

VKIST DGX-A100 DOCUMENTATION GUIDE

Đào Việt Anh, Researcher - IT Division of VKIST

1. Mục đích của tài liệu

Tài liệu kỹ thuật DGX-A100 gồm các mục đích chính sau:

- Tài liệu này cung cấp các kỹ thuật cơ bản để vận hành, sử dụng hệ thống máy chủ DGX-A100, NAS và cấu hình hệ thống phần mềm quản lý tài nguyên Kubernetes, Jupyterhub.
- Cung cấp một số khái niệm, định nghĩa cơ bản về hệ thống DGX-A100 và phần mềm quản lý tài nguyên.
- Đưa ra một số vấn đề kỹ thuật và cách khắc phục.
- Cơ sở cho việc nghiên cứu và phát triển hệ thống.

2. Các điểm chính

Tài liệu kỹ thuật được chia thành 2 phần chính, phần 1 là vận hành hệ thống DGX-A100 và hệ thống NAS, phần 2 là cấu hình cài đặt hệ thống phần mềm quản lý tài nguyên.

- Vận hành hệ thống DGX-A100
 - Vận hành thủ công
 - Vận hành thông qua phần mềm BMC
- Cài đặt và cấu hình hệ thống phần mềm quản lý tài nguyên
 - Quản lý tài nguyên GPU thông qua dòng lệnh Terminal
 - Hướng dẫn cài đặt và cấu hình Kubernetes
 - Hướng dẫn cài đặt và cấu hình Jupyterhub

3. Giới thiệu về DGX-A100



Hệ thống NVIDIA DGX™ A100 là một hệ thống đa năng, đa mục đích dành cho tất cả các cơ sở hạ tầng và công việc AI từ quá trình phân tích đến quá trình triển khai. Hệ thống được xây dựng trên 8 GPU NVIDIA A100 lõi Tensor.

Hiện tại VKIST đang có 1 máy chủ DGX-A100, 2 máy NAS với dung lượng mỗi máy là 60Tb, hệ lưu trữ và xử lý ảnh 240Tb, hệ thống đường truyền mạng, switch với băng thông 100mb/s đặt trên Hòa Lạc. Máy chủ DGX-A100 đã được cài đặt hệ thống máy ảo và cụm k8s sẵn sàng đưa vào cung cấp và vận hành Notebook thông qua Jupyterhub phục vụ nghiên cứu.

1.1 Tổng quan phần cứng

Có 2 loại hệ thống NVIDIA DGX™ A100, một loại là hệ thống NVIDIA DGX A100 640GB và một loại là hệ thống NVIDIA DGX A100 320GB. VKIST sử dụng hệ thống NVIDIA DGX A100 320GB.

Thành phần	NVIDIA DGX A100 320GB
GPU	8 NVIDIA A100 GPUs NVLinks thế hệ thứ 3
Tổng số bộ nhớ GPU	320GB (40GB 1 GPU)
NVIDIA NVSwitch	6 NVSwitch thế hệ thứ 2 nhanh gấp đôi thế hệ thứ 1
Mạng	9 cổng kết nối Mellanox ConnectX-6 VPI HDR IB/200 Gb/s
CPU	2 chip AMD Rome, Tổng 128 lõi
Bộ nhớ	1 TB
Lưu trữ	15 TB. 2 ổ cứng NVMe
Nguồn điện	200-240 volts AC 6.5 kW max. 3000 W @ 200-240 V, 16 A, 50-60 Hz

Thông số kỹ thuật

Thông số	Mô tả
Chiều cao	264 mm
Chiều rộng	428.3 mm
Chiều dài	897.1 mm
Trọng lượng	123.16 kg
Loại kiểu dáng	6U Rackmount

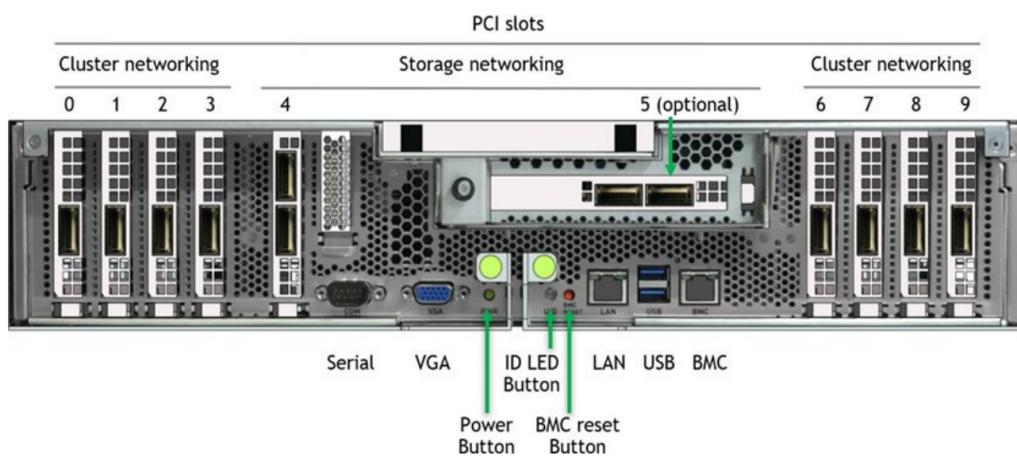
1.2 Sơ đồ mặt trước



Mặt trước của DGX-A100 bao gồm một tấm khung ngoài màu vàng ánh kim với nhiều lỗ không khí nhỏ làm nhiệm vụ thông khí và làm mát. Khung ngoài của DGX-A100 có khả năng tháo rời để kiểm tra các phần cứng bên trong. Bên ngoài khung là 4 lỗ vít để cố định DGX-A100 vào giá và một số nút chức năng khác. Chức năng của các nút trên mặt trước của DGX-A100 bao gồm:

Thành phần	Mô tả
Nút nguồn	Tắt bật hệ thống. Nháy xanh (1 Hz): Ché đô chờ (khởi động BMC); Nháy xanh (4 Hz): POST; Đèn xanh: Khởi động
Nút ID	Khởi động định danh. Nhấn và giữ đèn khi đèn xanh nuga点亮 hoặc nháy
Fault LED	Hệ thống lỗi

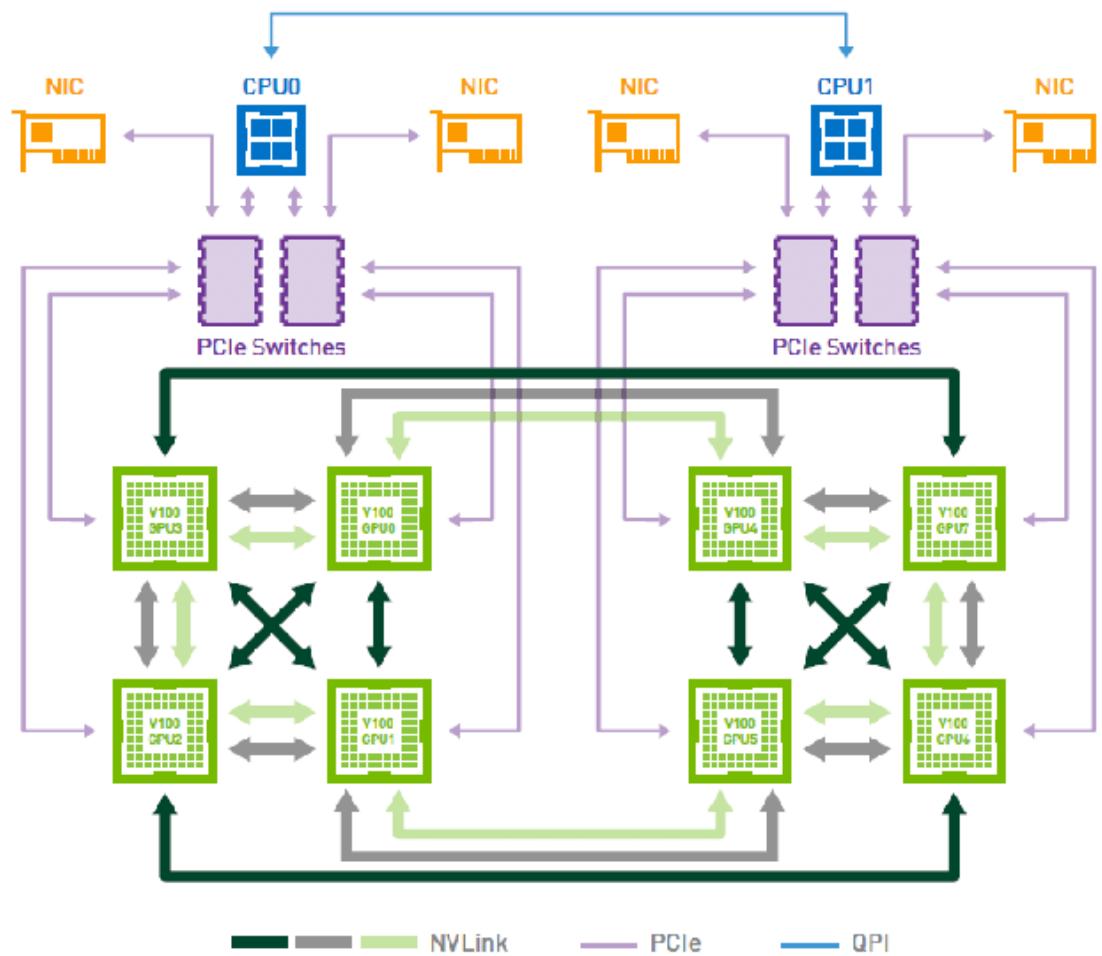
1.3 Sơ đồ cổng kết nối mặt sau



Mặt sau của DGX-A100 chứa hầu hết các cổng kết nối đến hệ thống, gồm các cổng mạng, cổng màn hình, cổng truyền dữ liệu fabric, cổng USB. Danh sách các cổng chức năng ở bảng dưới.

Thành phần	Mô tả
Nút Nguồn	Tắt bật hệ thống giống mặt trước
ID LED	Giống ID LED mặt trước
Nút BMC reset	Khởi động thủ công lại trình điều khiển BMC
Cổng từ 0 - 9	Cổng kết nối mạng
VGA	Cổng màn hình dùng để hiển thị DGX OS
LAN	Cổng kết nối LAN, tương ứng với cổng enp226s0 trong OS
BMC	Cổng kết nối đến trình điều khiển BMC

1.4 Sơ đồ hệ thống DGX A100



Hệ thống DGX-A100 bao gồm 8 GPU được kết nối thông qua các Nvswitch, các Nvswitch làm nhiệm vụ truyền dữ liệu trực tiếp giữa các GPU do đó các GPU có thể thực hiện Paralell, Distributed training. Ngoài ra mỗi NVSwitch có các cổng NIC kết nối đến các hệ DGX-A100 khác với tốc độ cao.

Thành phần	Số lượng
GPU	8
CPU	2
NVSwitches	6

1.5 Hệ điều hành

Hệ điều hành của hệ thống DGX A100 là thành phần không thể thiếu để quản lý và tận dụng hệ thống DGX-A100. Hệ điều hành của DGX A100 được build bởi đội ngũ của NVIDIA và được cứng hóa các phần mềm trong đó. Theo mặc định, hệ điều hành được sử dụng là Ubuntu server 20.04. Cài đặt các hệ điều hành khác có thể dẫn đến việc DGX-A100 không hoạt động hoặc hoạt động thiêu hiệu quả.

Hệ thống DGX A100 được cài đặt sẵn các phần mềm sau:

- Hệ điều hành Ubuntu 20.04
- Các phần mềm quản lý và theo dõi hệ thống:
 - NVIDIA System Management (NVSM): Cung cấp các thông tin về trạng thái và cảnh báo hệ thống NVIDIA DGX. Cung cấp các lệnh cơ bản để kiểm tra trạng thái hệ thống.
 - Data Center GPU Management (DCGM)
- Các gói hỗ trợ cho hệ thống DGX A100:
 - Driver cho NVIDIA GPU
 - Docker Engine
 - NVIDIA Container Toolkit
 - Mellanox OpenFabrics Enterprise Distribution for Linux (MOFED)
 - Mellanox Software Tools (MST)
 - cachefilesd (quản lý bộ nhớ cache)

2. Vận hành DGX A100

Tài liệu này hướng dẫn các bước khởi động, kết nối ban đầu để tiến hành sử dụng DGX-A100. Về cơ bản có 2 phương pháp để vận hành hệ thống là thủ công và thông qua phần mềm.

- Vận hành thủ công thông qua việc thao tác trực tiếp trên máy chủ DGX-A100. Cách khởi động và kết nối mạng cho máy chủ DGX-A100, cách boot lại hệ điều hành.
- Vận hành và quản lý thông qua phần mềm BMC, SSH. Phần mềm BMC - **Board Management Controller** được cài đặt mặc định bởi NVIDIA cho máy chủ DGX-A100, nó cung cấp đầy đủ các công cụ để điều khiển và quản lý DGX-A100 từ xa thông qua internet bao gồm cả việc tắt bật hệ thống.
-

2.1 Kết nối trực tiếp

Kết nối trực tiếp DGX A100 thông qua cổng VGA ở trước hoặc sau của hệ thống. Bàn phím và chuột có thể kết nối trực tiếp thông qua bất kỳ cổng USB nào trên thân máy.

Mặt trước



Mặt sau



Sau khi kết nối trực tiếp vào DGX-A100 khởi động bằng cách bấm vào nút nguồn của DGX-A100. Giao diện hiển thị trên màn hình là giao diện dạng dòng lệnh, đăng nhập bằng tài khoản quản trị của DGX-OS và trực tiếp điều khiển thông qua dòng lệnh.

2.2 Kết nối từ xa thông qua BMC

BMC là trình điều khiển DGX A100 từ xa, nằm trong cùng hệ thống với DGX-A100 nhưng tách biệt so với DGX-A100. Trình điều khiển BMC có IP riêng và tách biệt trong mạng LAN.

2.2.1 Cấu hình IP tĩnh cho BMC

Để cấu hình IP tĩnh cho BMC, tiến hành sử dụng lệnh ipmitool trong DGX OS.

Để xem cấu hình hiện tại, nhập lệnh:

```
$ sudo ipmitool lan print 1
```

Để cấu hình IP tĩnh trong LAN cho BMC, thực hiện các thao tác dưới.

1. Chuyển IP source về tĩnh.

```
$ sudo ipmitool lan set 1 ipsrc static
```

2. Cấu hình thông tin địa chỉ IP.

- Để cài địa chỉ IP, nhập lệnh dưới và thay <my-ip-address> bởi IP tĩnh trong dải mạng LAN

```
$ sudo ipmitool lan set 1 ipaddr <my-ip-address>
```

- Để cài Subnet Mask, nhập lệnh dưới và thay <my-netmask-address> bởi Netmask của mạng LAN

```
$ sudo ipmitool lan set 1 netmask <my-netmask-address>
```

- Để cài Default Gateway, nhập lệnh dưới và thay <my-default-gateway> bởi Default Gateway của mạng LAN

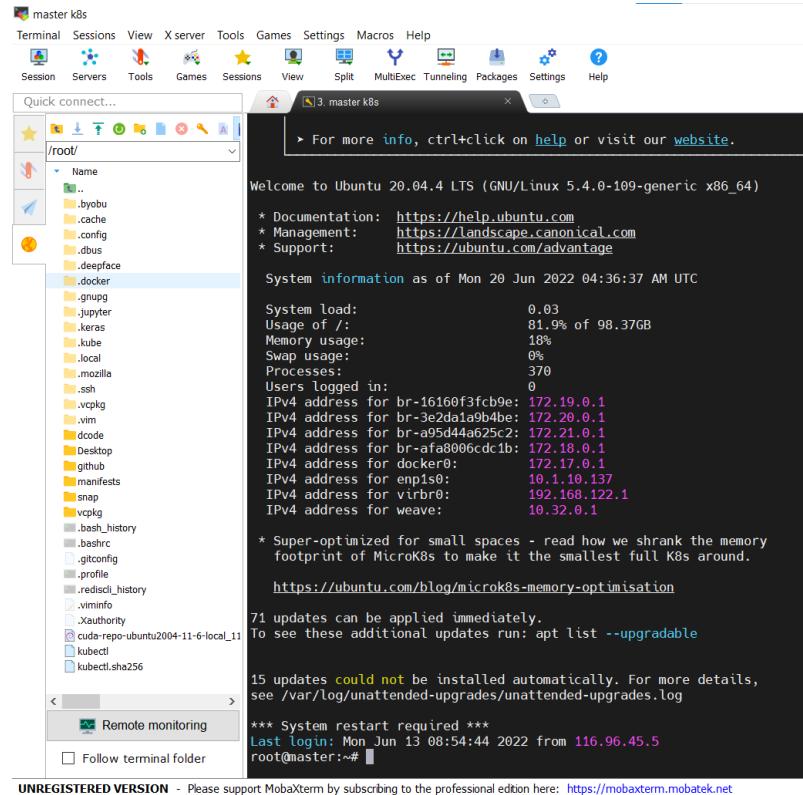
```
$ sudo ipmitool lan set 1 defgw ipaddr <my-default-gateway>
```

2.2.2 Sử dụng BMC để điều khiển DGX A100

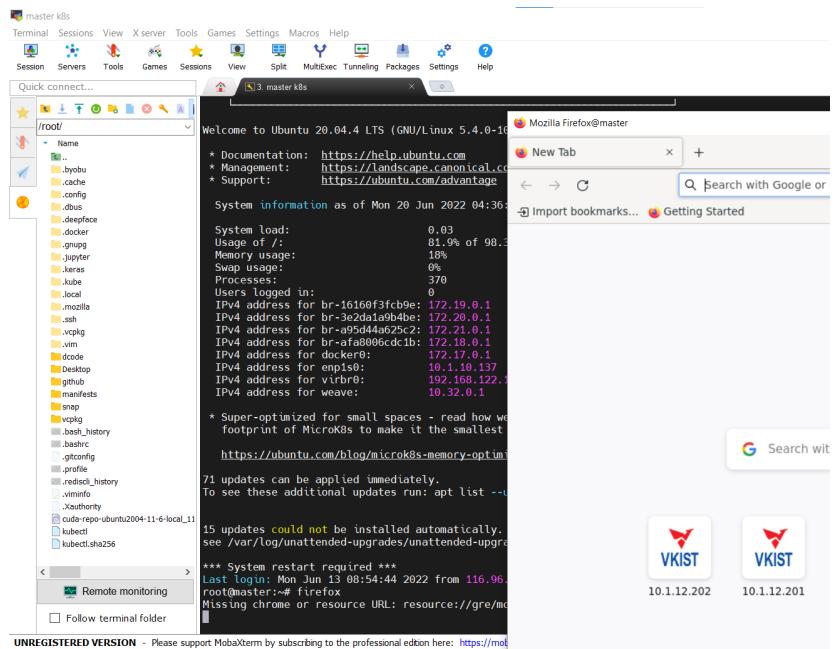
Đối với các máy trong LAN, để truy cập tiến hành gõ lệnh <https://<bmc-address>> vào trình duyệt và thay thế <bmc-address> bởi địa chỉ IP tĩnh của BMC.

Đối với các máy ngoài LAN, để truy cập cần ssh đến 1 máy trong LAN thông qua MobaXterm và gõ lệnh firefox để tiến hành remote trình duyệt trong LAN ra máy cá

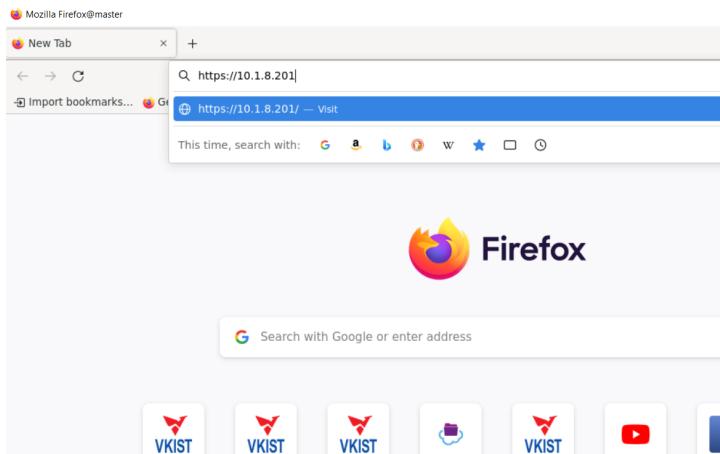
nhân sau đó truy cập giống như các máy trong LAN thông qua giao diện firefox đã được forward sang.



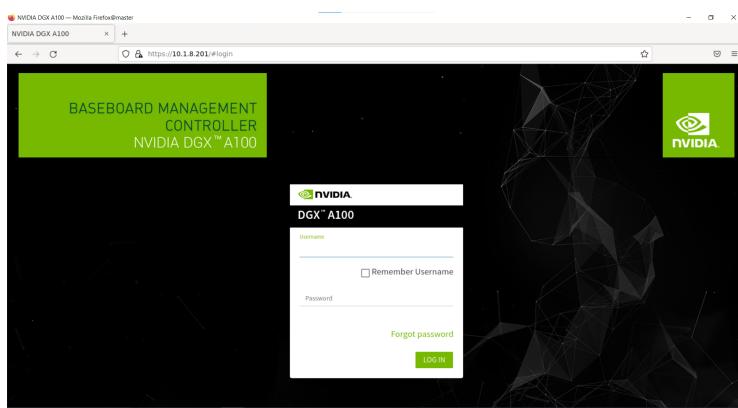
Kết nối đến một máy trong LAN thông qua ssh



Forward firefox máy trong LAN đến máy cá nhân thông qua lệnh firefox



Tại màn hình firefox gõ địa chỉ của BMC vào thanh tìm kiếm

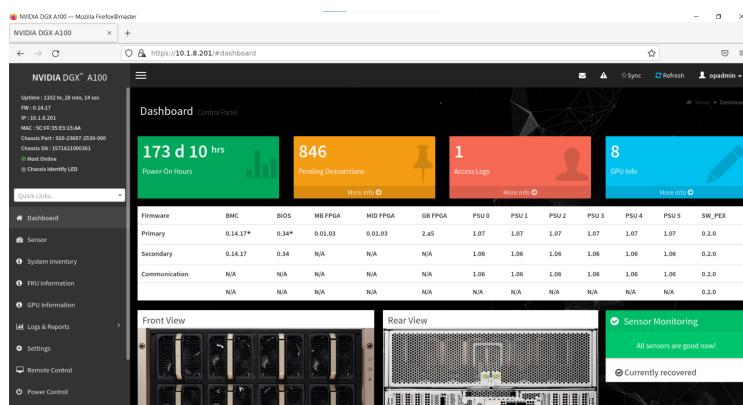


Màn hình đăng nhập của BMC

Hiện tại BMC đã được cấu hình sẵn tên đăng nhập và mật khẩu. Tên đăng nhập của BMC giống với tên đăng nhập của quản trị viên trong DGX-A100.

Username: <administrator-username>

Password: <bmc-password>

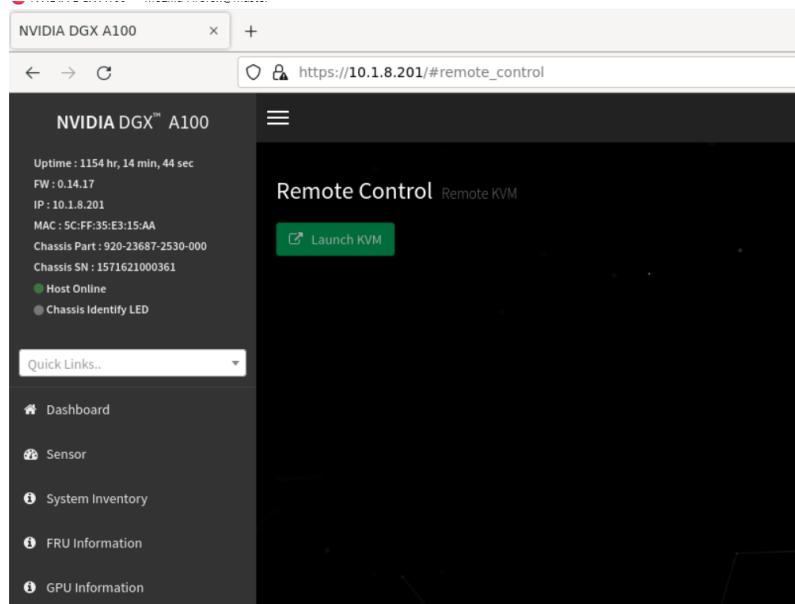


Giao diện chính của BMC

Hành động	Mô tả
Quick Links	Lối tắt đến các chức năng của BMC
Dashboard	Hiển thị thông tin tổng quan về trạng thái của thiết bị
Sensor	Cung cấp và đọc trạng thái các cảm biến của hệ thống, Ví dụ SSD, PSUs, Nguồn, nhiệt độ CPU, nhiệt độ DIMM, tốc độ quạt.
System Inventory	Thông tin sản xuất của thiết bị
FRU Information	PCIE, System, Processor, Storage, ...
Thông tin GPU	Cung cấp thông tin cho toàn bộ GPU trong hệ thống bao gồm GUID, phiên bản VBIOS, phiên bản InfoROM
Logs and Reports	Hiển thị, xóa IPMI event log, Hệ thống, Audit, Video, và POST Code logs.
Settings	Cấu hình các thông tin sau: Bản ghi BSOD, Dịch vụ người dùng, cài đặt cho chuột của KVM, cài đặt log, cài đặt chuyển hướng đa phương tiện, cài đặt mạng, cài đặt PAM Order, Bộ lọc sự kiện, dịch vụ, cài đặt SMTP, ...
Remote Control	Mở KVM để kết nối đến console của DGX A100 từ xa
Power Control	Thực hiện các hành động sau: Tắt nguồn, bật nguồn, tiết kiệm điện, reset cứng.
Chassis ID LED Control	Cho phép thay đổi trạng thái của chassis ID LED: Tắt, Bật, Nháy sáng (giá trị từ 5 - 255 lần 1 giây)
Maintenance	Thực hiện các hành động sau : Cấu hình Backup, cài đặt vị trí của Firmware Image, cập nhật Firmware, Khôi phục cài đặt gốc, Khôi phục mặc định nhà sản xuất, Quản trị hệ thống
Sign out	Đăng xuất khỏi giao diện web của BMC

Để điều khiển từ xa hệ thống DGX-A100 sử dụng BMC, thực hiện các bước sau:

1. Tại giao diện của BMC, Nhấp chuột vào mục Remote Control từ thanh điều hướng phía bên trái màn hình.
2. Nhấp chuột vào Launch KVM để khởi động KVM và truy cập vào trình điều khiển DGX A100.



3. Màn hình giao diện trình điều khiển DGX-A100 hiện ra và có thể trực tiếp điều khiển dưới dạng dòng lệnh.

3. Quản lý tài nguyên GPU

GPU của DGX-A100 là GPU A100, mỗi GPU bao gồm 40G ram. NVidia A100 TENSOR CORE GPU là một thiết bị điện toán đặc biệt. Không chỉ dành riêng cho các công việc Machine Learning / AI mà A100. Nó còn có thể đáp ứng cho các tác vụ tính toán khoa học đòi hỏi hiệu suất tính toán đại số tuyến tính hiệu năng cao. Các dòng GPU GeForce RTX 30xx và Quadro RTX Ax000 cũng rất tốt cho tính toán số học – Các phép tính không yêu cầu độ chính xác dấu phẩy động 64bit – tức là FP64 (floating point 64). Bên cạnh đó, A100 lại rất xuất sắc ở những công việc như thế bên cạnh khả năng tính toán số học với độ chính xác cao nhờ khả năng tăng tốc xử lý tính toán bằng GPU. Một đặc điểm của GPU A100 là chúng cho phép phân GPU thành các instance nhỏ hơn gọi là MIG, mỗi MIG là một tổ hợp của các nhân tính toán và nhân bộ nhớ của GPU.

3.1 Quản lý tài nguyên GPU thông qua lệnh

Để xem thông tin GPU và các tiến trình sử dụng bộ nhớ của GPU, chạy lệnh:

\$nvidia-smi

```
vkist@ubuntu-img:~$ nvidia-smi
Mon Jun 27 03:34:00 2022
+-----+
| NVIDIA-SMI 470.129.06    Driver Version: 470.129.06    CUDA Version: 11.4 |
+-----+
| GPU  Name      Persistence-M | Bus-Id      Disp.A  | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M. |
|                               |             |            MIG M. |
+-----+
| 0  NVIDIA A100-SXM... Off   | 00000000:04:00.0 Off |           On |
| N/A   31C     P0    43W / 400W |             0MiB / 40536MiB |       N/A  Default |
|                               |             |             Enabled |
+-----+
| 1  NVIDIA A100-SXM... Off   | 00000000:05:00.0 Off |           On |
| N/A   31C     P0    44W / 400W |             0MiB / 40536MiB |       N/A  Default |
|                               |             |             Enabled |
+-----+
| 2  NVIDIA A100-SXM... Off   | 00000000:06:00.0 Off |           On |
| N/A   32C     P0    49W / 400W |             20MiB / 40536MiB |       N/A  Default |
|                               |             |             Enabled |
+-----+
| 3  NVIDIA A100-SXM... Off   | 00000000:07:00.0 Off |           On |
| N/A   32C     P0    44W / 400W |             20MiB / 40536MiB |       N/A  Default |
|                               |             |             Enabled |
+-----+
| 4  NVIDIA A100-SXM... Off   | 00000000:08:00.0 Off |           On |
| N/A   34C     P0    51W / 400W |             3574MiB / 40536MiB |       N/A  Default |
|                               |             |             Enabled |
+-----+
```

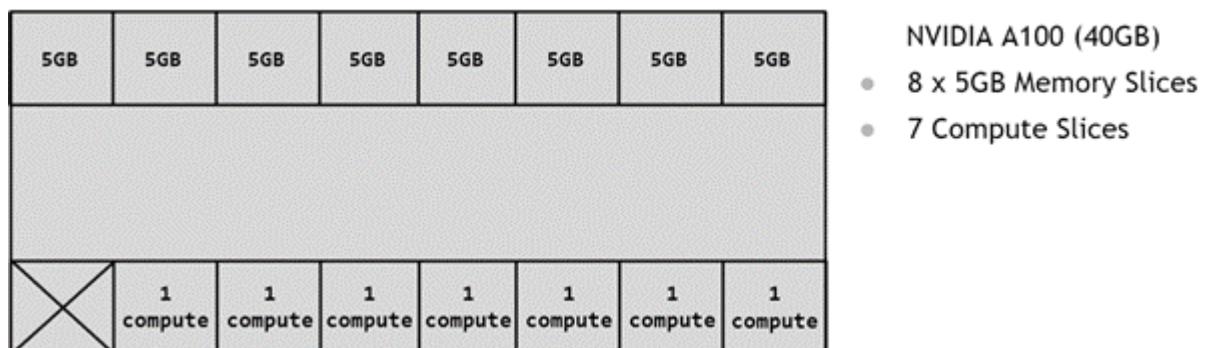
Để xem các tiến trình đang sử dụng hoặc truy cập vào GPU, chạy lệnh dưới và thay gpu_id bởi mã số của GPU (0,1,2,..)

```
$sudo lsof /dev/nvidia<gpu-id>
```

```
vkist@ubuntu-img:~$ sudo lsof /dev/nvidia4
[sudo] password for vkist:
COMMAND      PID  USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
kubelet      1568  root   16u  CHR  195,4      0t0  604 /dev/nvidia4
kubelet      1568  root   25u  CHR  195,4      0t0  604 /dev/nvidia4
kubelet      1568  root   26u  CHR  195,4      0t0  604 /dev/nvidia4
nvidia-de   14145  root   11u  CHR  195,4      0t0  604 /dev/nvidia4
nvidia-de   14145  root   24u  CHR  195,4      0t0  604 /dev/nvidia4
nvidia-de   14145  root   25u  CHR  195,4      0t0  604 /dev/nvidia4
nvidia-de   14145  root   38u  CHR  195,4      0t0  604 /dev/nvidia4
nvidia-de   14145  root   39u  CHR  195,4      0t0  604 /dev/nvidia4
nvidia-de   14145  root   40u  CHR  195,4      0t0  604 /dev/nvidia4
nvidia-de   14145  root   41u  CHR  195,4      0t0  604 /dev/nvidia4
nvidia-de   14145  root   42u  CHR  195,4      0t0  604 /dev/nvidia4
nvidia-de   14145  root   43u  CHR  195,4      0t0  604 /dev/nvidia4
nvidia-de   14145  root   44u  CHR  195,4      0t0  604 /dev/nvidia4
nvidia-de   14145  root   45u  CHR  195,4      0t0  604 /dev/nvidia4
nvidia-de   14145  root   56u  CHR  195,4      0t0  604 /dev/nvidia4
nvidia-de   14145  root   58u  CHR  195,4      0t0  604 /dev/nvidia4
python3    14612  root  mem   CHR  195,4      0t0  604 /dev/nvidia4
python3    14612  root   5u   CHR  195,4      0t0  604 /dev/nvidia4
python3    14612  root   6u   CHR  195,4      0t0  604 /dev/nvidia4
python3    14612  root   7u   CHR  195,4      0t0  604 /dev/nvidia4
python3    14612  root   13u  CHR  195,4      0t0  604 /dev/nvidia4
python3    14612  root   14u  CHR  195,4      0t0  604 /dev/nvidia4
```

3.2 Cài đặt MIG cho A100

3.2.1. Tổng quan



Một GPU A100 được chia thành 8 memory slice và 7 compute slides (instance), dùng cho mục đích phân chia và tận dụng tài nguyên GPU một cách hiệu quả.

Một số định nghĩa

GPU Engine

Một GPU engine chạy các công việc thực thi trên GPU. Engine được sử dụng phổ biến nhất là engine tính toán/ đồ họa dùng để thực hiện tính toán.

GPU Memory Slice

Một GPU Memory Slice là một phần nhỏ nhất của bộ nhớ GPU A100, bao gồm bộ điều khiển bộ nhớ (memory controllers) và cache. 1 GPU Memory Slice bằng 1/8 tổng tài nguyên của GPU, bao gồm cả dung lượng và băng thông.

GPU SM Slice

Một GPU SM Slice (streaming multiprocessor - thực hiện các tập lệnh trong GPU) là một phần nhỏ nhất của SM trong GPU A100. 1 GPU SM Slice có số lượng SM bằng 1/7 tổng số SM trong GPU

GPU Slice

Một GPU Slice là một phần nhỏ nhất của GPU A100, nó bao gồm 1 GPU Memory Slice và 1 GPU SM Slice.

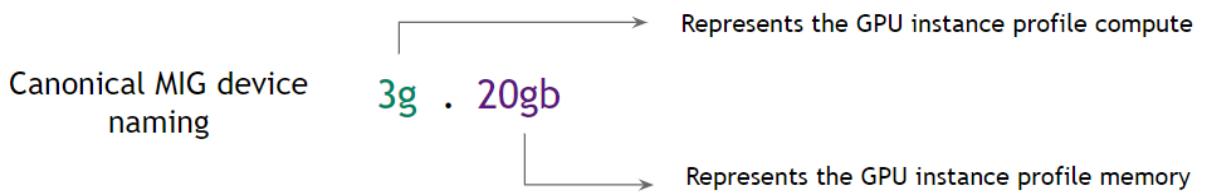
GPU Instance

Một GPU instance là tổ hợp của các GPU Slice và GPU engines

Compute Instance

Một GPU instance được chia nhỏ thành nhiều Compute instance. Một Compute Instance (CI) là tập con của SM slices và GPU engines của GPU instance của GPU A100. Các CI chia sẻ bộ nhớ và engine.

3.2.2. Ý nghĩa cách đặt tên của MIG



Một GPU instance bao gồm 3 thành phần chính là GPU Memory Slice và GPU SM Slice và các GPU engine. Trong đó các GPU SM Slice được phân vào các GPU compute instance khác nhau. Tên của GPU instance bao gồm 2 phần phần đầu tiên thể hiện số lượng compute instance trong 1 GPU instance và phần thứ 2 thể hiện số lượng bộ nhớ trong 1 GPU instance.

Ví dụ: 3g.20gb có ý nghĩa 1 GPU instance loại này sẽ có 3 compute instance và 20gb bộ nhớ.

Tổng số lượng compute instance của tất cả các GPU instance trong một GPU A100 không được vượt quá số lượng compute instance có trong A100 (7 compute instance)

3.2.3. Cài đặt MIG cho DGX-A100

YÊU CẦU PHẦN MỀM

- CUDA 11 và NVIDIA driver 450.80.02 trở lên
- Nếu chạy containers hoặc sử dụng Kubernetes, cần:
 - NVIDIA Container Toolkit (nvidia-docker2): v2.5.0 trở lên
 - NVIDIA K8s Device Plugin: v0.7.0 trở lên
 - NVIDIA gpu-feature-discovery: v0.2.0 trở lên

CÁC BƯỚC CÀI ĐẶT

- Enable MIG Mode
- Tạo GPU Instances

KỊCH BẢN CHIA MIG CHO DGX-A100

Có nhiều kịch bản cài đặt MIG cho DGX-A100, trong trường hợp này kịch bản cài đặt MIG cho DGX-A100 được thực hiện theo bảng cho thuận tiện.

GPU ID	MIG
GPU0, GPU1, GPU2, GPU3	Giữ nguyên, không chia MIG

GPU4	Chia thành 2 MIG 3g.20gb
GPU5	Chia thành 2 MIG 3g.20gb
GPU6	Chia thành 3 MIG 2g.10gb
GPU7	Chia thành 3 MIG 2g.10gb

Nguyên nhân GPU 6 và 7 chỉ có thể chia thành 3 MIG 2g.10Gb là do tổng số lượng nhân tính toán và nhân lưu trữ không được phép vượt quá số lượng nhân của 1 GPU A100 (7g và 40Gb) do đó 4 MIG 2g.10Gb yêu cầu một lượng tài nguyên bao gồm 8g và 40Gb sẽ khiến hệ thống báo lỗi.

THỰC HIỆN CÀI ĐẶT

Để cài đặt MIG cho A100, tiến hành enable các GPU muốn tạo MIG gồm các GPU 4,5,6,7.

Ví dụ enable cho GPU ID 4

\$nvidia-smi -i 4 -mig 1

Trong nhiều trường hợp, nếu đang có ứng dụng khác sử dụng GPU thì GPU sẽ không thể enable mig.

```
root@dgx-station:~# nvidia-smi -i 4 -mig 1
Warning: MIG mode is in pending enable state for GPU 00000000:87:00.0:In use by another client
00000000:87:00.0 is currently being used by one or more other processes (e.g. CUDA application or a monitoring application such as another instance of nvidia-smi). Please first kill all processes using the device and retry the command or reboot the system to make MIG mode effective.
All done.
```

Để tắt các ứng dụng này tiến hành chạy lệnh dưới để kiểm tra các tiến trình đang sử dụng GPU

```
opadmin@dgx-station:/raid/vkistuser$ sudo lsof /dev/nvidia4
COMMAND      PID USER   FD  TYPE DEVICE SIZE/OFF NODE NAME
nv-fabric  14492 root    2u  CHR  195,4      0t0 1466 /dev/nvidia4
nv-fabric  14492 root   32u  CHR  195,4      0t0 1466 /dev/nvidia4
nv-fabric  14492 root   33u  CHR  195,4      0t0 1466 /dev/nvidia4
nv-fabric  14492 root   34u  CHR  195,4      0t0 1466 /dev/nvidia4
kubelet   2442075 root   16u  CHR  195,4      0t0 1466 /dev/nvidia4
kubelet   2442075 root   28u  CHR  195,4      0t0 1466 /dev/nvidia4
kubelet   2442075 root   29u  CHR  195,4      0t0 1466 /dev/nvidia4
nvidia-de 3142145 root   11u  CHR  195,4      0t0 1466 /dev/nvidia4
nvidia-de 3142145 root   27u  CHR  195,4      0t0 1466 /dev/nvidia4
nvidia-de 3142145 root   28u  CHR  195,4      0t0 1466 /dev/nvidia4
nvidia-de 3142145 root   41u  CHR  195,4      0t0 1466 /dev/nvidia4
nvidia-de 3188941 root    9u  CHR  195,4      0t0 1466 /dev/nvidia4
nvidia-de 3188941 root   25u  CHR  195,4      0t0 1466 /dev/nvidia4
nvidia-de 3188941 root   26u  CHR  195,4      0t0 1466 /dev/nvidia4
nvidia-de 3188941 root   39u  CHR  195,4      0t0 1466 /dev/nvidia4
gpu-featu 3200851 root    9u  CHR  195,4      0t0 1466 /dev/nvidia4
gpu-featu 3200851 root   22u  CHR  195,4      0t0 1466 /dev/nvidia4
gpu-featu 3200851 root   23u  CHR  195,4      0t0 1466 /dev/nvidia4
```

Ngoại trừ tiến trình nv-fabric các tiến trình khác đều cần được tắt trước khi muốn enable mig cho GPU

Trong số đó bao gồm nvsm service, nvidia-dcgm service và các service do người dùng cài đặt hoặc các phần mềm do người dùng cài đặt. Sử dụng lệnh systemctl để tắt các service này và enable mig cho GPU lần nữa.

```
$ sudo nvidia-smi -i 4 -mig 1
Enabled MIG Mode for GPU 00000000:07:00.0
All done.
```

Sau khi enable mig thành công cho GPU tiến hành chạy lệnh dưới để kiểm tra danh sách các loại MIG có thể tạo

GPU instance profiles:								
GPU	Name	ID	Instances Free/Total	Memory GiB	P2P	SM CE	DEC JPEG	ENC OFA
3	MIG 1g.5gb	19	7/7	4.75	No	14 1	0 0	0 0
3	MIG 1g.5gb+me	20	1/1	4.75	No	14 1	1 1	0 1
3	MIG 2g.10gb	14	3/3	9.75	No	28 2	1 0	0 0
3	MIG 3g.20gb	9	2/2	19.62	No	42 3	2 0	0 0
3	MIG 4g.20gb	5	1/1	19.62	No	56 4	2 0	0 0
3	MIG 7g.40gb	0	1/1	39.50	No	98 7	5 1	0 1

Với 1 GPU (đã enabled) có thể thấy chỉ có 6 loại MIG có thể được tạo. Tuy nhiên MIG 1g.5gb+me yêu cầu Nvidia driver R470 do đó chỉ có 5 loại mig có thể được chọn để tạo instance.

Chú ý khi tạo số lượng compute instance của tất cả các gpu instance trong 1 GPU A100 không được vượt quá số lượng compute instance trong 1 GPU A100 là 7. Ví dụ:

Không thể tạo 1 mig loại 7g.40gb và 1 mig loại 4g.20gb cùng lúc do $7g+4g=11g>7g$.

Nhưng có thể tạo 1 mig loại 3g.20gb và 2 mig loại 2g.10gb do $3g+2g=7g$.

Tiến hành tạo instance cho GPU theo lệnh

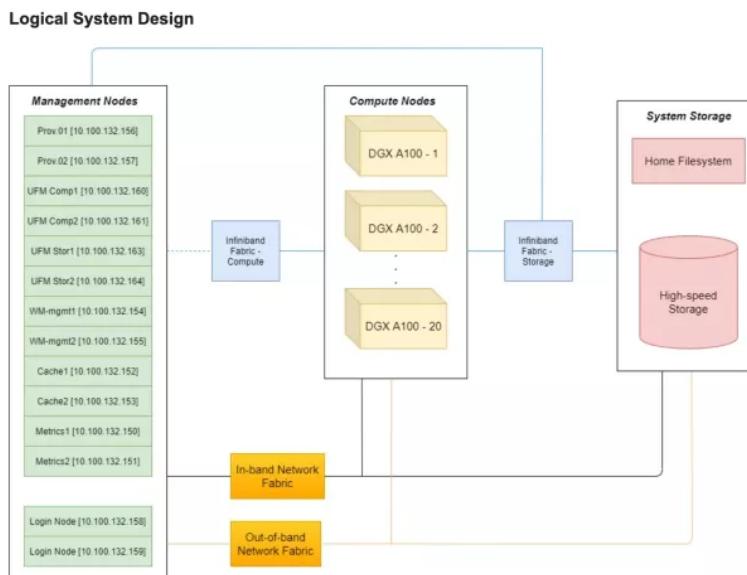
```
opadmin@gdx-station:/raid/vkistuser$ sudo nvidia-smi mig -i 4 -cgi 9,9 -C
Successfully created GPU instance ID 1 on GPU 4 using profile MIG 3g.20gb (ID 9)
Successfully created compute instance ID 0 on GPU 4 GPU instance ID 1 using profile MIG 3g.20gb (ID 2)
Successfully created GPU instance ID 2 on GPU 4 using profile MIG 3g.20gb (ID 9)
Successfully created compute instance ID 0 on GPU 4 GPU instance ID 2 using profile MIG 3g.20gb (ID 2)
opadmin@gdx-station:/raid/vkistuser$
```

Trong đó 9,9 là các ID của các loại MIG muốn tạo. Trong trường hợp này chỉ có thể tạo 2 GPU instance loại 9 hay 3g.20gb do đó ta ghi 9,9. Ta chỉ định GPU muốn tạo instance thông qua param -i 4

Sau khi thực hiện thành công 2 GPU instance loại 3g.20gb sẽ được tạo và có thể kiểm tra thông qua lệnh **nvidia-smi**

4. Thiết lập một hệ thống HPC (Tính toán hiệu năng cao)

Hệ thống tính toán hiệu năng cao có nhiệm vụ cung cấp các chức năng tính toán, lưu trữ, quản lý, tương tác với người dùng. Một hệ thống tính toán hiệu năng cao phức tạp hơn một hệ thống DGX-A100. Hệ thống DGX-A100 chỉ nên làm nhiệm vụ tính toán còn các nhiệm vụ quản lý hay lưu trữ sẽ do các hệ khác phối hợp đảm nhiệm nhằm tận dụng tối đa khả năng tính toán của DGX-A100. Một hệ thống tính toán hiệu năng cao khác với các hệ thống máy chủ thông thường là chúng có thêm các Compute Nodes. Các Compute Nodes làm nhiệm vụ tính toán khối lượng lớn các phép tính trong thời gian ngắn như các mô hình học sâu hoặc render. Đối với hệ thống HPC sử dụng DGX-A100 thì các hệ thống DGX-A100 sẽ là Compute Nodes. Ngoài ra, 2 thành phần khác là Management Nodes sẽ do một máy chủ thông thường đảm nhận và, Storage Nodes sẽ do hệ thống NAS đảm nhận.



4.1 Yêu cầu kỹ thuật

- Máy chủ quản lý k8s cluster
 - ❖ Yêu cầu phần cứng: Bộ nhớ tối thiểu 500G, Ram 64 GB, CPU 32 lõi
 - ❖ Yêu cầu phần mềm: Chạy hệ điều hành Ubuntu 20.04
 - ❖ Mục đích: Đóng vai trò là node quản lý của cluster, quản lý hệ thống HPC thông qua LAN. Chạy các dịch vụ NGC, Control Plane,...

- Switch mạng
 - ❖ Yêu cầu phần cứng: 5-10 Port mạng 1G - 10G, 10 - 24 Port LAN 100Mb hoặc switch chuyên dụng của Nvidia.
 - ❖ Mục đích: Đảm bảo chất lượng đường truyền cho hệ thống HPC. Các bài toán với chất lượng ảnh cao thì yêu cầu về đường truyền càng cao.
- Máy worker
 - ❖ Yêu cầu phần cứng: 1-5 máy chủ cấu hình bé hơn hoặc tương đương máy chủ quản lý.
 - ❖ Mục đích: Đóng vai trò là các node chạy dịch vụ hoặc các tác vụ không phải tác vụ hiệu năng cao, chạy dịch vụ cho phía người dùng, chạy inference các dịch vụ AI.

5. Cài đặt Kubernetes cluster

Kubernetes có nhiệm vụ quản lý, phân chia tài nguyên hệ tính toán hiệu năng cao. Có nhiều công nghệ có khả năng tương tự như Kubernetes nhưng kubernetes có một vài ưu điểm nổi bật bao gồm:

- Tiện lợi, dễ sử dụng và quản lý
- Đa dạng công cụ và mạnh mẽ
- Có khả năng áp dụng cho hệ thống với hầu hết phạm vi từ bé đến rất lớn
- Có cộng đồng hỗ trợ đông đảo
- Quản lý ứng dụng tập trung.
- Lập lịch tự động.
- Khả năng tự phục hồi.
- Triển khai và triển khai tự động.
- Cân bằng tải và cân ngang.
- Mật độ sử dụng tài nguyên cao hơn.



kubernetes

5.1 Tổng quan

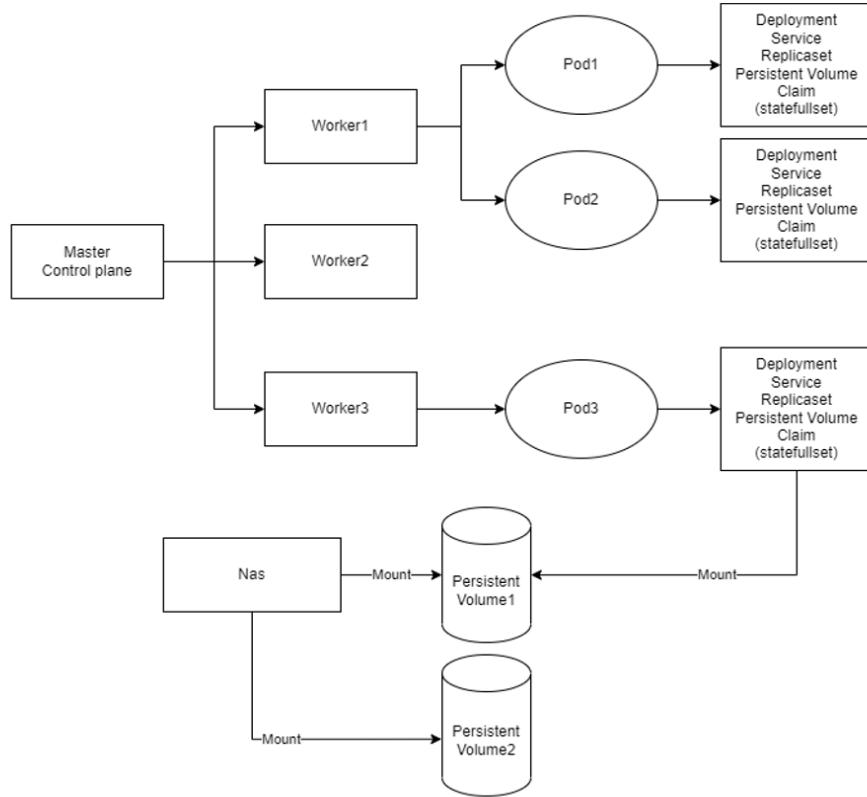
Kubernetes là một nền tảng nguồn mở, khả chuyển, có thể mở rộng để quản lý các ứng dụng được đóng gói và các service, giúp thuận lợi trong việc cấu hình và tự động hoá việc triển khai ứng dụng. Kubernetes là một hệ sinh thái lớn và phát triển nhanh chóng. Các dịch vụ, sự hỗ trợ và công cụ có sẵn rộng rãi.

Kubernetes quản lý theo dạng cluster. Cluster của Kubernetes được tạo nên bởi các node, mỗi node trong cluster có thể là máy ảo hoặc máy chủ vật lý. Trong cluster, một node sẽ được chọn làm node master có nhiệm vụ quản lý và điều khiển cluster, các node khác trong cluster, các ứng dụng và service. Phần mềm chạy trên node master được gọi là control plane, nó có nhiệm vụ quản lý các node và các pod.

Các định nghĩa:

- **API Group** là một tập những đường dẫn tương đến Kubernetes API. Có thể cho phép hay vô hiệu từng API group bằng cách thay đổi cấu hình trên API server của mình. API group đơn giản hóa việc mở rộng Kubernetes API, được chỉ định dưới dạng REST
- **API Server (kube-apiserver)**: được thiết kế để co giãn theo chiều ngang – có nghĩa là nó co giãn bằng cách triển khai thêm các thực thể.
- **Cgroup (control group)**: một nhóm các process trên Linux với sự tùy chọn trong cô lập tài nguyên, trách nhiệm và giới hạn. cgroup là một tính năng của Linux kernel giúp giới hạn, giao trách nhiệm, và cô lập việc sử dụng các tài nguyên trên máy (CPU, memory, disk I/O, network) cho một tập các process.
- **Cluster**: một tập các worker machine, được gọi là node, dùng để chạy các containerized application. Mỗi cụm (cluster) có ít nhất một worker node. Các worker node chứa các pod (là những thành phần của ứng dụng). Control Plane quản lí các worker node và pod trong cluster. Control Plane thường chạy trên nhiều máy tính và một cluster thường chạy trên nhiều node, cung cấp khả năng chịu lỗi và tính sẵn sàng cao.
- **Container**: Một image nhẹ, khả chuyển và có khả năng thực thi, chứa phần mềm và tất cả các dependencies của nó. Containers tách rời các ứng dụng khỏi hạ tầng máy chủ nhằm giúp cho việc triển khai dễ dàng hơn trên từng hệ thống cloud hay hệ điều hành khác nhau, và đơn giản hóa việc nhân rộng.
- **Container runtime interface (CRI)**: là một API phục vụ cho việc tích hợp container runtimes với kubelet trên một node
- **Control Plane**: Tầng điều khiển container, được dùng để đưa ra API, và các interface để định nghĩa, triển khai và quản lí vòng đời của các container.

- **DaemonSet:** Đảm bảo một bản sao của Pod đang chạy trên một tập các node của cluster.
- **Docker:** (cụ thể là Docker Engine): là một công nghệ phần mềm thực hiện việc ảo hóa tầng hệ điều hành, còn được gọi là container.
- **kube-proxy:** là một network proxy chạy trên mỗi node trong cluster, duy trì network rules trên các nodes. Những network rules này cho phép kết nối mạng đến các pods từ trong hoặc ngoài cluster. Kube-proxy sử dụng lớp packet filtering của hệ điều hành nếu có sẵn. Nếu không thì kube-proxy sẽ tự điều hướng network traffic.
- **Kubelet:** Một agent chạy trên mỗi node nằm trong cluster. Nó giúp đảm bảo rằng các containers đã chạy trong một pod. Kubelet sẽ nhận một tập các PodSpecs (đặc tính của Pod) được cung cấp thông qua các cơ chế khác nhau và bảo đảm rằng containers được mô tả trong những PodSpecs này chạy ổn định và khỏe mạnh. Kubelet không quản lý những containers không được tạo bởi Kubernetes.
- **Node:** Một node là một máy worker trong Kubernetes. Một worker node có thể là một máy tính ảo hay máy tính vật lý, tùy thuộc vào cluster. Nó bao gồm một số daemons hoặc services cần thiết để chạy các Pods và được quản lý bởi control plane. Daemons trên một node bao gồm kubelet, kube-proxy, và một container runtime triển khai theo CRI như Docker.
- **Pod:** Đối tượng nhỏ nhất và đơn giản nhất của Kubernetes. Một Pod đại diện cho một tập các containers đang chạy trên cluster. Một Pod thường được set up để chạy với một container chính yếu. Nó đồng thời có thể chạy kèm với các sidecar containers giúp bổ trợ thêm một số tính năng như thu thập log. Các Pods thường được quản lý bởi một Deployment.
- **Service:** Một cách để thể hiện ứng dụng đang chạy trong một tập các Pods dưới dạng dịch vụ mạng. Một tập các Pods được một Service nhắm đến (thường) được xác định với một selector. Nếu có nhiều Pods được thêm vào hay xóa đi, tập những Pods hợp với selector sẽ thay đổi. Service đảm bảo network traffic có thể đến tới tập những Pods để giải quyết công việc.
- **Vòng điều khiển (Controller):** Trong hệ thống Kubernetes, các bộ controllers là các vòng lặp điều khiển theo dõi trạng thái của mỗi cluster, sau đó chúng sẽ tạo hoặc yêu cầu sự thay đổi cần thiết. Mỗi controller cố thực hiện việc thay đổi để giúp hệ thống chuyển từ trạng thái hiện tại sang trạng thái mong muốn. Vòng điều khiển lặp lại theo dõi trạng thái chung của cluster thông qua Kubernetes API server



Sơ đồ một cụm K8s

5.2 Môi trường cài đặt

Hệ điều hành được sử dụng là Ubuntu 20.04, CPU tối thiểu 4 core, RAM từ 8G trở lên, drive tối thiểu 100G. Bên dưới là các lệnh kiểm tra cấu hình

```

vkist@master:/var/log$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:      Ubuntu 20.04.4 LTS
Release:         20.04
Codename:        focal

```

```

vkist@master:/var/log$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
Address sizes:         40 bits physical, 48 bits virtual
CPU(s):                8

```

Filesystem	Size	Used	Avail	Use%	Mounted on
udev	3.9G	0	3.9G	0%	/dev
tmpfs	796M	2.5M	794M	1%	/run
/dev/vda2	99G	5.1G	90G	6%	/
tmpfs	3.9G	0	3.9G	0%	/dev/shm
tmpfs	5.0M	0	5.0M	0%	/run/lock
tmpfs	3.9G	0	3.9G	0%	/sys/fs/cgroup
/dev/loop0	62M	62M	0	100%	/snap/core20/1328
/dev/loop1	68M	68M	0	100%	/snap/1xd/21835
/dev/loop2	44M	44M	0	100%	/snap/snapd/14978
tmpfs	796M	0	796M	0%	/run/user/0
/dev/loop3	62M	62M	0	100%	/snap/core20/1376
/dev/loop4	44M	44M	0	100%	/snap/snapd/15177
/dev/loop5	68M	68M	0	100%	/snap/1xd/22526
/dev/loop6	56M	56M	0	100%	/snap/core18/2344
/dev/loop7	118M	118M	0	100%	/snap/docker/1458
tmpfs	796M	0	796M	0%	/run/user/1001

5.3 Các bước cài đặt k8s cluster

1. Kiểm tra quyền của user

k8s cluster được cần được cài đặt dưới quyền của các user non root. Đối với các máy chưa có user non root ta tiến hành tạo một user mới và cung cấp quyền sudo cho user đó.

Để tạo một user mới ta sử dụng lệnh

```
root@master:/var/log# useradd vkist
```

Sau đó nhập mật khẩu cho user mới

Để thêm quyền sudo cho user mới, ta chạy lệnh sau dưới quyền root

```
root@master:/var/log# usermod -aG sudo vkist
```

Để chuyển từ root sang user mới tạo, ta chạy lệnh

```
root@master:/var/log# su vkist
```

2. Cài đặt các phần mềm liên quan

- ❖ Đối với Node master bao gồm: kubectl, kubeadm, kubelet, docker
- ❖ Đối với Node khác do không có Control Plane nên chỉ cần cài đặt: kubeadm, kubelet và docker

Các hướng dẫn cài đặt theo các đường link dưới :

- Đối với k8s

<https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>

The screenshot shows a section of the Kubernetes documentation titled "Install Kubeadm". It includes four numbered steps with command examples:

1. Update the `apt` package index and install packages needed to use the Kubernetes `apt` repository:

```
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl
```
2. Download the Google Cloud public signing key:

```
sudo curl -fsSL /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc
```
3. Add the Kubernetes `apt` repository:

```
echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes
```
4. Update `apt` package index, install kubelet, kubeadm and kubectl, and pin their version:

```
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

- Đối với Docker

<https://docs.docker.com/engine/install/ubuntu/>

Set up the repository

1. 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 \
  ca-certificates \
  curl \
  gnupg \
  lsb-release
```

2. 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
```

3. Use the following command to set up the `stable` repository. To add the `nightly` or `test` repository, add the word `nightly` OR `test` (or both) after the word `stable` in the commands below. [Learn about nightly and test channels.](#)

```
$ 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
```

3. Kiểm tra kubelet service và docker service đã chạy. Trong đa số trường hợp kubelet sẽ gặp lỗi khi khởi động.

```

root@master:~# systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
  Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: enabled)
  Drop-In: /etc/systemd/system/kubelet.service.d
    └─10-kubeadm.conf
   Active: inactive (dead) (Result: exit-code) since Fri 2022-03-18 08:38:14 UTC; 9min ago
     Docs: https://kubernetes.io/docs/home/
   Process: 28305 ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS $KUBELET_KUBEADM_ARGS $KUB...
   Main PID: 28305 (code=exited, status=1/FAILURE)

Mar 18 08:38:14 master systemd[1]: Stopped kubelet: The Kubernetes Node Agent.

```

Điều này xảy ra do cgroup-drive của kubelet và docker khác nhau.

Tiến hành đổi cgroup-drive cho kubelet theo hướng dẫn:

<https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/configure-cgroup-driver/>

hoặc trực tiếp thay đổi trên file config service của kubelet bằng cách chạy lệnh

```

vkist@master:/var/log$ systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
  Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: enabled)
  Drop-In: /etc/systemd/system/kubelet.service.d
    └─10-kubeadm.conf
   Active: active (running) since Fri 2022-03-18 10:18:46 UTC; 17h ago
     Docs: https://kubernetes.io/docs/home/
   Main PID: 51920 (kubelet)
     Tasks: 21 (limit: 9441)
    Memory: 53.1M
      CGroup: /system.slice/kubelet.service
              └─51920 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/...

```

Như vậy ta sẽ sửa config trong file 10-kubeadm.conf. Tiến hành chạy lệnh

```

vkist@master:/var/log$ sudo cat /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
# Note: This dropin only works with kubeadm and kubelet v1.11+
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf"
Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml"
Environment="KUBELET_EXTRA_ARGS=--cgroup-driver=systemd"

```

và thêm dòng

Environment="KUBELET_EXTRA_ARGS=--cgroup-driver=systemd"

sau đó khởi động lại kubelet service bằng lệnh

systemctl stop kubelet

systemctl start kubelet

Đối với docker, ta sẽ sửa trực tiếp trên file service của docker để thay đổi cgroup-drive đồng bộ với kubelet bằng cách sửa file

/lib/systemd/system/docker.service

Thêm param **--exec-opt native.cgroupdriver=systemd** vào sau lệnh ExecStart sau đó khởi động lại docker service bằng lệnh

systemctl stop docker

systemctl start docker

systemctl stop docker.socket

systemctl start docker.socket

```

vkist@master:/var/log$ cat /lib/systemd/system/docker.service
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
After=network-online.target docker.socket firewalld.service containerd.service
Wants=network-online.target
Requires=docker.socket containerd.service

[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --exec-opt native.cgroupdriver=systemd
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always

```

- Sau khi kubelet và docker đã chạy, tiếp tục tiến hành khởi tạo Control Plane trên Node master bằng kubeadm. Kubeadm là công cụ để tạo k8s cluster. Đầu tiên, chạy lệnh **kubeadm reset** để khởi tạo lại tất cả các cấu hình của cluster.

Để khởi tạo Control Plane cho Node master, chạy lệnh **kubeadm init**. Lưu ý chỉ chạy lệnh này trên Node master. Khi kubelet và docker đã chạy thành công và các bước thực hiện chính xác thì lệnh init sẽ chạy preflight và không có lỗi nào xảy ra.

Sau khi Control Plane khởi tạo thành công kubeadm sẽ cung cấp 1 register token cho việc đăng ký các Node khác vào Control Plane này.

```

kubeadm join 10.1.10.137:6443 --token mdet7z.n5wy3qnlgizflw2y \
--discovery-token-ca-cert-hash
sha256:b69e07410fd9fc8c180a4c1683e38eec604d384813156a70773b6a7a6ad8
4c13

```

Để khởi tạo register token mới, tiến hành chạy lệnh dưới. Token được khởi tạo có thời gian 24h

```

vkist@master:/var/log$ kubeadm token create --print-join-command
kubeadm join 10.1.10.137:6443 --token 4q5f1u.a3gfmh3h7qj81jw1 --discovery-token-ca-cert-hash sha256:b69e07410fd9fc8c180a4c168
3e38eec604d384813156a70773b6a7a6ad84c13

```

- Tiến hành đăng ký Node khác vào Control Plane

Các Node khác cũng được thực hiện các bước từ 1 đến 3 giống với Node master. Thay vì dùng kubeadm init để khởi tạo Control Plane mới giống Node master, các Node khác sẽ đăng ký vào Control Plane của Node master thông qua resgister token khởi tạo ở bước 4 bằng lệnh kubeadm join.

Copy kết quả của token được tạo ở bước 4 và chạy trong terminal của Node muốn đăng ký, trong trường hợp trên câu lệnh có dạng:

```

kubeadm join 10.1.10.137:6443 --token mdet7z.n5wy3qnlgizflw2y
--discovery-token-ca-cert-hash
sha256:b69e07410fd9fc8c180a4c1683e38eec604d384813156a70773b6a7a6ad
84c13

```

Nếu chạy thành công Node mới sẽ được đăng ký vào Control Plane của Node master.

- Sử dụng kubectl để quản lý và điều khiển Node và Pod.

Chạy lệnh **kubectl cluster-config** để kiểm tra Control Plane vẫn hoạt động bình thường. Trong nhiều trường hợp, lỗi sau sẽ xuất hiện

```
vkist@master:~/kube/config$ kubectl get node
The connection to the server 10.1.10.137:6443 was refused - did you specify the right host or port?
```

Lỗi này do kubectl không thể kết nối đến kube-apiserver của Control Plane để thực hiện việc lấy thông tin về cluster.

Thông thường kube-apiserver được đóng gói và chạy trên nền tảng container của Node master, ở đây là Docker. Do đó lỗi này thường xảy ra do 3 nguyên nhân chủ yếu

- Do Docker không hoạt động hoặc hoạt động lỗi
 - Do kube-apiserver container không hoạt động hoặc bị k8s pause
 - Do file admin.conf không thuộc quyền hạn của non user
- Đối với lỗi do Docker không hoạt động hoặc hoạt động lỗi tiến hành kiểm tra lại Docker bằng cách gõ lệnh **docker** vào thanh công cụ. Sửa lỗi Docker bằng cách chạy lại service hoặc kiểm tra cgroup-drive đã đồng bộ với kubelet.
 - Đối với lỗi kube-apiserver container không hoạt động hoặc bị k8s pause, ta tiến hành kiểm tra bằng lệnh dưới

```
root@master:/var/log# docker ps | grep kube-apiserver
204042ff5aba 3fc1d62d6587 "kube-apiserver --ad..." 5 hours ago Up 5 hours k8s_kube-apiserver
be-apiserver_kube-apiserver-master_kube-system_12569c218896461a0a8cd165d59e9e78_3
a35dfcd435d3 3fc1d62d6587 "kube-apiserver --ad..." 21 hours ago Up 21 hours k8s_kube-apiserver
be-apiserver_kube-apiserver-master_kube-system_12569c218896461a0a8cd165d59e9e78_2
fbac0e5fad80 k8s.gcr.io/pause:3.6 "/pause" 21 hours ago Up 21 hours k8s_POD
D_kube-apiserver-master_kube-system_12569c218896461a0a8cd165d59e9e78_0
```

Nếu kube-apiserver container bị tắt thì khởi động lại. Trong trường hợp kube-apiserver container vẫn chạy thì nguyên nhân do ổ cứng bị tràn hoặc do bộ nhớ swap đang chạy. Tiến hành sửa lỗi bằng cách tắt bộ nhớ swap và resize lại thư mục ổ cứng (đối với trường hợp dùng VM) với quyền root

```
root@master:/var/log# sudo mount -o size=10M,rw,nodev,nosuid -t tmpfs tmpfs /tmp
root@master:/var/log# growpart /dev/vda 2
CHANGED: partition=2 start=4096 old: size=10479616 end=10483712 new: size=209711071 end=209715167
```

```
root@master:/var/log# resize2fs /dev/vda2
resize2fs 1.45.5 (07-Jan-2020)
Filesystem at /dev/vda2 is mounted on /; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 13
The filesystem on /dev/vda2 is now 26213883 (4k) blocks long.
```

```
root@master:/var/log# swapoff -a
```

- Đối với lỗi file admin.conf không thuộc quyền hạn của non root user. Lỗi này xảy ra do file admin.conf được kubeadm init tạo ra cho Node master nằm trong thư mục /etc/kubernetes do đó kubectl của non root user không thể truy cập vào

file này. Ta tiến hành chuyển file admin.conf sang thư mục \$HOME/.kube/config của user bằng lệnh dưới dùng quyền root

```
root@master:/var/log# sudo cp /etc/kubernetes/admin.conf $HOME/.kube/config
```

Sau đó thay đổi biến môi trường KUBECONFIG

```
root@master:/var/log# export KUBECONFIG=/home/vkist/.kube/config/admin.conf
```

7. Sau khi kiểm tra kubectl đã chạy mà không có lỗi ta tiến hành chạy **kubectl get node** nếu thành công ta sẽ thu được kết quả như hình dưới. Cluster bao gồm 1 Node master và 1 Node worker

```
root@master:/etc/kubernetes# kubectl get nodes
NAME     STATUS    ROLES          AGE     VERSION
master   NotReady control-plane,master   26m    v1.23.5
worker   NotReady <none>           7m39s   v1.23.5
root@master:/etc/kubernetes#
```

có thể thấy trạng thái của cả 2 Node là NotReady điều này xảy ra do chưa cài đặt Pod phụ trợ WeaveNet (nguyên nhân do k8s yêu cầu các network interface đặc biệt cho các Pod) hoặc do config bị sai.

Kiểm tra trạng thái cài đặt của các Pod WeaveNet bằng lệnh

```
root@master:/etc/kubernetes# kubectl get pod --all-namespaces
NAMESPACE     NAME           READY   STATUS    RESTARTS   AGE
kube-system   coredns-64897985d-g542l   0/1     Pending   0          31m
kube-system   coredns-64897985d-mdn7f   0/1     Pending   0          31m
kube-system   etcd-master            1/1     Running   0          32m
kube-system   kube-apiserver-master   1/1     Running   0          32m
kube-system   kube-controller-manager   1/1     Running   0          32m
kube-system   kube-proxy-4v8w7        1/1     Running   0          31m
kube-system   kube-proxy-68ztt        1/1     Running   0          13m
kube-system   kube-scheduler-master   1/1     Running   0          32m
root@master:/etc/kubernetes#
```

Để cài WeaveNet Pod ta phải thực hiện trong non root user bằng lệnh dưới

```
vkist@master:~/.kube/config$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"
serviceaccount/weave-net created
clusterrole.rbac.authorization.k8s.io/weave-net created
clusterrolebinding.rbac.authorization.k8s.io/weave-net created
role.rbac.authorization.k8s.io/weave-net created
rolebinding.rbac.authorization.k8s.io/weave-net created
daemonset.apps/weave-net created
vkist@master:~/.kube/config$ kubectl get nodes
```

Nếu thành công, trạng thái của các Node sẽ trở thành Ready

```
vkist@master:/var/log$ kubectl get node
NAME     STATUS    ROLES          AGE     VERSION
master   Ready    control-plane,master   21h    v1.23.5
worker   Ready    <none>           21h    v1.23.5
vkist@master:/var/log$
```

Tạo 1 Pod và deploy thử lên Node worker

```
vkist@master:/var/log$ kubectl run --image=weaveworks/hello-world hello
```

```
vkist@master:/var/log$ kubectl get pods -o wide
NAME      READY   STATUS             RESTARTS   AGE     IP      NODE      NOMINATED NODE   READINESS GATES
hello     0/1     ContainerStatusUnknown   1         21h    <none>   worker    <none>    <none>
```

6. Cài đặt Jupyterhub

Kubernetes làm nhiệm vụ quản lý và phân chia tài nguyên của DGX-A100 để tận dụng nguồn tài nguyên này cho các mục đích như huấn luyện, chạy infer hay chạy demo, dịch vụ ta có đa dạng các giải pháp như kubeflow, jupyterhub, virtual machine. Tài liệu này sẽ hướng dẫn cài đặt và cấu hình jupyterhub, so với các công nghệ còn lại Jupyter hub có những ưu điểm nổi bật như :

- Mã nguồn mở
- Đơn giản, dễ dàng cài đặt và sử dụng
- Có khả năng cung cấp cho số lượng lớn người dùng
- Có giao diện tiện lợi cho việc quản lý cũng như sử dụng



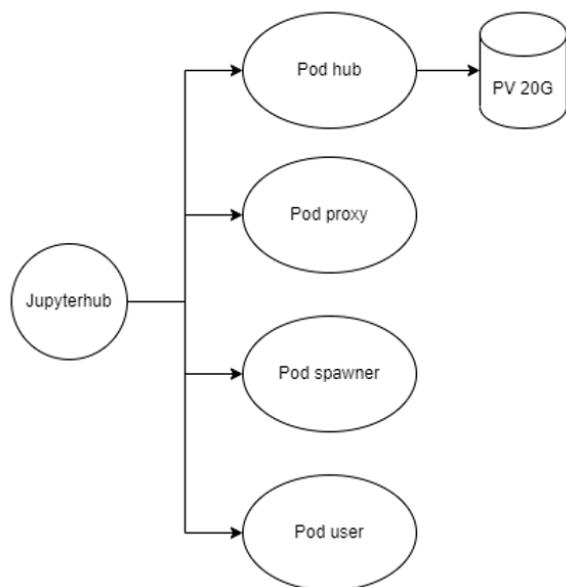
6.1 Tổng quan

Jupyterhub là dự án phần mềm giúp cung cấp các notebook cho đa người dùng và được thiết kế cho các công ty, lớp học hay các phòng nghiên cứu. Jupyterhub có khả năng cấp phát các notebook với môi trường tính toán và tài nguyên đầy đủ thông qua sức mạnh của k8s cluster. Qua đó người dùng có khả năng tập trung vào công năng sử dụng mà không phải tự tay cài đặt và điều khiển.

Một số đặc điểm của Jupyterhub:

- Tùy biến: Jupyterhub có khả năng cung cấp đa dạng các loại môi trường phục vụ các mục đích nghiên cứu khác nhau. Jupyterhub hỗ trợ các image và kernel thông qua Jupyter server và các giao diện người dùng bao gồm Jupyter Notebook, Jupyter Lab, RStudio, nteract.
- Linh hoạt: Jupyterhub có thể tùy chỉnh các phương thức xác thực nhằm mục đích phân quyền phân loại người dùng.

- Có khả năng mở rộng: Jupyterhub là dự án được đóng gói và được triển khai trên các công nghệ đóng gói như Docker. Nó cũng được chạy trên k8s thông qua các pod và tận dụng khả năng của k8s để cung cấp cho hàng vạn người dùng.
- Tích hợp: Jupyterhub là dự án hoàn toàn mở và được thiết kế chạy trên đa nền tảng từ các nền tảng điện toán đám mây thương mại cho đến máy tính cá nhân đều có thể chạy Jupyterhub.



Sơ đồ hệ thống Jupyterhub

Một hệ thống Jupyterhub đầy đủ bao gồm 4 Pod chính là hub, proxy, spawner và user. Pod hub làm nhiệm vụ quản lý các pod user, cung cấp giao diện cho admin, xác thực và phân quyền user. Pod proxy làm nhiệm vụ cấp mạng và IP cho các pod user và cho pod hub. Pod spawner có chức năng tạo ra một pod user mới với đầy đủ môi trường, tài nguyên mà user đó request. Pod user là pod cung cấp giao diện notebook cho từng người dùng, mỗi người dùng sẽ có 1 Pod loại này và có tên Pod trùng với tên username.

6.2 Cài đặt Jupyterhub

Để cài đặt Jupyterhub trên k8s cluster các yêu cầu sau cần được hoàn thiện:

- Nhà cung cấp điện toán đám mây: trong trường hợp này cloud được cài đặt trên hệ thống DGX-A100 của viện VKIST thông qua Kubernetes cluster.
- K8s
- Helm v3 để tùy chỉnh và điều khiển việc cài đặt các gói của Jupyterhub

- Docker và Docker registration để quản lý và xây dựng các container của Jupyterhub cũng như các container tùy biến của người dùng

Khi đã có K8s cluster, ta có thể cài đặt Jupyterhub trên K8s thông qua Helm chart. Helm sẽ cài đặt các pod, các gói và các container của Jupyterhub một cách tự động lên K8s dựa trên tùy chỉnh của người dùng đặt trong file config.yaml. Để cài đặt Helm, tác giả khuyên dùng các gói binary đã được xây dựng sẵn. Tìm hiểu thêm tại <https://helm.sh/docs/intro/install/>. Helm được cài đặt từ binary thông qua 3 bước.

1. Tải phiên bản của helm trên trang git release chính thức của helm về máy master.
Phiên bản được dùng là 3.8.0
2. Mở gói thông qua lệnh (tar -zxvf helm-v3.8.0-linux-amd64.tar.gz)
3. Tìm file binary của helm và chuyển đến thư mục đích thông qua lệnh (mv linux-amd64/helm /usr/local/bin/helm)

Tiến hành cài đặt Jupyterhub thông qua Helm chart. Helm chart chứa các template của các dự án và render chúng vào K8s cluster. Tìm hiểu thêm tại <https://zero-to-jupyterhub.readthedocs.io/en/latest/jupyterhub/installation.html>

Đăng ký [JupyterHub Helm chart repository](#) vào helm do đó tránh việc cài đặt dùng các đường dẫn phức tạp sau này.

```
helm repo add jupyterhub https://jupyterhub.github.io/helm-chart/
helm repo update
```

Ta sẽ thu được kết quả :

Hang tight while we grab the latest from your chart repositories...

...Skip local chart repository

...Successfully got an update from the "stable" chart repository

...Successfully got an update from the "jupyterhub" chart repository

Update Complete. ※ Happy Helming!※

Cài đặt chart và tùy chỉnh bằng file config.yaml đã tạo bằng cách chạy lệnh dưới trong cùng thư mục với file config.yaml. Ta sẽ tùy chỉnh file config.yaml trong phần sau:

```
helm upgrade --cleanup-on-fail \
--install <helm-release-name> jupyterhub/jupyterhub \
--namespace <k8s-namespace> \
--create-namespace \
--version=<chart-version> \
--values config.yaml
```

Trong đó :

- <helm-release-name> là tên của [Helm release](#), nhằm mục đích định danh phiên bản của dự án. Để xem tên phiên bản dùng lệnh helm list.
- <k8s-namespace> là tên của [Kubernetes namespace](#), nhằm mục đích định danh nhóm Kubernetes resources. Để xem tên nhóm dùng lệnh kubectl get pod -ALL
- Bước này sẽ mất một khoảng thời gian, khi đó các pod, package và container của Jupyterhub được cài đặt bên dưới.
- Tham số --version đại diện cho phiên bản của *Helm chart*, không phải phiên bản của JupyterHub. Mỗi phiên bản của JupyterHub Helm chart ánh xạ đến một phiên bản của JupyterHub. Ví dụ., 0.11.1 của Helm chart chạy JupyterHub 1.3.0. Để xem phiên bản của Helm chart và Jupyterhub tương ứng tham khảo [Helm Chart repository](#).

Trong khi bước 2 đang chạy, ta có thể xem các pod đang được tạo bằng cách nhập lệnh. Đợi trạng thái pod của hub và proxy chuyển sang Running.

```
vkist@master:~$ kubectl get pod -n jhub
NAME                  READY STATUS RESTARTS AGE
continuous-image-puller-8pzrw  1/1   Running 0      16h
hub-5f8cbfbfd-lj6wg     1/1   Running 0      16h
proxy-794b66dc49-lfgnb   1/1   Running 0      16h
user-scheduler-79c85f98dd-kmgcz 1/1   Running 1      16h
user-scheduler-79c85f98dd-s8nn9  1/1   Running 1      16h
```

Để truy cập Jupyterhub. Chạy lệnh dưới để tìm EXTERNAL-IP của proxy-public [service](#).

```
vkist@master:~$ kubectl get service -n jhub
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
hub       ClusterIP  10.102.242.102  <none>        8081/TCP    17h
proxy-api  ClusterIP  10.108.132.173  <none>        8001/TCP    17h
proxy-public LoadBalancer  10.105.73.24   10.1.12.200  80:31219/TCP  17h
```

Trong trường hợp EXTERNAL-IP ở trạng thái <pending> điều này có nghĩa là service proxy-public không tìm được dải IP. Ta phải cài metallB phụ trợ cho LoadBalancer này. Tham khảo thêm tại <https://metallb.universe.tf/installation/>

Chạy lệnh

```
kubectl apply -f
```

```
https://raw.githubusercontent.com/metallb/metallb/v0.12.1/manifests/namespace.yaml
```

```
kubectl apply -f
```

```
https://raw.githubusercontent.com/metallb/metallb/v0.12.1/manifests/metallb.yaml
```

Tiến hành tạo file metalLB-config.yaml để cấu hình cho metallB

```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 10.1.12.200-10.1.12.250
```

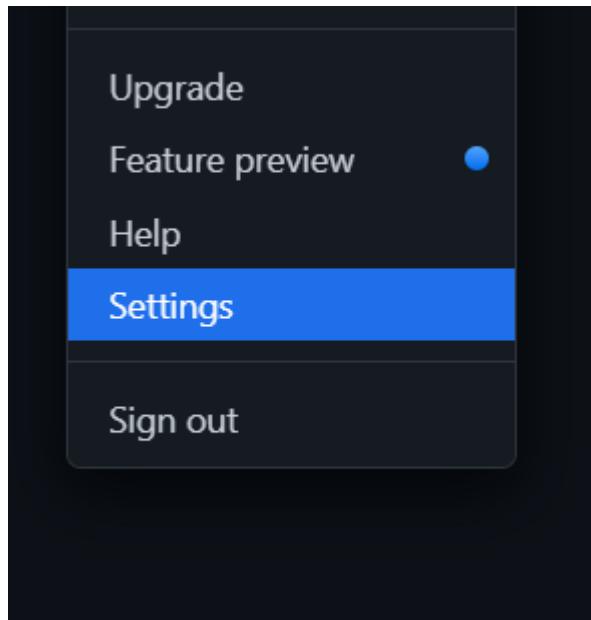
Thay thế dải 10.1.12.200-10.1.12.250 bằng dải IP local của LAN.

Chạy lệnh kubectl apply -f metalLB-config.yaml, service proxy-public sẽ tự động được đăng ký đến một IP trong dải IP đã config. Trong trường hợp này IP được đăng ký là 10.1.12.200 sau khi truy cập IP này ta sẽ thấy Jupyterhub đang chạy trong chế độ xác thực dummy do đó người dùng có thể điền username, password bất kỳ để truy cập hub.

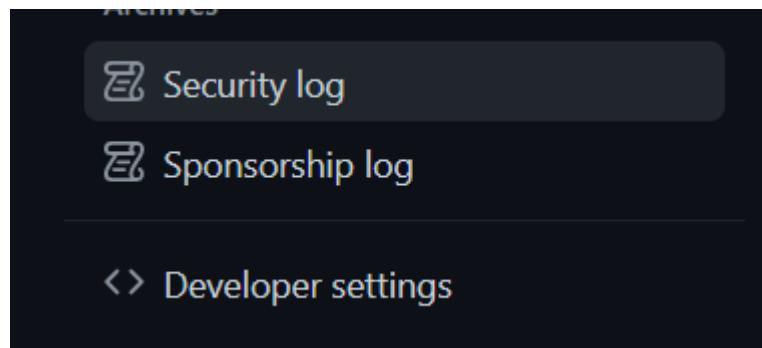
6.3 Cấu hình Jupyterhub

Jupyterhub sau khi cài đặt sẽ chạy ở cấu hình đơn giản nhất, với cấu hình này mọi tài khoản đều có thể đăng nhập và tạo tài khoản.

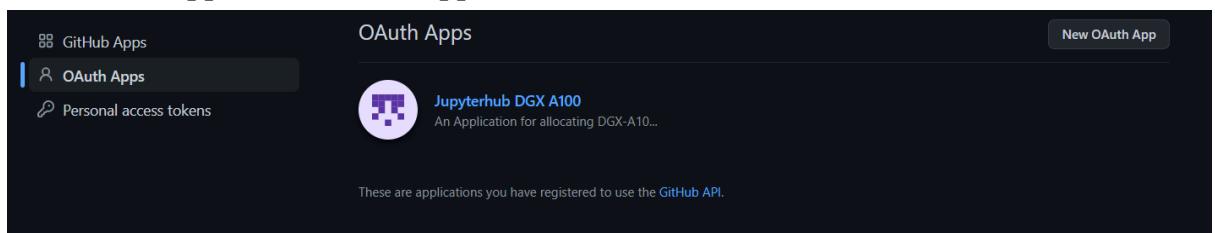
Để phân quyền người dùng bằng Github, đầu tiên ta cần tạo một app oauth trên github. Tạo tài khoản trên github và bấm vào mục Settings



Trong setting bấm chọn Developer settings



Chọn oauth app và new oauth app



Điền các thông số vào form hiện ra

Register a new OAuth application

Application name *

Something users will recognize and trust.

Homepage URL *

The full URL to your application homepage.

Application description

Application description is optional

This is displayed to all users of your application.

Authorization callback URL *

Your application's callback URL. Read our [OAuth documentation](#) for more information.

Enable Device Flow

Allow this OAuth App to authorize users via the Device Flow. Read the [Device Flow documentation](#) for more information.

Trong đó, chú ý mục Authorization callback URL sẽ điền đường dẫn đến API oauth callback của Jupyterhub và API này phải được public, thông thường có dạng http://<your_url_to_jupyterhub_homepage>/hub/oauth_callback

Cuối cùng bấm Register application và một oauth app được tạo ra như dưới

The screenshot shows the GitHub OAuth Apps interface. At the top, there's a button labeled "New OAuth App". Below it, a list of registered apps is shown, starting with "Jupyterhub DGX A100". This app has a purple icon and a description: "An Application for allocating DGX-A10...". Below the list, a message says "These are applications you have registered to use the GitHub API.".

Bấm vào oauth app vừa tạo và copy 2 tham số client id và client secret

The screenshot shows the GitHub OAuth Client secrets page. It displays a "Client ID" field containing "6c0a1d059a39cea43c1d" and a "Client secrets" section. In the secrets list, there is one entry: a key icon followed by "*****f84cd07a", with the note "Added 3 days ago by daovietanh190499" and "Last used within the last week". A "Delete" button is visible next to the secret. A note at the bottom states "You cannot delete the only client secret. Generate a new client secret first." There is also a "Generate a new client secret" button.

Đồng thời để phân quyền người dùng bằng Github ta tiến hành thay đổi cấu hình trong file config.yaml và update lại Jupyterhub trong k8s theo cấu hình này dùng helm chart

Trong file config.yaml, thay đổi cấu hình xác thực trong mục hub.config hub:

config:

GitHubOAuthenticator:

client_id: 6c0a1d059a39cea43c1d

client_secret: ced85cab3963bd6336a2c349ecd2204f84cd07a

oauth_callback_url: http://14.232.152.54:1611/hub/oauth_callback

JupyterHub:

authenticator_class: github

Authenticator:

admin_users:

- daovietanh190499

- gungui98

Trong đó:

- client_id và client_secret là 2 tham số được lấy từ trang dành cho nhà phát triển của github ở bước đầu tiên
- oauth_callback_url là đường dẫn đến api oauth callback của Jupyterhub
- authenticator_class là tên loại xác thực
- Trong mục Authenticator, tham số admin_users nhằm xác định các user có quyền admin và tham số allowed_users nhằm xác định các user có quyền truy cập Jupyterhub

7. Kết quả và triển khai thử nghiệm của hệ thống DGX-A100

Hiện tại hệ thống DGX-A100 đã đưa vào hoạt động và đạt một số kết quả sau:

- Hệ thống DGX-A100, hệ thống NAS và hệ thống mạng được kết nối với nhau và vận hành trơn tru.
- Cài đặt thành công DGX OS cho máy chủ DGX-A100.
- ssh, vpn, bmc, máy ảo hoạt động 24/7, luôn sẵn sàng để kết nối, quản lý mỗi khi hệ thống gặp sự cố (high available).
- Hệ thống cụm K8s cơ bản hoạt động 24/7, có khả năng khôi phục sau khi hệ thống lỗi hoặc mất điện
- Phần mềm Jupyterhub hoạt động 24/7, hiện tại đang cung cấp tài nguyên cho nghiên cứu, huấn luyện mô hình nhận diện khuôn mặt, xây dựng mô hình 3D, mô hình ngôn ngữ, tiếng nói.

8. Một số vấn đề cần giải quyết

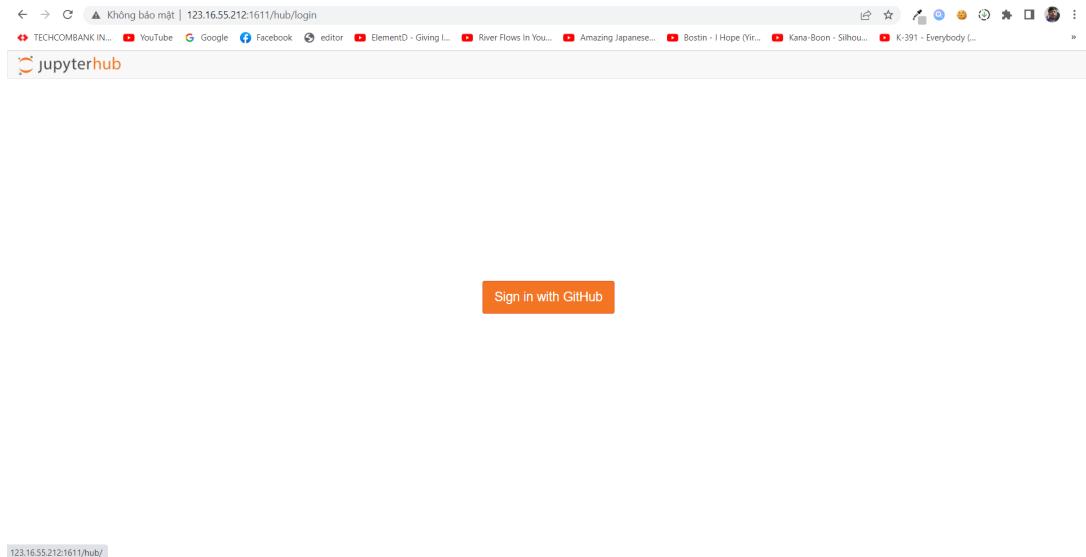
- Về hệ thống DGX-A100
 - Vấn đề về đường truyền, cần nâng cấp đường truyền (Switch) để đáp ứng đủ nhu cầu và tận dụng tối đa khả năng của DGX-A100.
- Về phần mềm quản lý tài nguyên
 - Kubernetes
 - Hiện tại máy master là máy chủ ảo chạy trên DGX-A100 do đó hiệu quả không cao và tài nguyên bị phân chia từ A100 về máy chủ ảo khiến cho không tận dụng hết khả năng của GPU. Cần thay thế bằng máy chủ vật lý.
 - Chưa có giải pháp cho việc phân chia tài nguyên cho Distributed Training.
 - Jupyterhub
 - Mỗi trường notebook chưa hoàn toàn đáp ứng đầy đủ các nhu cầu. Cần build và đóng gói lại notebook hoàn chỉnh hơn thông qua bản khảo sát người dùng.
 - Notebook vẫn gặp lỗi mất kết nối với GPU.

JUPYTER NOTEBOOK DOCUMENTATION GUIDE

Đào Việt Anh, Researcher - IT Division of VKIST

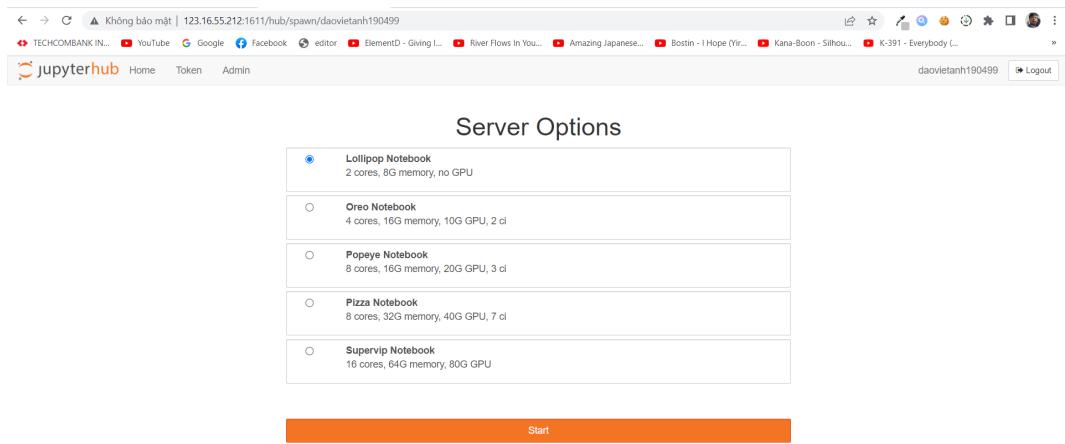
1. Đăng nhập vào notebook

Truy cập vào địa chỉ <http://123.16.55.212:1611> để vào giao diện trang đăng nhập. Tiến hành đăng nhập bằng tài khoản Github cá nhân. Đối với các lần đăng nhập sau hệ thống sẽ tự động đăng nhập vào tài khoản Github lần trước.



2. Khởi tạo Notebook

Sau khi đăng nhập, người dùng sẽ được điều hướng sang trang lựa chọn notebook. Tại đây người dùng sẽ được phân quyền việc sử dụng notebook theo chỉ định của admin

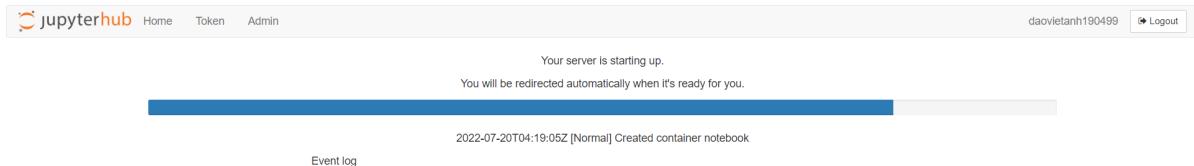


Đối với các tài khoản có đầy đủ các lựa chọn thì có khả năng tạo các notebook bao gồm:

- Lollipop Notebook: 2 nhân CPU, 8G ram, không có GPU

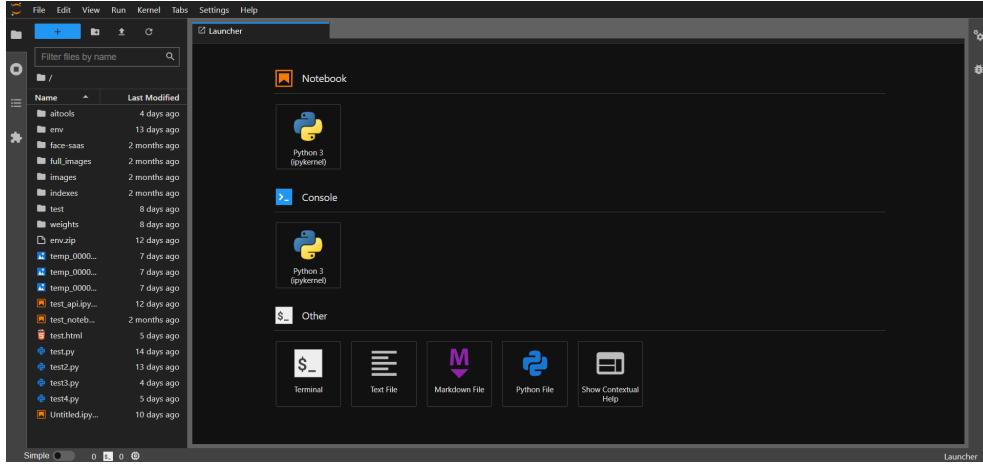
- Oreo Notebook: 4 nhân CPU, 16G ram, 10G GPU
- Popeye Notebook: 8 nhân CPU, 16G ram, 20G GPU
- Pizza Notebook: 8 nhân CPU, 32G ram, 40G GPU
- Supervip Notebook: 16 nhân CPU, 64G ram, 80G GPU

Sau khi chọn notebook, spawner hub của hệ thống Jupyterhub sẽ tiến hành khởi tạo môi trường notebook và giao diện giống với cấu hình đã chọn.



Trong trường hợp cấu hình còn lại của máy chủ không thể đáp ứng, màn hình sẽ thông báo lỗi cho người dùng và đợi timeout trong 5 phút. Trong thời gian đợi 5 phút nếu máy chủ có khả năng cung cấp đủ cấu hình thì notebook sẽ được tiếp tục khởi tạo. Trong thực tế notebook là một môi trường web đóng kín và được chứa trong Docker container. Docker container chứa đầy đủ các môi trường, thư viện cần thiết để có thể sử dụng được ngay như Pytorch, opencv, cuda, nvidia-cuda-toolkit, ... Docker container được sử dụng là gungui/deep-learning-cuda.

3. Khởi chạy một số tác vụ đơn giản của Notebook



Giao diện jupyterhub

Tại cột phía bên trái màn hình giao diện là cột thư mục home của Notebook, đường dẫn thư mục home của Notebook mặc định là /home/jovyan. Thư mục home được mount tới thư mục trên máy NAS do đó dữ liệu sẽ không bị mất đi nếu được lưu trữ vào thư mục này. Ngoài thư mục home tất cả các thư mục còn lại đều thuộc về ổ cứng của máy chủ, do đó sẽ bị reset mỗi khi khởi động lại.

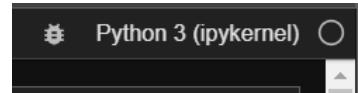
Phía bên phải màn hình là tab Launcher chứa menu các tác vụ để khởi chạy các ứng dụng của Notebook bao gồm: Notebook để tạo một notebook mới chạy theo từng dòng lệnh, Console và Terminal để mở cửa sổ dòng lệnh giống trong hệ điều hành Ubuntu, Text file Markdown file và Python file để tạo file text, markdown và python mới.

A screenshot of a Jupyter Notebook interface. It shows four code cells in a horizontal tab bar: 'Untitled.ipynb', 'comicsub.py', 'test4.py', and 'test3.py'. The first cell contains Python code for a text detection model. The output of the first cell shows several warning messages from PyTorch and TorchVision. The other three cells are empty.

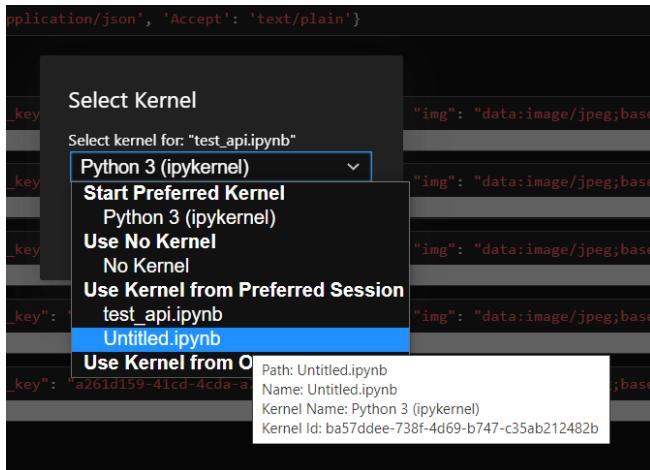
Giao diện notebook

3.1. Mở mới Notebook

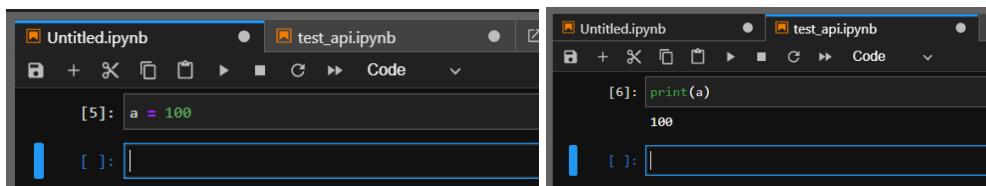
Để mở một Notebook mới bấm vào biểu tượng Python dưới danh mục Notebook trong mục Launcher. Nhân mặc định của notebook là Python3. Lúc này hệ thống sẽ tạo một file Untitled.ipynb trong thư mục home và chạy kernel. Khi 2 notebook được mở thì tương ứng có 2 kernel được mở, điều này khiến cho 2 notebook hoàn toàn tách biệt nhau do đó trong một số trường hợp để liên kết dữ liệu, chia sẻ kernel giữa 2 notebook ta tiến hành thay đổi kernel của 1 trong 2 notebook bằng cách bấm vào biểu tượng



kernel ở góc trên bên trái của notebook sau đó chọn kernel của notebook còn lại trong mục Use Kernel from Preferred Session.



Khi này dữ liệu giữa 2 notebook sẽ được chia sẻ với nhau

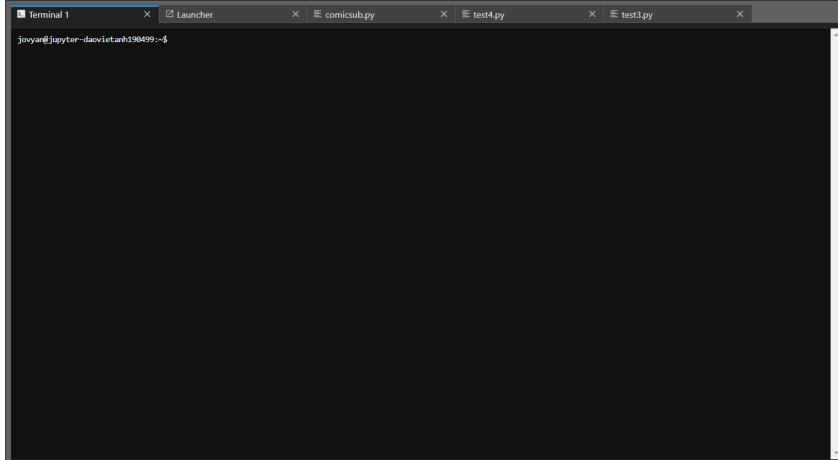


Các ô lệnh trong Notebook được sử dụng để nhập lệnh Python hoặc lệnh hệ thống. Đối với lệnh hệ thống thêm ký tự ! trước mỗi dòng lệnh để phân biệt với lệnh Python. Trong trường hợp muốn chạy một file python có lệnh hiển thị như imshow, plot của matplotlib thì ta chạy file đó trong notebook theo cú pháp %run <tên file python>

3.2. Mở mới Terminal

Kể từ phiên bản Jupyterhub 2.0, giao diện đã được cập nhật hoàn chỉnh, tiện lợi cho terminal và notebook. Để mở mới một terminal bấm vào biểu tượng Terminal trong mục Launcher. Giống với terminal của Ubuntu, terminal của Jupyterhub có đầy đủ các công cụ cơ bản để quản lý thư mục, tập tin, networking và phân quyền user. Terminal của Jupyterhub được khởi động dưới quyền mặc định của user jovyan, có đầy đủ quyền sudo như một tài khoản root. Do Terminal hiển thị bằng thẻ canvas của web

browser do đó sẽ có một số trường hợp bị lỗi font chữ như khoảng cách các chữ cái bị lỗi hoặc không hiển thị được chữ cái thì cần cài font monospace vào hệ điều hành của máy đó.



Giao diện terminal

Một số lệnh terminal để kiểm tra hệ thống:

- df -h : kiểm tra dung lượng ổ đĩa, thư mục, tệp
- free : kiểm tra dung lượng bộ nhớ RAM theo thời gian thực
- lscpu: kiểm tra thông số CPU
- nvidia-smi: kiểm tra thông số GPU NVIDIA
- htop: kiểm tra dung lượng RAM, CPU đã sử dụng theo thời gian thực

Để cài đặt các package mới cho Ubuntu của Notebook bắt buộc phải thực hiện dưới quyền root bằng cách thêm lệnh sudo trước lệnh cài đặt. Các package được cài đặt sẽ bị xóa sau mỗi lần reset notebook do đó để lưu trữ lâu dài các package cần được cài đặt vào thư mục home.

4. Khởi tạo và lưu trữ môi trường ảo

Môi trường ảo có mục đích đóng gói và tách biệt môi trường cho một dịch vụ, ứng dụng hay tác vụ. Môi trường ảo rất quan trọng trong việc tách biệt các package, thư viện để tránh xung đột khi 2 hay nhiều ứng dụng yêu cầu các phiên bản khác nhau của cùng một package, thư viện. Ngoài ra, môi trường ảo giúp lưu trữ trạng thái của môi trường tránh việc cài lại nhiều lần gây mất thời gian, tại vì môi trường notebook cung cấp không thể đáp ứng nhu cầu của toàn bộ người dùng nên việc sử dụng môi trường

ảo là rất quan trọng trong việc giảm thiểu lỗi khi cài đặt và tránh lãng phí thời gian cài đặt. Tài liệu này sẽ hướng dẫn tạo môi trường ảo trên Jupyterhub notebook của VKIST theo 2 phương pháp là sử dụng conda và sử dụng virtualenv.

4.1. Tạo môi trường ảo bằng conda

Conda là phần mềm tạo và quản lý môi trường ảo dành cho python phổ biến hiện nay. Jupyter notebook cũng đã hỗ trợ cài sẵn conda trong môi trường mặc định. Tuy nhiên, các lệnh conda của notebook là chưa đầy đủ, để sử dụng đầy đủ các lệnh của conda bao gồm lệnh activate ta cần kích hoạt conda trong notebook.

Khi chạy conda trên terminal notebook lỗi chưa được kích hoạt của conda sẽ hiện lên

```
jovyan@jupyter-daovietanh190499:~$ conda activate
CommandNotFoundError: Your shell has not been properly configured to use 'conda activate'.
To initialize your shell, run

    $ conda init <SHELL_NAME>

Currently supported shells are:
- bash
- fish
- tcsh
- xonsh
- zsh
- powershell

See 'conda init --help' for more information and options.

IMPORTANT: You may need to close and restart your shell after running 'conda init'.
```

Câu lệnh sẽ hiện ra các gợi ý lựa chọn kích hoạt, do hệ điều hành của notebook là Ubuntu vì vậy ta sẽ chọn option bash. Sau khi chạy lệnh conda init bash ta sẽ thu được kết quả bao gồm đường dẫn đến thư mục gốc của conda.

```
jovyan@jupyter-daovietanh190499:~$ conda init bash
no change    /opt/conda/condabin/conda
no change    /opt/conda/bin/conda
no change    /opt/conda/bin/conda-env
no change    /opt/conda/bin/activate
no change    /opt/conda/bin/deactivate
no change    /opt/conda/etc/profile.d/conda.sh
no change    /opt/conda/etc/fish/conf.d/conda.fish
no change    /opt/conda/shell/condabin/Conda.ps1
no change    /opt/conda/shell/condabin/conda-hook.ps1
no change    /opt/conda/lib/python3.9/site-packages/xontrib/conda.xsh
no change    /opt/conda/etc/profile.d/conda.csh
modified     /home/jovyan/.bashrc

==> For changes to take effect, close and re-open your current shell. <=
jovyan@jupyter-daovietanh190499:~$
```

Tiến hành khởi tạo conda bằng lệnh

\$source /opt/conda/etc/profile.d/conda.sh

Lúc này lệnh activate của conda đã được kích hoạt và có thể sử dụng bình thường

```
jovyan@jupyter-daovietanh190499:~$ conda create -n test
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <=
      current version: 4.12.0
      latest version: 4.13.0

Please update conda by running

$ conda update -n base conda

## Package Plan ##

environment location: /opt/conda/envs/test

Proceed ([y]/n)? y

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
# $ conda activate test
#
# To deactivate an active environment, use
#
# $ conda deactivate

jovyan@jupyter-daovietanh190499:~$ conda activate test
(test) jovyan@jupyter-daovietanh190499:~$
```

Tuy nhiên lệnh khởi tạo môi trường của conda vẫn mặc định vào ô cứng cache của máy host notebook nên sẽ bị reset sau mỗi lần khởi động lại notebook. Để tạo một môi trường mới tại thư mục home của notebook tiến hành chạy lệnh

```
$conda create --prefix /home/jovyan/<env-name>
```

Để kích hoạt môi trường này chạy lệnh

```
$conda activate /home/jovyan/<env-name>
```

Thay thế env-name bằng tên của môi trường muốn khởi tạo.

```
(test) jovyan@jupyter-daovietanh190499:~$ conda create --prefix /home/jovyan/test-new-folder
Collecting package metadata (current_repotdata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
    current version: 4.12.0
    latest version: 4.13.0

Please update conda by running

$ conda update -n base conda


## Package Plan ##

environment location: /home/jovyan/test-new-folder

Proceed ([y]/n)? y

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate /home/jovyan/test-new-folder
#
# To deactivate an active environment, use
#
#     $ conda deactivate

(test) jovyan@jupyter-daovietanh190499:~$ conda activate /home/jovyan/test-new-folder
(/home/jovyan/test-new-folder) jovyan@jupyter-daovietanh190499:~$
```

Như vậy thư mục của môi trường mới đã được chuyển vào thư mục home của notebook và sẽ không bị xóa sau mỗi lần notebook bị reset.

4.2. Tạo môi trường ảo bằng virtualenv

Virtualenv là phần mềm tạo môi trường ảo nhỏ gọn tiện lợi chạy trên nền python. Hiện tại môi trường jupyter notebook chưa cài đặt sẵn phần mềm này.

Để cài đặt virtualenv tiến hành chạy lệnh

```
$ pip install virtualenv
```

Để tạo môi trường mới trong thư mục hiện tại, chạy lệnh dưới

```
$ virtualenv <env-name>
```

thay <env-name> bằng tên môi trường muốn khởi tạo.

```
jovyan@jupyter-daovietanh190499:~$ virtualenv test-venv
created virtual environment CPython3.9.10.final.0-64 in 16196ms
  creator CPython3Posix(dest=/home/jovyan/test-venv, clear=False, no_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=/home/jovyan/.local/share/virtualenv)
    added seed packages: pip==22.1.1, setuptools==62.3.2, wheel==0.37.1
  activators BashActivator,CShellActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator
jovyan@jupyter-daovietanh190499:~$ []
```

Để kích hoạt môi trường mới tạo, chạy lệnh dưới

```
$ source <env-name>/bin/activate
```

thay <env-name> bằng tên môi trường mới khởi tạo.

```
jovyan@jupyter-daovietanh190499:~$ source test-venv/bin/activate
(test-venv) jovyan@jupyter-daovietanh190499:~$
```

Để thoát khỏi môi trường hiện tại chạy lệnh

```
$ deactivate
```

```
(test-venv) jovyan@jupyter-daovietanh190499:~$ deactivate
jovyan@jupyter-daovietanh190499:~$ []
```

Để lấy danh sách và version của từng thư viện đã cài trong môi trường ảo tiến hành chạy lệnh

```
$ pip freeze
```

```
(test-venv) jovyan@jupyter-daovietanh190499:~$ pip freeze
hnswlib==0.6.2
numpy==1.23.1
opencv-python==4.6.0.66
torch==1.12.0
typing_extensions==4.3.0
```

Lưu ý do toàn bộ môi trường của virtualenv được lưu trong 1 thư mục, vì vậy có thể di chuyển, lưu trữ, copy, đóng gói toàn bộ môi trường này sang bất kỳ máy khác với cùng hệ điều hành và có phần mềm virtualenv là có thể chạy môi trường này.