**Live Session III**

Tips/tricks, commands…

Midnight Commander

Midnight Commander is a powerful text-based file manager that provides a dual-pane interface, making it easy to navigate your filesystem, copy/move files, and perform various file operations.

To install MC (Midnight Commander) in Ubuntu, you can use the following command in your terminal:

**sudo apt update**

**sudo apt install mc**

After installation, you can launch Midnight Commander by simply typing:

**mc**

[ Tips/tricks, commands… ]

Grep Recursively Without a Pipe

Very often, we will use find's **exec** option to grep for something. This is a basic way of emulating a "find in files" task:

find . -exec grep 'hello' {} \;

But this command is lengthy, this command is much more convenient and easier to learn:

**grep -r 'hello'**

Switch Between Two Directories Instantly

Each time you run "**cd -**" it will flip between the previous two directories you navigated to, making it very fast and easy to move back and forth.

"cd -" will print the path of the directory it changes to, even if you have the current directory in your prompt.

Copy Directories With Maximum Fidelity

cp -R docs backup-docs

This command lets you copy entire directory structures, producing a full copy of a directory and all its contents, including files and other directories. It's a great way of backing up files or creating a copy of a larger project to work on.

However, you may notice that the copies aren't exactly the same as the originals. Their modification times, owners, and permissions may all be different.

These times only differ by a minute, but the problem can be much greater. The fix is simple: use **-a** (for "archive") instead of -R:

**cp -a docs backup-docs**

[ bandwith testing … ]

Real Time Bandwith manager!

sudo apt-get update

sudo apt-get -y install vnstat

sudo vnstat -l -i wlan0 (ex. listen in on wlan0 socket for ex.)

nmcli connection show (to check socket names if nec.)

Surf the web in browser and watch for bit transfer rates

**Ctrl^C** when complete for summary

[ docker ]

Docker (for httpd/php/mysql services, etc.)

Docker simplifies the process of application deployment by packaging software and its dependencies into containers. These containers are lightweight, portable, and can run consistently across *different* computing environments, **ensuring** that applications behave the same way regardless of where they are deployed. This capability is particularly valuable in modern software development, where applications are often distributed across various platforms and infrastructures.

Containers vs. Virtual Machines

Containers are often compared to virtual machines (VMs), but they operate differently. While VMs virtualize the entire hardware stack, containers share the host operating system's kernel and utilize its resources more efficiently. This results in faster startup times and lower overhead, allowing multiple containers to run on a single host without the resource constraints typically associated with VMs

Components of Docker

Docker consists of several key components:

* **Docker Engine**: The core service that runs and manages containers.
* **Docker Images**: Read-only templates used to create containers. They include everything needed to run an application, such as code, libraries, and environment variables.
* **Docker Containers**: Instances of Docker images that run the application in an isolated environment.
* [**Docker Hub**](https://hub.docker.com/): A cloud repository where users can share and access Docker images

Benefits of Using Docker

1. **Portability**: Docker containers can run on any system that supports Docker, making it easy to move applications between development, testing, and production environments.
2. **Efficiency**: Containers share the host OS kernel, reducing resource consumption compared to traditional VMs. This allows for higher density of applications on a single host.
3. **Speed**: Containers can be started and stopped quickly, facilitating rapid deployment and scaling of applications.
4. **Isolation**: Each container runs in its own environment, which enhances security and stability by preventing applications from interfering with one another

Let’s do it!

Install docker

# Update the apt package index and install packages to allow apt to use a repository over HTTPS:

**sudo apt-get update**

**sudo apt-get install -y apt-transport-https ca-certificates curl gnupg lsb-release**

Add Docker’s official GPG key:

**curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg**

Set up the stable repository

**echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null**

Update the apt package index, and install the latest version of Docker Engine and containerd

**sudo apt-get update**

**sudo apt-get install -y docker-ce docker-ce-cli containerd.io**

Start Docker and enable it to run at boot:

**sudo systemctl start docker**

**sudo systemctl enable docker**

Create a Docker Container with httpd, PHP, and MySQL

You can create a Docker Compose file to define your services. Create a file named   
docker-compose.yml:

Open editor to compose .yml file

Create file and enter script info as follows:

**sudo nano docker-compose.yml**

version: '3.8'

services:

web:

image: nginx:latest

ports:

- "80:80"

db:

image: mysql:latest

environment:

MYSQL\_ROOT\_PASSWORD: rootpassword

MYSQL\_DATABASE: mydatabase

MYSQL\_USER: user

MYSQL\_PASSWORD: userpassword

volumes:

- db\_data:/var/lib/mysql

volumes:

db\_data:

**\*Note port 80 above installs webpages that will run on the default listed 80 port. Can be also another port like 8080:80**

Create a Simple PHP Page

Create a directory named **html**in your current working directory and add a file named index.php with the following content:

**mkdir html**

**cd html**

**sudo nano index.php**

Add the following contents

<?php

phpinfo();

?>

Save contents and exit editor.

Run Docker Compose

Run the following commands to start the containers:

First download docker compose as follows

**sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose**

**sudo chmod +x /usr/local/bin/docker-compose**

Second apply permissions to docker-compose binary

**sudo chmod +x /usr/local/bin/docker-compose**

Then execute the following command

**sudo docker-compose up -d**

**View web page in Firefox, type**

[**http://localhost:8080**](http://localhost:8080)

or do a curl request command

**curl localhost:8080**

Vagrant setup to allow for VMs to communicate with each other via IP addresses, especially for Vault-server creation.

**VM Networking in Vagrant**

When using Vagrant, the VMs can communicate with each other in several ways:

**Private Network**: This is the most common approach for VM-to-VM communication.

Vagrant.configure("2") do |config|

# Vault Server VM

config.vm.define "vault-server" do |server|

server.vm.box = "ubuntu/focal64"

server.vm.hostname = "vault-server"

server.vm.network "private\_network", ip: "192.168.56.10"

end

# Client VM

config.vm.define "client-vm" do |client|

client.vm.box = "ubuntu/focal64"

client.vm.hostname = "client-vm"

client.vm.network "private\_network", ip: "192.168.56.11"

end

end

**Public Network**: If you need the VMs to be accessible on your local network.

For the Vault server

server.vm.network "public\_network", ip: "192.168.1.10"

For the client

client.vm.network "public\_network", ip: "192.168.1.11"

**Vault Configuration Updates**

When setting up Vault in this environment, make these adjustments:

1. In the Vault server configuration, update the api\_addr to use the fixed IP:

api\_addr = "http://192.168.56.10:8200" # Use your server's IP

1. On the client VM, set the environment variable to point to the server's IP:

export VAULT\_ADDR='http://192.168.56.10:8200'

**Testing Connectivity**

To ensure your VMs can communicate:

1. From each VM, ping the other:

ping 192.168.56.10 # From client to server

ping 192.168.56.11 # From server to client

1. Test TCP connectivity to the Vault port:

From client VM

nc -zv 192.168.56.10 8200

**Troubleshooting**

If connectivity fails:

1. Check firewall settings on both VMs:

sudo ufw status

If enabled, allow the Vault port

sudo ufw allow 8200/tcp

2. Verify network settings in Vagrant:

vagrant ssh vault-server -c "ip addr show"

vagrant ssh client-vm -c "ip addr show"

These configurations will ensure your Vault server and client VMs can properly communicate using their IP addresses in a Vagrant environment.

Simple Vault setup on Ubuntu Server example including how to set up authentication between VMs.

**Vault Server Setup on Ubuntu**

**1. Install Vault**

Add HashiCorp GPG key

wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

Add HashiCorp repository

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

Update and install

sudo apt update

sudo apt install vault

**2. Configure Vault Server**

Create Vault configuration directory

sudo mkdir -p /etc/vault.d

sudo chmod 750 /etc/vault.d

Create configuration file

sudo tee /etc/vault.d/vault.hcl > /dev/null << EOF

storage "file" {

path = "/opt/vault/data"

}

listener "tcp" {

address = "0.0.0.0:8200"

tls\_disable = 1 # For production, enable TLS

}

api\_addr = "http://127.0.0.1:8200"

ui = true

EOF

Create data directory

sudo mkdir -p /opt/vault/data

sudo chown -R vault:vault /opt/vault

**3. Start Vault Server**

Start the Vault server

sudo systemctl enable vault

sudo systemctl start vault

Check status

sudo systemctl status vault

**4. Initialize Vault**

Set environment variable

export VAULT\_ADDR='http://127.0.0.1:8200'

Initialize Vault

vault operator init -key-shares=3 -key-threshold=2

This will output initialization keys and a root token. SAVE THESE SECURELY - you'll need them to unseal the vault and authenticate as root.

**5. Unseal Vault**

You need to run this command with different unseal keys (at least 2 in this setup):

vault operator unseal [unseal-key-1]

vault operator unseal [unseal-key-2]

**6. Login as Root**

vault login [root-token]

**7. Set Up AppRole Authentication for VM-to-VM Authentication**

Enable AppRole auth method

vault auth enable approle

Create a policy for the client VM

vault policy write ubuntu-client - << EOF

path "secret/data/ubuntu/\*" {

capabilities = ["read"]

}

EOF

Create a role for the client VM

vault write auth/approle/role/ubuntu-client \

token\_policies="ubuntu-client" \

token\_ttl=1h \

token\_max\_ttl=4h

Get the RoleID

vault read auth/approle/role/ubuntu-client/role-id

# Save the role\_id output

# Generate a SecretID

vault write -f auth/approle/role/ubuntu-client/secret-id

# Save the secret\_id output

**8. Store a Sample Secret**

Enable the KV secrets engine v2

vault secrets enable -version=2 secret

Store a sample secret

vault kv put secret/ubuntu/credentials \

username="ubuntu-user" \

password="StrongP@ssw0rd123"

**9. On the Client Ubuntu VM**

Install Vault client:

Add HashiCorp GPG key and repository (same as above)

wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update

sudo apt install vault

**10. Authenticate from Client VM to Vault Server**

Set environment variable to point to your Vault server

export VAULT\_ADDR='http://your-vault-server-ip:8200'

Login using AppRole

vault write auth/approle/login \

role\_id="YOUR\_ROLE\_ID" \

secret\_id="YOUR\_SECRET\_ID"

**11. Retrieve the Secret**

# Use the token from the login response

export VAULT\_TOKEN="YOUR\_TOKEN\_FROM\_LOGIN"

Read the secret

vault kv get secret/ubuntu/credentials

This setup creates a simple Vault server, initializes it, and configures AppRole authentication for secure VM-to-VM communication. The sample secret demonstrates how to store and retrieve credentials securely.

-For production use, you should enable TLS, implement proper backup strategies, and consider high-availability configurations.

Create a Database and Table in MySQL

You can access the MySQL container and create a database and table. Use the following commands:

Run status and notate container id number for MySQL container

**sudo docker ps**

# Access the MySQL container

**sudo docker exec -it <your\_container\_id\_of\_db> mysql -u root -p**

Enter **root password** per your .yml file you created above.

# Inside the MySQL shell, execute the following:

**CREATE DATABASE testdb;**

**USE testdb;**

**CREATE TABLE users (id INT AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(255), email VARCHAR(255));**

**INSERT INTO users (name, email) VALUES ('John Doe', 'john@example.com'), ('Jane Doe', 'jane@example.com');**

SELECT \* FROM users;

Refs

<https://www.zdnet.com/article/kde-neon-shows-that-the-plasma-6-linux-distro-is-something-truly-special/>

<https://www.bing.com/videos/riverview/relatedvideo?q=kde%20Desktop%20Cube&mid=4A279441455BF03A49D04A279441455BF03A49D0&ajaxhist=0>

<https://www.bing.com/videos/riverview/relatedvideo?q=kde%20Desktop%20Cube&mid=4F9520E7EC29DE6B05A64F9520E7EC29DE6B05A6&ajaxhist=0>

<https://www.bing.com/videos/riverview/relatedvideo?q=kde%20Desktop%20Cube&mid=3ED909A7878B6BB8AC8F3ED909A7878B6BB8AC8F&ajaxhist=0>

Alt distros

NIXOS

<https://www.zdnet.com/article/sparky-linux-is-a-blazing-fast-distro-that-can-keep-your-older-machines-running-for-years/>

<https://www.zdnet.com/article/how-to-install-ubuntu-linux/>

Kali Purple