visudo -f /etc/sudoers.d/admin => add below config
```

```
tuto ALL=(ALL) NOPASSWD: /usr/bin/yum => tuto user can execute yum
command without password prompt.

now run -> sudo yum install nginx - now the package downloads


sudo group on linux is wheel on almalinux:
sudo usermod -aG wheel tuto => now tuto is added to wheel (sudoer)
group who can perfrom all sudo operatios

now as tuto user: (exit from it and login again as su - tuto)
sudo fdisk -l => should provide the proper output


Add custom group:
sudo groupadd devops =>
sudo usermod -a -G devops tuto => add tuto user to devops group and
cat /etc/group will show the new group created and tuto user is
associated this group.
```

**Linux File system:**

```
/ => is the File system root.

First level directories:
/boot, /etc, /opt, /bin, /sbin, /mnt, /home, /dev etc ..

In Linux, the /mnt directory is a special directory used for
temporarily mounting file systems that are not part of the main file
system hierarchy. It's intended for manual mounting of removable or
temporary storage devices, like USB drives or network shares.

File system tree is common across Linux distributions.

ssh into vagrant:

cd /
ls => lists all first level directories
[vagrant@sloopstash-dkr-swm-mgr-1 /]$ ls
afs bin boot dev etc home lib lib64 media mnt opt proc root run sbin
srv sys tmp usr var
```

```
ls /dev/ => pty, tty vcs etc .. tty are character devices, sda..
sda1.. sdb .. are block devices

The first charcter in permission crw--w----. is C - which represent
File type and here character device
crw--w----. 1 root tty 4, 0 Mar 13 2025 tty0
crw--w----. 1 root tty 4, 1 Mar 13 2025 tty1
crw--w----. 1 root tty 4, 10 Mar 13 2025 tty10
crw--w----. 1 root tty 4, 11 Mar 13 2025 tty11
crw--w----. 1 root tty 4, 12 Mar 13 2025 tty12
crw--w----. 1 root tty 4, 13 Mar 13 2025 tty13
crw--w----. 1 root tty 4, 14 Mar 13 2025 tty14
crw--w----. 1 root tty 4, 15 Mar 13 2025 tty15


Similarly drwxr-xr-x. - d represents directory

ls -la /var/run/ - we can see socket file like below (S - represents
socket file type)
srw-rw-rw-. 1 root root 0 Mar 13 2025 rpcbind.sock


regular files starts with hyphen(-) -rw-r--r--.
```

**File Permissions:**

```
Binary matrix based permissions

chmod 755 system-metrics.sh => 7 for user, 5 for group and 5 for
others (anonymous)

r w x
0 0 0 => 0
0 0 1 => 1
0 1 0 => 2
0 1 1 => 3
1 0 0 => 4
1 0 1 => 5
1 1 0 => 6
1 1 1 => 7

Isolated permissions:
```

chmod ugo+x system-metrics.sh => add execute permission for user, group and others

to remove => -x ex: chmpd g-x => remove execute permission for group

sudo chown -R vagrant:tuto script => apply tuto group to script folder and inside directory recursiverly

**Linux Process:**

All workloads or processes run as system services inside the machine (prod server)
how many CPU cores in computer determines computing capacity

A process creates a thread which can be executed
A thread is executed at the cpu core
A cpu core can execute one thread at a time.

nproc -> command to get numbe of cores
1 -> shows up in vagrant (bcz we configured 1 cpu core in vagrant configuration)
lscpu -> all information about cpu

sudo yum install nginx

sudo systemctl status nginx

sudo systemctl stop nginx => start nginx
sudo systemctl stop nginx => stop nginx

execute nginx as foreground mode:
sudo nginx -g 'daemon off;' => it blocks the pty terminal

Access => http://192.168.101.5/

in another terminal:
ps -ef | grep nginx => shows nginx is running

ctrl+z => SIGNSTOP => signal number 19 (POSIX Signal) -> pause a process fg 1 => it starts the process again (1 refers to process number)

ctrl+c => SIGNINT => signal number 2 (POSIX Signal) => means terminal interrrupt

Running nginx as background process:
sudo nginx -g 'daemon off;' & => & here shows background and it shows process ID

It does not block the pty terminal

SIGNTERM => signal number 15 (POSIX Signal) => Graceful termination of process => no data/state loss

sudo kill -15 {process_id} => output will be +Done (graceful shutdown)

sudo kill -9 {process_id} => output will be +killed (force exit) - we can see nginx is still running - means it has not terminated properly and essentially we have to kill each process manually which is overhead.

**Package:**

Traditional way of installing package: Example is Redis - which does not provide option to download as package instead it provides source code.
Build/Compile
- Download software source code
- Build and compile source code (convert source code to machine code)
- Intsall binary executable programs in bin paths

compliation tool is => gcc
build tool is => make => make install => takes the binary executable programs and put into bin paths

To install redis:
cd /tmp => best directory to compile the code

wget http://download.redis ......
tar xvzf redis-4.0.9.tar.gz -> uncompress the source code

```
cd redis-4.0.9
sudo make distclean => not needed when installing for the first time
sudo make

# Install Redis.
$ sudo make install => it copies all binaries from tmp folder where
source code was downloaded and compiled to bin paths

to find where it is installed:
which redis-server => would output /usr/local/bin/redis-server as it
is binary path


to start the redis:
$ redis-server

which nginx => /usr/sbin/nginx -> automatically compliled and saved to
sbin directory when insatlled the package using apt install => debian
package
(yum install => RPM package)
```

**Service:**

```
init system -> start, stop, restart, reload, status etc -> contols the
lifecycle of sysetm services
Daemon program -> ends with d => kthreadd, mysqld, systemd, sshd,
dockerd etc
system service -> Init system uses the daemon to run system services

SysV -> Init script + runlevel => is an init system works based on
runlevels
systemd -> Unit file + Target => is moderna day init system

first process to start is init system
2nd would be kthreadd

Daemon program is default to run in background

Init system reads the system configuration for process and run/start
the system service

To check daemon program - lets install Docker
https://sloopstash.com/knowledge-base/how-to-install-docker-on-
linux.html
```

- which dockerd
=> /usr/bin/dockerd

sudo systemctl status docker.service

we can see /lib/systemd/system/docker.service => Docker is system service where systemd is the init system which has docker.service config file loaded and uses it to run the daemon program of system service.

sysytemctl -> is the client pgm which helps to operate serivices -> start, stop, status, reload, restart etc.

cat /usr/lib/systemd/system/docker.service => system configuration file

To understand old implementation Sysv (whereas systemd is modern way)
$ runlevel
=> ex: N 3 => N3 mode - multi user with network mode - default for RPM linux

for ubuntu => N 5 => N5 mode - multi user with GUI with networking mode - default for Debian ubuntu linux

systemctl get-default => check target of system service
=> graphical.target => means multi user, gui with networking on ubuntu linux
=> multi-user.target => means multi user with networking on RPM linux

SysV method of handling service:
sudo /etc/init.d/docker start (this was not possible in alma linux possibly it was removed - this command worked only on ubuntu linux)

**Network:**

Router - will have some OS/Kernel
All devices run some light weight OS in network
every device/machine inside n/w will have IP address

IP is connected to wifi n/w interface

we will have 65535 IP ports

N/W socket is combination of IP address + port

Number of IPs per machine supported * 65535 ports => result number of socket combinations

who gives IP?:
DHCP server inside the Router device provide the IP addresses - the pattern of IP addresses can be configured in router device config
32 bit IPV4 netwrok namesapce is used to frame the IP address
It follows CIDR format
Ex: 11.1.0.0/16
N/W classes:
Class A => /8 => ~ IP addresses
Class B => /16 => 65534 IP addresses
Class C => /24 => 254 IP addresses

Docker uses => container network model to run containers
Kubernetes => container network interface to run containers

Amazon cloud uses => Amazon vcp
Azure uses => Azure virtual netwrok

How to find network drivers in linux kernel:
modinfo bridge => default nwking driver in kernel (Ethernet)
modinfo ipvlan => driver used by docker
modinfo macvlan => Driver for MAC address based VLANs
(/lib/modules/5.14.0-503.26.1.el9_5.aarch64/kernel/drivers/net/macvlan.ko.xz)

netstat -i => Provides Kernel interface table

```
Iface MTU RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0 1500 1056 0 0 0 937 0 0 0 BMRU
eth1 1500 0 0 0 0 28 0 0 0 BMRU
lo 65536 6 0 0 0 6 0 0 0 LRU
```

Rx =Reciveing packets

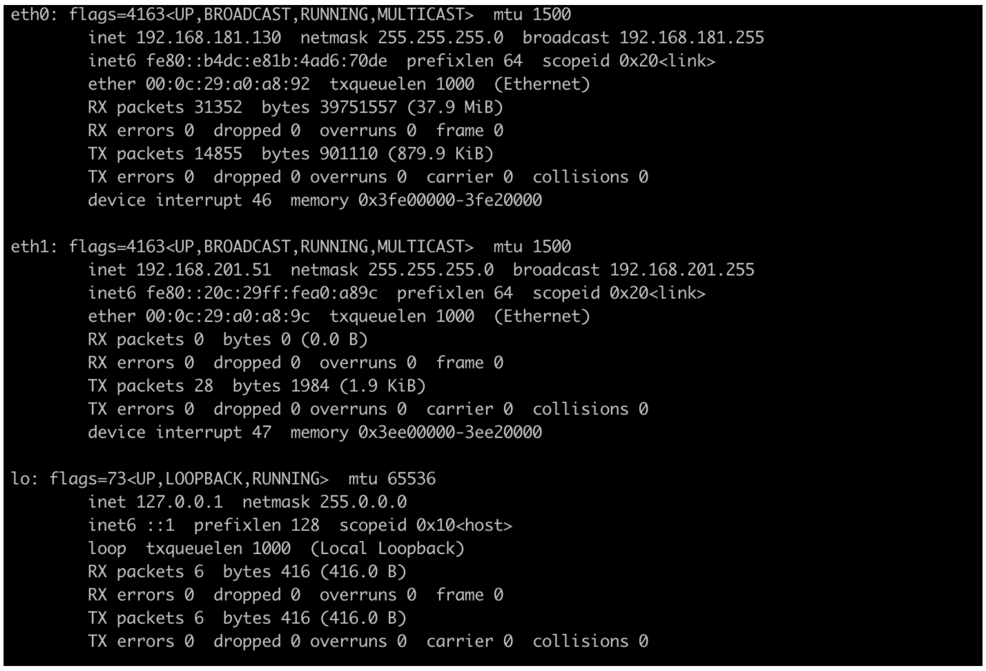
TX= Transmitting packets

sudo docker network ls

$ ifconfig => shows detailed nwk interface > lo - loop back, eth1, eth0 => ethernet each of these has IP addresses

inet ex: 192.168.201.51 => IPV4 address
inet6 ex: fe80::20c:29ff:fea0:a89c => IPV6 address
and MAC address => ex: ether 00:0c:29:a0:a8:9c cannot be changed.

Refer below screenshot



```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.181.130  netmask 255.255.255.0  broadcast 192.168.181.255
        inet6 fe80::b4dc:e81b:4ad6:70de  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:a0:a8:92  txqueuelen 1000  (Ethernet)
        RX packets 31352  bytes 39751557 (37.9 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 14855  bytes 901110 (879.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 46  memory 0x3fe00000-3fe20000

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.201.51  netmask 255.255.255.0  broadcast 192.168.201.255
        inet6 fe80::20c:29ff:fea0:a89c  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:a0:a8:9c  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 28  bytes 1984 (1.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 47  memory 0x3ee00000-3ee20000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 6  bytes 416 (416.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6  bytes 416 (416.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

sudo netstat -ntlp => shows listner sockets
when nginx server is accessed
TCP socket is created with > 0.0.0.0:80
0 0.0.0.0:22 => ssh socket

tcp 0 0 0.0.0.0:22 0.0.0.0:* LISTEN 883/sshd: /usr/sbin

**SSH:**

is a protocol widely used in cloud servers and containers
is client server model

ssh server => is a daemo prgm runs on server side
ssh client => pgm runs on client machine

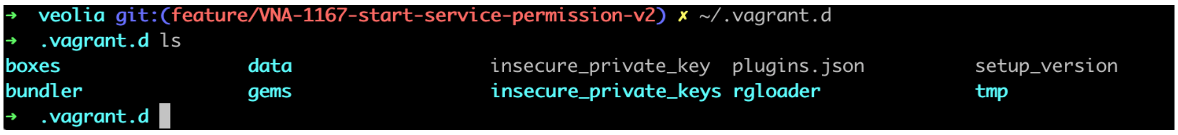
We should not use password based ssh on prod

we should use key based access
ssh-keygen is used to create ssh key pair (Private and Public key) using available algorithm

Local machine will have private key
Remote machine or server will have public key

setting up key based authentication with ssh:

Vagrant comes with default insecure private and public key.
vagrant file has configuration related to ssh credentials
config.ssh.username => vagrant
config.ssh.private_key_path => '~/.vagrant.d/insecure_private_key'
screenshot below



```
→  veolia git:(feature/VNA-1167-start-service-permission-v2) ✗ ~/.vagrant.d
→  .vagrant.d ls
boxes              data               insecure_private_key  plugins.json       setup_version
bundler            gems               insecure_private_keys rgloader           tmp
→  .vagrant.d
```

Inside vagrant => cat ~/.ssh/authorized_keys - vagrants insecure public key

ssh key pair is machine independant/ User independant => if the same key is pasted into another user ssh directory - the connection will work.

Generate new ssh key pair fo tuto user:
ssh-keygen -t rsa -f tuto
=> your identification is saved tuto
=> your public key is saved to tuto.pub

copy the public key and paste it to tuto home directory inside vagrant
$ su - tuto
$ mkdir .ssh
$ nano ~/.ssh/authorized_keys

```
vagrant ssh converts itself to => ssh -i vagrant vagrant@192.168.101.5
for tuto => ssh -i tuto tuto@192.168.101.5

where 192.168.101.5 is default IP address for vagrant configured in
config file.
```

inside vagrant => python3 -i
=> opens interactive python console. It does inline execution of python code
=> load the code - and then execute the code by calling function for example

Copy the files from computer to VM using scp command

- Run ifconfig inside vagrant to see IP address of VM (on my VM instance => 192.168.201.51) and use it in scp command

```
sudo scp -i ~/.vagrant.d/insecure_private_key www/my-devops-
project/script/manage-redis.py vagrant@192.168.201.51:~/

to exeucte the script make it executable (pyhton script for ex:)
chmod +x manage-redis.py => then ls -la will show the file in green
colour indicating executable.
use => ./manage-redis.py

now pyhton3 is being used on Linux:
which pyhton3 => /usr/bin/python3 => this should be used as
interpretor


Tip: to truncate the file content => > manage-redis.py



almlinux install redis using either package manager or make file
https://www.liquidweb.com/help-docs/install-redis-on-linux-almalinux/


Task for weekend:
```

# DevOps Notes: Containerisation Part 1:

Containerisation is an ecosystem or assemble and tools used around it
learn it in on premise and then in cloud infrastructure.

Containerise => Build OCI image (OCI compliant container image) => (tools: Docker)

- **Open Container Initiative (OCI):**
  A Linux Foundation project that aims to create open standards for container formats and runtimes.

Orchestrate => Orchestrate containerised work loads => Automate OCI containers in sequence way (tools: Kubernetes)

- Take OCI image and automate it as to run OCI containers.

Use bottom to top approach for orchestrating the stacks

Install CRM fullstack:
https://sloopstash.sharepoint.com/sites/training/SitePages/Library/Fullstack/CRM/01-DEV-environment.aspx

```
to start redis:
$ redis-server

cd /opt/app/source/

Run => sudo pip install -r requirements.txt

Start the CRM app => python3 init.py --port=2000

nano /opt/app/source/conf/redis.conf

=> update endpoint to localhost and redis port which 6379

nano /opt/app/source/conf/static.conf

=> remvoe the port 8001 at the last and save it.

Start the CRM app again => python3 init.py --port=2000
Try to access python flask app server with URL now
http://192.168.201.51:2000/ - it gives requested url not found
```

Instead access => http://192.168.201.51:2000/health
shows OK

http://192.168.201.51:2000/dashboard Loads some content with broken UI - it loads without any look and feel

Web servers handle static content (HTML, CSS, images) and focus on fast, efficient delivery using HTTP/HTTPS. Application servers manage dynamic content, execute business logic, and provide features like transaction management, security, and scalability.

So we need nginx webserver here.

```
sudo dnf install nginx
```

```
systemctl status nginx
```

configure nginx to connect with python flask: COnfgure virtual host configuration

```
sudo nano /etc/nginx/conf.d/app.conf
```

```
upstream app-servers {
server localhost:2000;
}
```

means connect to backend python server running on 2000 port

Nginx was not starting:

```
ls -ld /opt/nginx/log
ls -l /opt/nginx/log/error.log

# Set the right owner
sudo chown -R vagrant:vagrant /opt/nginx/log

# Or if you're using www-data
# sudo chown -R www-data:www-data /opt/nginx/log

sestatus
sudo semanage fcontext -a -t httpd_log_t "/opt/nginx/log(/.*)?"
sudo restorecon -Rv /opt/nginx/log
sudo systemctl restart nginx

$ hostname -I => shows your server IP
192.168.181.130 192.168.201.51


/etc/nginx/conf.d/app.conf => Add server mapping
upstream app-servers {
server localhost:2000;
}
server {
server_name app.crm.sloopstash.*;
root /opt/app/source;
-----
}



Login into redis cli
$ redis-cli

=> keys *
1) "a"
2) "ac"
shows data from app server

$ HGETALL a
1) "1"
2) "{\"name\": \"Rakshith\", \"email\": \"rakshthdavngre@gmail.com\",
\"phone\": \"9538432357\", \"website\": \"https://test.com\",
\"description\": \"Test description\"}"

HTTP status code
----------------
40X -> App logic errors (Headache of fullstack devs) - Layer 7 error
---
404 -> Url not found
403 -> Access denied
401 -> Not authorized
50X -> Backend errors (Headache of DevOps) - Layer 7 error
---
502 -> Bad gateway
500 -> Internal server error
```

Network errors (TCP/IP connection) -> Layer 4, 5 error => for example if NGINX is stopped

to access the log => tail -f /opt/nginx/log/access.log

```
$ sudo netstat -ntlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program
name
tcp 0 0 0.0.0.0:2000 0.0.0.0:* LISTEN 6764/python3
tcp 0 0 0.0.0.0:6379 0.0.0.0:* LISTEN 6303/redis-server *
tcp 0 0 0.0.0.0:22 0.0.0.0:* LISTEN 881/sshd: /usr/sbin
tcp 0 0 0.0.0.0:80 0.0.0.0:* LISTEN 6786/nginx: master
tcp 0 0 0.0.0.0:111 0.0.0.0:* LISTEN 1/systemd
tcp6 0 0 :::6379 :::* LISTEN 6303/redis-server *
tcp6 0 0 :::22 :::* LISTEN 881/sshd: /usr/sbin
tcp6 0 0 :::80 :::* LISTEN 6786/nginx: master
tcp6 0 0 :::111 :::* LISTEN 1/systemd
```

http://192.168.201.51:2000/dashboard

Access app url bypassing nginx =>
http://app.crm.sloopstash.dv:2000/dashboard using port will bypass nginx and access the app URL directly

**Docker:**

Install Docker in VM (RPM in my case):

```
# Install required system packages.
$ sudo yum install yum-utils device-mapper-persistent-data lvm2

# Add Docker repository to package manager source list.
$ sudo yum-config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo

# Install Docker, Docker client, and Containerd.
$ sudo yum install docker-ce docker-ce-cli containerd.io
```

=> ce -means community additon
=> containerd - container life cycle manager
=> docker-ce-cli and containerd.io are by default dependecies of docker-ce (explicitly metioned here just for understanding)

```
# Check Docker version.
$ docker --version

# Check status of Docker service.
$ sudo systemctl status docker.service

# Enable Docker service at boot.
$ sudo systemctl enable docker.service
```

To check the programs installed:

```
$ docker(=> press tab two time)
=> shows = docker, dockerd, dockerd-rootless-setuptool.sh, dockerd-
rootless.sh, docker-proxy

$ docker info
$ docker context ls
```

To expose docker to internet:

```
$ sudo nano /etc/docker/daemon.json
add:
{
"hosts": [
"tcp://0.0.0.0:2375", => exposes to outside via port 2375
"unix:///var/run/docker.sock"
],
"containerd": "/run/containerd/containerd.sock",
"storage-driver": "overlay2",
"log-driver": "json-file",
"log-opts": {
"max-size": "100m"
}
}
```

Also:

```
$ sudo mkdir /etc/systemd/system/docker.service.d
$ sudo nano /etc/systemd/system/docker.service.d/override.conf
Add below content:
[Service]
ExecStart=
ExecStart=/usr/bin/dockerd
```

After this reload:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart docker.service

now check: sudo netstat -ntlp => now we can see dockerd is starting
via tcp: 2375 port
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program
name
tcp 0 0 0.0.0.0:22 0.0.0.0:* LISTEN 893/sshd: /usr/sbin
tcp 0 0 0.0.0.0:80 0.0.0.0:* LISTEN 940/nginx: master p
tcp 0 0 0.0.0.0:111 0.0.0.0:* LISTEN 1/systemd
tcp6 0 0 :::2375 :::* LISTEN 9784/dockerd
tcp6 0 0 :::22 :::* LISTEN 893/sshd: /usr/sbin
tcp6 0 0 :::80 :::* LISTEN 940/nginx: master p
tcp6 0 0 :::111 :::* LISTEN 1/systemd
```

# DevOps Notes: Containerisation Docker Part 2:

Mounting or sync folder from local to virtual machine:
https://sloopstash.com/knowledge-base/configuring-host-vm-folder-path-sync-mount-in-virtualbox-and-vagrant.html

```
Inside your Vagrantfile, add:

Vagrant.configure("2") do |config|
config.vm.box = "generic/ubuntu2204"

config.vm.synced_folder "/opt/source/Devel/SLST/",
"/opt/source/Devel/SLST"
end



Replace the path /Users/yourname/Projects with your local folder.
```

**Containerisation:**

- **OCI Image - is also called docker image or container image =>** is container image to run the container (equivalent to Vagrant box for VM)
- Sequential collection of filesystem layers
- $ docker composer -> lets automate and orchestrate the container images.

**Storage Drivers (filesystem):**
- Docker uses storage drivers to manage how images and containers are stored on the host system.
- When working with OCI images, Docker uses drivers like overlay2, fuse-overlayfs, btrfs, and zfs. The default storage driver is often overlay2, but other options like fuse-overlayfs and btrfs are also available.

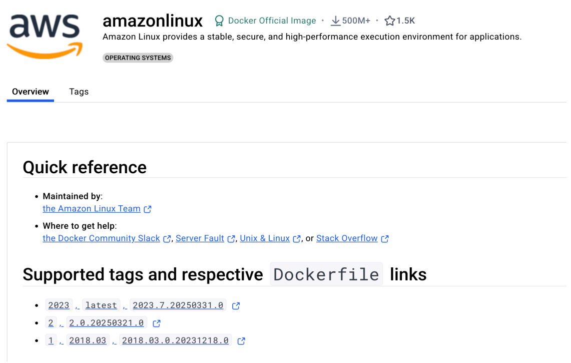
OCI image also consists of platform (process architecture): amd64, arm64 etc

Example: Redis OCI Image: it will have
- Linux OS
- System packages
- Redis programs

Similarly nginx will be another OCI Image.

Group of these OCI Images is OCI containers
Run OCI containers:
$ docker container run
$ runc

**Building OCI/Docker Images:**
Docker hub provides minimal OCI images which we can download from
Ex: AWS recommends Amazon linux docker image which is cloud native
Similarly we have different docker images for ubuntu, almalinux etc **(try alpine linux as task which is minimal weight - but we need to install extra packages)**
https://hub.docker.com/_/amazonlinux


amazonlinux Docker Official Image · 500M+ · 1.5K
Amazon Linux provides a stable, secure, and high-performance execution environment for applications.
OPERATING SYSTEMS

Overview Tags

Quick reference

- **Maintained by:**
  the Amazon Linux Team
- **Where to get help:**
  the Docker Community Slack, Server Fault, Unix & Linux, or Stack Overflow

Supported tags and respective `Dockerfile` links

- `2023`, `latest`, `2023.7.20250331.0`
- `2`, `2.0.20250321.0`
- `1`, `2018.03`, `2018.03.0.20231218.0`

## Build docker image manually:

```
# Pull amazonlinux:2 Docker image from registry
(https://hub.docker.com).
$ sudo docker image pull amazonlinux:2 --platform linux/arm64 =>
linux/amd64 is default

Check pulled images
$ sudo docker image ls

Check if container is running:
$ sudo docker container ls => will show if container is up and time

# Start interactive container using amazonlinux:2 Docker image. it
will start the container and open bash interactive
$ sudo docker container run -t -i --rm amazonlinux:2 /bin/bash

Inside the bash of container - we can check OS information:
bash-4.2# cat /etc/os-release
NAME="Amazon Linux"
VERSION="2"
ID="amzn"
ID_LIKE="centos rhel fedora"
VERSION_ID="2"
PRETTY_NAME="Amazon Linux 2"
ANSI_COLOR="0;33"
CPE_NAME="cpe:2.3:o:amazon:amazon_linux:2"
HOME_URL="https://amazonlinux.com/"
SUPPORT_END="2026-06-30"


# Install system packages.
$ yum update -y
$ yum install -y wget vim net-tools gcc make tar git unzip sysstat
tree initscripts bind-utils nc nmap logrotate crontabs
```

```
$ yum install -y python-devel python-pip python-setuptools
$ yum clean all
$ rm -rf /var/cache/yum
# Install Supervisor.
$ python -m pip install supervisor
$ mkdir /etc/supervisord.d
$ history -c
```

Now snapshot or commit the container to create the OCI image

```
# List Docker containers.
$ sudo docker container ls
# Commit the interactive container into a new Docker image.
$ sudo docker container commit abb63b3c14b5 sloopstash/base:v1.1.1 =>
where abb63b3c14b5 is containder ID from containder ls command to get
the containder ID
=> output will be filesystem layer is created =>
"sha256:0c9527d16a814cdd11490461f4c12a0e494e40f12623401b6ef455c3e26e23
53"

$ sudo docker image ls => check if image is commited
REPOSITORY TAG IMAGE ID CREATED SIZE
sloopstash/base v1.1.1 f078e65c2b83 15 seconds ago 672MB
amazonlinux 2 e57c74d00e62 3 weeks ago 196MB
```

To check file system layers from snapshot:

```
$ sudo docker image inspect sloopstash/base:v1.1.1
```

```
[
{
"Id":
"sha256:f078e65c2b83a416dcf07e3df9872f9ca1d9fede2a5e2a684e81a9394ca2fa
31",
"RepoTags": [
"sloopstash/base:v1.1.1"
],
"RepoDigests": [],
"Parent":
"sha256:e57c74d00e620510b94d88f8d4be7d8be515389b641e8644eeb51bc91874e9
a9",
"Comment": "",
"Created": "2025-04-23T03:11:10.304176987Z",
"DockerVersion": "28.0.4",
"Author": "",
"Config": {
"Hostname": "abb63b3c14b5",
"Domainname": "",
"User": "",
"AttachStdin": true,
"AttachStdout": true,
"AttachStderr": true,
"Tty": true,
"OpenStdin": true,
"StdinOnce": true,
"Env": [
"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
],
"Cmd": [
"/bin/bash"
],
"Image": "amazonlinux:2",
"Volumes": null,
"WorkingDir": "/",
"Entrypoint": null,
"OnBuild": null,
"Labels": {}
},
"Architecture": "arm64",
"Variant": "v8",
"Os": "linux",
"Size": 671988257,
"GraphDriver": {
"Data": {
"LowerDir":
"/var/lib/docker/overlay2/808bdd6860e6d0c2d4fe6c936a53b211c68540a838e4
faa36852b664f64b8b60/diff",
"MergedDir":
"/var/lib/docker/overlay2/f868e1f01217cb0758d8a4ca85293a72195d4f41cd71
76ce8145eb2533767a98/merged",
"UpperDir":
"/var/lib/docker/overlay2/f868e1f01217cb0758d8a4ca85293a72195d4f41cd71
76ce8145eb2533767a98/diff",
"WorkDir":
"/var/lib/docker/overlay2/f868e1f01217cb0758d8a4ca85293a72195d4f41cd71
76ce8145eb2533767a98/work"
},
"Name": "overlay2"
},
"RootFS": {
"Type": "layers",
"Layers": [
"sha256:40e36a1944f94a1d56ae295fdc6999f8955bfa1e42197723de196d7c32c846
4f",
"sha256:0c9527d16a814cdd11490461f4c12a0e494e40f12623401b6ef455c3e26e23
53"
]
},
"Metadata": {
"LastTagTime": "2025-04-23T03:11:10.313137824Z"
}
}
]

Above one layer belongs to base linux (amazon linux2) and the other
layer belongs to our sanpshot image - so file systems or layers are
shared to improve the boot which is core concept of container.

overlay2 - is the storgae driver for OCI image - this was configured
by default in docker confgiration
# Modify Docker daemon configuration.
$ sudo nano /etc/docker/daemon.json
{
"hosts": [
"tcp://0.0.0.0:2375",
"unix:///var/run/docker.sock"
],
"containerd": "/run/containerd/containerd.sock",
"storage-driver": "overlay2",
"log-driver": "json-file",
"log-opts": {
"max-size": "100m"
}
```

**Automated way of building docker image: using Dockerfile**

Two ways =>

- Linear Docker file
- Multi stage Docker file

Dockerfile is part of IAC (infrastructure as automation code) => instead of building inside shell as per manual way - we do it via dockerfile

Project structure:

- Inside docker project - create folders image => redis => 7.2.1
- 7.2.1 => context and docker files for linear and multi stage

```
To run the Docker file: inside VM go to /opt/source/Devel/SLST/docker-
project/

Command with explaination: Build the image/snapshot wuth command
sudo docker image build \
-t sloopstash/redis-linear:v7.2.1 \ # Name of the resultant OCI image
-f image/redis/7.2.1/amazon-linux-2-linear.dockerfile \ # Dockerfile
path.
image/redis/7.2.1/context # Build context diectory.

$ sudo docker image build -t sloopstash/redis-linear:v7.2.1 -f
image/redis/7.2.1/amazon-linux-2-linear.dockerfile
image/redis/7.2.1/context

=> [internal] load build definition from amazon-linux-2-
linear.dockerfile
=> => transferring dockerfile: 315B => being transferred to docker
deamon
=> [internal] load metadata for docker.io/sloopstash/base:v1.1.1
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/sloopstash/base:v1.1.1
=> CACHED [2/5] WORKDIR /tmp
=> [3/5] RUN wget http://download.redis.io/releases/redis-7.2.1.tar.gz
&& tar xvzf redis-7.2.1.tar.gz
=> [4/5] WORKDIR redis-7.2.1
=> [5/5] RUN make distclean && make && make install
=> exporting to image
=> => exporting layers
=> => writing image
sha256:d43da49c89cdafcea2596858bbf470abc83aab7949a598f05b63c4d40e26291
8
=> => naming to docker.io/sloopstash/redis-linear:v7.2.1

Docker deamon run the commands and script inside RUN statement from
dockerfile
```

```
Check if redis is installed inside image:
$ sudo docker container run -ti --rm sloopstash/redis-linear:v7.2.1
/bin/bash
Check redis commands => $ redis-server or $ redis-cli
bash-4.2# redis-
redis-benchmark redis-check-aof redis-check-rdb redis-cli redis-
sentinel redis-server (All binaries exists here)

Multistage: we can see image size is reduced
$ sudo docker image build -t sloopstash/redis-multi-stage:v7.2.1 -f
image/redis/7.2.1/amazon-linux-2-multi-stage.dockerfile
image/redis/7.2.1/context --no-cache

$ sudo docker image ls
REPOSITORY TAG IMAGE ID CREATED SIZE
sloopstash/redis-multi-stage v7.2.1 97d9860cf386 31 seconds ago 695MB
<none> <none> 4b45a77e02b7 8 minutes ago 953MB
sloopstash/redis-linear v7.2.1 3c2f0b01f4c8 33 minutes ago 929MB
<none> <none> d43da49c89cd 47 minutes ago 675MB
sloopstash/base v1.1.1 f078e65c2b83 24 hours ago 672MB
amazonlinux 2 e57c74d00e62 3 weeks ago 196MB

$ sudo docker container run -ti --rm sloopstash/redis-multi-
stage:v7.2.1 /bin/bash
bash-4.2# redis-
redis-cli redis-server => we can see only two redis binaries are
copied (redis-cli and redis-server)
```

Multistage is recommended for if need of downloading binary and make commands (compiling binaries) for ex: redis. In case of direct apt or yum downloads - we can got for Linear builds.

amazon-linux-2.dockerfile: https://github.com/sloopstash/kickstart-docker/blob/master/image/redis/7.2.1/amazon-linux-2.dockerfile

- In multi stage if stages starts with **FROM sloopstash/base:v1.1.1 AS** these will be executed in parallel
- If there is a dependency on previous stage - it will execute in sequence.

Either parallel or dependant stage - will create intermediate OCI images (not visible) - the final OCI image/stage will copy the data from these intermediate images. **Parallel stages will start first and then dependant stages.**
in linear build command && rm -rf redis-7.2.1* \ is needed bcz it might result in bigger image - in multistage this is not needed as we are just copying binaries from stage to stage into final OCI image.

```
# Build Docker image for Redis using Dockerfile.
$ sudo docker image build -t sloopstash/redis:v7.2.1 -f
image/redis/7.2.1/amazon-linux-2.dockerfile image/redis/7.2.1/context

[+] Building 103.1s (16/16) FINISHED docker:default
=> [internal] load build definition from amazon-linux-2.dockerfile
=> => transferring dockerfile: 1.96kB
=> WARN: ConsistentInstructionCasing: Command 'From' should match the
case of the command majority (uppercase) (line 12)
=> [internal] load metadata for docker.io/sloopstash/base:v1.1.1
=> [internal] load .dockerignore
=> => transferring context: 2B
=> CACHED [create_redis_directories 1/2] FROM
docker.io/sloopstash/base:v1.1.1
=> [install_system_packages 2/2] RUN yum install -y tcl
=> [create_redis_directories 2/2] RUN set -x && mkdir /opt/redis &&
mkdir /opt/redis/data && mkdir /opt/redis/log && mkdir /opt/redis/conf
&& mkdir /opt/redis/script && mkdir /opt/redis/sy 0.5s
=> [install_redis 1/4] WORKDIR /tmp
=> [install_redis 2/4] RUN set -x && wget
http://download.redis.io/releases/redis-7.2.1.tar.gz --quiet && tar
xvzf redis-7.2.1.tar.gz > /dev/null
=> [install_redis 3/4] WORKDIR redis-7.2.1
=> [install_redis 4/4] RUN set -x && make distclean && make && make
install
=> [finalize_redis_oci_image 2/6] COPY --from=install_redis
/usr/local/bin/redis-server /usr/local/bin/redis-server
=> [finalize_redis_oci_image 3/6] COPY --from=install_redis
/usr/local/bin/redis-cli /usr/local/bin/redis-cli
=> [finalize_redis_oci_image 4/6] COPY --from=create_redis_directories
/opt/redis /opt/redis
=> [finalize_redis_oci_image 5/6] RUN set -x && ln -s
/opt/redis/system/supervisor.ini /etc/supervisord.d/redis.ini &&
history -c
=> [finalize_redis_oci_image 6/6] WORKDIR /opt/redis
=> exporting to image
=> => exporting layers
=> => writing image
sha256:9f5e1cc3b4cdcf527afb7bfa77924aefdb79dd21a659147b727d246556a1e8d
1
=> => naming to docker.io/sloopstash/redis:v7.2.1
```

nginx Docker file: https://github.com/sloopstash/kickstart-docker/blob/master/image/nginx/1.24.0/amazon-linux-2.dockerfile

```
# Build Docker image for Nginx using Dockerfile.
$ sudo docker image build -t sloopstash/nginx:v1.24.0 -f
image/nginx/1.24.0/amazon-linux-2.dockerfile
image/nginx/1.24.0/context
```

Python Docker file: https://github.com/sloopstash/kickstart-docker/blob/master/image/python/3.12.0/amazon-linux-2.dockerfile

```
# Build Docker image for App using Dockerfile.
$ sudo docker image build -t sloopstash/python:v3.12.0 -f
image/python/3.12.0/amazon-linux-2.dockerfile
image/python/3.12.0/context
```

**Docker Hub:**
**Create Account on docker**
Docker Account: https://app.docker.com/
**rakshithtb1989**
rakshthdavngre@gmail.com

Go to Docker hub and create repository and in repository settings select Scout.

Rename/alias/tagging to other namespace for existing OCI images:

```
$ sudo docker image tag sloopstash/redis:v7.2.1
rakshithtb1989/redis:v7.2.1
```

To push this to docker login - authenticate with docker hub

```
$ sudo docker login
Waiting for authentication in the browser…

WARNING! Your credentials are stored unencrypted in
'/root/.docker/config.json'.
Configure a credential helper to remove this warning. See
```

```
https://docs.docker.com/go/credential-store/

Login Succeeded

Push this OCI image:
$ sudo docker image push rakshithtb1989/redis:v7.2.1
```

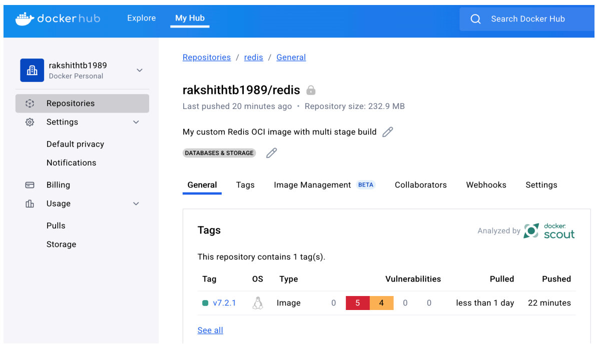
As we push the image Docker scout will start analysing and shows vulnariabilies

**Integrating with Docker Desktop:**
Open Docker desktop - login
Click on Images - click on Hub - we can pull the image pushed above - the image will be pushed.
When we click on Run for the image
Docker file should have Command or entrypoint for container to run when started - else it will show exited.

```
ex: CMD /user/local/bin/redis-server
EXPOSE 6379
or ENTRYPOINT /user/local/bin/redis-server
```

**Above is legacy code - however this is moved to container config file instead.**
Do not put app source code or sensitive information in OCI image.
Docker image can be imported or exported in archived format so using sensitive data is a security flaw.

Inside Docker desktop - run Docker scout analysis for the pulled image - that shows scan results within our image layer similar to what we did in Docker hub.



rakshithtb1989/redis
Last pushed 20 minutes ago · Repository size: 232.9 MB

My custom Redis OCI image with multi stage build

Tags

This repository contains 1 tag(s).

| Tag | OS | Type | Vulnerabilities | | | | | Pulled | Pushed |
|---|---|---|---|---|---|---|---|---|---|
| v7.2.1 | | Image | 0 | 5 | 4 | 0 | 0 | less than 1 day | 22 minutes |

Task PostgresSql - https://github.com/sloopstash/kickstart-docker/blob/master/image/postgresql/16.4/amazon-linux-2.dockerfile
Try to refer Sloopstash git repo and start building the images and push to our repo to improve the profile.

**Container Network Model (CNM):**

- CNM integrates core linux network drivers to enable nwk capabilities for the OCI containers.
- create and manage nwk interface for OCI containers
- Assign IP address to OCI containers - enables nwk traffic routing

```
# View Linux kernel networking module info. - check these commands
inside Linux VM
$ modinfo bridge => shows bridge driver module on linux and maps to
CNM bridge driver.
filename: /lib/modules/5.14.0-
503.26.1.el9_5.aarch64/kernel/net/bridge/bridge.ko.xz
description: Ethernet bridge driver
alias: rtnl-link-bridge
version: 2.3
license: GPL
rhelversion: 9.5
srcversion: F066C06A3F63323D13AE1D9

Similarly below driver modules on linux maps with respctive driver on
CNM
$ modinfo ipvlan
$ modinfo macvlan
```

Managing nwks in Docker:
https://sloopstash.sharepoint.com/sites/training/SitePages/Library/DevOps/Docker/10-Docker-network.aspx

```
In another terminal: run
$ sudo docker network ls => lists the default nwk interfaces of docker
engine
NETWORK ID NAME DRIVER SCOPE
8c9f1ef3f062 bridge bridge local
0cf97c306153 host host local
1aacc04a389e none null local

$ sudo docker network inspect bridge => where bridge is name of docker
nwk
....
"Network": ""
},
"ConfigOnly": false,
"Containers": {},
"Options": {
"com.docker.network.bridge.default_bridge": "true",
"com.docker.network.bridge.enable_icc": "true",
"com.docker.network.bridge.enable_ip_masquerade": "true",
"com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
"com.docker.network.bridge.name": "docker0", => this is what being
displayed in igconfig output as docker0 => bridge nwk for Docker
"com.docker.network.driver.mtu": "1500"
},
"Labels": {}
```

Similarly => $ sudo docker network inspect host => we do not see any name as above - it won't create newk interface for host and it consumes the host nwk adpater

Likewise => $ sudo docker network inspect none => none nwk means totally disabling IP address and nwetworking for docker container

```
....
"EnableIPv6": false, => means no nwking enabled
.....
```

```
# Start container within the host Docker network.
$ sudo docker container run -t -i --rm --network=host
sloopstash/nginx:v1.24.0 /bin/bash
```

=> --network=host => docker nwk host is chosen here which is sutitable for webservers/load balancers

Inside bash run => ifconfig => we can see there is no isolation with host (VM) and container - we see all ips => docker0 is default

```
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
ether 92:cc:f0:6a:96:64 txqueuelen 0 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.181.130 netmask 255.255.255.0 broadcast 192.168.181.255
inet6 fe80::b4dc:e81b:4ad6:70de prefixlen 64 scopeid 0x20<link>
```

```
ether 00:0c:29:a0:a8:92 txqueuelen 1000 (Ethernet)
RX packets 458983 bytes 74331933 (70.8 MiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 354298 bytes 494768402 (471.8 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
device interrupt 46 memory 0x3fe00000-3fe20000

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.201.51 netmask 255.255.255.0 broadcast 192.168.201.255
inet6 fe80::20c:29ff:fea0:a89c prefixlen 64 scopeid 0x20<link>
ether 00:0c:29:a0:a8:9c txqueuelen 1000 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 31 bytes 2194 (2.1 KiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
device interrupt 47 memory 0x3ee00000-3ee20000

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 6 bytes 416 (416.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 6 bytes 416 (416.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

# Start Nginx.
$ nginx => now can be accessed using IP of VM eth1 => 192.168.201.51
from the host machine (our computer browser) => means inbound traffic
inside container.
- IP address from docker cannot be accessed on host computer browser
instead we need to use IP address of VM eth1

bash-4.2# ping google.com => means from the container able to connect
to internet - outbound traffic
PING google.com (142.250.192.14) 56(84) bytes of data.
64 bytes from bom12s14-in-f14.1e100.net (142.250.192.14): icmp_seq=1
ttl=128 time=84.0 ms
64 bytes from bom12s14-in-f14.1e100.net (142.250.192.14): icmp_seq=2
ttl=128 time=96.6 ms
```

Using none network:

```
# Start container within the none Docker network. Not used most of the
time. mostly for running scripts etc.
$ sudo docker container run -t -i --rm --network=none
sloopstash/nginx:v1.24.0 /bin/bash

bash-4.2# ifconfig => only one nwk interfaces which is loopback -
which cannot be reached from the internet
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

# Start Nginx.
$ nginx => but cannot be accessed using the VM IP address as well

bash-4.2# ping google.com
ping: google.com: Name or service not known
```

Using Bridge network:

```
Allows to run Production and Dev envs in standalone docker engine
works well on cloud as well as on premise infrastructure.

Docker does not allow create another nwk interface for Host and None
nwk interfaces. But we can create many Bridge nwks as requuired for
isolation purpose.

$ sudo docker network create -d bridge --subnet=14.1.0.0/16 --ip-
range=14.1.1.0/24 --gateway=14.1.1.1 sloopstash-dev-crm_common

=> --gateway=14.1.1.1 above indicates how we can access the container
traffic

[vagrant@sloopstash-dkr-swm-mgr-1 ~]$ sudo docker network ls => we can
see our custom docker nwk of CNM type bridge is created. => subnet
14.1.0.0/16 above represents - IPV4 namespace to specify the IP
address range for containers and combinations with class A/B/C. refer
to linux session for IPV4 nwk namespace for class and range formula.
NETWORK ID NAME DRIVER SCOPE
```

```
8c9f1ef3f062 bridge bridge local
0cf97c306153 host host local
1aacc04a389e none null local
dd3366b9bd48 sloopstash-dev-crm_common bridge local

$ sudo docker network inspect sloopstash-dev-crm_common
{
"Name": "sloopstash-dev-crm_common",
"Id":
"dd3366b9bd48675e536c5b62747df5946d7c192efc4eec0162f35811454ba286",
"Created": "2025-05-01T02:56:48.384182199Z",
"Scope": "local", => standalone Docker engine - it will be overlay for
Docker swarm
"Driver": "bridge",
"EnableIPv4": true,
"EnableIPv6": false,
"IPAM": {
"Driver": "default",
"Options": {},
"Config": [
{
"Subnet": "14.1.0.0/16",
"IPRange": "14.1.1.0/24",
"Gateway": "14.1.1.1"
}
]
},
"Internal": false,
"Attachable": false,
"Ingress": false,
"ConfigFrom": {
"Network": ""
},
"ConfigOnly": false,
"Containers": {},
"Options": {},
"Labels": {}
}

In another terminal: run $ ifconfig
....
br-dd3366b9bd48: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500 => this
is the custom nwk created by us above - we can see IP address 14.1.1.1
inet 14.1.1.1 netmask 255.255.0.0 broadcast 14.1.255.255
```

```
ether f6:d9:91:6c:7f:f9 txqueuelen 0 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
ether 62:d2:3e:5d:6c:35 txqueuelen 0 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
......
```

Now start the Docker container with custom Docker nwk:

```
$ sudo docker container run -t -i --rm --network=sloopstash-dev-
crm_common sloopstash/nginx:v1.24.0 /bin/bash

# List network interfaces within the Docker container.
bash-4.2# ifconfig => we can see the IP address assigned is 14.1.1.0
and nwk is isolated with VM.
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 14.1.1.0 netmask 255.255.0.0 broadcast 14.1.255.255
ether 8a:02:b8:fc:ab:a8 txqueuelen 0 (Ethernet)
RX packets 12 bytes 1112 (1.0 KiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 3 bytes 126 (126.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0


# Start Nginx.
$ nginx

bash-4.2# ps -ef => we cam nginx is running
UID PID PPID C STIME TTY TIME CMD
root 1 0 0 03:08 pts/0 00:00:00 /bin/bash
root 8 1 0 03:09 ? 00:00:00 nginx: master process nginx
nginx 9 8 0 03:09 ? 00:00:00 nginx: worker process
root 12 1 0 03:19 pts/0 00:00:00 ps -ef
```

Now we can access the niginx inside VM (we still cannot access the container IP from HOST browser (our computer)) This is how we need to isolate dev, prod and non prod environments for more secured implementation.
in another ssh vm session:

```
[vagrant@sloopstash-dkr-swm-mgr-1 ~]$ curl -XGET http://14.1.1.0
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully
installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

To enable access to container from external (host - our computer) - we need to use port mapping

```
$ sudo docker container run -t -i --rm --network=sloopstash-dev-crm_common -p 80:80 sloopstash/nginx:v1.24.0 /bin/bash => did not work
$ sudo docker container run -t -i --rm --network=sloopstash-dev-crm_common -p 8080:80 sloopstash/nginx:v1.24.0 /bin/bash => try this but not sure
```

After this we can access the nginx using VM IP on host machine (browser our computer)

Similarly we have vlan and Macvlan are docker networks - we can try if needed - https://sloopstash.sharepoint.com/sites/training/SitePages/Library/DevOps/Docker/10-Docker-network.aspx

**Docker Storage:**

Docker storage uses the Linux kernel file system modules like the networking model uses Linux kernel drivers.

```
$ modinfo overlay
filename: /lib/modules/5.14.0-
503.26.1.el9_5.aarch64/kernel/fs/overlayfs/overlay.ko.xz
alias: fs-overlay
license: GPL
description: Overlay filesystem
author: Miklos Szeredi <miklos@szeredi.hu>
rhelversion: 9.5
srcversion: 665446FDC325EDBF7294FBE
depends:
intree: Y
name: overlay
........
```

How persistent is data in OCI container ?

As compared to OCI image the data in OCI container will be gone once container is shutdown - so the concept of Docker storage.

**Types of Docker storage:**

**Ephemeral =>**

- When container is created => thin read/write layer which forms the root filesystem of the OCI container which gets deleted upon shutdown
- **Suitable for tmp data - no persistence.**

```
$ sudo docker container run -t -i --rm amazonlinux:2 /bin/bash

In another ssh terminal
$ sudo docker container ls
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
da0784bce9b6 amazonlinux:2 "/bin/bash" 25 seconds ago Up 24 seconds
flamboyant_vaughan


$ sudo docker container inspect da0784bce9b6
Check the Graph driver section => this is called thin read/write layer
forms root filesystem of the container
```

```
......
"GraphDriver": {
"Data": {
"ID":
"da0784bce9b6cd07e78b867e635155face29c5d9127b520084bb14d5ee5784a0",
"LowerDir":
"/var/lib/docker/overlay2/3db03565f08c16ad3f99979772a2da6bda8737bee075
800ec052e9d066db49f2-
init/diff:/var/lib/docker/overlay2/808bdd6860e6d0c2d4fe6c936a53b211c68
540a838e4faa36852b664f64b8b60/diff",
"MergedDir":
"/var/lib/docker/overlay2/3db03565f08c16ad3f99979772a2da6bda8737bee075
800ec052e9d066db49f2/merged",
"UpperDir":
"/var/lib/docker/overlay2/3db03565f08c16ad3f99979772a2da6bda8737bee075
800ec052e9d066db49f2/diff",
"WorkDir":
"/var/lib/docker/overlay2/3db03565f08c16ad3f99979772a2da6bda8737bee075
800ec052e9d066db49f2/work"
},
"Name": "overlay2"
},
............

$ sudo df -Th
Filesystem Type Size Used Avail Use% Mounted on
....
overlay overlay 19G 5.5G 13G 30%
/var/lib/docker/overlay2/3db03565f08c16ad3f99979772a2da6bda8737bee0758
00ec052e9d066db49f2/merged => this is ephimeral volume which is not
persistent
...

If we exit container - the above mount will be gone !!!!!

Files can be synced while container is running - but once the
container is shutdown - the mount will be deleted !!
```

**tmpFs** =>

- Data written in the system memory RAM - low latency read/writes
- Best suited for storing sensitive data such as passwords, API keys, and key files.
- Good for serving cached static content like HTML, JS, CSS, images, videos, etc., via HTTP servers like Nginx, Apache HTTPD.

```
$ sudo docker container run -t -i --rm --mount
'type=tmpfs,dst=/opt/app/cache,tmpfs-size=100m'
sloopstash/nginx:v1.24.0 /bin/bash

=> we can see there is no src attribute in command as thr will be no
sync - memmpry size cam also be given in the command => depends on
system memory not the host machine memory.

bash-4.2# cd /opt/app/cache/
bash-4.2# df -Th
Filesystem Type Size Used Avail Use% Mounted on
overlay overlay 19G 5.5G 13G 30% /
tmpfs tmpfs 64M 0 64M 0% /dev
shm tmpfs 64M 0 64M 0% /dev/shm
/dev/nvme0n1p3 xfs 19G 5.5G 13G 30% /etc/hosts
tmpfs tmpfs 100M 0 100M 0% /opt/app/cache
tmpfs tmpfs 820M 0 820M 0% /proc/acpi
tmpfs tmpfs 820M 0 820M 0% /proc/scsi
tmpfs tmpfs 820M 0 820M 0% /sys/firmware
```

**Bind mount** =>

- Enables bidirectional filesystem sync btw host and OCI container - the data persists here.
- **We need manage** the bind mount and  storage and the data written in it
- recommended for syncing source code and configuration files to the OCI container from host machine.

```
Clone the CRM app inside VM:
$ sudo git clone https://github.com/sloopstash/sloopstash-crm-app.git
/opt/sloopstash-crm-app
$ sudo chown -R $USER:$USER /opt/sloopstash-crm-app

Use the above path for bind mount storage type: Bcz docker does not
manages it.
$ sudo docker container run -t -i --rm --mount
'type=bind,src=/opt/sloopstash-crm-app,dst=/opt/app/source'
sloopstash/python:v3.12.0 /bin/bash

We can see the code inside /opt/app/source:
bash-4.2# cd /opt/app/source
bash-4.2# ls
LICENSE README.md asset conf config.py controller helper init.py
library model requirements.txt script theme view
```

```
bash-4.2# touch readme.txt

Now check in VM host => /opt/sloopstash-crm-app - readme.txt will
exist => this is also birectional sync.
```

**Volume** =>

- Enables bidirectional filesystem sync btw host and OCI container - the data persists here.
- **Docker manages** the bind mount and  storage and the data written in it
- Suitable for Data related workloads like MangoDB, Redis,, Hadoop. enables third party storage drivers and storage.

```
$ sudo docker volume ls

# Create Docker volume.
$ sudo docker volume create sloopstash-dev-crm_redis-data
$ sudo docker volume create sloopstash-dev-crm_redis-log
$ sudo docker volume create sloopstash-dev-crm_app-log
$ sudo docker volume create sloopstash-dev-crm_nginx-log

$ sudo docker volume ls
DRIVER VOLUME NAME
local sloopstash-dev-crm_app-log
local sloopstash-dev-crm_nginx-log
local sloopstash-dev-crm_redis-data
local sloopstash-dev-crm_redis-log

$ sudo docker volume inspect sloopstash-dev-crm_redis-data
[
{
"CreatedAt": "2025-05-06T02:22:48Z",
"Driver": "local",
"Labels": null,
"Mountpoint": "/var/lib/docker/volumes/sloopstash-dev-crm_redis-
data/_data", => Docker manages this volume
"Name": "sloopstash-dev-crm_redis-data",
"Options": null,
"Scope": "local"
}
]

Just to check
$ sudo su - root
$ cd /var/lib/docker/volumes/sloopstash-dev-crm_redis-data/_data => it
will be empty intially

# Start container with Docker volume.
$ sudo docker container run -t -i --rm --mount
'type=volume,src=sloopstash-dev-crm_redis-data,dst=/opt/redis/data'
sloopstash/redis:v7.2.1 /bin/bash

=> where src=sloopstash-dev-crm_redis-data is the mount path for
volume storage => name of the volume created
=> dst=/opt/redis/data => container path

Go to /opt/redis/data
bash-4.2# cd /opt/redis/data
bash-4.2# touch redis.rdb
bash-4.2# ls
redis.rdb

The above file will persist even if we exit the container and run it
again

Now check if the same files exists in Mount path same file exists here
as well:
[vagrant@sloopstash-dkr-swm-mgr-1 ~]$ sudo su - root
Last login: Tue May 6 02:26:36 UTC 2025 on pts/0
[root@sloopstash-dkr-swm-mgr-1 ~]# cd
/var/lib/docker/volumes/sloopstash-dev-crm_redis-data/_data
[root@sloopstash-dkr-swm-mgr-1 _data]# ls
redis.rdb
[root@sloopstash-dkr-swm-mgr-1 _data]#

Whatever file created here - will be synced to container path as well
/opt/redis/data = so it is bidirectional sync.
```

# DevOps Notes: Containerisation Docker Part 3:

## Differences between VM and Container:

For example:
Software stack => 20 micro services
=> 1 environment => 20 micro services
=> 1 service = 1 VM = 1 Container

Non production env will require more VMs or containers based on teams and team size say 2500 containers or VMs
This volume of VMs is very difficult and so we go to Container
Containers consume less hardware and cost effective
VM is needed to run the container as it needs Linux kernel. if we have Linux host machine - we can run container directly on it.
In essence: While containers are generally run without VMs, they can be used alongside VMs for various reasons like isolation, hardware requirements, and leveraging existing infrastructure
https://m.youtube.com/watch?v=Sb1LEaoW1vY
1. Kernel bootstrapping:
Start the VM and ssh and we know how it bootstraps with kthreadd and it is heavy weight

```
ps -ef
```

Now run the container in another terminal:

```
$ sudo docker container run -t -i --rm sloopstash/base:v1.1.1
/bin/bash
bash-4.2# ps -ef
UID PID PPID C STIME TTY TIME CMD
root 1 0 0 02:00 pts/0 00:00:00 /bin/bash
root 6 1 0 02:01 pts/0 00:00:00 ps -ef

We can see above big list of process listed as compared to VM -
meaning container uses the host machine (VMs) bootstrapping. So it is
light weight.
```

2. Sharing Kernel:
Inside VM - go to boot directory:

```
[vagrant@sloopstash-dkr-swm-mgr-1 ~]$ cd /boot/
[vagrant@sloopstash-dkr-swm-mgr-1 boot]$ ls
It will have all components grub loader etc.
```

In container - got to boot directory - it will be empty

```
bash-4.2# cd /boot/
bash-4.2# ls
Empty

Below command shows how container inherits kernel from host machine:
bash-4.2# uname -a
Linux 92a20595179f 5.14.0-503.26.1.el9_5.aarch64 #1 SMP
PREEMPT_DYNAMIC Mon Mar 3 10:49:04 UTC 2025 aarch64 aarch64 aarch64
GNU/Linux
```

3. Memory locking:

We need to allocate memory for Vagrant in vagrant config file ex: vf.memory = {2048} this is concept of **malock** - memory locking (check more on internet) - even if VM uses all of it memory allocated or not but still we need to give it - it blocks that unused memory on Host (our laptop - mac)

Whereas container does not use memory locking - but use memory limiting - we can use this option while running the container:

```
$ sudo docker container run --help | grep memory
--kernel-memory bytes Kernel memory limit
-m, --memory bytes Memory limit
--memory-reservation bytes Memory soft limit
--memory-swap bytes Swap limit equal to memory plus swap: '-1' to
enable unlimited swap
--memory-swappiness int Tune container memory swappiness (0 to 100)
(default -1)
```

4. Resource isolation:

Container does not isolate processor, memory and storage. VMs provide realistic isolation.

```
bash-4.2# df -Th
Filesystem Type Size Used Avail Use% Mounted on
overlay overlay 19G 5.5G 13G 30% /
tmpfs tmpfs 64M 0 64M 0% /dev
shm tmpfs 64M 0 64M 0% /dev/shm
/dev/nvme0n1p3 xfs 19G 5.5G 13G 30% /etc/hosts
tmpfs tmpfs 820M 0 820M 0% /proc/acpi
tmpfs tmpfs 820M 0 820M 0% /proc/scsi
tmpfs tmpfs 820M 0 820M 0% /sys/firmware

It will be same output on VM as well for the root mount / which is 19G
```

```
bash-4.2# free -m
total used free shared buff/cache available
Mem: 1638 253 808 9 575 1294
Swap: 0 0 0

Same output in VM as well - so no isolation

Similarly for processor.
```

Container implements abstracted isolation. no hardware level isolation and isolation in terms of multiple environments which are light weight.

5. Cascading Failure: (Production grade issue)
Out of 10 VMs if any one VM is error - other VMs will not be affected - bcz of VMs feature of isolation. but in container infrastructure might get affected as it does not have realistic isolation.
6. Boot Performance:
Container boots in 0-1 seconds - VM takes more time as it is heavy weight.

## How Docker container is created and modes:

- **runc** creates the OCI container's root filesystem (overlay fs layer)
- Execute Docker cmd or entry point to create running OCI container.

**modes**:

- Service or workload containers (97% of industry use case)
- Interactive container => debugging troubleshooting and test OCI containers used for learning purpose
  - using container run -t -i
- Intermediate container => used during OCI build image process => created by RUN statement in DockerFile which is handled by docker daemon process.

## Container Supervisor:

- Helps to manage micro services inside container
- Acts like an init system for the container DOCKER CMD or entry point
- Python community is developing or contributing to this - this is the reason we installed supervisor using pip when building base OCI image.
- It is good to avoid **CMD** and **ENTRYPOINT** statements in your Dockerfile, as they can be set during the creation of a Docker container based on demand.

```
# Run default CMD in Docker container.
$ sudo docker container run -i -t --rm sloopstash/base:v1.1.1
```

```
=> we get access to Bash shell
bash-4.2#

=> we did not give any Docker CMD to go to bin/bash - but the amazon
linux2 image out of which our base image was formed has default CMD
/bin/bash

# Run Bash shell as CMD in Docker container.
$ sudo docker container run -i -t --rm sloopstash/base:v1.1.1
/bin/bash
=> here /bin/bash is Docker CMD which opens bash shell and takes the
precedence over amazon linux2 Docker CMD.

for other ex:
$ sudo docker container run -i -t --rm sloopstash/base:v1.1.1 /bin/sh

# Run Nginx in foreground mode as CMD in Docker container.
$ sudo docker container run -i -t --rm sloopstash/nginx:v1.24.0
/usr/sbin/nginx -g 'daemon off;'
=> nginx runs and blocks the terminal
# Run Nginx in foreground mode as CMD and ENTRYPOINT in Docker
container.
$ sudo docker container run -i -t --rm --entrypoint /usr/sbin/nginx
sloopstash/nginx:v1.24.0 -g 'daemon off;'
=> nginx runs and blocks the terminal where entrypoint is new release
std as compared to just Docker CMD
=> --entrypoint /usr/sbin/nginx and docker CMD => daemon off
```

## Build Docker containers for each stack of CRM App:

Create Directory structure under docker project referring to https://github.com/sloopstash/kickstart-docker

- Common configuration for stacks goes inside workload directory
- App specific configuration will go to **_stack** directory

custom nwk and storage we have already created, which we will use to start the container in the command.

```
Design fully configured Redis workload OCI container
........................................................

sudo docker container run -d -i -t --rm \
--name sloopstash-dev-crm-redis \
-h redis \
--network sloopstash-dev-crm_common \
--ip 14.1.1.10 \
--mount 'type=volume,src=sloopstash-dev-crm_redis-
data,dst=/opt/redis/data' \
--mount 'type=volume,src=sloopstash-dev-crm_redis-
log,dst=/opt/redis/log' \
-v /opt/source/Devel/SLST/docker-
project/workload/supervisor/conf/server.conf:/etc/supervisord.conf \
-v /opt/source/Devel/SLST/docker-
project/workload/redis/7.2.1/conf/supervisor.ini:/opt/redis/system/sup
ervisor.ini \
-v /opt/source/Devel/SLST/docker-
project/workload/redis/7.2.1/conf/server.conf:/opt/redis/conf/server.c
onf \
--entrypoint /usr/bin/supervisord \
sloopstash/redis:v7.2.1 \
-c /etc/supervisord.conf

=> output =>
14fff5c5a14b9c2efe9fd7be2ead14cef25a9880f8272afc27fbd80dfbae133e
```

-d => means run conatiner in detached mode and run in background and terminal is released.

```
[vagrant@sloopstash-dkr-swm-mgr-1 ~]$ sudo docker container ls
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
14fff5c5a14b sloopstash/redis:v7.2.1 "/usr/bin/supervisor…" 11 seconds
ago Up 10 seconds sloopstash-dev-crm-redis
```

To access container bash:

```
# Access Bash shell of existing Docker container.
$ sudo docker container exec -ti sloopstash-dev-crm-redis /bin/bash
bash-4.2# ps -ef
UID PID PPID C STIME TTY TIME CMD
root 1 0 0 03:16 pts/0 00:00:00 /usr/bin/python /usr/bin/supervisord -
c /etc/supervisord.conf
root 8 1 0 03:16 pts/0 00:00:02 redis-server 0.0.0.0:3000
root 26 0 0 03:27 pts/1 00:00:00 /bin/bash
root 32 26 0 03:27 pts/1 00:00:00 ps -ef
```

We can see above first process is supervisor which is init system and it takes care of starting redis server.

```
How supervisor is starting redis server ? this is configured in
workload supervisor.ini for redis.
bash-4.2# cat /etc/supervisord.d/redis.ini
[program:redis]
command=bash -c "redis-server /opt/redis/conf/server.conf"
process_name=%(program_name)s
pidfile=/opt/redis/system/server.pid

We can use supervisorctl to control the process or workload.
ex: supervisorctl stop redis or supervisorctl start redis.

# Run command in existing Docker container.
$ sudo docker container exec -ti sloopstash-dev-crm-redis
supervisorctl status
redis RUNNING pid 8, uptime 0:09:44
$ sudo docker container exec -ti sloopstash-dev-crm-redis redis-cli

# Stop Docker container.
$ sudo docker container stop sloopstash-dev-crm-redis

# Start Docker container.
$ sudo docker container start sloopstash-dev-crm-redis
```

# DevOps Notes: Containerisation Part 4:

## Docker Compose:

- YAML is used for Docker compose
- YAML is not a programming language and it is a key value data structure like JSON
- YAML scoping is based on indentation.

Docker compose YAML configuration is the infrastructure as code (Iac) implementation in docker.

- Built-in keys
- User defined keys

Docker resources in standalone docker engine (good for development and testing purposes) is combination of => Docker volumes, storage and containers.

Multiple environments of Docker are isolated using namespaces.

So, **Docker compose is a tool** which allows to use docker resources in YAML to create multiple environments isolated using namespace and enable container orchestration.

Last week we build OCI container manually - now we are going to automate it with Docker compose YAML.
Refer compose directory https://github.com/sloopstash/kickstart-docker/tree/master/compose
Create compose directory => crm.yml, DEV.ENV, QAA.ENV, QAB.ENV

```
DEV.ENV => define env variables

# common configuration
ENVIRONMENT=DEV
EXTERNAL_DOMAIN=sloopstash.dv
INTERNAL_DOMAIN=sloopstash-dev.internal
HOME_DIR=/opt/source/Devel/SLST/docker-project

# Stack specific configuration
CRM_REDIS_VERSION=7.2.1
CRM_NETWORK=14.1.1.0/24
CRM_REDIS_IP=14.1.1.10

similarly add deatails for QAA, QAB environments as well.

Now start with compose YAML => crm.yml

version => is a built in key (mandatory)
services => is a built in key (mandatory)
volumes => is a built in key (optional)
networks => is a built in key (optional)

redis => user defined key (anything pattern can be used)

Compose creates/deletes the resources on the fly => means it creates
docker volumes etc

networks keyword here is mimicking the below command:
$ sudo docker network create -d bridge --subnet=14.1.0.0/16 --ip-
range=14.1.1.0/24 --gateway=14.1.1.1 sloopstash-dev-crm_common

volumes keyword here is mimicking the below command:
$ sudo docker volume create sloopstash-dev-crm_redis-data
```

Now run the composer yaml

```
Insider VM terminal - go to /opt/source/Devel/SLST/docker-project/

# Store environment variables.
$ export ENVIRONMENT=dev

$ cd /opt/source/Devel/SLST/docker-project

# Provision OCI containers using Docker compose.
$ sudo docker compose -f compose/crm.yml --env-file
compose/${ENVIRONMENT^^}.env -p sloopstash-${ENVIRONMENT}-crm up -d

=> where -p sloopstash-${ENVIRONMENT}-crm => project name which
enables isolation between envrionments (namespace isolation).

to check if the container is working properly:
$ sudo docker container exec -ti sloopstash-dev-crm-redis-1 /bin/bash

bash-4.2# ps -ef
UID PID PPID C STIME TTY TIME CMD
root 1 0 0 03:21 ? 00:00:00 /usr/bin/python /usr/bin/supervisord -c
/etc/supervisord.conf
root 8 1 0 03:21 ? 00:00:00 redis-server 0.0.0.0:3000
root 14 0 0 03:22 pts/0 00:00:00 /bin/bash
root 20 14 0 03:22 pts/0 00:00:00 ps -ef

to down the container:
$ sudo docker compose -f compose/crm.yml --env-file
compose/${ENVIRONMENT^^}.env -p sloopstash-${ENVIRONMENT}-crm down


to run for another environment:
$ export ENVIRONMENT=qaa

run the same docker compose command
$ sudo docker compose -f compose/crm.yml --env-file
compose/${ENVIRONMENT^^}.env -p sloopstash-${ENVIRONMENT}-crm up -d

below output: => it first creates netwrok, volume and bind the volume
and then container is created (orchestrating the resource creation)
[+] Running 4/4
✔ Network sloopstash-qaa-crm_common Created
✔ Volume "sloopstash-qaa-crm_redis-data" Created
✔ Volume "sloopstash-qaa-crm_redis-log" Created
✔ Container sloopstash-qaa-crm-redis-1 Started

[vagrant@sloopstash-dkr-swm-mgr-1 ~]$ sudo docker network ls
NETWORK ID NAME DRIVER SCOPE
098f6f1b68c2 bridge bridge local
0cf97c306153 host host local
1aacc04a389e none null local
d943ded38fcb sloopstash-qaa-crm_common bridge local

[vagrant@sloopstash-dkr-swm-mgr-1 ~]$ sudo docker volume ls
DRIVER VOLUME NAME
local sloopstash-dev-crm_app-log
local sloopstash-dev-crm_nginx-log
local sloopstash-dev-crm_redis-data
local sloopstash-dev-crm_redis-log
local sloopstash-qaa-crm_redis-data
local sloopstash-qaa-crm_redis-log


[vagrant@sloopstash-dkr-swm-mgr-1 ~]$ sudo docker container ls

similarly we can run it for other environment as well

Evern after down - volumes will persist - but netwroks will get
deleted.
```

## Setting up CRM full stack using Docker compose YAML:

Update the crm.yml and DEV.env, QAA.env, QAB.env taking code from https://github.com/sloopstash/kickstart-docker/blob/master/compose alter any code which is relevant - specifically Python version for my setup 3.12 whereas the code has 2.7 which is legacy

```
The below code in docker compose YAMl orchestrate or sequences the
automation.
depends_on:
- redis
If the redis container is not started - it will not start the app
container which is depedant on Redis.

In one terminal:
```

```
$ export ENVIRONMENT=dev
$ cd /opt/source/Devel/SLST/docker-project

# Provision OCI containers using Docker compose.
$ sudo docker compose -f compose/crm.yml --env-file
compose/${ENVIRONMENT^^}.env -p sloopstash-${ENVIRONMENT}-crm up -d

In another terminal:
$ export ENVIRONMENT=qaa
$ cd /opt/source/Devel/SLST/docker-project

# Provision OCI containers using Docker compose.
$ sudo docker compose -f compose/crm.yml --env-file
compose/${ENVIRONMENT^^}.env -p sloopstash-${ENVIRONMENT}-crm up -d

**The exported variables are only scope of current ssh session - thats
the reason we can export two different variable values and run the
container.

The containers are isolated with namespace and also with network model
abstraction.

DEV environment: http://app.crm.sloopstash.dv:8001/dashboard
QAA environment: http://app.crm.sloopstash.qaa:8002/dashboard
QAB environment: http://app.crm.sloopstash.qab:8003/dashboard


App container is not running ? debug why ?

Check error log for container name:
$ sudo docker logs sloopstash-dev-crm-app-1
Traceback (most recent call last):
File "/usr/bin/supervisord", line 5, in <module>
from supervisor.supervisord import main
ModuleNotFoundError: No module named 'supervisor'
```

## Hadoop cluster setup:

https://github.com/sloopstash/kickstart-docker/wiki/Deploy-Hadoop-cluster-(Data-Lake-stack)

```
Insider terminal:
```

```
$ sudo git clone https://github.com/sloopstash/kickstart-docker.git
/opt/kickstart-docker

$ sudo chown -R $USER:$USER /opt/kickstart-docker

# Switch to Docker starter-kit directory.
$ cd /opt/kickstart-docker

$ export ENVIRONMENT=dev

# Provision OCI containers using Docker compose.
$ sudo docker compose -f compose/data-lake/hadoop/main.yml --env-file
compose/${ENVIRONMENT^^}.env -p sloopstash-${ENVIRONMENT}-data-lake-s1
up -d
```

Errors:

```
! hadoop-data-2 The requested image's platform (linux/amd64) does not
match the detected host platform (linux/arm64/v8) and no specific
platform was requested 0.0s
! hadoop-data-3 The requested image's platform (linux/amd64) does not
match the detected host platform (linux/arm64/v8) and no specific
platform was requested 0.0s
! hadoop-data-1 The requested image's platform (linux/amd64) does not
match the detected host platform (linux/arm64/v8) and no specific
platform was requested
```

Verify Hadoop cluster:

```
# Access Bash shell of existing OCI container running Hadoop name node
1.
$ sudo docker container exec -ti sloopstash-${ENVIRONMENT}-data-lake-
s1-hadoop-name-1-1 /bin/bash

# List Hadoop data nodes.
$ hdfs dfsadmin -report

# Exit shell.
$ exit
```

Write data to HDFS filesystem

```
# Access Bash shell of existing OCI container running Hadoop data node
1.
```

```
$ sudo docker container exec -ti sloopstash-${ENVIRONMENT}-data-lake-s1-hadoop-data-1-1 /bin/bash

# Write data to HDFS filesystem.
$ hdfs dfs -mkdir -p /nginx/log/14-07-2024
$ touch access.log
$ echo "[14-07-2024 10:50:23] 14.1.1.1 app.crm.sloopstash.dv GET /dashboard HTTP/1.1 200 http://app.crm.sloopstash.dv/dashboard 950 - Mozilla Firefox - 0.034" > access.log
$ hdfs dfs -put -f access.log /nginx/log/14-07-2024

# Exit shell.
$ exit
```

Read data from HDFS filesystem

```
# Access Bash shell of existing OCI container running Hadoop data node 2.
$ sudo docker container exec -ti sloopstash-${ENVIRONMENT}-data-lake-s1-hadoop-data-2-1 /bin/bash

# Read data from HDFS filesystem.
$ hdfs dfs -ls -R /
$ hdfs dfs -cat /nginx/log/14-07-2024/access.log

# Exit shell.
$ exit
```

Manage Data Lake stack (Hadoop cluster) environments

Docker

```
# Switch to Docker starter-kit directory.
$ cd /opt/kickstart-docker

# Stop OCI containers using Docker compose.
$ sudo docker compose -f compose/data-lake/hadoop/main.yml --env-file compose/${ENVIRONMENT^^}.env -p sloopstash-${ENVIRONMENT}-data-lake-s1 down

# Restart OCI containers using Docker compose.
$ sudo docker compose -f compose/data-lake/hadoop/main.yml --env-file compose/${ENVIRONMENT^^}.env -p sloopstash-${ENVIRONMENT}-data-lake-s1 restart
```

# DevOps Notes: Kubernetes Part 1:

## Setting Up 3 VMs for Docker Swarm and Kubernetes

**Docker:**

```
# Download Docker starter-kit from GitHub to local filesystem path.
$ sudo git clone https://github.com/sloopstash/kickstart-docker.git
/opt/kickstart-docker-new

# Change ownership of Docker starter-kit directory.
$ sudo chown -R $USER /opt/kickstart-docker-new

$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-dkr-mgr-1

$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-dkr-wkr-1

$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-dkr-wkr-2


# Boot Linux VM of Docker node using Vagrant.
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-dkr-mgr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-dkr-wkr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-dkr-wkr-2

# Halt Linux VM of Docker node using Vagrant.
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
halt sloopstash-dkr-mgr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
halt sloopstash-dkr-wkr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
halt sloopstash-dkr-wkr-2

# SSH to Linux VM of Docker node using Vagrant.
```

```
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
ssh sloopstash-dkr-mgr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
ssh sloopstash-dkr-wkr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
ssh sloopstash-dkr-wkr-2
```

**Kubernetes:**

```
# Download Kubernetes starter-kit from GitHub to local filesystem
path.
$ sudo git clone https://github.com/sloopstash/kickstart-
kubernetes.git /opt/kickstart-kubernetes

# Change ownership of Kubernetes starter-kit directory.
$ sudo chown -R $USER /opt/kickstart-kubernetes

# Boot Linux VM of Kubernetes node using Vagrant.
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-k8s-mtr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-k8s-wkr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-k8s-wkr-2

# Halt Linux VM of Kubernetes node using Vagrant.
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
halt sloopstash-k8s-mtr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
halt sloopstash-k8s-wkr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
halt sloopstash-k8s-wkr-2

# SSH to Linux VM of Kubernetes node using Vagrant.
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
ssh sloopstash-k8s-mtr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
ssh sloopstash-k8s-wkr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
ssh sloopstash-k8s-wkr-2
```

```
$ hostname
$ free -h
```

Docker stand alone is good for running containers (mostly building the stack containers). but for production grade Kubernetes/Docker swarm is important for orchestrating and run these containers.
Node:

- mostly a VM or else a host machine
- Container runtime or container engine

Cluster => node to node / multi node communication forms the cluster.
Docker compose YAML/Kubernetes YAML is container orchestrator => sequential provisioning and automation of resources in container cluster.
Core features:

- High availability and distribution
- load-balancing and routing
- rollout and rollback deployments
- Resource isolation => nwk, storage, containers etc
- Service discovery => connect to micro service endpoints
- scheduling => map containers to specific node in cluster
- Autoscaling => both up and down
- Fault tolerance

## Docker Swarm vs Kubernetes:

| | |
|---|---|
| Docker swarm is Docker native => it works only with docker engine (container runtime) | Kubernetes is framework => works with different container runtimes not only specific to Docker. supports:<br>• containerd<br>• docker<br>• mirantis<br>• CRI-O<br>Kubernetes is cluster as well as orchestrator. |
| Container nwk model<br>• overlay | Container nwking interface<br>• cilium<br>• calcio<br>• Flannel<br>• Amazon VPC CNI<br>• Azure CNI etc. |

## Setting up Kubernetes master VM: (Install tools in Kubernetes VM)

```
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-k8s-mtr-1

# ssh into VM
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
ssh sloopstash-k8s-mtr-1

$ ifconfig (shows ip configured in vagrant file => 192.168.201.61)
=> ....
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.201.61 netmask 255.255.255.0 broadcast 192.168.201.255
......
```

**Install Docker on Master VM** => https://sloopstash.com/knowledge-base/how-to-install-docker-on-linux.html
https://sloopstash.sharepoint.com/sites/training/SitePages/Library/DevOps/Kubernetes/07-Setup-Kubernetes.aspx

```
# Install required system packages.
```

```
$ sudo yum install yum-utils device-mapper-persistent-data lvm2

# Add Docker repository to package manager source list.
$ sudo yum-config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo

# Install Docker, Docker client, and Containerd.
$ sudo yum install docker-ce docker-ce-cli containerd.io

# Check Docker version.
$ docker --version

# Check status of Docker service.
$ sudo systemctl status docker.service

# Enable Docker service at boot.
$ sudo systemctl enable docker.service
```

**Kubernetes specific configuration for Docker:**

```
$ sudo nano /etc/docker/daemon.json

{
"hosts": [
"tcp://0.0.0.0:2375",
"unix:///var/run/docker.sock"
],
"exec-opts": ["native.cgroupdriver=systemd"],
"containerd": "/run/containerd/containerd.sock",
"storage-driver": "overlay2",
"storage-opts": [],
"log-driver": "json-file",
"log-opts": {
"max-size": "100m"
}
}

# Override docker systemd configuration.
$ sudo mkdir /etc/systemd/system/docker.service.d
$ sudo nano /etc/systemd/system/docker.service.d/override.conf

[Service]
ExecStart=
ExecStart=/usr/bin/dockerd
```

```
# reolad systemd service.
$ sudo systemctl daemon-reload

# restart docker service
$ sudo systemctl restart docker.service

# check status if running => it should show active/enabled
$ sudo systemctl status docker.service
```

**Install and configure Docker CRI: couples Docker with Kubernetes:**

```
# Download Cgroup library.
$ wget
https://raw.repo.almalinux.org/almalinux/8/BaseOS/aarch64/os/Packages/
libcgroup-0.41-19.el8.aarch64.rpm -P /tmp

# Install Cgroup library.
$ sudo yum install -y /tmp/libcgroup-0.41-19.el8.aarch64.rpm


Need to do from below:check proper version for aarch64 here =>
https://github.com/Mirantis/cri-dockerd/releases/tag/v0.3.14

# Download Docker CRI. => coupling b/w docker engine and kublet
$ wget https://github.com/Mirantis/cri-
dockerd/releases/download/v0.3.14/cri-dockerd-0.3.14.arm64.tgz -P /tmp

$ tar xzvf /tmp/cri-dockerd-0.3.14.arm64.tgz
$ sudo mv /tmp/cri-dockerd/cri-dockerd /usr/local/bin/

# 1) Download the official socket definition:
sudo wget -O /etc/systemd/system/cri-docker.socket \
https://raw.githubusercontent.com/Mirantis/cri-
dockerd/master/packaging/systemd/cri-docker.socket

# 2) (Optional but recommended) Also grab the service unit:
sudo wget -O /etc/systemd/system/cri-docker.service \
https://raw.githubusercontent.com/Mirantis/cri-
dockerd/master/packaging/systemd/cri-docker.service

# 3) Edit the service if you installed cri-dockerd to a non-standard
path
# e.g. if your binary is in /usr/local/bin instead of /usr/bin:
```

```
sudo sed -i 's|/usr/bin/cri-dockerd|/usr/local/bin/cri-dockerd|'
/etc/systemd/system/cri-docker.service


PROMPT USED ON CHATGPT:
cri-docker.service: Failed to locate executable /usr/local/bin/cri-
dockerd: Permission denied
cri-docker.service: Failed at step EXEC spawning /usr/local/bin/cri-
dockerd: Permission denied

If you're on an SELinux-enforcing host (e.g. CentOS/RHEL), the default
install might label it unconfined_u:object_r:usr_t which isn't
executable by systemd. Reset it to the proper bin label:

$ sudo restorecon -v /usr/local/bin/cri-dockerd

ls -Z /usr/local/bin/cri-dockerd => should show below with bin_t label
# Should show something like: system_u:object_r:bin_t:s0

$ sudo systemctl daemon-reload

# Start Docker CRI service at boot.
$ sudo systemctl start cri-docker.service

# Check status of Docker CRI service.
$ sudo systemctl status cri-docker.service

# Enable Docker CRI service at boot.
$ sudo systemctl enable cri-docker.service

$ sudo nano /etc/crictl.yaml

runtime-endpoint: unix:///var/run/cri-dockerd.sock
image-endpoint: unix:///var/run/cri-dockerd.sock
timeout: 20
debug: false
```

**Install Kubernetes tools:**

```
# Set SELinux in permissive mode.
$ sudo setenforce 0

$ sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/'
/etc/selinux/config
```

```
# Permanently disable Linux swap memory.
$ sudo swapoff -a

$ sudo nano /etc/fstab

#/swapfile none swap defaults 0 0

# Enable bridge netfilter Linux kernel module.
$ sudo modprobe br_netfilter
$ sudo nano /etc/modules-load.d/kubernetes.conf
br_netfilter

# Permanently set required Linux kernel params.
$ sudo nano /etc/sysctl.d/kubernetes.conf

vm.swappiness=0
net.bridge.bridge-nf-call-ip6tables=1
net.ipv4.ip_forward=1
net.bridge.bridge-nf-call-iptables=1

# Apply required Linux kernel params.
$ sudo sysctl --system


# Add Kubernetes repository to package manager source list.
$ sudo nano /etc/yum.repos.d/kubernetes.repo

[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.28/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.28/rpm/repodata/repomd.xml
.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni

# Install Kubernetes packages from repository.
$ sudo yum install -y kubelet kubeadm kubectl --
disableexcludes=kubernetes

# Explicitly set private IP on Kubernetes master node 1.
$ sudo nano /etc/sysconfig/kubelet
```

```
KUBELET_EXTRA_ARGS=--node-ip=192.168.201.61
=> VMs IP
```

**Worker node specific settings:**

```
# Explicitly set private IP on Kubernetes worker node 1.
$ sudo nano /etc/sysconfig/kubelet
KUBELET_EXTRA_ARGS=--node-ip=192.168.201.64

# Explicitly set private IP on Kubernetes worker node 2.
$ sudo nano /etc/sysconfig/kubelet
KUBELET_EXTRA_ARGS=--node-ip=192.168.201.65
```

**Check Kubelet service status:**

```
# Check status of Kubelet service.
$ sudo systemctl status kubelet.service

# Enable Kubelet service at boot. not needed for now
# $ sudo systemctl enable kubelet.service
```

Master node is also referred as Kubernetes control plane.
$ **kubeadm init**
=> Initiates the cluster bootstrapping in the master node - this is run only on master node.
=> creates the cluster metadata (state of cluster => stores info of all worker nodes in cluster => containers etc.)
=> clustered metadata is stored in **Etcd** service.
=> raft consensus cluster => if any master node fails => the other master node will become leader. Etcd will sync across these master nodes.

Worker node => to run workload container ex: nginx, app etc.
=> We should not run workload on any master nodes as it is solely for monitoring cluster.

Kubernetes agents runs on all nodes in cluster and sync the cluster metadata => worker nodes state changes to master => some commands run on master node sync it ti worker node.

## Initialise Kuberentes Cluster

https://sloopstash.sharepoint.com/sites/training/SitePages/Library/DevOps/Kubernetes/08-Configure-Kubernetes.aspx

```
# Initialize Kubernetes cluster. IMPORTANT: Save the output of below
command somewhere.
for master node:
$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --apiserver-
advertise-address=192.168.201.61 --cri-socket=unix:///var/run/cri-
dockerd.sock

=> where pod nwk range is set to be 10.244.0.0/16 bcz we will be using
Flannel network which recommends 10.244 namespace - which will be
learnt after.

=> where apiserver-advertise-address is kubernetese api server address
to interact with client.

=> where --cri-socket is not needed for containerd runtime (which is
default supported by Kubernetes) - but for Docker we need to specify
this socket to communicate with kubelet and Docker.

below error is shown:
error execution phase preflight: [preflight] Some fatal errors
occurred:
[ERROR FileContent--proc-sys-net-bridge-bridge-nf-call-iptables]:
"/proc/sys/net/bridge/bridge-nf-call-iptables does not exist
[preflight] If you know what you are doing, you can make a check non-
fatal with `--ignore-preflight-errors=..."

run:
$ sudo modprobe br_netfilter

and run the above init command again:

facing issue with init command not completing with
Waiting for the kubelet to boot up the control plane as static Pods
from directory "/etc/kubernetes/manifests". This can take up to 4m0s

Unfortunately, an error has occurred:
timed out waiting for the condition

This error is likely caused by:
- The kubelet is not running
- The kubelet is unhealthy due to a misconfiguration of the node in
some way (required cgroups disabled)

If you are on a systemd-powered system, you can try to troubleshoot
the error with the following commands:
- 'systemctl status kubelet'
- 'journalctl -xeu kubelet'

Additionally, a control plane component may have crashed or exited
when started by the container runtime.
To troubleshoot, list all containers using your preferred container
runtimes CLI.
Here is one example how you may list all running Kubernetes containers
by using crictl:
- "crictl --runtime-endpoint unix:///var/run/cri-dockerd.sock ps -a |
grep kube | grep -v pause'
Once you have found the failing container, you can inspect its logs
with:
- 'crictl --runtime-endpoint unix:///var/run/cri-dockerd.sock logs
CONTAINERID'
error execution phase wait-control-plane: couldn't initialize a
Kubernetes cluster
To see the stack trace of this error execute with --v=5 or higher"


another error:
Unfortunately, an error has occurred:
timed out waiting for the condition

This error is likely caused by:
- The kubelet is not running
- The kubelet is unhealthy due to a misconfiguration of the node in
some way (required cgroups disabled)

If you are on a systemd-powered system, you can try to troubleshoot
the error with the following commands:
- 'systemctl status kubelet'
- 'journalctl -xeu kubelet'

Additionally, a control plane component may have crashed or exited
when started by the container runtime.
To troubleshoot, list all containers using your preferred container
runtimes CLI.
```

```
Here is one example how you may list all running Kubernetes containers
by using crictl:
- 'crictl --runtime-endpoint unix:///var/run/cri-dockerd.sock ps -a |
grep kube | grep -v pause'
Once you have found the failing container, you can inspect its logs
with:
- 'crictl --runtime-endpoint unix:///var/run/cri-dockerd.sock logs
CONTAINERID'
error execution phase wait-control-plane: couldn't initialize a
Kubernetes cluster
To see the stack trace of this error execute with --v=5 or higher

# to reset the above command.
$ sudo kubeadm reset --cri-socket=unix:///var/run/cri-dockerd.sock
```

Successful output is stored in kubenetes-project/tmp/commands.txt
token below:

```
kubeadm join 192.168.201.61:6443 --token 1klcwv.q1j9az61dwths4qg \
--discovery-token-ca-cert-hash
sha256:9f5bb27ba8e6d0a3c7da1d41dc468ce17207f0ecf50737caad30acc964be12c
5
```

```
$ sudo docker image ls

$ sudo crictl pods

$ sudo crictl images

$ sudo systemctl status kubelet.service
```

```
# Configure Kubernetes client with admin user credentials.
$ mkdir -p $HOME/.kube
$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
$ sudo chown $USER:$USER $HOME/.kube/config
$ sudo chmod go+r /etc/kubernetes/pki/ca.crt
$ sudo chmod go+r /etc/kubernetes/pki/ca.key
# Create Flannel network in Kubernetes cluster.
```

```
$ kubectl apply -f
https://raw.githubusercontent.com/sloopstash/kickstart-
kubernetes/master/cni/flannel/main.yml

Error with old ip address was showing:
kubectl config set-cluster kubernetes --
server=https://192.168.201.61:6443

# List Kubernetes nodes.
$ kubectl get nodes -o wide
```

## Graceful Shutdown of master node:

- Always shutdown worker node first.
- on master node - best practice is to stop kubelet and docker and then halting the VM.

## Understanding the output of kubeadm init (bootstrap process):

1. Preflight checks:
    - Checks the OS and system checks for compatibility of running control plane
    - pull the images required for setting up kubernetes cluster.

```
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes
cluster
[preflight] This might take a minute or two, depending on the speed of
your internet connection
[preflight] You can also perform this action in beforehand using
'kubeadm config images pull'
```

2. Generate SSL certificates:
    - Implements authentication between cluster resources and components.
    - enables encrypted communication between cluster resources and components => **/etc/kubernetes/pki**

```
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [kubernetes
kubernetes.default kubernetes.default.svc
kubernetes.default.svc.cluster.local sloopstash-k8s-mtr-1] and IPs
[10.96.0.1 192.168.201.61]
[certs] Generating "apiserver-kubelet-client" certificate and key
```

```
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [localhost
sloopstash-k8s-mtr-1] and IPs [192.168.201.61 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [localhost
sloopstash-k8s-mtr-1] and IPs [192.168.201.61 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "sa" key and public key
```

3. Creates Kubernetes config files:

```
kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
```

4. Creates Kubernetes Manifest files (YAML):
   - To run kubernetes control plane or master node services.

```
[etcd] Creating static Pod manifest for local etcd in
"/etc/kubernetes/manifests"
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-
manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
```

5. Starts Kubernetes agent (kubelet):

```
[kubelet-start] Writing kubelet environment file with flags to file
"/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file
"/var/lib/kubelet/config.yaml"
[kubelet-start] Starting the kubelet
```

6. Starts Kubernetes control plane:
   - Kubelet starts the control plane as service pods or containers.
   - performs tainting labelling of master node

```
[wait-control-plane] Waiting for the kubelet to boot up the control
plane as static Pods from directory "/etc/kubernetes/manifests". This
can take up to 4m0s
```

```
[apiclient] All control plane components are healthy after 3.501651
seconds
[upload-config] Storing the configuration used in ConfigMap "kubeadm-
config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config" in namespace kube-
system with the configuration for the kubelets in the cluster
[upload-certs] Skipping phase. Please see --upload-certs
[mark-control-plane] Marking the node sloopstash-k8s-mtr-1 as control-
plane by adding the labels: [node-role.kubernetes.io/control-plane
node.kubernetes.io/exclude-from-external-load-balancers]
[mark-control-plane] Marking the node sloopstash-k8s-mtr-1 as control-
plane by adding the taints [node-role.kubernetes.io/control-
plane:NoSchedule]
```

7. Generate bootstrap tokens:
    - creates token to be ables to join with worker nodes and this is to be saved.

```
[bootstrap-token] Using token: e60vor.qjj5rfg3rf1xpuc1
[bootstrap-token] Configuring bootstrap tokens, cluster-info
ConfigMap, RBAC Roles
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens
to get nodes
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens
to post CSRs in order for nodes to get long term certificate
credentials
[bootstrap-token] Configured RBAC rules to allow the csrapprover
controller automatically approve CSRs from a Node Bootstrap Token
[bootstrap-token] Configured RBAC rules to allow certificate rotation
for all node client certificates in the cluster
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-
public" namespace
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to
a rotatable kubelet client certificate and key
```

8. Start Kubernetes add-ons:
    - CoreDNS and Kube-proxy

```
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy
```

- And at last output will show tokens and these needs to be saved. (token is saved in commands.txt)

Fix for modprobe br_netfilter:

```
ssh into terminal:
```

```
$ sudo systemctl stop kubelet.service
$ sudo modprobe br_netfilter
$ sudo systemctl start kubelet.service

$ kubectl get nodes -o wide

$ sudo crictl pods => should show all ready

$ sudo nano /etc/modules-load.d/kubernetes.conf
br_netfilter

exit ther terminal and halt the master node and do vagrant up again.
```

## worker node setup:

```
Complete all steps upto install Kubernetes tools - do not run
kubeadm init and all.

$ sudo systemctl enable kubelet.service

# Store Kubernetes bootstrap tokens as environment variables.
$ K8S_WORKER_TOKEN=<TOKEN>
$ K8S_WORKER_TOKEN_HASH=<TOKEN_HASH>

take the tokens saved from master node
$ K8S_WORKER_TOKEN=1klcwv.q1j9az61dwths4qg
$
K8S_WORKER_TOKEN_HASH=sha256:9f5bb27ba8e6d0a3c7da1d41dc468ce17207f0ecf
50737caad30acc964be12c5


# Join Kubernetes worker node to the cluster.
$ sudo kubeadm join 192.168.201.61:6443 --token $K8S_WORKER_TOKEN --
discovery-token-ca-cert-hash $K8S_WORKER_TOKEN_HASH --cri-
socket=unix:///var/run/cri-dockerd.sock

or

$ sudo kubeadm join 192.168.201.61:6443 --token
e60vor.qjj5rfg3rf1xpuc1 --discovery-token-ca-cert-hash
sha256:2ebb1646487ce4964f70fa86a7274cc930175568a40c6c6e39985ce9e7971e4
0 --cri-socket=unix:///var/run/cri-dockerd.sock
```

```
[preflight] Running pre-flight checks
[WARNING Service-Kubelet]: kubelet service is not enabled, please run
'systemctl enable kubelet.service'
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n
kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file
"/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file
"/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS
Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was
received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the
cluster.

If any issue with token => create the join command from master:
$ sudo kubeadm token create --print-join-command --cri-
socket=unix:///var/run/cri-dockerd.sock

after joining: run below command on master node
$ kubectl get nodes -o wide > should show the worker node which is
joined.
```

# DevOps Notes: Docker Cluster

## Setting Up 3 VMs for Docker Swarm

```
# Download Docker starter-kit from GitHub to local filesystem path.
$ sudo git clone https://github.com/sloopstash/kickstart-docker.git
/opt/kickstart-docker-new

# Change ownership of Docker starter-kit directory.
$ sudo chown -R $USER /opt/kickstart-docker-new
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-dkr-mgr-1

$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-dkr-wkr-1

$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-dkr-wkr-2
```

```
# Boot Linux VM of Docker node using Vagrant.
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-dkr-mgr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-dkr-wkr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-dkr-wkr-2

# Halt Linux VM of Docker node using Vagrant.
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
halt sloopstash-dkr-mgr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
halt sloopstash-dkr-wkr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
halt sloopstash-dkr-wkr-2

# SSH to Linux VM of Docker node using Vagrant.
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
ssh sloopstash-dkr-mgr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
ssh sloopstash-dkr-wkr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
ssh sloopstash-dkr-wkr-2
```

Sometimes VM up does not work:
[msg.disk.noBackEnd] Cannot open the disk '/opt/kickstart-docker-new/vagrant/alma-linux-9/vmware/arm64/server/.vagrant/machines/sloopstash-dkr-mgr-1/vmware_fusion/040afbe5-b6f3-4902-ab6b-53faa90d6ff5/disk-cl1.vmdk' or one of the snapshot disks it depends on.

```
sudo dnf install -y open-vm-tools
sudo systemctl enable --now vmtoolsd
sudo systemctl status vmtoolsd

cd /opt/kickstart-docker-new/vagrant/alma-linux-9/vmware/arm64/server

# Remove only the VMware Fusion disk folder
rm -rf .vagrant/machines/sloopstash-dkr-mgr-1/vmware_fusion

or destory the VM and run up agin.

# Re-provision that single machine
vagrant up sloopstash-dkr-mgr-1
```

## Docker Swarm

```
Install docker - https://sloopstash.com/knowledge-base/how-to-install-
docker-on-linux.html

# Install required system packages.
$ sudo yum install yum-utils device-mapper-persistent-data lvm2

# Add Docker repository to package manager source list.
$ sudo yum-config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo

# Install Docker, Docker client, and Containerd.
$ sudo yum install docker-ce docker-ce-cli containerd.io

# Check Docker version.
$ docker --version

# Check status of Docker service.
$ sudo systemctl status docker.service

# Enable Docker service at boot.
$ sudo systemctl enable docker.service

# Modify Docker daemon configuration.
$ sudo nano /etc/docker/daemon.json

{
"hosts": [
"tcp://0.0.0.0:2375",
"unix:///var/run/docker.sock"
],
"containerd": "/run/containerd/containerd.sock",
"storage-driver": "overlay2",
"log-driver": "json-file",
"log-opts": {
"max-size": "100m"
}
}

# Override Docker service configuration in Systemd.
$ sudo mkdir /etc/systemd/system/docker.service.d
$ sudo nano /etc/systemd/system/docker.service.d/override.conf
[Service]
ExecStart=
ExecStart=/usr/bin/dockerd

# Reload Systemd service.
$ sudo systemctl daemon-reload

# Apply configuration changes to Docker service by restarting it.
$ sudo systemctl restart docker.service


# Do not run the below command for worker node.
# Initialize Docker swarm cluster.
$ sudo docker swarm init --advertise-addr 192.168.201.51

Swarm initialized: current node (vc23q2032nrgpvlsmykr4snsi) is now a
manager.
To add a worker to this swarm, run the following command:
docker swarm join --token SWMTKN-1-
5u07b09nscjmxdpfyk5xpwj9fjh4emhgm4tl37fks2t3lai4hj-
3pxoh86j3xm6ypyhzwq2jqfzb 192.168.201.51:2377

To add a manager to this swarm, run 'docker swarm join-token manager'
and follow the instructions.

$ sudo docker node ls => see the list of nodes in docker
```

## Join nodes to Docker swarm cluster

```
# Store Docker swarm join token as environment variable. take the
below token from swam manager init output.
$ DOCKER_WORKER_TOKEN=SWMTKN-1-
5u07b09nscjmxdpfyk5xpwj9fjh4emhgm4tl37fks2t3lai4hj-
```

```
3pxoh86j3xm6ypyhzwq2jqfzb
# Join Docker worker node to the cluster.
$ sudo docker swarm join --token $DOCKER_WORKER_TOKEN
192.168.201.51:2377
=> This node joined a swarm as a worker.
```

### Deep dive into Docker swarm cluster

Its always necessary to have some kind of depth understanding about the internal implementation of Docker swarm cluster. Let's use below commands to dive deep into the internals of Docker swarm cluster implementation.

- **7946** port is used for node-node communication and **2377** port is used to sync cluster state between the Docker manager nodes.
- **2375** => communication b/w docker client to docker daemon (remote) recommended by docker. and this is configured in docker configuration /etc/docker/daemon.json

```
On Master node:
# List Docker sockets and connections. on Master node
[vagrant@sloopstash-dkr-mgr-1 ~]$ sudo netstat -nalp | grep docker
tcp6 0 0 :::2375 :::* LISTEN 27790/dockerd
tcp6 0 0 :::2377 :::* LISTEN 27790/dockerd
tcp6 0 0 :::7946 :::* LISTEN 27790/dockerd

On worker node:
[vagrant@sloopstash-dkr-wkr-1 ~]$ sudo netstat -nalp | grep docker
tcp 0 0 192.168.201.54:43988 192.168.201.51:2377 ESTABLISHED
27758/dockerd
tcp6 0 0 :::7946 :::* LISTEN 27758/dockerd
tcp6 0 0 :::2375 :::* LISTEN 27758/dockerd
udp6 0 0 :::7946 :::* 27758/dockerd
```

Try to setup worker 2 as well.

# DevOps Notes: Kubernetes Part 2:

## Container Runtime interface (CRI):

- Library developed and maintnd. by CNCF community

- couples Kubelet agent and container run time (container d (default in Kubernetes, thst why we use --cri-socket in command to specify Docker), Docker, Mirantis etc.)

## Different container run times and why we chose Docker:

- Docker
  - Docker tools are easy to use offers better community support - stable container engine
- Container-d
  - Devloped by docker's moby project and CNCF community - not much easy to use tools
- Mirantis
  - Enterprise addition of docker and container runtime
- CRI-O
  - not a matured container runtime like docker or containerd

## Container Network interface (CNI):

- Creates and manages nwk interfaces for pods (containers)
- Assign IP address to the pod containers
- enables network traffic routing

## N/W drivers and why do we chose Flannel:

- AWS VPC
  - developed by AWS for Kubernetes pods running in Amazon EKS cluster
- Azure CNI
  - Runs on Azure Kubernetes cluster
- Flannel
  - Uses  bridge nwk driver in linux kernel to implement nwk implementations - works with cloud and on premise.
- Calico
  - Implemented on Azure kubernetes cluster
- Cilium

## Hands on:

```
Start the master node:
$ cd /opt/kickstart-kubernetes

$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-k8s-mtr-1

# start the kubelet service
```

```
sudo systemctl start kubelet.service

Start 1 worker node:
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-k8s-wkr-1

# start the kubelet service
sudo systemctl start kubelet.service

On maste node - check nodes status:
[vagrant@sloopstash-k8s-mtr-1 ~]$ kubectl get nodes
NAME STATUS ROLES AGE VERSION
sloopstash-k8s-mtr-1 Ready control-plane 5d19h v1.28.15
sloopstash-k8s-wkr-1 Ready <none> 5d19h v1.28.15
```

## How Kubelet couples with CRI

```
In the worker node:
[vagrant@sloopstash-k8s-wkr-1 ~]$ sudo ps -ef | grep kubelet
root 7073 1 1 02:27 ? 00:00:02 /usr/bin/kubelet --bootstrap-
kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --
kubeconfig=/etc/kubernetes/kubelet.conf --
config=/var/lib/kubelet/config.yaml --container-runtime-
endpoint=unix:///var/run/cri-dockerd.sock --pod-infra-container-
image=registry.k8s.io/pause:3.9 --node-ip=192.168.201.64
vagrant 8042 6927 0 02:29 pts/0 00:00:00 grep --color=auto kubelet

=> where Kubelet is Kubernetes agent
We can see above "-container-runtime-endpoint=unix:///var/run/cri-
dockerd.sock" is selected by Kubelet and this is done by Docker CRI

[vagrant@sloopstash-k8s-wkr-1 ~]$ sudo ps -ef | grep dockerd
root 1011 1 0 02:25 ? 00:00:07 /usr/bin/dockerd
root 4817 1 0 02:25 ? 00:00:07 /usr/local/bin/cri-dockerd --container-
runtime-endpoint fd://
root 7073 1 1 02:27 ? 00:00:14 /usr/bin/kubelet --bootstrap-
kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --
kubeconfig=/etc/kubernetes/kubelet.conf --
config=/var/lib/kubelet/config.yaml --container-runtime-
endpoint=unix:///var/run/cri-dockerd.sock --pod-infra-container-
image=registry.k8s.io/pause:3.9 --node-ip=192.168.201.64
vagrant 10045 6927 0 02:40 pts/0 00:00:00 grep --color=auto dockerd
```

```
fd:// => referes to local unix socket used by CRI-docker

[vagrant@sloopstash-k8s-wkr-1 ~]$ ls /var/run => we can see cri-
dockerd.sock
-------
agetty.reload chrony-dhcp cri-dockerd.sock dbus
-------
```

## How CRI couples with container runtime

```
[vagrant@sloopstash-k8s-wkr-1 ~]$ ls /var/run/
agetty.reload chrony-dhcp cri-dockerd.sock dbus docker.pid
gssproxy.pid lock motd.d rpcbind.sock sepermit sudo tuned vmware
auditd.pid console crond.pid dmeventd-client docker.sock

=> Kubelet connects with CRI using cri-docker socket and CRI connects
with Docker daemon using docker.sock
```

## Verifying CNI implementation:

```
$ cd /etc/cni/net.d
[vagrant@sloopstash-k8s-wkr-1 net.d]$ ls
10-flannel.conflist

=> We have used flannel nwk

On master node:
[vagrant@sloopstash-k8s-mtr-1 ~]$ kubectl get
configmaps,deployments,replicasets,daemonsets,pods,services -o wide -n
kube-flannel
NAME DATA AGE
configmap/kube-flannel-cfg 2 5d20h
configmap/kube-root-ca.crt 1 5d20h

NAME DESIRED CURRENT READY UP-TO-DATE AVAILABLE NODE SELECTOR AGE
CONTAINERS IMAGES SELECTOR
daemonset.apps/kube-flannel-ds 2 2 2 2 2 <none> 5d20h kube-flannel
docker.io/rancher/mirrored-flannelcni-flannel:v0.19.1 app=flannel

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
pod/kube-flannel-ds-f6884 1/1 Running 1 (5d20h ago) 5d20h
192.168.201.64 sloopstash-k8s-wkr-1 <none> <none>
```

```
pod/kube-flannel-ds-l66vb 1/1 Running 1 (5d20h ago) 5d20h
192.168.201.61 sloopstash-k8s-mtr-1 <none> <none>
```

from the above output => We can see flannel nwk works on both master and worker nodes => it runs pod container in every single node => using the configuration 10-flannel.conflist mentioned above.

```
$ sudo su
$ cd /var/lib/cni

[root@sloopstash-k8s-mtr-1 cni]# ls -la
total 4
drwx------. 5 root root 50 May 22 06:30 .
drwxr-xr-x. 37 root root 4096 May 28 2025 ..
drwx------. 3 root root 21 May 22 06:30 cache
drwx------. 2 root root 150 May 28 02:23 flannel
drwxr-xr-x. 3 root root 18 May 22 06:30 networks

[root@sloopstash-k8s-mtr-1 cni]# cd flannel/
[root@sloopstash-k8s-mtr-1 flannel]# ls
7aae30f94c5cb6056472bddddb8e87b3cafad0f5c4fb6af7ae4d678fffba0d8a
e640952d72eb07d02db3d6ec433ad98f32f36e2e83fe721f9d0c966a6bdd3e19
[root@sloopstash-k8s-mtr-1 flannel]# cat
7aae30f94c5cb6056472bddddb8e87b3cafad0f5c4fb6af7ae4d678fffba0d8a
{"cniVersion":"0.3.1","hairpinMode":true,"ipMasq":false,"ipam":
{"ranges":[[{"subnet":"10.244.0.0/24"}]],"routes":
[{"dst":"10.244.0.0/16"}],"type":"host-
local"},"isDefaultGateway":true,"isGateway":true,"mtu":1450,"name":"cb
r0","type":"bridge"}[root@sloopstash-k8s-mtr-1 flannel]
```

So from the above output in file => we can see flannel is bridge nwk type.

In worker node or master node - we can see flannel nwk created using ifconfig:

```
[vagrant@sloopstash-k8s-wkr-1 net.d]$ ifconfig
..........
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.201.64 netmask 255.255.255.0 broadcast 192.168.201.255
inet6 fe80::20c:29ff:fe0c:1000 prefixlen 64 scopeid 0x20<link>
ether 00:0c:29:0c:10:00 txqueuelen 1000 (Ethernet)
RX packets 4364 bytes 2338372 (2.2 MiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 4040 bytes 515939 (503.8 KiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
device interrupt 47 memory 0x3ee00000-3ee20000

flannel.1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1450
inet 10.244.1.0 netmask 255.255.255.255 broadcast 0.0.0.0
inet6 fe80::ccbd:9ff:fe68:35dc prefixlen 64 scopeid 0x20<link>
ether ce:bd:09:68:35:dc txqueuelen 0 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 15 overruns 0 carrier 0 collisions 0
.........
```

## Kubernetes Client and Kubernetes User:

Install Kubernetes client - https://sloopstash.com/knowledge-base/how-to-install-kubernetes-client-on-windows-mac-and-linux.html

```
Install Kubernetes client.
$ brew install kubernetes-cli@1.28
=> 28 vesion is not avaialable - so install 32 version
$ brew install kubernetes-cli@1.32

# Check Kubernetes client version.
$ kubectl version --client
```

- Do not allow SSH access to any nodes in the kubernetes cluster for developers or QA etc.
- We have to provide access to them using **kubeconfig** file whereas they have to access using kubernetes client.
- SSL certificates created during kubeadm init - will be used to generate self signed ssl certificates using openssl command and provide them to be able to access control plane.

```
on the Master node:
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-k8s-mtr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
ssh sloopstash-k8s-mtr-1

[vagrant@sloopstash-k8s-mtr-1 ~]$ sudo systemctl start kubelet.service
```

```
[vagrant@sloopstash-k8s-mtr-1 ~]$ kubectl get nodes

$ sudo git clone https://github.com/sloopstash/kickstart-
kubernetes.git /opt/kickstart-kubernetes
$ sudo chown -R $USER:$USER /opt/kickstart-kubernetes

# Switch to Kubernetes starter-kit directory.
$ cd /opt/kickstart-kubernetes

# Generate SSL private key.
$ openssl genrsa -out secret/tuto.key 2048

[vagrant@sloopstash-k8s-mtr-1 kickstart-kubernetes]$ ls secret/
tuto.key
# Generate SSL CSR (certificate signing request file)
$ openssl req -new -key secret/tuto.key -out secret/tuto.csr -subj
"/CN=tuto"

[vagrant@sloopstash-k8s-mtr-1 kickstart-kubernetes]$ ls secret/
tuto.csr tuto.key
# Generate SSL certificate using SSL CA certificate and CA private key
of Kubernetes master node 1
# CA certicate => certificates generated during cluster bootstrap/root
certificate will be used to generate self signed certificate.
$ openssl x509 -req -days 365 -in secret/tuto.csr -CA
/etc/kubernetes/pki/ca.crt -CAkey /etc/kubernetes/pki/ca.key -
CAcreateserial -CAserial secret/tuto.srl -out secret/tuto.crt

secret/tuto.crt
Certificate request self-signature ok
subject=CN=tuto

[vagrant@sloopstash-k8s-mtr-1 kickstart-kubernetes]$ ls secret/
tuto.crt tuto.csr tuto.key tuto.srl
```

## Generate configuration for Kubernetes user

Open SSH connection to the Kubernetes **master** node, then execute the below commands to generate a Kubernetes client configuration for the Kubernetes user named **tuto**.

- Create **credentials** for the Kubernetes user using the SSL certificate and SSL private key.

- Create **cluster configuration** for the Kubernetes user using the SSL CA certificate available in the **master** Kubernetes node.
- Create **context** for the Kubernetes user by mapping it with the cluster configuration.

```
# Switch to Kubernetes starter-kit directory.
$ cd /opt/kickstart-kubernetes

# Create credentials in Kubernetes client.
$ kubectl config set-credentials tuto --client-
certificate=secret/tuto.crt --client-key=secret/tuto.key --kubeconfig
secret/tuto.conf
=> User "tuto" set.

[vagrant@sloopstash-k8s-mtr-1 kickstart-kubernetes]$ ls secret/
tuto.conf tuto.crt tuto.csr tuto.key tuto.srl
=> tuto.conf is kubeconfig file generated for developers to be able to
access kubeternetes control plane

# Create cluster configuration in Kubernetes client.
$ kubectl config set-cluster kubernetes --server
https://192.168.201.61:6443 --certificate-authority
/etc/kubernetes/pki/ca.crt --kubeconfig secret/tuto.conf
=> Cluster "kubernetes" set.
with this the secret/tuto.conf will be update with more details like -
cluster details, server etc

[vagrant@sloopstash-k8s-mtr-1 kickstart-kubernetes]$ cat
secret/tuto.conf
apiVersion: v1
clusters:
- cluster:
certificate-authority: /etc/kubernetes/pki/ca.crt
server: https://192.168.201.61:6443
name: kubernetes
contexts: null
current-context: ""
kind: Config
preferences: {}
users:
- name: tuto
user:
client-certificate: tuto.crt
client-key: tuto.key
```

```
# Create context in Kubernetes client.
$ kubectl config set-context tuto@kubernetes --cluster=kubernetes --
user=tuto --kubeconfig secret/tuto.conf
=> Context "tuto@kubernetes" created.

Now the tuto.config will have more updates with context:
[vagrant@sloopstash-k8s-mtr-1 kickstart-kubernetes]$ cat
secret/tuto.conf
apiVersion: v1
clusters:
- cluster:
certificate-authority: /etc/kubernetes/pki/ca.crt
server: https://192.168.201.61:6443
name: kubernetes
contexts:
- context:
cluster: kubernetes
user: tuto
name: tuto@kubernetes
current-context: ""
kind: Config
preferences: {}
users:
- name: tuto
user:
client-certificate: tuto.crt
client-key: tuto.key


# Use context in Kubernetes client => So developers will have access
to this context (cluster) not any other cliusters which is more secure
in production grade.
$ kubectl config use-context tuto@kubernetes --kubeconfig
secret/tuto.conf
=> Switched to context "tuto@kubernetes".

# View Kubernetes client configuration.
$ kubectl config view --flatten --kubeconfig secret/tuto.conf
```

## Now developer has to configure in his laptop:

```
In MAC laptop/our machine:
```

```
$ mkdir ~/.kube

$ nano ~/.kube/config
=> (copy the contents from secret/tuto.conf from master node and paste
here. kubectl config view --flatten --kubeconfig secret/tuto.conf)

➜ ~ kubectl get nodes
Error from server (Forbidden): nodes is forbidden: User "tuto" cannot
list resource "nodes" in API group "" at the cluster scope
The erros is bcz of limited access.

Now provide access to tuto on master node:
[vagrant@sloopstash-k8s-mtr-1 kickstart-kubernetes]$ kubectl apply -f
cluster-role/devops.yml
clusterrole.rbac.authorization.k8s.io/devops created

[vagrant@sloopstash-k8s-mtr-1 kickstart-kubernetes]$ kubectl apply -f
cluster-role/binding/devops.yml
clusterrolebinding.rbac.authorization.k8s.io/devops created

[vagrant@sloopstash-k8s-mtr-1 kickstart-kubernetes]$ kubectl describe
clusterrolebinding devops
Name: devops
Labels: <none>
Annotations: <none>
Role:
Kind: ClusterRole
Name: devops
Subjects:
Kind Name Namespace
---- ---- ---------
User tuto

Now on Mac kubernetes client:
➜ ~ kubectl get nodes
NAME STATUS ROLES AGE VERSION
sloopstash-k8s-mtr-1 Ready control-plane 6d20h v1.28.15
sloopstash-k8s-wkr-1 NotReady <none> 6d20h v1.28.15
```

## Kubernetes Dashboard:

Kubernetes dashboard can be used to deploy containerized applications to a Kubernetes cluster, troubleshoot your containerized applications, and manage the cluster resources. You can also use Kubernetes dashboard to get an overview of applications running on your cluster, as well as for creating or modifying individual Kubernetes resources such as deployments, jobs, daemon-sets, etc. For example, you can scale a deployment, initiate a rolling update, restart a pod or deploy new applications using a deploy wizard. Please checkout the Kubernetes dashboard documentation to know more on it.

Kubernetes dashboard can be used as monitoring solution for kubernetes cluster for top grade solutions - use observability tools like ELK, Grafana, Prometheus etc.
Not recommended for production.

```
Start the wokrer node as well. As we need one worker node to be
availble
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
up sloopstash-k8s-wkr-1
$ sudo VAGRANT_CWD=./vagrant/alma-linux-9/vmware/arm64/server vagrant
ssh sloopstash-k8s-wkr-1

on master node/control plance - Add labels to worker node as worker as
per stds:
$ kubectl label nodes sloopstash-k8s-mtr-1 type=on-premise
provider=host service=virtualbox region=local availability_zone=local-
a
=> node/sloopstash-k8s-mtr-1 labeled
$ kubectl label nodes sloopstash-k8s-wkr-1 type=on-premise
provider=host service=virtualbox region=local availability_zone=local-
b node-role.kubernetes.io/worker=worker
=> node/sloopstash-k8s-wkr-1 labeled


On master worker node.
# Switch to Kubernetes starter-kit directory.
$ cd /opt/kickstart-kubernetes

# Create Kubernetes dashboard.
$ kubectl apply -f dashboard/main.yml

# Create service-account for Kubernetes dashboard.
$ kubectl apply -f service-account/web.yml

# Create cluster-role-binding for Kubernetes dashboard.
$ kubectl apply -f cluster-role/binding/noc.yml
```

```
# Create role for Kubernetes dashboard.
$ kubectl apply -f role/noc.yml -n kubernetes-dashboard

# Create role-binding for Kubernetes dashboard.
$ kubectl apply -f role/binding/noc.yml -n kubernetes-dashboard

# Create token to access Kubernetes dashboard.
$ kubectl create token web --duration 12h -n kubernetes-dashboard
=> generates the token with 12h expiry

Run below command on Mac/host machine
# Start Kubernetes proxy service on host machine.
➜ ~ kubectl proxy --address='0.0.0.0' --port=8001 --accept-hosts='.*'
Starting to serve on [::]:8001
```

Now access the dashboard using the URL =>
http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/
Paste the token generated on control plane and paste here for login => then we should be able to login - the token might expire in 12 hours and we can regenerate again.

```
on master node check:

$ kubectl get
configmaps,deployments,replicasets,daemonsets,pods,services -o wide -n
kubernetes-dashboard
```

# DevOps Notes: Kubernetes Part 3:

## Kubernetes Metrics server:

- This is not monitoring system
- CAdvisor collects node level metrics and resources level data to metrics sever
- The metrics server sends data to Kubernetes API server (control plane) needed for autoscaling
- Not recommended for production.

## Kubernetes Resource or object:

- Orchestration in Kubernetes is orchestration of resources or objects similar to orchestration of containers in Docker.
- Access Management:
  - Docker does not provide access management.
- Storage Management:
- Network Management:
- Workload Management:

## Iac Implementation in the Kubernetes cluster:

- Kubernetes YAML similar to Docker compose YAML.
- Isolation in Docker is based on directory and in Kubernetes it is based on namespaces.

```
# Get list of resources/ API objects
$ kubectl api-resources

=> will have colum "namespaced" => true or false

So two types:
Namespaced => Ex: pods
and Non-namespaced => Ex: configmaps, ClusterRoleBinding
```

- Namespaced => True  => components all belong to specific Environment
- Namespaced => False => components all belong to cluster level => and any environment can use.
- Kubernetes API server is made of multiple APIs => Ex: RBAC, Networking, authentication, authorization, autoscaling etc.

```
[vagrant@sloopstash-k8s-mtr-1 ~]$ kubectl api-versions
admissionregistration.k8s.io/v1
apiextensions.k8s.io/v1
apiregistration.k8s.io/v1
apps/v1
authentication.k8s.io/v1
authorization.k8s.io/v1
autoscaling/v1
autoscaling/v2
batch/v1
certificates.k8s.io/v1
coordination.k8s.io/v1
discovery.k8s.io/v1
```

```
events.k8s.io/v1
flowcontrol.apiserver.k8s.io/v1beta2
flowcontrol.apiserver.k8s.io/v1beta3
networking.k8s.io/v1
node.k8s.io/v1
policy/v1
rbac.authorization.k8s.io/v1
scheduling.k8s.io/v1
storage.k8s.io/v1
v1
```

Namespaces:

```
[vagrant@sloopstash-k8s-mtr-1 ~]$ kubectl get namespaces
NAME STATUS AGE
default Active 11d
kube-flannel Active 11d
kube-node-lease Active 11d
kube-public Active 11d
kube-system Active 11d
kubernetes-dashboard Active 4d23h


=> default, public, node-lease comes by default from Kubernetes

List the resources under the specifi namespace (-n kubernetes-
dashboard)
$ kubectl get
configmaps,deployments,replicasets,daemonsets,pods,services -o wide -n
kubernetes-dashboard
=> The resources shown in the list were create using the namespace key
earlier.

if -n is not given in command => default namespace is considered
$ kubectl get
configmaps,deployments,replicasets,daemonsets,pods,services -o wide
=> Lists only the resources with default namespace.
```

Docker v/s Kubernetes resource isolation:

```
Docker Compose:
# Provision OCI containers using Docker compose.
```

```
$ sudo docker compose -f compose/crm.yml --env-file
compose/${ENVIRONMENT^^}.env -p sloopstash-${ENVIRONMENT}-crm up -d
=> where -p (project directory based on environment) argument serves
the purpose of isolation

Kubernetes:
# Create Kubernetes namespace.
$ kubectl create namespace sloopstash-${ENVIRONMENT}-crm


$ kubectl create namespace sloopstash-stg-crm
$ kubectl create namespace sloopstash-qaa-crm
$ kubectl create namespace sloopstash-qab-crm

[vagrant@sloopstash-k8s-mtr-1 ~]$ kubectl get namespaces
NAME STATUS AGE
default Active 11d
kube-flannel Active 11d
kube-node-lease Active 11d
kube-public Active 11d
kube-system Active 11d
kubernetes-dashboard Active 5d
sloopstash-qaa-crm Active 8s
sloopstash-qab-crm Active 4s
sloopstash-stg-crm Active 13s
```

## Building Kubernetes project:

https://github.com/sloopstash/kickstart-kubernetes => build it on /opt/source/Devel/SLST/kubernetes-project

- Kubernetes doc recommends camelCasing for directory structure - but normal - separated directory is ok => persistent-volume-claim

### Kubernetes resources in the context of storage:

- Data persistence in Kubernetes pods/workloads is achieved using **persistent volumes**
  - StorageClass => cluster level
  - PersistentVolume => cluster level
  - PersistentVolumeClaim => name-spaced (environment specific) => will attach the storage to the pod container (means claim)

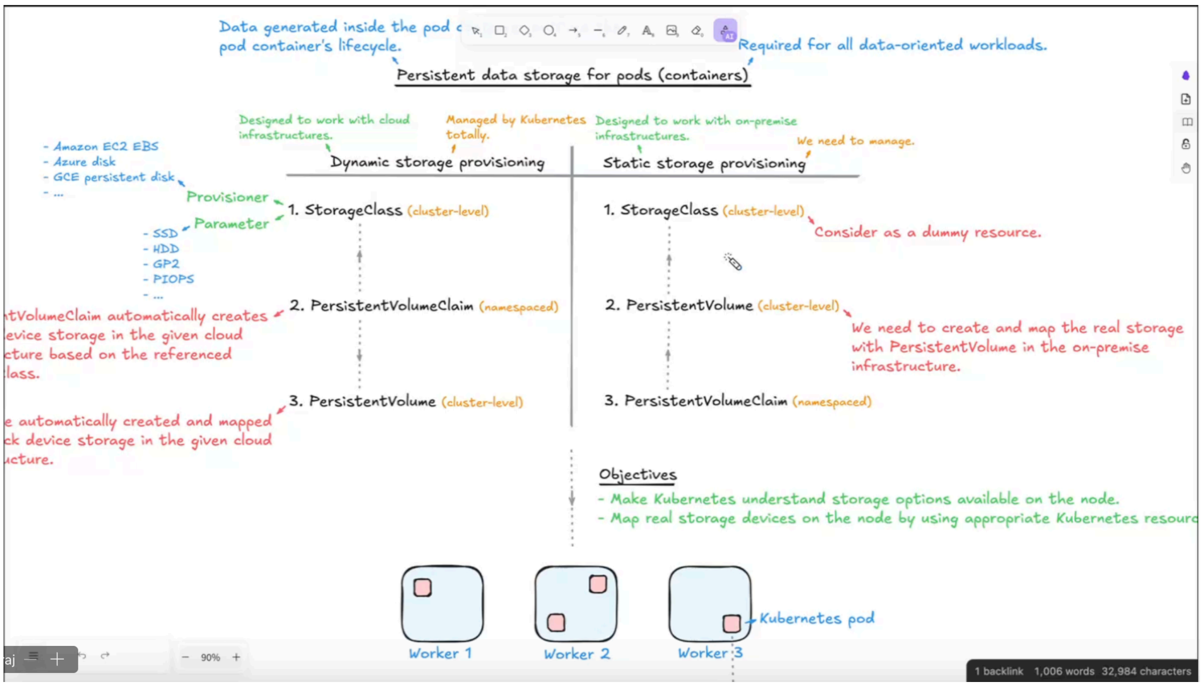
**PersistentVolume Types:**

- local => for on premise infrastructure
- NFS => n/w based file system should be avoided
- Host path => Similar to Docker bind mount storage
- Fibre channel
- ISCSI
- CSI (container storage interface) => recommended for cloud infrastructure.
  - Amazon EC2 EBS
  - Azure disk
  - GCE persistent disk

**Temporary storage + sharing**
- **Projected Volumes**
  - **ConfigMap:**
    - => Kubernetes resource => store and sync configuration files with pod containers.
  - **Secret:**
    - => Kubernetes resource => store and sync sensitive data with pod containers.
- Ephemeral Volumes: => similar to Docker once pods are down - data is gone.
  - EmtyDir
  - CSI ephimeral volume

Refer to whiteboard for Dynamic and static storage provisioning in Kubernetes:
**Dynamic Storage provisioning:**
- for cloud infrastructure
- Devops will create storageClass and provide persistenetvolumeclaim template to developer
- Developer run the template which creates persistentVolume and attaches the storage to pod container.

why dynamic storage provisioning does not wok on premise ? => bcz kubernetes is cloud native unlike docker which on premise native
**Static storage provisioning:**
- for on premise or local
- Devops will create storageclass and persistentVolume and provide claim template to developer.



Data generated inside the pod c
pod container's lifecycle.
Required for all data-oriented workloads.
Persistent data storage for pods (containers)
Designed to work with cloud infrastructures.
Managed by Kubernetes totally.
Designed to work with on-premise infrastructures.
We need to manage.
Dynamic storage provisioning
Static storage provisioning
- Amazon EC2 EBS
- Azure disk
- GCE persistent disk
- ...
Provisioner
Parameter
- SSD
- HDD
- GP2
- PIOPS
- ...
1. StorageClass (cluster-level)
2. PersistentVolumeClaim (namespaced)
3. PersistentVolume (cluster-level)
1. StorageClass (cluster-level)
Consider as a dummy resource.
2. PersistentVolume (cluster-level)
We need to create and map the real storage with PersistentVolume in the on-premise infrastructure.
3. PersistentVolumeClaim (namespaced)
tVolumeClaim automatically creates
evice storage in the given cloud
cture based on the referenced
lass.
e automatically created and mapped
ck device storage in the given cloud
ucture.
Objectives
- Make Kubernetes understand storage options available on the node.
- Map real storage devices on the node by using appropriate Kubernetes resourc
Worker 1
Worker 2
Worker 3
Kubernetes pod
90%
1 backlink 1,006 words 32,984 characters

We will use kubectl client on laptop as developer to apply the YAML and create resources in Kubernetes control plane - as we do not give ssh access for developers to control plane.
Create storage-class directory in kubernetes project - for reference check kubernetes doc https://kubernetes.io/docs/concepts/storage/storage-classes/#local (local for on premise)

```
apiVersion: storage.k8s.io/v1 => take this reference for avaialble API
versions using (kubectl api-versions)
kind: StorageClass => Kind of resource
metadata:
name: sloopstash-${ENVIRONMENT}-crm-redis-data => (identifier used by
Kubernetes for the resoucrce - these logs are created corelating with
voumes from docker compose - we need twi logs for it)
provisioner: kubernetes.io/no-provisioner # indicates that this
StorageClass does not support automatic provisioning
volumeBindingMode: WaitForFirstConsumer (create resource when POD
requests it )



Run the below commands from laptop /opt/source/Devel/SLST/kubernetes-
project directory
$ export ENVIRONMENT=stg

Redis:
```

```
$ kubernetes-project git:(main) ✗ envsubst < storage-
class/crm/redis.yml | kubectl apply -f -
storageclass.storage.k8s.io/sloopstash-stg-crm-redis-data created
storageclass.storage.k8s.io/sloopstash-stg-crm-redis-log created

Verify:
→ kubernetes-project git:(main) ✗ kubectl get storageclass
NAME PROVISIONER RECLAIMPOLICY VOLUMEBINDINGMODE ALLOWVOLUMEEXPANSION
AGE
sloopstash-stg-crm-redis-data kubernetes.io/no-provisioner Delete
WaitForFirstConsumer false
sloopstash-stg-crm-redis-log kubernetes.io/no-provisioner Delete
WaitForFirstConsumer false

Nginx:
$ kubernetes-project git:(main) ✗ envsubst < storage-
class/crm/nginx.yml | kubectl apply -f -
storageclass.storage.k8s.io/sloopstash-stg-crm-nginx-log created

App:
$ kubernetes-project git:(main) ✗ envsubst < storage-class/crm/app.yml
| kubectl apply -f -
storageclass.storage.k8s.io/sloopstash-stg-crm-app-log created

→ kubernetes-project git:(main) ✗ kubectl get storageclass
NAME PROVISIONER RECLAIMPOLICY VOLUMEBINDINGMODE ALLOWVOLUMEEXPANSION
AGE
sloopstash-stg-crm-app-log kubernetes.io/no-provisioner Delete
WaitForFirstConsumer false
sloopstash-stg-crm-nginx-log kubernetes.io/no-provisioner Delete
WaitForFirstConsumer false
sloopstash-stg-crm-redis-data kubernetes.io/no-provisioner Delete
WaitForFirstConsumer false
sloopstash-stg-crm-redis-log kubernetes.io/no-provisioner Delete
WaitForFirstConsumer false
```

Now start writing YAMl for persistentVolume

```
apiVersion: v1
kind: PersistentVolume
metadata:
name: sloopstash-${ENVIRONMENT}-crm-redis-0-data (0 - is used bcz at
cluster level we can have multiple redis containers)
spec:
```

```
storageClassName: sloopstash-${ENVIRONMENT}-crm-redis-data (this is
the name of storage class created for redis above)


nodeAffinity => scheduling specific resource to specific node


→ kubernetes-project git:(main) ✗ kubectl describe node sloopstash-
k8s-wkr-1

Name: sloopstash-k8s-wkr-1
Roles: worker
Labels: availability_zone=local-b
beta.kubernetes.io/arch=arm64
beta.kubernetes.io/os=linux
kubernetes.io/arch=arm64
kubernetes.io/hostname=sloopstash-k8s-wkr-1
kubernetes.io/os=linux => (this will be selected in nodeAffinity in
template)


run the persistent volume for redis:
→ kubernetes-project git:(main) ✗ envsubst < persistent-
volume/crm/redis.yml | kubectl apply -f -
persistentvolume/sloopstash-stg-crm-redis-0-data created
persistentvolume/sloopstash-stg-crm-redis-0-log created

verify if created:
→ kubernetes-project git:(main) ✗ kubectl get
persistentvolumes,storageclasses -o wide
NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS
REASON AGE VOLUMEMODE
persistentvolume/sloopstash-stg-crm-redis-0-data 5Gi RWO Retain
Available sloopstash-stg-crm-redis-data 78s Filesystem
persistentvolume/sloopstash-stg-crm-redis-0-log 2Gi RWO Retain
Available sloopstash-stg-crm-redis-log 78s Filesystem

NAME PROVISIONER RECLAIMPOLICY VOLUMEBINDINGMODE ALLOWVOLUMEEXPANSION
AGE
storageclass.storage.k8s.io/sloopstash-stg-crm-app-log
kubernetes.io/no-provisioner Delete WaitForFirstConsumer false 34m
storageclass.storage.k8s.io/sloopstash-stg-crm-nginx-log
kubernetes.io/no-provisioner Delete WaitForFirstConsumer false 36m
```

```
storageclass.storage.k8s.io/sloopstash-stg-crm-redis-data
kubernetes.io/no-provisioner Delete WaitForFirstConsumer false 41m
storageclass.storage.k8s.io/sloopstash-stg-crm-redis-log
kubernetes.io/no-provisioner Delete WaitForFirstConsumer false 41m

Run the persistent volume templates for nginx and app as well.

➜ kubernetes-project git:(main) ✗ envsubst < persistent-
volume/crm/app.yml | kubectl apply -f -
persistentvolume/sloopstash-stg-crm-app-log created

➜ kubernetes-project git:(main) ✗ envsubst < persistent-
volume/crm/nginx.yml | kubectl apply -f -
persistentvolume/sloopstash-stg-crm-nginx-log created
```

Now create Persistent volume claim:
**If we observe - redis.yml is not created inside persistent volume claim dir - bcz redis is to be run as stateful resource whereas app.yml and nginx.yml is run as deployment workloads**

```
Go to /opt/source/Devel/SLST/kubernetes-project

Apply role binding devops role for tuto user

➜ kubernetes-project git:(main) ✗ envsubst < persistent-volume-
claim/crm/app.yml | kubectl apply -f - -n sloopstash-${ENVIRONMENT}-
crm

➜ kubernetes-project git:(main) ✗ envsubst < persistent-volume-
claim/crm/nginx.yml | kubectl apply -f - -n sloopstash-${ENVIRONMENT}-
crm


➜ kubernetes-project git:(main) ✗ kubectl get sc,pv,pvc -n
sloopstash-${ENVIRONMENT}-crm
```