



NARLabs 財團法人國家實驗研究院

國家高速網路與計算中心

National Center for High-performance Computing

[課程錄影檔](#)

[part1](#)

[課程錄影檔](#)

[part2](#)

[課程錄影檔](#)

[part3](#)

GPU HPC 教育訓練

以Yolov9 進行跨節點訓練實務

方育斌 陳威宇 周朝宜

waue0920@gmail.com

Apr.2025

文件



<https://ppt.cc/fWW9Ex>

範例程式

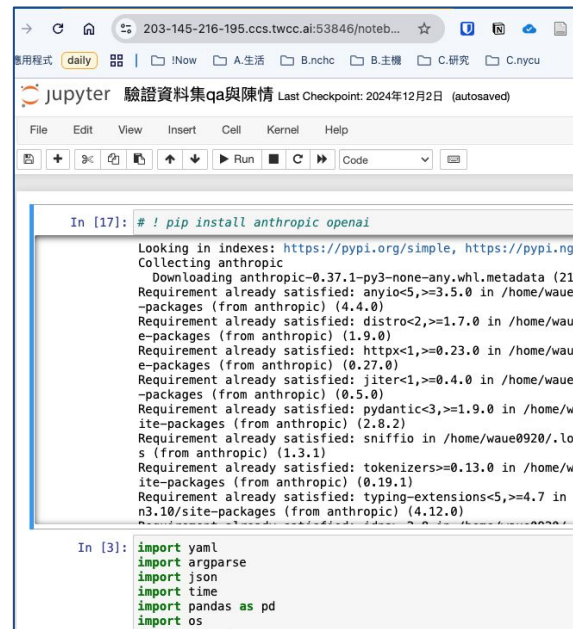


https://github.com/waue0920/nchc_hpc_slurm_example

www.nchc.narlabs.org.tw

使用國網中心 <開發型容器> 運算的日常

- 開container -> jupyter notebook (or vscode)
-> 跑實驗 -> 關 container
- meeting ... -> 被一陣思想輸出 -> 接著上次的實驗做新一輪實驗
- 開container -> jupyter notebook (or vscode)
-> 跑實驗 -> 關 container
-



The screenshot shows a Jupyter Notebook window with a browser address bar at the top. The notebook title is "驗證資料集qa與陳情" and it shows the last checkpoint from 2024年12月2日. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Help) and a toolbar with icons for saving, undo, redo, and running code. The code cell shows the command `! pip install anthropic openai` being executed. The output displays the process of looking in indexes, collecting metadata, and downloading the `anthropic-0.37.1-py3-none-any.whl` file. It also lists several requirements that are already satisfied, such as `anyio<5, >=3.5.0`, `distro<2, >=1.7.0`, `httpx<1, >=0.23.0`, `jiter<1, >=0.4.0`, `pydantic<3, >=1.9.0`, `sniffio`, `tokenizers<0.13.0, >=0.19.1`, and `typing-extensions<5, >=4.7`. Below the first cell, a second code cell is partially visible, showing `import yaml`, `import argparse`, `import json`, `import time`, `import pandas as pd`, and `import os`.

```
In [17]: # ! pip install anthropic openai

Looking in indexes: https://pypi.org/simple, https://pypi.ngc
Collecting anthropic
  Downloading anthropic-0.37.1-py3-none-any.whl.metadata (21
Requirement already satisfied: anyio<5,>=3.5.0 in /home/waue0
-packages (from anthropic) (4.4.0)
Requirement already satisfied: distro<2,>=1.7.0 in /home/waue
e-packages (from anthropic) (1.9.0)
Requirement already satisfied: httpx<1,>=0.23.0 in /home/waue
e-packages (from anthropic) (0.27.0)
Requirement already satisfied: jiter<1,>=0.4.0 in /home/waue0
-packages (from anthropic) (0.5.0)
Requirement already satisfied: pydantic<3,>=1.9.0 in /home/wa
ite-packages (from anthropic) (2.8.2)
Requirement already satisfied: sniffio in /home/waue0920/.loc
s (from anthropic) (1.3.1)
Requirement already satisfied: tokenizers<0.13.0, >=0.19.1 in /home/wa
ite-packages (from anthropic) (0.19.1)
Requirement already satisfied: typing-extensions<5,>=4.7 in /
n3.10/site-packages (from anthropic) (4.12.0)

In [3]: import yaml
import argparse
import json
import time
import pandas as pd
import os
```

- 雖然資料、程式都在, 但 apt install 的程式都要重新裝
 - 記得做snapshot -> 還原
 - 難度 [😞] 感受 [😞]
- 雖然有用 conda 創建各自的運算環境, 但時間久了某些library 出錯, 甚至 jupyter notebook 開不起來
 - 會被要求刪掉 .local .cache , 砍掉重練
 - 難度 [😞😞] 感受[😞😞]
- 雖然最高有8 張 gpu 的container, 但不是always 選得到
 - 戲棚下等久了就是你的
 - 難度 [😞😞] 感受[😞😞😞]
- 我的研究要用到超過 8 張 gpu 卡的記憶體才裝得下
 - 多開幾個 gpu vm 來串成cluster
 - 難度 [😞😞😞] 感受[😞😞]

使用國網HPC服務，打通研究的任督二脈

- ~~雖然資料、程式都在，但 apt install 的程式都要重新裝~~
 - HPC 可以使用封裝好的客製化容器來當執行環境
- ~~雖然有用 conda 創建各自的運算環境，但時間久了某些library 出錯，甚至 jupyter notebook 開不起來~~
 - HPC 的執行環境彼此獨立
- ~~雖然最高有8 張 gpu 的container，但不是always 選得到~~
 - HPC內有大pool, 採用排班機制送job
- ~~我的研究要用到超過 8 張 gpu 卡的記憶體才裝得下~~
 - HPC 天生支援 跨節點運算

- 使用 Container 注意的地方

- 。 運算完後需要自行刪除容器, 以免持續產生費用
- 。 要還原容器環境: 1.先快照 -> 2.選取客製化容器 -> 3.挑選快照 -> 還原容器完成

1

開發型容器詳細資料

配置

監控

映像檔

刪除

重新整理

基本資訊

ID 5098569

名稱 wag121

2

Custom Image ⓘ

TWCC提供基本的容器作業環境，並提供容器複本服務，能打造專屬的容器，相同的工作環境也可迅速建立。

3

映像檔 * ⓘ

基本設定 * ⓘ

pytorch-24.08-py3:llmdata20241213

pytorch-21.06-py3:yolo9ngc2106t19

pytorch-21.06-py3:fedcoin20241122

這門課會講解

- * 國網中心 HPC 平台介紹
- * Slurm 任務派送與瀏覽
 - 3 種sbatch 非pytorch 任務
- * 使用大型專案 Yolov9 跨節點平行訓練
 - 程式運作流程
 - 多個 pytorch 任務
- * Singularity 製作與運作原理
 - 3種製作 sif 方法

預期上完這門課你會

- * 了解國網中心的HPC平台
- * 了解提交與查看 Slurm 任務與執行日誌
- * 如何將實際的大型專案yolov9跨節點運算
- * 如何用singularity打包需要的計算環境到平台上使用

預期你已經有

- 有iService 帳號, 且有計畫
- 熟悉 Linux 指令與SSH連線
 - vim文字編輯器

這門課不會提到

- Python / Pytorch 程式碼
- Yolov9 電腦視覺技術原理
- 平行分散任務技術細節
- mpi / openacc 等使用gpu 方式
- VS code IDE 整合工具開發

國網其他補充課程

- CPU 主機課程
 - 實體 8hrs / 線上 2 hrs
- GPU 主機課程
 - 實體 8hrs / 線上 2 hrs

有效計畫

https://iservice.nchc.org.tw/module_page.php?module=nchc_service#nchc_service/nchc_service.php?action=nchc_motp_unix_account_edit_v3

服務狀態啟用

iService登入->會員中心->計畫管理->我的計畫
點選某一計畫->服務啟用狀態

iService登入->會員中心 -> 計畫管理 ->我的計畫



NARLabs 財團法人國家實驗研究院

國家高速網路與計算中心

National Center for High-performance Computing

國網中心HPC跨節點運算 (一)

[課程錄影檔part1](#)

國網中心 陳威宇

waue0920@gmail.com

Mar.2025

文件



<https://ppt.cc/fWW9Ex>

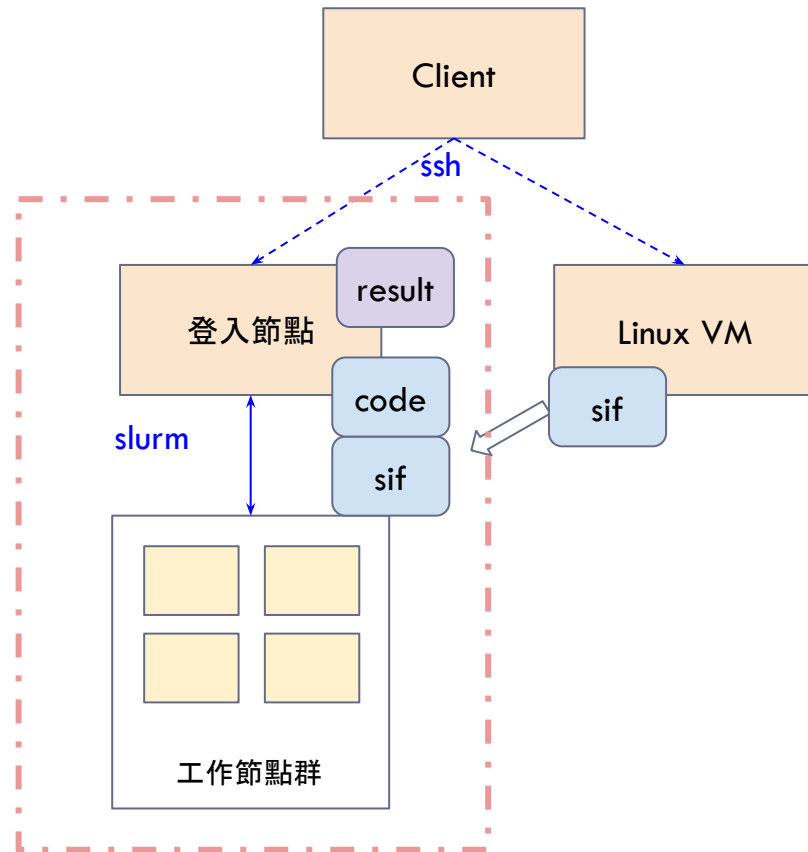
範例程式



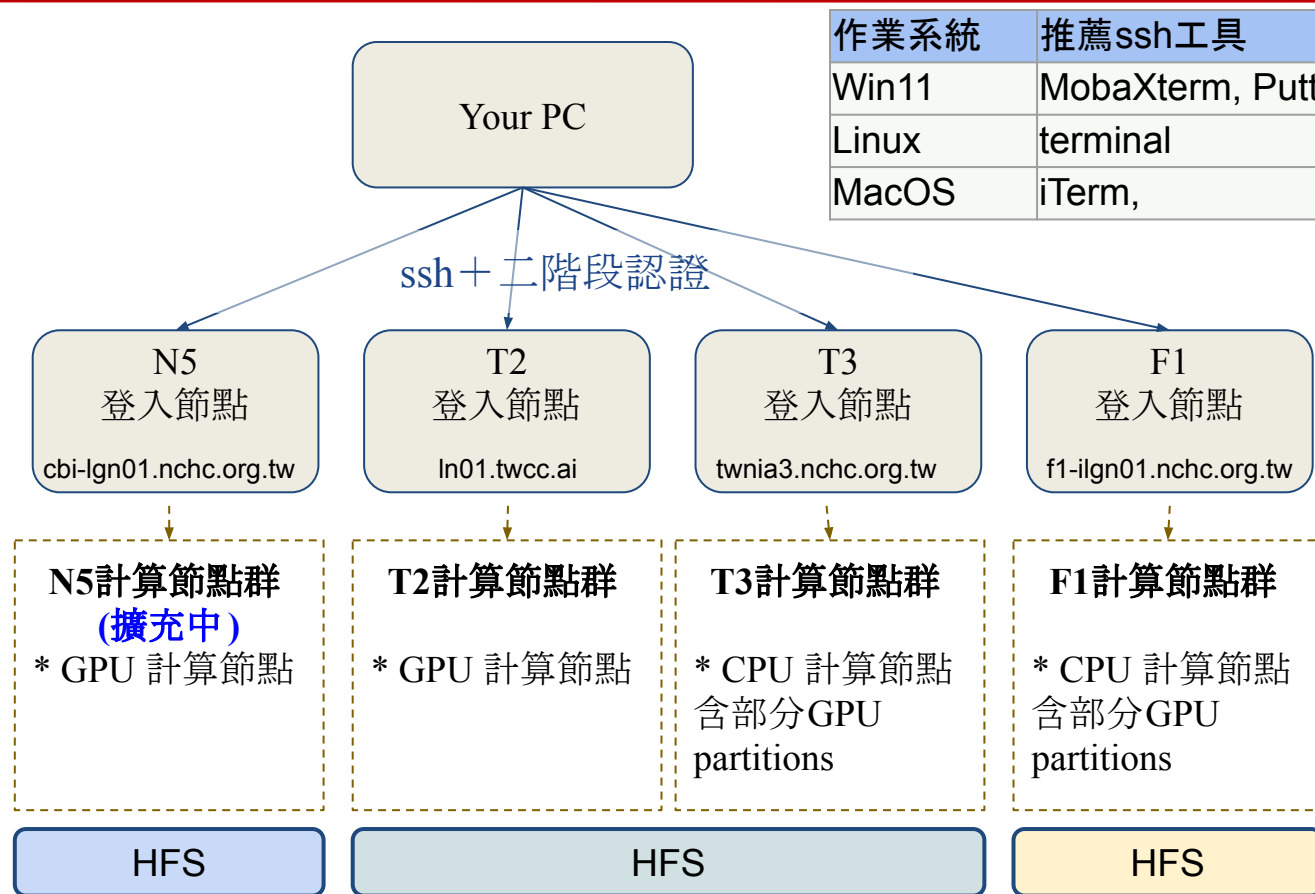
https://github.com/waue0920/nchc_hpc_slurm_example

www.nchc.narlabs.org.tw

- 引言
- NCHC 服務一覽表
- NCHC GPU 運算介紹
- Slurm 介紹
- TWCC HPC Jobs
- 動手做
- 提醒與整理



國網中心 運算服務 一覽表



作業系統	推薦ssh工具	執行方式
Win11	MobaXterm, Putty	圖形化介面
Linux	terminal	指令 ssh
MacOS	iTerm,	指令 ssh

本課程的重點：

- * 如何在國網中心的HPC平台上, 提交Slurm 任務
- * 如何將實際的大型專案放到HPC平台上計算
- * 如何打包需要的計算環境到HPC平台上使用

T1	T2 TWCC	T3	F1	N5 晶創主機	...
台灣衫一號 <ul style="list-style-type: none">已下線cpu 主機slurm	台灣衫二號 (TWCC) <ul style="list-style-type: none">gpu 主機2016張 NVIDIA V100 卡slurm + VM + container	台灣衫三號 <ul style="list-style-type: none">900 cpu nodesslurm	創進一號 <ul style="list-style-type: none">558 cpu nodesslurm	晶創主機 <ul style="list-style-type: none">gpu 主機NVIDIA H100slurm持續擴充中	

運算

開發型容器

虛擬運算 (VCS)

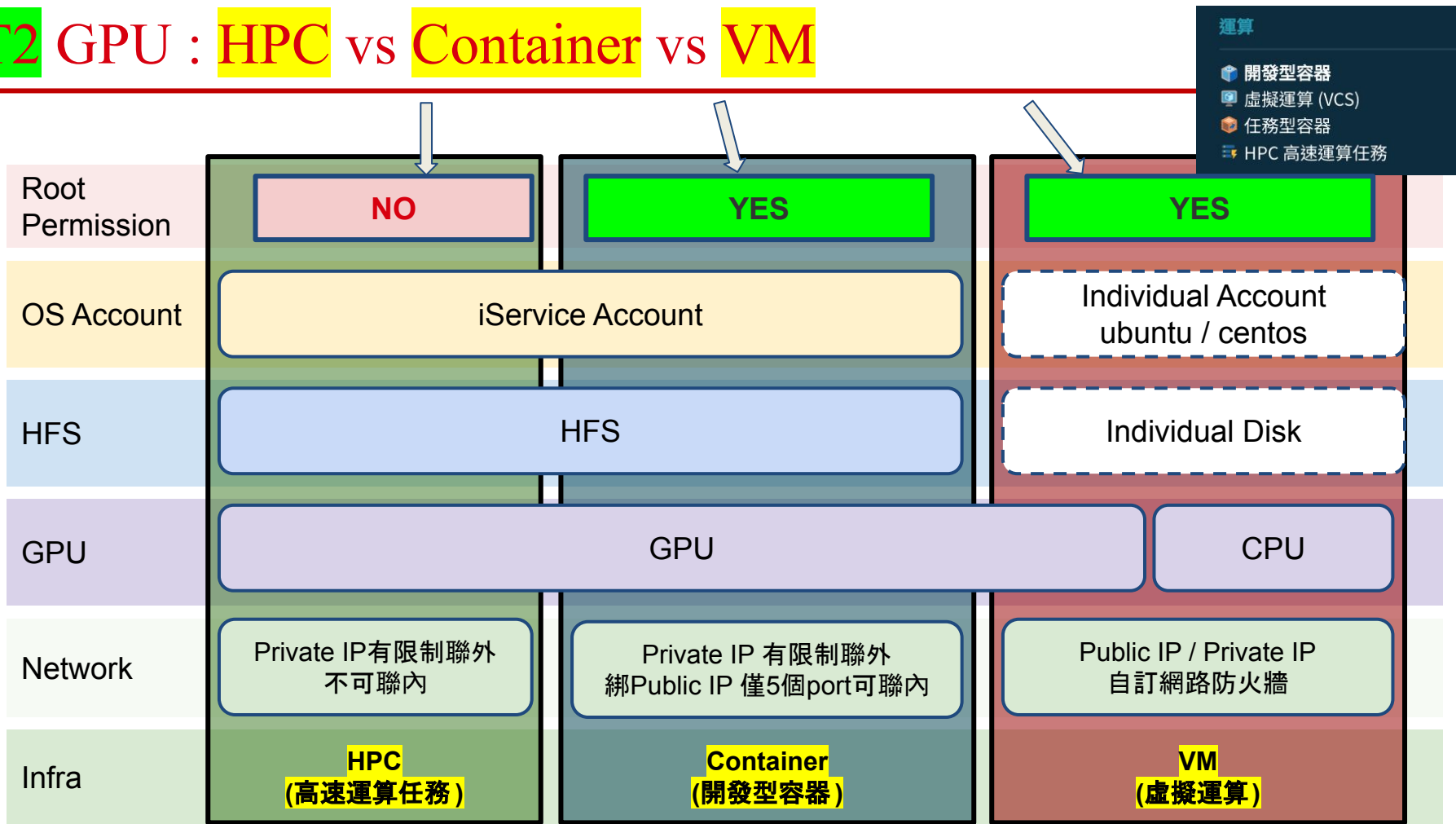
任務型容器

HPC 高速運算任務

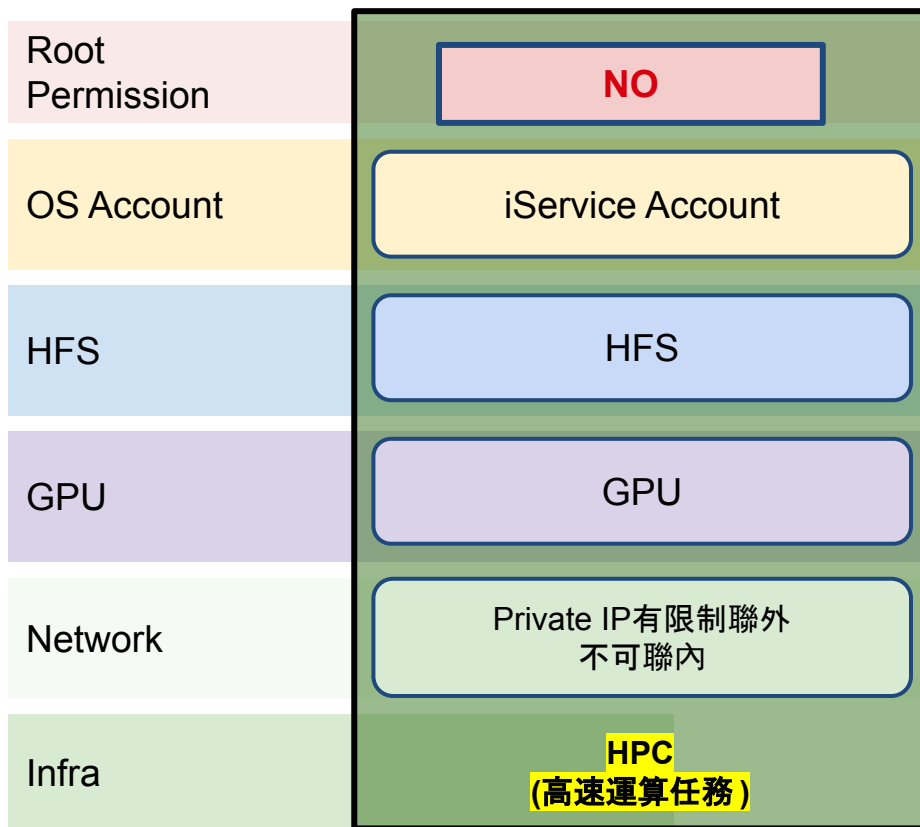
- 國網中心仿照 colab 的概念設計 T2 的使用環境
- 超級電腦的計算領域中，還是需要用排班程式 (slurm) 來負責執行管理用戶的運算

資料日期 (2024.12.31)
[2] <https://www.nchc.org.tw/Page?itemid=6&mid=10>
[3] <https://man.twcc.ai/@twnia3/rJM5qk3Aw>
[4] https://iservice.nchc.org.tw/nchc_service/nchc_service_forerunnerone.php
[5] https://iservice.nchc.org.tw/nchc_service/nchc_service_news_content.php?contentId=1007571&type=all_content&newsId=59649

T2 GPU : HPC vs Container vs VM



- 運算
- 開發型容器
 - 虛擬運算 (VCS)
 - 任務型容器
 - HPC 高速運算任務



A 晶創主機之正式名稱為晶創25（英文：NANO 5），將於 2025年3月20日正式收費資源，有助於生成式AI技術研發與應用服務。主機規格相關內容，請參考 [國網中心](#) 以下提供節點規格、收費方式、使用說明與注意事項，敬請用戶參考。

[使用額度與收費費率]

帳號之使用額度與收費標準皆以GPU小時為計算服務單位

GPU小時 (GPU小時) = 執行小時數 × GPU 數

[節點規格] 每台計算節點規格：

- 8 片 NVIDIA H100 GPU
- 8 個 InfiniBand 400Gb/s 網路埠
- 2TB 記憶體

[收費方式] 各類計畫之費率：(單位：元)

計畫類型	每GPU小時費率
國科會計畫	25
學術計畫	50
政府與法人計畫	50
企業與個人計畫	120

https://iservice.nchc.org.tw/nchc_service/nchc_service_ga.php?target=54

- Slurm (Simple Linux Utility for Resource Management) 是一個開源的、具有容錯性和高度可擴展的資源管理和作業調度系統
- 它主要用於大型和小型Linux集群，提供資源分配、作業排程和監控等功能
- Top500中有60%使用Slurm

排名	主機名稱	國家	排班工具	網址
1	El Capitan	美國	Slurm	El Capitan
2	Frontier	美國	Slurm	Frontier
3	Aurora	美國	Slurm	Aurora
4	LUMI	芬蘭	Slurm	LUMI
5	Fugaku	日本	富士通專有系統	Fugaku
6	Perlmutter	美國	Slurm	Perlmutter
7	Selene	美國	Slurm	Selene
8	JUWELS Booster Module	德國	Slurm	JUWELS
9	HPC5	意大利	PBS Pro	HPC5
10	SuperMUC-NG	德國	Slurm	SuperMUC-NG

以上資料參考自2024年11月的TOP500列表，參考資料：

[1] Slurm官網: <https://slurm.schedmd.com/documentation.html>

[2] TOP500: <https://www.top500.org/>

[3] Wikipedia - TOP500: <https://en.wikipedia.org/wiki/TOP500>

[4] Data Center Dynamics: <https://www.datacenterdynamics.com/>

- 特色：

- 全局管理與任務管控：由 Slurm 提供平台的統一管理與排程。
- 任務派送：透過排班機制 (Job Scheduling) 完成任務分配。
- 容器化環境支持：由 Singularity 提供高效執行環境。
 - why not docker ? => singularity 可攜式性比 docker 更好

- 優點：

- 跨節點運算
- 更經濟有效地使用資源
- 送job簡便，方便做微調參數的實驗

- 缺點：

- 登入節點非運算節點，job 不會即時執行
- 無法取得 root
 - 故環境要先用 Singularity 包好

```
(base) waue0920@un-ln01:~/slurm/t2yolov9$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE MODELIST
gtest      up        30:00      2    idle gn[1001-1002]
gp1d       up    1-00:00:00     15    mix gn[0602-0603,0606-0608,0611,0613,0718,0817,1004-1009]
gp1d       up    1-00:00:00     19    idle gn[0409,0415,0417,0502-0503,0505-0506,0513-0514,0610,0615-0617,0703,0707,0709,0812,0815,1010]
gp2d*      up    2-00:00:00     15    mix gn[0602-0603,0606-0608,0611,0613,0718,0817,1004-1009]
gp2d*      up    2-00:00:00     19    idle gn[0409,0415,0417,0502-0503,0505-0506,0513-0514,0610,0615-0617,0703,0707,0709,0812,0815,1010]
gp4d       up    4-00:00:00     15    mix gn[0602-0603,0606-0608,0611,0613,0718,0817,1004-1009]
gp4d       up    4-00:00:00     19    idle gn[0409,0415,0417,0502-0503,0505-0506,0513-0514,0610,0615-0617,0703,0707,0709,0812,0815,1010]
express    up    4-00:00:00     15    mix gn[0602-0603,0606-0608,0611,0613,0718,0817,1004-1009]
express    up    4-00:00:00     19    idle gn[0409,0415,0417,0502-0503,0505-0506,0513-0514,0610,0615-0617,0703,0707,0709,0812,0815,1010]
```

T2

詞彙	定義
Partition	指一組以對大時間長度為區隔的計算節點，例如 gp1d, gp2d, p4d 等。
Timelimit	指派送至叢集的作業允許運行的最大時間長度。如果作業超過時間限制，它將被 Slurm 終止。
Node	HPC 叢集中的單一計算機。
State	指節點的當前狀態，例如閒置 (idle)、運算中 (mix) 或關閉 (down)。
Nodelist	節點名稱，例如：ln01.twcc.ai

```
(base) [waue0920@lgn01 salloc_4331 bin]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE MODELIST
dev        up        2:00:00     12    mix hgpn[04-06,13-21]
normal     up    2-00:00:00     12    mix hgpn[04-06,13-21]
taide      up    3-00:00:00     12    mix hgpn[04-06,13-21]
trust      up    2-00:00:00      1    idle hgpn01
```

N5

動手做: 登入 HPC 登入節點

<https://man.twcc.ai/@twccdocs/doc-twnia2-main-zh/https%3A%2F%2Fman.twcc.ai%2F%40twccdocs%2Fguide-twnia2-login-and-logout-zh>

NAR Labs

- 登入節點、iservice 帳號
 - 登入到 ln01.twcc.ai (需二階段認證)

登入後, 輸入\$ 後的指令

```
$ sinfo
$ wallet
```

```
ssh waue0920@ln01.twcc.ai
Warning: Permanently added 'ln01.twcc.ai' (ED25519) to the list of known hosts.

#####
#
#      ~ WELCOME TO TAIWANIA 2 ~
#
#      How to log in?
#      https://docs.twcc.ai/docs/login-logout/
#
#####

(waue0920@ln01.twcc.ai) Please select the 2FA login method.
1. Mobile APP OTP
2. Mobile APP PUSH
3. Email OTP
Login method: 2
(waue0920@ln01.twcc.ai) Password:
Please check your push token.
[PASS] The push verification succeeded.
Last login: Thu Jan  2 15:23:57 2025 from 140.110.136.70
```

```
(base) waue0920@un-ln01:~$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
gtest      up       30:00      2   idle gn[1001-1002]
gp1d       up  1-00:00:00    12  mix  gn[0602-0603,0718,0815,0817,1004-1010]
gp1d       up  1-00:00:00    22  idle gn[0409,0415,0417,0502-0503,0505-0506,0513-0514,0606-0608,0610-0611,0613,0615-0617,0703,0707,0709,0812]
gp2d*      up  2-00:00:00    12  mix  gn[0602-0603,0718,0815,0817,1004-1010]
gp2d*      up  2-00:00:00    22  idle gn[0409,0415,0417,0502-0503,0505-0506,0513-0514,0606-0608,0610-0611,0613,0615-0617,0703,0707,0709,0812]
gp4d       up  4-00:00:00    12  mix  gn[0602-0603,0718,0815,0817,1004-1010]
gp4d       up  4-00:00:00    22  idle gn[0409,0415,0417,0502-0503,0505-0506,0513-0514,0606-0608,0610-0611,0613,0615-0617,0703,0707,0709,0812]
express    up  4-00:00:00    12  mix  gn[0602-0603,0718,0815,0817,1004-1010]
express    up  4-00:00:00    22  idle gn[0409,0415,0417,0502-0503,0505-0506,0513-0514,0606-0608,0610-0611,0613,0615-0617,0703,0707,0709,0812]
(base) waue0920@un-ln01:~$ wallet
INFO: If there are many projects, you may need to wait about 5 seconds
INFO: Specify a project, it should be faster. `wallet $PROJECT_ID`
PROJECT_ID: GOV113 8, PROJECT_NAME: 運用GPU發展 評估, SU_BALANCE: 698
```

登入節點 -> HFS -> 共用檔案

```
$ ls /work/TWCC_cntr
```

登入後，輸入\$ 後的指令

```
$ ls -al /work/waue0920/open_access/
```

```
(base) waue0920@un-ln01:~$ ls -al /work/waue0920/open_access/
total 112706917
drwxr-xr-x+ 2 waue0920 TRI107129      4096 Mar 20 16:39 .
drwxr-xr-x+ 6 waue0920 TRI107129      4096 Feb 15 20:59 ..
-rwxr-xr-x  1 waue0920 TRI107129 8827060224 Feb 15 18:31 gyolo_ngc2306_20250215.sif
-rwxr-xr-x  1 waue0920 TRI107129 8827625472 Feb 17 12:20 gyolo_ngc2306_20250217.sif
-rwxr-xr-x  1 waue0920 TRI107129 8830595072 Feb 18 00:01 gyolo_ngc2306_20250218.sif
-rwxr-xr-x  1 waue0920 TRI107129 8830562304 Mar 10 11:41 gyolo_ngc2306_20250310.sif
-rwxr-xr-x  1 waue0920 TRI107129 8990957568 Mar 13 14:13 gyolo_ngc2306_20250313.sif
lrwxrwxrwx  1 waue0920 TRI107129      26 Mar 20 16:37 gyolo_ngc2306.sif -> gyolo_ngc2306_20250313.sif
-rwxr-xr-x  1 waue0920 TRI107129 9241235456 Aug 29 2024 nemo_ngc2405_20240829.sif
-rwxrwxr-x  1 waue0920 TRI107129 25394143232 Sep  3 2024 nemo_ngc2407_20240901.sif
lrwxrwxrwx  1 waue0920 TRI107129      25 Mar 20 16:38 nemo.sif -> nemo_ngc2407_20240901.sif
-rwxr-xr-x  1 waue0920 TRI107129 7000821760 Nov 11 17:42 ngc2111_cv28074.sif
-rwxr-xr-x  1 waue0920 TRI107129 5390626816 Feb 18 15:11 openacc_nvidia_20250218.sif
-rwxr-xr-x  1 waue0920 TRI107129 8528822272 Dec 25 09:37 pytorch_23.06.sif
-rwxr-xr-x  1 waue0920 TRI107129 6864506880 Nov 15 17:55 yolo9t1_ngc2111_20241115.sif
-rwxr-xr-x  1 waue0920 TRI107129 8684834816 Dec 26 15:40 yolo9t2_ngc2306_20241226.sif
lrwxrwxrwx  1 waue0920 TRI107129      28 Mar 20 16:39 yolo9t2_ngc2306.sif -> yolo9t2_ngc2306_20241226.sif
```

動手做: Slurm - 平台狀況 查看與取消

// 查看叢集狀態
\$ sinfo

// 查看任務狀態
\$ squeue
\$ sacct

輸入指令試看看

```
(base) waue0920@un-ln01:~$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE MODELIST
gtest      up        30:00      2     idle gn[1001-1002]
gp1d       up        1-00:00:00  27    mix  gn[0308,0503,0601-0603,0607-0608,0610-0611,0613,0615-0617,0703,0707,0709,0812,0815,0817,1003-1010]
gp1d       up        1-00:00:00   8     idle gn[0408,0413,0506-0507,0514-0515,0605,0718]
gp2d*      up        2-00:00:00  27    mix  gn[0308,0503,0601-0603,0607-0608,0610-0611,0613,0615-0617,0703,0707,0709,0812,0815,0817,1003-1010]
gp2d*      up        2-00:00:00   8     idle gn[0408,0413,0506-0507,0514-0515,0605,0718]
gp4d       up        4-00:00:00  27    mix  gn[0308,0503,0601-0603,0607-0608,0610-0611,0613,0615-0617,0703,0707,0709,0812,0815,0817,1003-1010]
gp4d       up        4-00:00:00   8     idle gn[0408,0413,0506-0507,0514-0515,0605,0718]
express    up        4-00:00:00  27    mix  gn[0308,0503,0601-0603,0607-0608,0610-0611,0613,0615-0617,0703,0707,0709,0812,0815,0817,1003-1010]
express    up        4-00:00:00   8     idle gn[0408,0413,0506-0507,0514-0515,0605,0718]
(base) waue0920@un-ln01:~$ squeue
      JOBID PARTITION  NAME      USER  ST      TIME  NODES MODELIST(REASON)
      704680 gp1d      bash     s6300121 R       21:47      1 gn0608
      704563 gp4d      FourCast u1673339 R      3:57:15      1 gn1009
      704672 gp2d      tf_vit   u6477228 R       34:41      1 gn0308
      704582 gp2d      run.sh   u3237749 R      3:23:20      1 gn1004
      704652 gp4d      hello    u1976090 R       1:02:00      1 gn1003
      704399 gp4d      FADA_ipi jgtf0322 R      20:16:54      1 gn0308
      704522 gp4d      FADA_ipi jgtf0322 R       5:34:44      1 gn1010
      703623 gp4d      lsImL    u5671665 R      2-02:58:49      4 gn[0815,0817,1005,1008]
      704592 gp4d      sugar    u5185961 R       2:59:10      1 gn1010
      704591 gp1d      sugar    u5185961 R       3:01:08      1 gn1010
      704594 gp4d      sugar    u5185961 R       2:58:10      1 gn1010
      704595 gp4d      SLEET_ga sean0204 R       2:47:35      1 gn1004
      703561 gp4d      sugar    caohieu9 R      2-05:58:41      1 gn1007
      703113 gp4d      sleep    c00cjz00 R      3-13:08:28      1 gn0308
      702953 gp4d      qwq32b   c00cjz00 R      3-23:29:36      1 gn1006

(base) waue0920@un-ln01:~$ sacct
JobID      JobName    Partition      Account      AllocCPUS      State ExitCode
-----
(base) waue0920@un-ln01:~$
```

動手做: Slurm 派送任務

```
// 下載課程練習專案
```

```
$ cd ~
```

```
$ git clone https://github.com/waue0920/nhc_hpc_slurm_example
```

```
$ cd nhc_hpc_slurm_example/twcc/example1_checkenv
```

```
// 派送任務:
```

```
$ sbatch 1check_gpu_single.sb
```

```
// 查看執行狀況
```

```
$ cat 1check_gpu_single.sb
```

```
$ sacct
```

```
// 取消任務
```

```
$ scancel <s_pid>
```

本範例預設使用 TWCC

https://github.com/waue0920/nhc_hpc_slurm_example/blob/main/twcc/example1_checkenv/1check_gpu_single.sb

- 觀察參數
- 觀察log
- 觀察運作機制
- 觀察各種參數的變化

```
(base) waue0920@un-ln01:~/nhc_hpc_slurm_example/twcc/example1_checkenv$ sacct
```

JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
704694	check_gpu	gtest	gov113038	32	COMPLETED	0:0
704694.batch	batch		gov113038	16	COMPLETED	0:0
704694.exte+	extern		gov113038	32	COMPLETED	0:0
704700	check_gpu	gp2d	gov113038	160	COMPLETED	0:0
704700.batch	batch		gov113038	32	COMPLETED	0:0
704700.exte+	extern		gov113038	160	COMPLETED	0:0

程式碼 ex1

```
#!/bin/bash
#SBATCH -A <你的計畫錢包>          ## 錢包id
#SBATCH --job-name=check_gpu        ## 工作名稱
#SBATCH --output=z_1check.log        ## 標準輸出與錯誤輸出同時記錄到此檔案
#SBATCH --error=z_1check.log        ## 標準錯誤輸出記錄同一檔案
#SBATCH --nodes=2                   ## 請求n個節點
#SBATCH --gres=gpu:4                ## 請求n張GPU
#SBATCH --cpus-per-task=16          ## 單個任務請求n 個 CPU (GPU的4倍)
#SBATCH --time=00:10:00             ## 最長執行時間 n 分鐘 (可不用)
#SBATCH --partition=gtest           ## 測試分區 (gtest / gp2d / gp4d / express)
```

```
# 載入必要的模組 (根據叢集需求)
module purge
module load singularity
```

```
echo "=====
echo "1. 節點資訊"
echo "目前執行節點: $(hostname)"
```

```
echo "=====
echo "2. GPU 數量"
nvidia-smi --list-gpus
```

```
echo "=====
echo "3. SLURM 系統參數"
scontrol show job $SLURM_JOB_ID
```

```
echo "=====
echo "4. SLURM Nodes"
echo "Allocated Nodes: $SLURM_JOB_NODELIST"
```

```
echo "=====
echo "5. SLURM 環境參數"
env | grep SLURM
```

```
echo "=====
```

輸出

1. 節點資訊

目前執行節點: gn0506.twcc.ai

2. GPU 數量

```
GPU 0: Tesla V100-SXM2-32GB (UUID: GPU-901e2d66-2839-61e6-d9d1-845090cbbce9)
GPU 1: Tesla V100-SXM2-32GB (UUID: GPU-e963edad-bf72-feff-dadd-850de44bf59b)
GPU 2: Tesla V100-SXM2-32GB (UUID: GPU-61868fd5-4c44-1da8-d42f-4f1ebff5781f)
GPU 3: Tesla V100-SXM2-32GB (UUID: GPU-718c9cd8-3f39-5a0c-a1b7-ffaa25f5e754)
GPU 4: Tesla V100-SXM2-32GB (UUID: GPU-3dcddc80-3ecb-c74b-d2aa-274160e944bf)
GPU 5: Tesla V100-SXM2-32GB (UUID: GPU-8527990b-7b78-dd53-f3ad-0b22dc42f441)
GPU 6: Tesla V100-SXM2-32GB (UUID: GPU-2eec042a-3b48-2ccb-07c5-3fca08e9c826)
GPU 7: Tesla V100-SXM2-32GB (UUID: GPU-0360b7da-e210-3128-698f-9d4542336177)
```

3. SLURM 系統參數

```
JobId=704700 JobName=check_gpu
UserId=waue0920(10191) GroupId=TRI107129(3278) MCS_Label=N/A
Priority=10063310 Nice=0 Account=gov113038 QOS=normal
JobState=RUNNING Reason=None Dependency=(null)
Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
RunTime=00:00:02 TimeLimit=00:10:00 TimeMin=N/A
SubmitTime=2025-03-20T17:13:11 EligibleTime=2025-03-20T17:13:48
AccrueTime=2025-03-20T17:13:48
StartTime=2025-03-20T17:13:48 EndTime=2025-03-20T17:23:48 Deadline=N/A
PreemptTime=None SuspendTime=None SecsPreSuspend=0
LastSchedEval=2025-03-20T17:13:48
Partition=gp2d AllocNode:Sid=un-ln01:593115
ReqNodeList=(null) ExcNodeList=(null)
NodeList=gn[0506-0507,0514-0515,0605]
BatchHost=gn0506
NumNodes=5 NumCPUs=160 NumTasks=5 CPUs/Task=32 ReqB:S:C:T=0:0:*:*
TRES=cpu=160,mem=3600G,node=5,billing=160,gres/gpu=40
Socks/Node=* NtasksPerN:B:S:C=1:0:*:* CoreSpec=*
MinCPUsNode=32 MinMemoryNode=720G MinTmpDiskNode=0
Features=(null) DelayBoot=00:00:00
OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
Command=/home/waue0920/waue/git/nchc_hpc_slurm_example/twcc/example1_checkenv/1check_gpu_single.sb
WorkDir=/home/waue0920/waue/git/nchc_hpc_slurm_example/twcc/example1_checkenv
StdErr=/home/waue0920/waue/git/nchc_hpc_slurm_example/twcc/example1_checkenv/z_1check.log
StdIn=/dev/null
StdOut=/home/waue0920/waue/git/nchc_hpc_slurm_example/twcc/example1_checkenv/z_1check.log
Power=
TresPerNode=gpu:8
```

4. SLURM Nodes

Allocated Nodes: gn[0506-0507,0514-0515,0605]

5. SLURM 環境參數

```
SLURM_NODEID=0
SLURM_TASK_PID=3523965
SLURM_PRIO_PROCESS=0
SLURM_SUBMIT_DIR=/home/waue0920/waue/git/nchc_hpc_slurm_example/twcc/example1_checkenv
SLURM_CPUS_PER_TASK=32
SLURM_CHECKPOINT_IMAGE_DIR=/var/slurm/checkpoint
SLURM_PROCID=0
SLURM_JOB_GID=3278
SLURMD_NODENAME=gn0506
SLURM_TASKS_PER_NODE=1(x5)
```

waue0920 init the project

程式碼 ex2

Code Blame 43 lines (33 loc) · 1.39 KB

```
1 #!/bin/bash
2 #SBATCH -A GOV113038
3 #SBATCH --job-name=check_gpu ## 工作名稱
4 #SBATCH --output=z_2check_env.log ## 標準輸出與錯誤輸出同時記錄到此檔案
5 #SBATCH --error=z_2check_env.log ## 標準錯誤輸出記錄同一檔案
6 #SBATCH --nodes=2 ## 請求n個節點
7 #SBATCH --gres=gpu:4 ## 請求n張 GPU
8 #SBATCH --cpus-per-task=16 ## 單個任務請求 n 個 CPU
9 #SBATCH --time=00:10:00 ## 最長執行時間 n 分鐘
10 #SBATCH --partition=gtest ## 測試分區
11
12
13 # 模組載入 (根據叢集需求)
14 module purge
15 module load singularity
16
17
18 echo "=====
19 echo "BatchNode : $(hostname)"
20 echo "=====
21 echo "
22
23 * GPU 資源檢查"
24 echo "SLURM_JOB_GPUS=$SLURM_JOB_GPUS"
25 echo "SLURM_GPUS_PER_NODE=$SLURM_GPUS_PER_NODE"
26 echo "SLURM_GPUS_ON_NODE=$SLURM_GPUS_ON_NODE"
27 echo ""
28 echo "可用 GPU 檢查 (nvidia-smi)"
29 nvidia-smi --list-gpus
30
31
32 echo "=====
33 # 執行 `env.sh`, 將環境資訊記錄到 log
34
35 ## * 這指令未帶入gpu 資源
36 # srun --mpi=pmix $SINGULARITY bash env.sh
37
38 ## * 這指令在 t2 會錯誤, H100可執行, 因為 $SLURM_GPUS_ON_NODE
39 # srun --gres=gpu:$SLURM_GPUS_ON_NODE --mpi=pmix bash env.sh
40
41 ## * 這指令要解決 t2 上沒有 $SLURM_GPUS_ON_NODE 而做
```

waue0920 init the project

程式碼 ex3

d018636 · last month History

Code Blame 37 lines (29 loc) · 1.25 KB

```
1 #!/bin/bash
2 #SBATCH -A GOV113038 ## 錢包id
3 #SBATCH --job-name=check_gpu ## 工作名稱
4 #SBATCH --output=z_3check_env.log ## 標準輸出與錯誤輸出同時記錄到此檔案
5 #SBATCH --error=z_3check_env.log ## 標準錯誤輸出記錄同一檔案
6 #SBATCH --nodes=2 ## 請求n個節點
7 #SBATCH --gres=gpu:4 ## 請求n張 GPU
8 #SBATCH --cpus-per-task=16 ## 單個任務請求 n 個 CPU
9 #SBATCH --time=00:10:00 ## 最長執行時間 n 分鐘
10 #SBATCH --partition=gtest ## 測試分區
11
12
13 # 模組載入 (根據叢集需求)
14 module purge
15 module load singularity
16
17 # sif
18 SIF=/work/waue0920/open_access/yolo9t2_ngc2306_def-20241226.sif
19 SINGULARITY="singularity run --nv $SIF"
20
21 # defind master
22 MASTER_ADDR=$(scontrol show hostname $SLURM_NODENAME | head -n 1)
23 export MASTER_ADDR
24
25 ## twcc 不知為何SLURM_GPUS_ON_NODE 為空
26 nvidia-smi --list-gpus
27 NGPU=$(nvidia-smi -L | wc -l) # $SLURM_GPUS_ON_NODE
28 export NGPU
29
30 echo "=====
31
32 ## 呼叫 yolo train torchrun 版本
33 # srun --gres=gpu:$SLURM_GPUS_ON_NODE --mpi=pmix $SINGULARITY bash yolotrain_segment.sh #
34
35 cmd="srun --gres=gpu:$NGPU --mpi=pmix $SINGULARITY bash run_python.sh"
```

實作提醒與限制

- Slurm job 的計價 = 從任務開始執行 -> 任務結束
- 登入節點可以**看到**別人share的檔案,
 - ex: everyone can read /work/waue0920/open_access/yolo9t2_ngc2306_20241226.sif
- HPC 上每個人送的job有限制
 - T2 上: 每account 最多 5 node x 8gpu (最長4天)
 - gtest 只能 2 node x 8gpu (最長 30min)
 - N5 上: 每個account 最多 4 node x 8gpu (最長2天)
 - dev 只能 2 node x 8gpu (最長30min)

Queue 名稱	最長執行時間 (小時)	高優先權	每位用戶最多可提交計算工作數上限	適用計畫	節點類型	每位用戶最多可取用 GPU 數上限
gp1d	24		20	各式計畫	GPU 計算節點	40
gp2d	48		20	各式計畫	GPU 計算節點	40
gp4d	96		20	各式計畫	GPU 計算節點	40
gtest	0.5		5	各式計畫	GPU 計算節點	40
express	96	v	20	企業與個人計畫	GPU 計算節點	256

<https://man.twcc.ai/@twccdocs/doc-twnia2-main-zh/https%3A%2F%2Fman.twcc.ai%2F%40twccdocs%2Fguide-twnia2-queue-zh>

佇列名稱	每個計畫最多可用GPU總數	每個Job最大執行時間	每個計畫同一時間	
			最多可執行job的數量	最多可進到主機排隊的工作數
dev	8	2 (小時)	2	2
normal	16	48 (小時)	2	2
taide	40	72 (小時)	5	10

依據你的計畫(Account)來選擇可用的partition。
表格更新日期為2025/01/10。

<https://man.twcc.ai/aPiCU8VXS7SZgFJBOoSqBQ>



NARLabs 財團法人國家實驗研究院

國家高速網路與計算中心

National Center for High-performance Computing

國網中心HPC跨節點運算 (二)

[課程錄影檔part2](#)

國網中心 陳威宇

waue0920@gmail.com

文件



<https://ppt.cc/fWW9Ex>

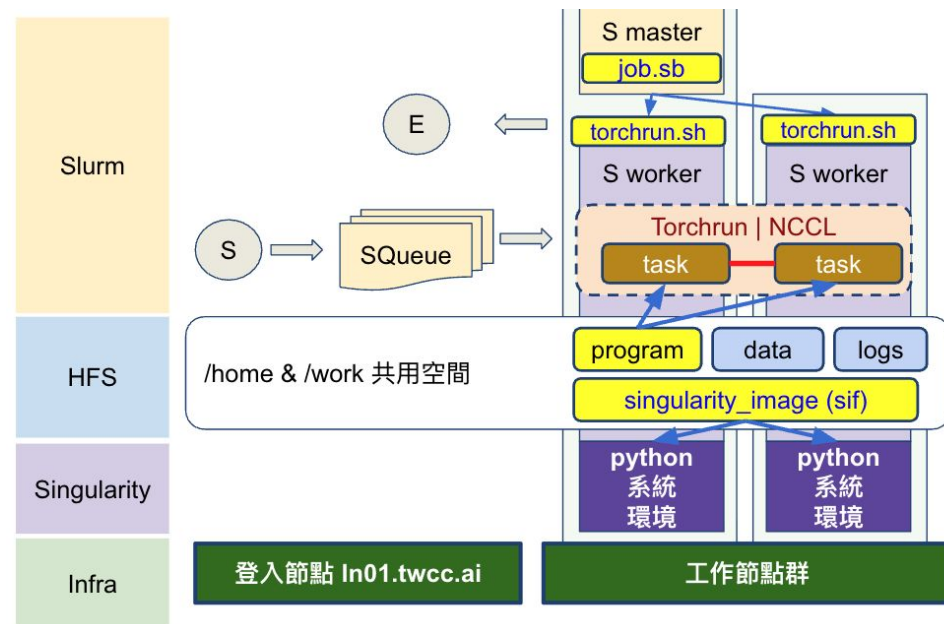
範例程式



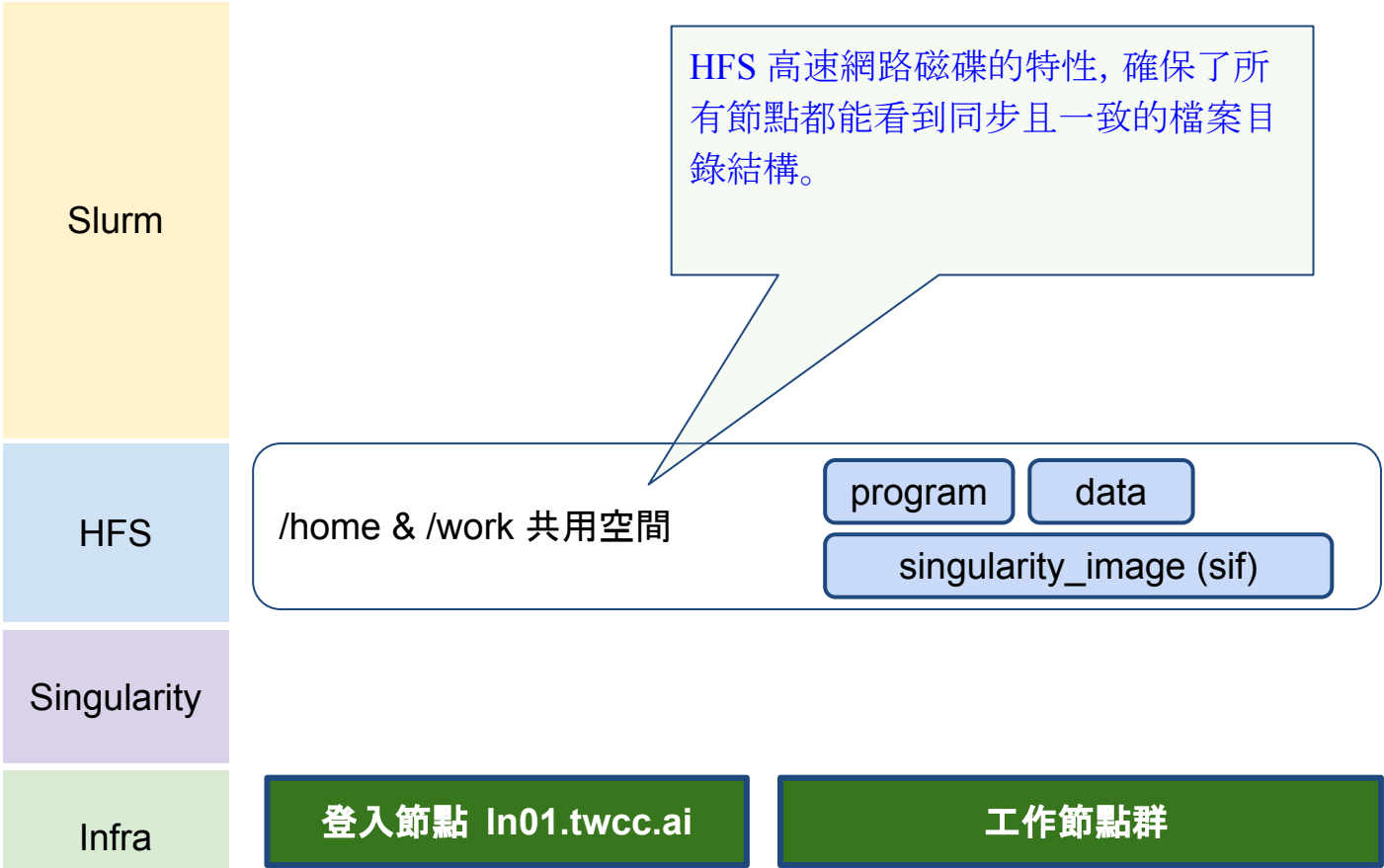
https://github.com/waue0920/nchc_hpc_slurm_example

www.nchc.narlabs.org.tw

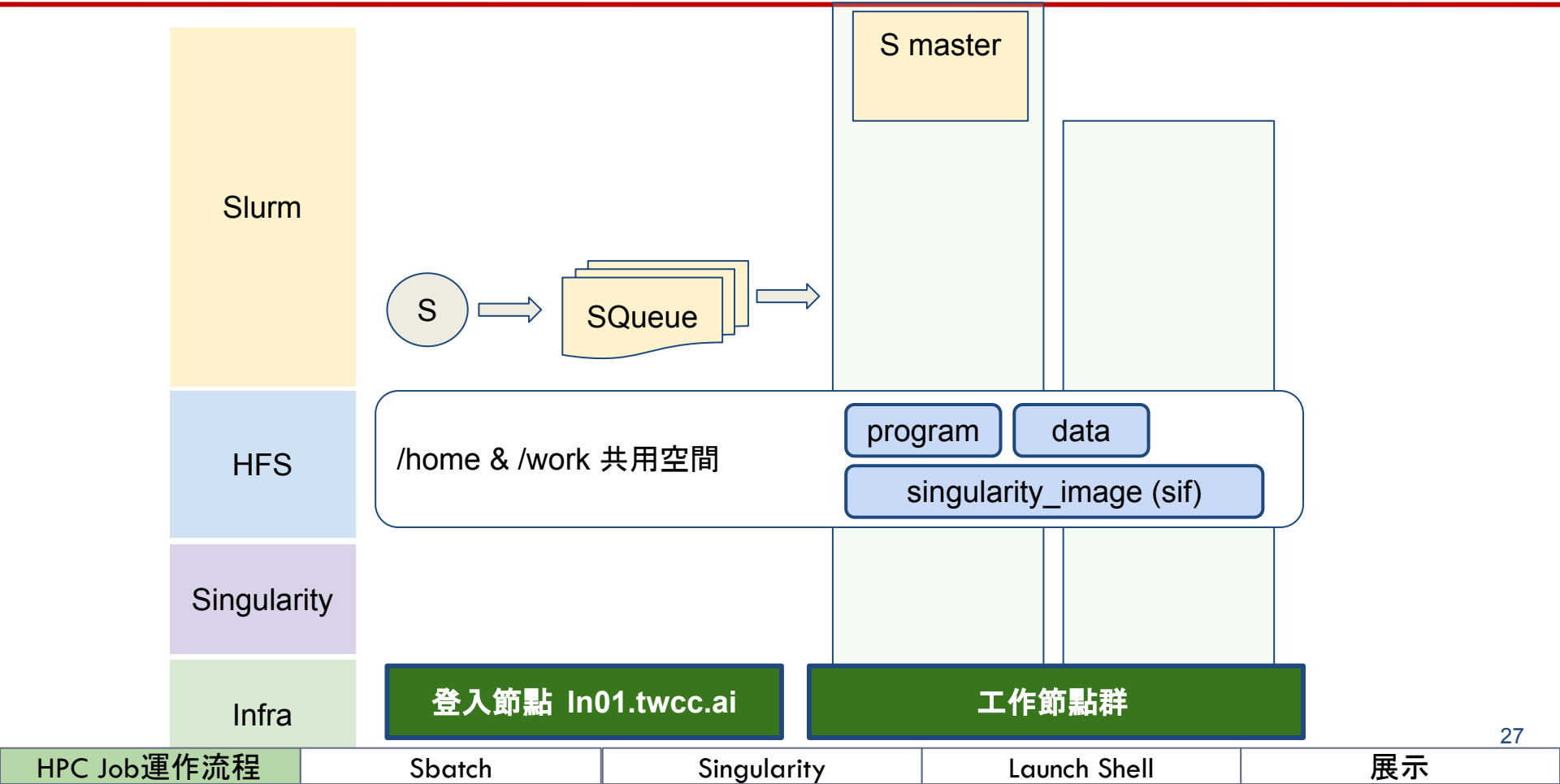
- HPC Job 運作機制
- Slurm Sbatch Script (job.sb)
- Singularity 環境製作 (sif)
- Slurm Launch Shell (torchrun.sh)
- 展示 yolo跨節點運算
- 整理與提醒



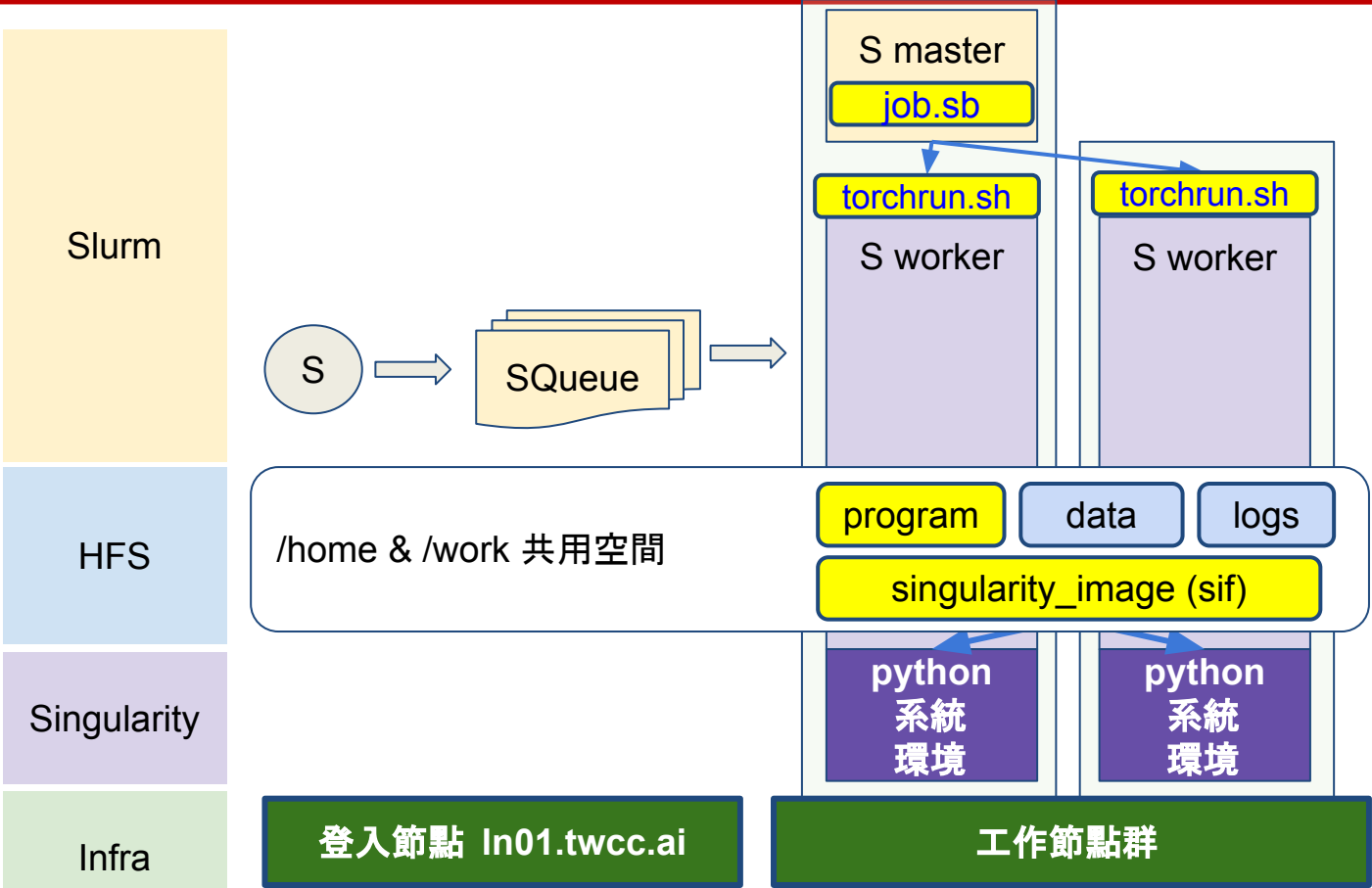
HPC 架構運作機制(1)



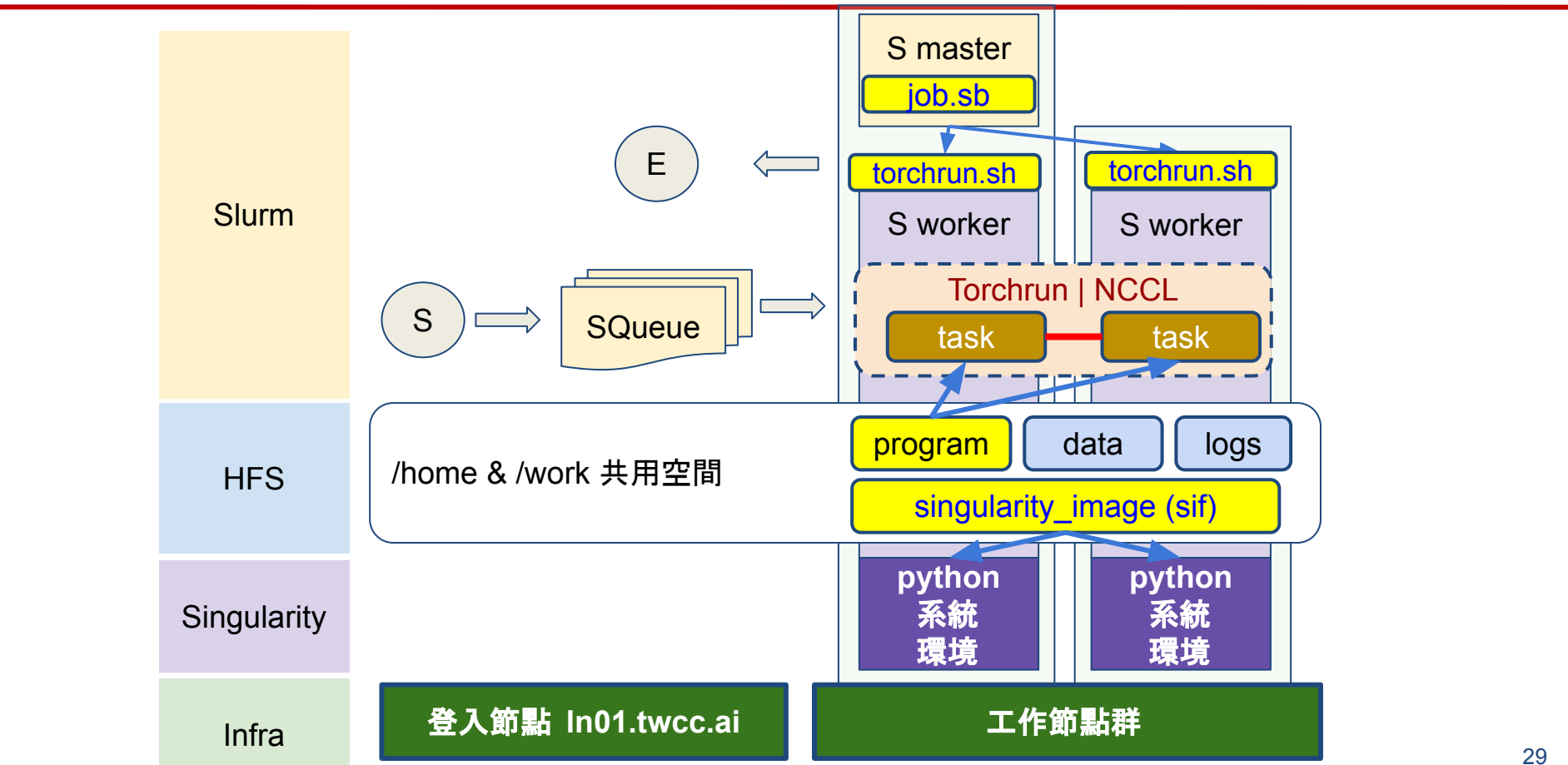
HPC 架構運作機制(2)



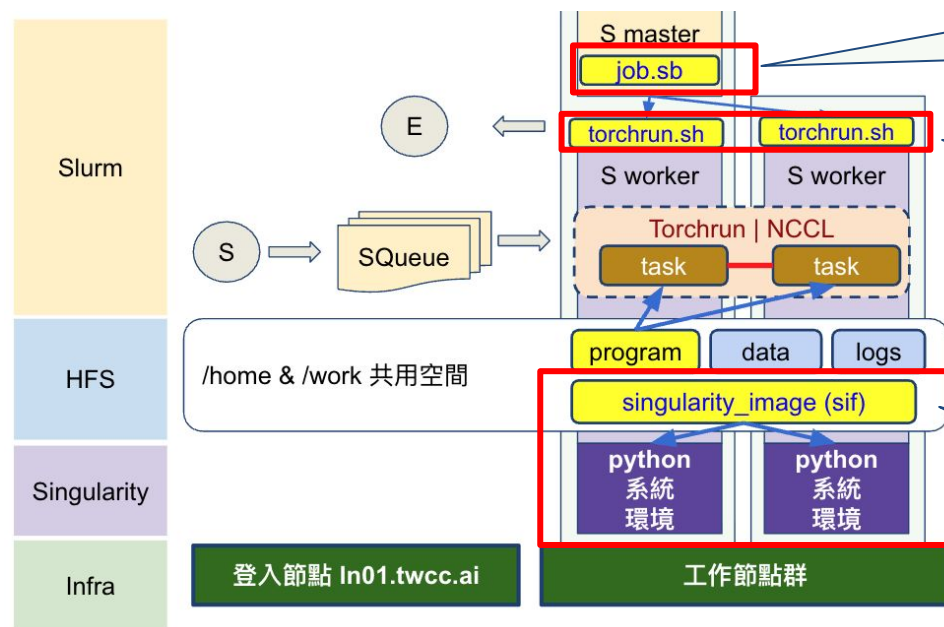
HPC 架構運作機制(3)



HPC 架構運作機制(4)



流程細部說明



`job.sb` 為 slurm 任務腳本。

語言格式為 Bash Shell, 但開頭註解`SBATCH xxx` 可視為Slurm 的Config。只有S master 會執行。
可以透過 `export <Name>=<Value>` 方式將需要的參數傳達給工作節點

`torchrun.sh` 為每個工作節點都會執行的執行啟動腳本。

語言格式為 Bash Shell, 其實並非必要, 但其存在可以簡化`job.sb` 的訊息量。

`singularity_image (sif)` 為使用 singularity 所製作的容器環境。

後有詳細運作與製作細節。

- slurm 會在allocate 的node中, 挑某一個node 為 master_node, 而在allocate的所有node都是 worker_node
- job.sb 只會在master_node上執行。
- srun 內的執行程式 (**torchrun.sh**) 才會在每個 worker node上運行。
- 因此 sb 與 sh 內宣告的變數是獨立的, 但在 sb 宣告 export 的變數, 會被slurm帶到 worker node裡面去

- 小的job 可以直接編寫成這樣:
srun --gres=gpu:2 --mpi=pmix singularity run --nv \$SIF python test.py
- 分出 sh的好處是可以將命令行變得精簡

```
#!/bin/bash
#SBATCH --job-name=<name>    ## Job 名稱
#SBATCH --mail-type=ALL      ## 收到通知條件
#SBATCH --mail-user=xxx@gmail.com
#SBATCH --nodes=2            ## 索取 x 個節點
#SBATCH --cpus-per-task=32    ## 每個 task 索取 x 顆 CPU, 1gpu : 4cpu
#SBATCH --gres=gpu:8          ## 每個節點索取 x 張 GPU
#SBATCH --account="GOV113xxx" ## iService_ID 計畫 ID
#SBATCH --partition=gp2d      ## 使用測試 queue
#SBATCH --output=<log_name>.log ## 將標準輸出記錄到 log
#SBATCH --error=<log_name>.log ## 將錯誤輸出記錄到同一個log

# needed in H100
module purge
module load singularity

# sif
SIF=/work/waue0920/open_access/yolo9t2_nge2306_20241226.sif
SINGULARITY="singularity run --nv $SIF"

# define master
MASTER_ADDR=$(scontrol show hostname $SLURM_NODELIST | head -n 1)
export MASTER_ADDR

## twcc 的 SLURM_GPUS_ON_NODE 為空, 所以直接算
nvidia-smi --list-gpus
NGPU=$(nvidia-smi -L | wc -l) # $SLURM_GPUS_ON_NODE
export NGPU

## 呼叫 yolo train torchrun 版本
cmd="srun --gres=gpu:$NGPU --mpi=pmix $SINGULARITY bash torchrun.sh"
echo $cmd
$cmd
```



- Singularity 是一種專為高性能計算 (HPC) 環境設計的容器技術, Singularity CE 版本是100% opensource 工具
- 它允許用戶在不需要root權限的情況下運行容器, 並且能夠輕鬆地將Docker容器轉換為Singularity容器。
- Singularity的設計目的是為了提供更高的安全性和可移植性, 特別適合在科研和學術環境中使用。
- 與Slurm 的關係:
 - 大多數使用Slurm的單位都可以搭配使用Singularity, 因為Singularity設計之初就是為了在高性能計算 (HPC) 環境中運行, 這些環境通常使用Slurm作為資源管理和作業調度系統。
 - 至於Docker, 雖然許多HPC環境支持Docker, 但並不是所有的HPC環境都會直接使用Docker。這是因為Docker需要root權限, 這在多用戶環境中可能會帶來安全問題。
 - 不過, Singularity可以從Docker容器映像構建Singularity容器, 這樣就能在HPC環境中安全地使用Docker映像

參考資料

- [1] Singularity 網址: <https://sylabs.io/docs/>
- [2] <https://www.top500.org/>
- [3] <https://en.wikipedia.org/wiki/TOP500>
- [4] <https://www.datacenterdynamics.com/>
- [5] <https://docs.sylabs.io/guides/3.7/user-guide/>



ml 為 module load 的縮寫

// 查看 有什麼 module 可以載入
\$ ml avail
// 載入某個 module
\$ ml load singularity
// 查看載入什麼模組
\$ ml
// 清除
\$ ml purge

```
(base) [waue0920@cbi-lgn01 ~]$ singularity
bash: singularity: command not found...
Packages providing this file are:
'apptainer'
'singularity-ce'
(base) [waue0920@cbi-lgn01 ~]$ ml load singularity
(base) [waue0920@cbi-lgn01 ~]$ singularity
Usage:
singularity [global options...] <command>
```

```
(base) [waue0920@cbi-lgn01 ~]$ ml
Currently Loaded Modules:
  1) singularity/3.7.1
```

```
(base) [waue0920@cbi-lgn01 ~]$ ml avail

----- /opt/ohpc/pub/modulefiles -----
cmake/3.24.2    hwloc/2.7.2    os             pmix/4.2.9     singularity/3.7.1 (L)

----- /work/HPC_software/LMOD/nvidia/modulefiles -----
cuda/12.2      nvhpc-hpcx-cuda12/24.7    nvhpc/24.7

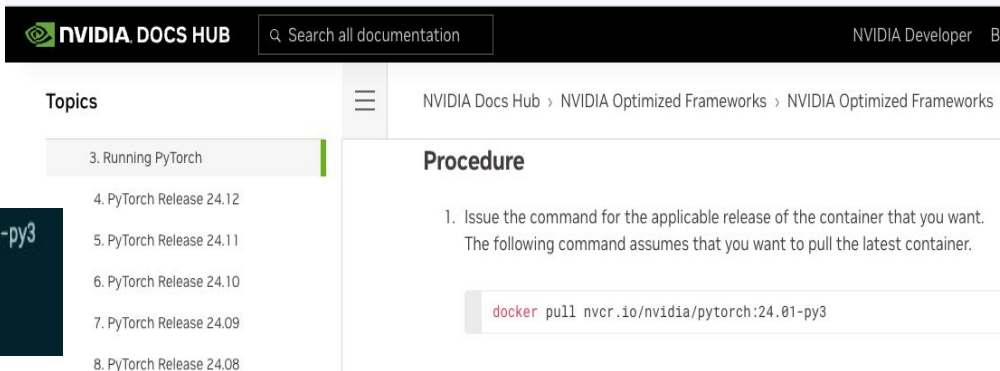
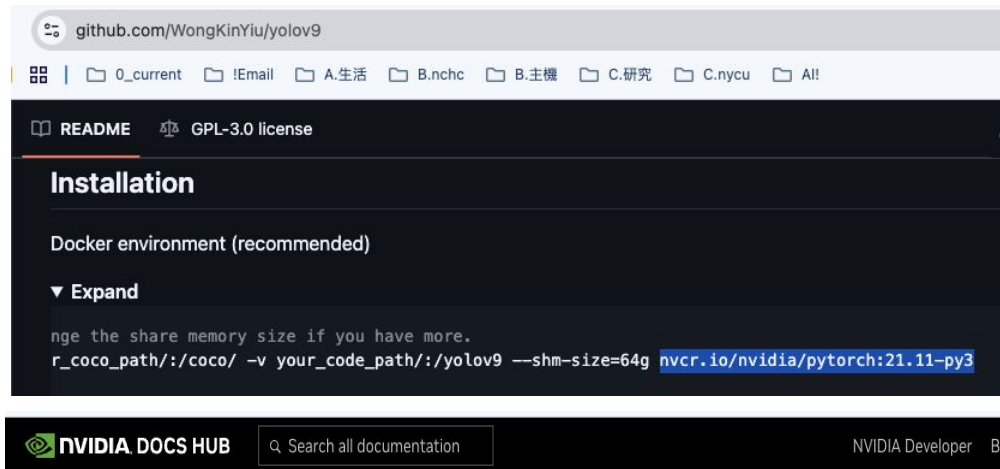
----- /work/HPC_software/LMOD/intel/modulefiles -----
advisor/latest                debugger/2024.2.1 (D)    intel_ippcp_ia32/latest
advisor/2024.2                (D)    dev-utilities/latest
ccl/latest                    dev-utilities/2024.2.0 (D)    intel_ippcp_intel64/latest
ccl/2021.13.1                  (D)    dnnl/latest                                intel_ippcp_intel64/2021.12 (D)
compiler-intel-llvm/latest     (D)    dnnl/3.5.0                                mkl/latest
compiler-intel-llvm/2024.2.1 (D)    dpct/latest                                mkl/2024.2 (D)
compiler-intel-llvm32/latest   (D)    dpct/2024.2.0                             mkl32/latest
compiler-intel-llvm32/2024.2.1 (D)    dpl/latest                                mkl32/2024.2 (D)
compiler-rt/latest             (D)    dpl/2022.6                                mpi/latest
compiler-rt/2024.2.1           (D)    ifort/latest                                mpi/2021.13 (D)
compiler-rt32/latest           (D)    ifort/2024.2.1                             tbb/latest
compiler-rt32/2024.2.1        (D)    ifort32/latest                             tbb/2021.13 (D)
compiler/latest                (D)    ifort32/2024.2.1                         tbb32/latest
compiler/2024.2.1              (D)    intel_ipp_ia32/latest                     tbb32/2021.13 (D)
compiler32/latest              (D)    intel_ipp_ia32/2021.12                   vtune/latest
compiler32/2024.2.1           (D)    intel_ipp_intel64/latest                  vtune/2024.2 (D)
debugger/latest                intel_ipp_intel64/2021.12 (D)

Where:
D: Default Module
L: Module is loaded
```

singularity 容器 印象檔製作方法與指令

- 1. 專案支援的docker環境
- 2. 去docker hub / ngc 網站上確認
- 3. 拉下來使用

```
$ singularity pull test.sif
docker://nvcr.io/nvidia/pytorch:23.06-py3
```



torchrn.sh

- 每個worker node 都會執行的程式碼
- torchrn 中有些參數要執行期間才能知道, 因此參數不能寫死, 要引用或當場計算
如: nnodes, node_rank, master_addr

前面其實都只是在把參數準備好
有些參數可以直接引用slurm帶來的預設參數

```
torchrn \  
--nproc_per_node=$NGPU \  
--nnodes=$NNODES \  
--node_rank=$NODE_RANK \  
--master_addr=$MASTER_ADDR \  
--master_port=$MASTER_PORT \  
segment/train_dual.py \  
--workers $NWorker \  
--device $DEVICE_LIST \  
--batch $NBatch \  
--data coco.yaml \  
--img 640 \  
--cfg models/segment/yolov9-c-dseg.yaml \  
--weights "" \  
--name gelan-c-seg \  
--hyp hyp.scratch-high.yaml \  
--no-overlap \  
--epochs $NEPOCH \  
--close-mosaic 10
```

```
#!/bin/bash  
### 參數設定區 ###  
## 工作目錄  
WORKDIR=/home/waue0920/yolov9  
cd $WORKDIR  
## 設定 NCCL  
export NCCL_DEBUG=INFO  
## SLURM 環境  
NNODES=${SLURM_NNODES:-1}      # 節點總數, 默認為 1  
NODE_RANK=${SLURM_NODEID:-0}    # 當前節點的 rank, 默認為 0  
if [ -z "$MASTER_ADDR" ]; then  
    echo "oh! why MASTER_ADDR not found!"  
    MASTER_ADDR=$(scontrol show hostname $SLURM_NODELIST | head -n 1)  
fi  
if [ -z "$NGPU" ]; then  
    echo "oh! why NPROC_PER_NODE not found!"  
    NGPU=$(nvidia-smi -L | wc -l) # 等於 $SLURM_GPUS_ON_NODE  
fi  
MASTER_PORT=9527  
DEVICE_LIST=$(seq -s, 0 $((NGPU-1)) | paste -sd, -) # 0,1,...n-1  
echo "Debug Information:"  
echo "=====  
echo "SLURM_NODEID: $NODE_RANK"  
echo "SLURM_NNODES: $NNODES"  
echo "=====  
### 執行訓練命令 ###  
## 超參數設定  
NBatch=128    # v100 超過 254會failed  
NEPOCH=100    # 約 20 mins / per Epoch  
NWorker=16    # cpu = gpu x 4, worker < cpu  
## 訓練 segment/train_dual.py 命令 (動態設置 nproc_per_node 和 nnodes)  
TRAIN_CMD="torchrn --nproc_per_node=$NGPU --nnodes=$NNODES --node_rank=$NODE_RANK \  
--master_addr=$MASTER_ADDR --master_port=$MASTER_PORT \  
segment/train_dual.py --workers $NWorker --device $DEVICE_LIST --batch $NBatch \  
--data coco.yaml --img 640 --cfg models/segment/yolov9-c-dseg.yaml \  
--weights "" --name gelan-c-seg --hyp hyp.scratch-high.yaml --no-overlap \  
--epochs $NEPOCH --close-mosaic 10"  
  
## 印出完整的訓練命令  
echo "$TRAIN_CMD"  
echo "=====  
$TRAIN_CMD  
## 檢查執行結果  
if [ $? -ne 0 ]; then  
    echo "Error: TRAIN_CMD execution failed on node $(hostname)" >&2  
    exit 1  
fi
```

任務 yolov9 的跨節點訓練

目標：

- 訓練電腦視覺 yolov9 任務，需要使用超過16張V100 GPU來訓練，該專案有特定的python、pytorch與套件版本。
- <https://github.com/WongKinYiu/yolov9>

條件：

- python==3.8
- torch==2.1
- pillow==9.5.0,
opencv-python-headless==4.8.0.74

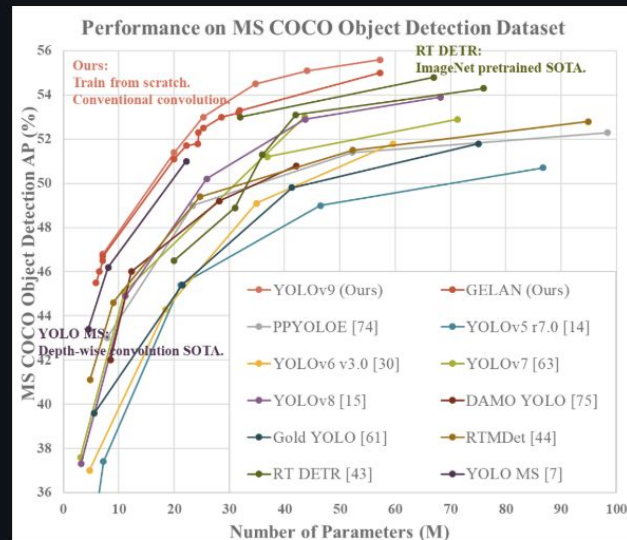
實驗：

1. 實測 8張 x 2 node 的 訓練任務
2. 實測 不同 超參數 (Batchsize, ...) 的多種組合 Benchmark
3. 實測 訓練任務在 N (node) x M (gpu) 多種組合的 Benchmark

YOLOv9

Implementation of paper - YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information

cs.CV arXiv:2402.13616 Hugging Face Spaces Hugging Face Spaces CC Open in Colab OpenCV BlogPost



Performance

MS COCO

Model	Test Size	Apval	AP ₅₀ ^{val}	AP ₇₅ ^{val}	Param.	FLOPs
YOLOv9-T	640	38.3%	53.1%	41.3%	2.0M	7.7G
YOLOv9-S	640	46.8%	63.4%	50.7%	7.1M	26.4G
YOLOv9-M	640	51.4%	68.1%	56.1%	20.0M	76.3G
YOLOv9-C	640	53.0%	70.2%	57.8%	25.3M	102.1G

展示:派送 yolov9 訓練任務

- 準備好資料
- 準備好程式
- 修改 slurm scripts
- 發送 slurm jobs
- 觀察log情況
- 同時間再送一個job

可先執行 1xx.sb後, 觀察行為與結果後, 依此類推接續分別執行 2xx.sb , 3xx.sb 來理解訓練行為

nchc_hpc_slurm_example / twcc /

waue0920 fix sif path

Name	Last commit message
..	
example1_checkenv	fix sif path
example2_yolov9	init the project
yolov9	init the project
README.md	init the project

nchc_hpc_slurm_example / twcc / example2_yolov9 /

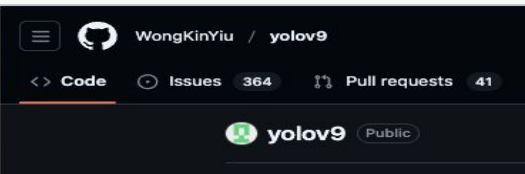
waue0920 init the project

Name	Last commit message
..	
1torch2run_object.sb	init the project
2torch2run_segment_local.sb	init the project
3torch2run_segment_2node.sb	init the project
env.py	init the project
readme.txt	init the project
yolotrain_object.sh	init the project
yolotrain_segment.sh	init the project
yolotrain_segment_local.sh	init the project

此資料夾為 github 上的 yolov9 專案。

<https://github.com/WongKinYiu/yolov9>

使用 git clone 下載yolov9專案, 並按照說明準備mscoco 資料集



- 專案支援slurm 訓練
 - 程式支援 torchrun, deepspeed, ... 等平行能力 (包含 torch早期版本 `torch.distributed.launch`)
 - 環境有推薦的 Docker 環境
- 派送 slurm job的三要件
 - singularity 環境 *
 - 程式的平行驅動方法: torchrun.sh
 - slurm sbatch script : job.sb
- 觀察 job的現況

Q: 是否所有專案都可以”順利”改成跨節點HPC運算？

A: yolov9 滿足以下條件

- 訓練需要大量GPU, 因此增加GPU除了可加速也可放大參數量, 讓跨節點運算有足夠效益。
- 程式原本就支援 `torch.distributed.launch` , 因此為改成torchrun 降低了門檻
- 專案說明提供docker 執行環境, 因此為改成singularity 環境降低了門檻

實作提醒

- T2 HPC 上每個 account 最多 5node x 8gpu 。
 - queue: gtest 只有兩個節點，每個job最多能跑30min(不計入運算上限中)
 - 若要拿到完整的運算節點，需要求 --gres=gpu:8，而系統會從sinfo state = idle 的 node要資源
- 程式運作的過程中，都可以從登入節點，ssh 到 所屬的運算節點內查看當前狀況

```
(base) waue0920@un-ln01:~$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
gtest      up        30:00      2    idle gn[1001-1002]
gp1d       up        1-00:00:00    12    mix  gn[0602-0603,
gp1d       up        1-00:00:00    22    idle gn[0409,0415,
gp2d*      up        2-00:00:00    12    mix  gn[0602-0603,
gp2d*      up        2-00:00:00    22    idle gn[0409,0415]
```

```
(base) waue0920@un-ln01:~/slurm/t2yolov9$ squeue -u waue0920
JOBID PARTITION  NAME      USER ST      TIME  NODES NODELIST(REASON)
686379  gp2d  T2Yolo9S waue0920 R      22:18    2 gn[0709,0812]
(base) waue0920@un-ln01:~/slurm/t2yolov9$ ssh gn0709
Warning: Permanently added 'gn0709,172.17.7.9' (ECDSA) to the list of known hosts.
```



NARLabs 財團法人國家實驗研究院

國家高速網路與計算中心

National Center for High-performance Computing

國網中心HPC跨節點運算 (三)

[課程錄影檔part3](#)

國網中心 陳威宇

waue0920@gmail.com

文件



<https://ppt.cc/fWW9Ex>

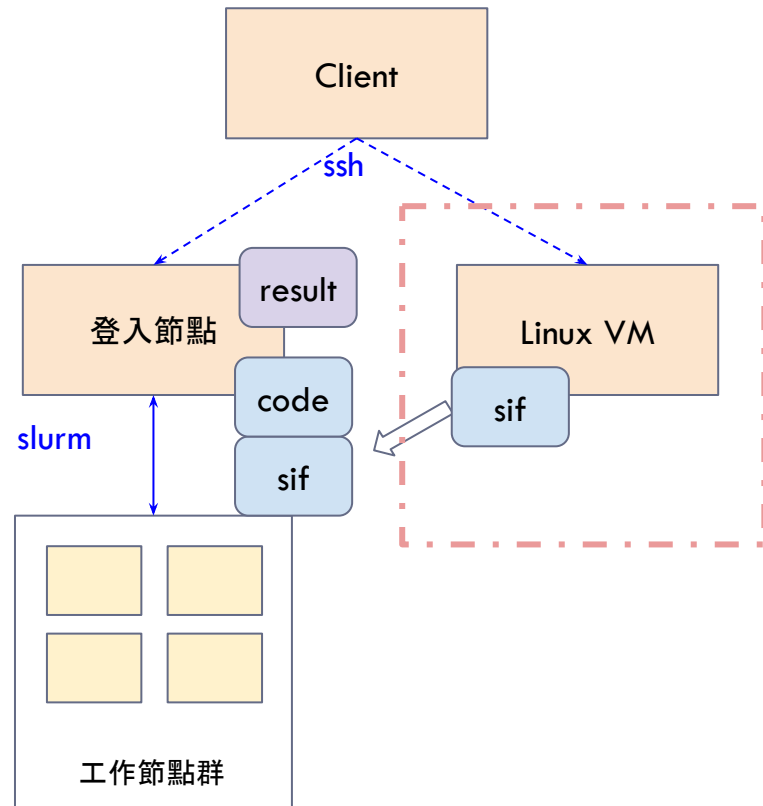
範例程式



https://github.com/waue0920/nchc_hpc_slurm_example

www.nchc.narlabs.org.tw

- 任務說明：
 - 打造一個專屬的singularity image 檔
- 背景說明: Singularity vs Docker
- Singularity 指令與作法一覽表
- 展示
- 總結



目標：

- 打造一個專屬的singularity image 檔

條件：

- python==3.8
- torch==2.1
- pillow==9.5.0,
opencv-python-headless==4.8.0.
74
- 額外加其他套件
 - Wandb

singularity 容器 印象檔製作方法與指令

NARLabs

- 方法一：直接拉下 docker hub 上的印象檔 -> 轉成 singularity 格式的印象檔
 - 直接從 Docker Hub 或 Singularity Hub 下載預建的映像檔，然後使用 `singularity build` 指令來生成 Singularity 格式的映像檔

```
singularity pull my_image.sif docker://url/repo_name
```

1. 原則上，所有docker hub 上的 image 都可以用這個方法拉下來轉換成 sif
 - <https://hub.docker.com/r/xxx/yolov9-gpu>
2. 由於需要gpu 的支援，最好去 nvidia 的 docker hub 上下載，有較好的gpu 對應
 - <https://docs.nvidia.com/deeplearning/frameworks/pytorch-release-notes/running.html>
3. 若專案有提到使用哪個docker image，就可以直接選擇他了

Installation

Docker environment (recommended)

▼ Expand

```
# create the docker container, you can change the share memory size if you have more.  
nvidia-docker run --name yolov9 -it --v your_coco_path:/coco/ --v your_code_path:/yolov9 --shm
```

9 --shm-size=64g nvidia.io/nvidia/pytorch:21.11-py3

特性	Singularity	Docker
權限	<ul style="list-style-type: none">- 無需 root 權限即可執行容器- 無需啟用daemon程序	<ul style="list-style-type: none">- 需 root 權限, (HPC 系統禁止)- 需要啟用 daemon (HPC 系統中不易配置)
社群生態	<ul style="list-style-type: none">- 專注於科研和 HPC 領域, 生態系統較小- 社群更新速度較慢	<ul style="list-style-type: none">- 擁有更廣泛的社群支持和豐富的工具及插件- 更適合雲端應用及微服務部署
目的	<ul style="list-style-type: none">- 提供便攜式的容器運行環境, 支持 HPC 工作流	<ul style="list-style-type: none">- 靈活的容器化解決方案, 適合開發和雲端應用- 著重於快速開發、測試和部署
運行模式	<ul style="list-style-type: none">- 單一 .sif 文件運行, 無需 daemon 支持- 容器直接使用主機資源(文件系統、網絡、GPU)	<ul style="list-style-type: none">- 基於 daemon 的模式, 需拉取容器映像並啟動- 容器與主機隔離性更強, 依賴專門的資源橋接

Install Singularity 3.8.4 (w/ Go 1.17.3)

```
// Ensure repositories are up-to-date
$ sudo apt-get update
// Install debian packages for dependencies
$ sudo apt-get install -y \
    build-essential \
    libseccomp-dev \
    pkg-config \
    squashfs-tools \
    cryptsetup \
    curl wget git

// Install GO
$ export GOVERSION=1.17.3 OS=linux ARCH=amd64 # change this as you
need

$ wget -O /tmp/go${GOVERSION}.${OS}-${ARCH}.tar.gz \
    https://dl.google.com/go/go${GOVERSION}.${OS}-${ARCH}.tar.gz

$ sudo tar -C /usr/local -xzf /tmp/go${GOVERSION}.${OS}-${ARCH}.tar.gz

// add /usr/local/go/bin to the PATH environment variable
$ echo 'export PATH=$PATH:/usr/local/go/bin' >> ~/.bashrc
$ source ~/.bashrc
```

<https://github.com/apptainer/singularity/blob/master/INSTALL.md>

```
// Install singularity
$ git clone https://github.com/hpcng/singularity.git
$ cd singularity
$ git checkout v3.8.4

// Compiling Singularity
$ ./mconfig
$ cd ./builddir
$ make
$ sudo make install
$ singularity --version
$ singularity run library://godlovedc/funny/lolcow
```

```
ubuntu@vm1741942883569-5210262-iaas:~/
INFO: Downloading library image
89.2MiB / 89.2MiB [=====]

< You will be run over by a bus. >
-----
      ^ ^
      (oo)\_____
      (__) \       )\/\
           ||----w |
           ||     ||
```

- 方法一:直接拉下docker hub 上的印象檔 -> 轉成 singularity 格式的印象檔
 - 直接從 Docker Hub 或 Singularity Hub 下載預建的映像檔, 然後使用`singularity build` 指令來生成 Singularity 格式的映像檔

```
$ sudo singularity pull my_image.sif docker://url/repo_name  
$ sudo singularity pull my_image.sif shub://url/repo_name
```

- 方法二:使用 Docker 容器 ID 生成 Singularity 格式映像檔
 - docker hub -> docker ps -> docker images -> singularity.sif

```
$ docker run -d --name <docker_ubuntu> ubuntu:latest  
$ docker ps -a          ## 找到 docker_ubuntu 的 container_id => <docker_pid>  
$ docker commit <docker_pid> <your_docker_image>:<tag>  
$ sudo singularity build method2.sif docker-daemon://<your_docker_image>:<tag>
```

- 方法三: singularity 腳本def檔生成 singularity 格式印象檔
 - 使用 Singularity 定義檔(.def 檔)來生成 Singularity 格式的映像檔

```
$ sudo singularity build my_image.sif my_definition.def
```

singularity 印象檔製作方法比較表

特性	方法一：Docker Hub 映像檔轉換	方法二：從容器 ID 生成映像檔	方法三：使用 Singularity 定義檔
名稱	Docker Hub 到 Singularity 映像檔	修改後容器保存並轉換為 Singularity 映像檔	基於 .def 檔案的 Singularity 映像檔構建
特色	標準化、快速獲取現有映像檔並轉換	允許高度靈活的即時修改與保存	完全可重現、適用於多主機環境
優點	<ul style="list-style-type: none">- 穩定性高：Docker Hub 提供的映像檔通常經過驗證- 效率高：直接轉換，不需額外修改- 廣泛支持：適合標準化需求的項目	<ul style="list-style-type: none">- 成功率高：幾乎不會因相容性問題失敗- 靈活性高：允許即時修改環境，保留所有細節- 快速測試：生成後可立即進行功能驗證	<ul style="list-style-type: none">- 高度可重現：可在不同主機、環境下重建相同映像檔- 可控性強：可詳細記錄環境構建步驟- 長期維護方便：適合需要頻繁調整的專案
缺點	<ul style="list-style-type: none">- 靈活性不足：若需調整環境需重新修改並構建映像檔- 相容性風險：Docker 映像檔可能不完全支援 Singularity	<ul style="list-style-type: none">- 難以復現：映像檔高度依賴原始容器，難以在不同主機上重頭構建	<ul style="list-style-type: none">- 初始成本高：撰寫 .def 檔案需專業技能，耗時長- 測試周期長：若有遺漏或錯誤，需多次重構映像檔
適用情境	<ul style="list-style-type: none">- 標準化環境需求：如yolov9已在以下docker環境下驗證過 nvcv.io/nvidia/pytorch:21.11-py3	<ul style="list-style-type: none">- 個性化需求：如需要精確控制環境配置，並且短期內需要測試和調整- 單機部署：項目無需跨主機復現	<ul style="list-style-type: none">- 長期維護專案：適用於需在任一台主機，任一個時間點都能覆現的專案
實作	<ul style="list-style-type: none">- 可以直接在登入節點執行- singularity pull xxx.sifdocker://<repo_url>	<ul style="list-style-type: none">- 需要另外在 GPU VM 裡執行- 在VM裡安裝docker與singularity- sudo singularity build xxx.sifdocker-daemon:///<tag>	<ul style="list-style-type: none">- 需要另外在 GPU VM 裡執行- 在VM裡安裝 singularity- sudo singularity build xxx.sif ooo.def

方法一:singularity 容器 印象檔製作

- 方法一:直接拉下 docker hub 上的印象檔 -> 轉成 singularity 格式的印象檔
 - 直接從 Docker Hub 或 Singularity Hub 下載預建的映像檔, 然後使用 `singularity build` 指令來生成 Singularity 格式的映像檔

```
$ singularity pull my_image.sif docker://url/repo_name
```

```
➡ singularity pull docker://nvcr.io/nvidia/pytorch:23.06-py3
```

1. 原則上, 所有docker hub 上的 image 都可以用這個方法拉下來轉換成 sif
 - <https://hub.docker.com/r/xxxx/yolov9-gpu>
2. 由於需要gpu 的支援, 最好去 nvidia 的 docker hub 上下載, 有較好的gpu 對應
 - <https://docs.nvidia.com/deeplearning/frameworks/pytorch-release-notes/running.html>
3. 若專案有提到使用哪個docker image, 就可以直接選擇他了

Installation

Docker environment (recommended)

▼ Expand

```
# create the docker container, you can change the share memory size if you have more.  
nvidia-docker run --name yolov9 -it -v your_coco_path:/coco/ -v your_code_path:/yolov9 --shm
```

```
9 --shm-size=64g nvcr.io/nvidia/pytorch:21.11-py3
```

方法三:singularity 容器 印象檔製作

展示 :singularity 製作yolo9t2_ngc2306.sif

@ TWCC GPU VM

- VM環境可以考量(執行環境>套件環境) (Ubuntu 20.04)
 - 安裝nvidia_driver/ cuda / docker / singularity
 - (container內無法安裝docker)

```
$ singularity pull docker://nvcr.io/nvidia/pytorch:23.06-py3
$ vim yolo9_2306.def
$ singularity build yolo9t2_ngc2306.sif yolov9_ngc2306.def
$ singularity shell - -nv yolo9t2_ngc2306.sif
```

@ HPC login node

- 將 yolo9t2_ngc2306.sif 複製到 HPC node 上
- 在 torch2run_segment_2node.sb 內使用剛剛傳來的 .sif 檔
- 送出任務

```
$ sbatch torch2run_segment_2node.sb
```

Bootstrap: docker

From: nvcr.io/nvidia/pytorch:23.06-py3

%post

更新並安裝必要的系統套件

apt-get update

apt-get install -y wget git vim build-essential \

libssl-dev libffi-dev zip htop screen libgl1-mesa-glx libc6

apt-get clean && rm -rf /var/lib/apt/lists/*

移除不需要的舊版 OpenCV 並升級 pip, 否則import cv2 error

rm -rf /usr/local/lib/python3.10/dist-packages/cv2

安裝Python套件, opencv < 4.9, pillow==9.5.0, wandb support

pip install --no-cache-dir --upgrade pip

pip install --no-cache-dir gitpython thop seaborn albumentations \

opencv-python-headless==4.8.0.74 opencv-python==4.8.0.74 \

ipython tqdm tensorboard pycocotools>=2.0 pillow==9.5.0 \

wandb

%environment

設置 CUDA 環境變數

export

LD_LIBRARY_PATH=/usr/local/cuda/lib64:\$LD_LIBRARY_PATH

%labels

Maintainer "wychen <wychen@narlabs.org.tw>"

Version "1.0"

%runscript

echo "Customised for yolov9 by pytorch:23.06-py3"

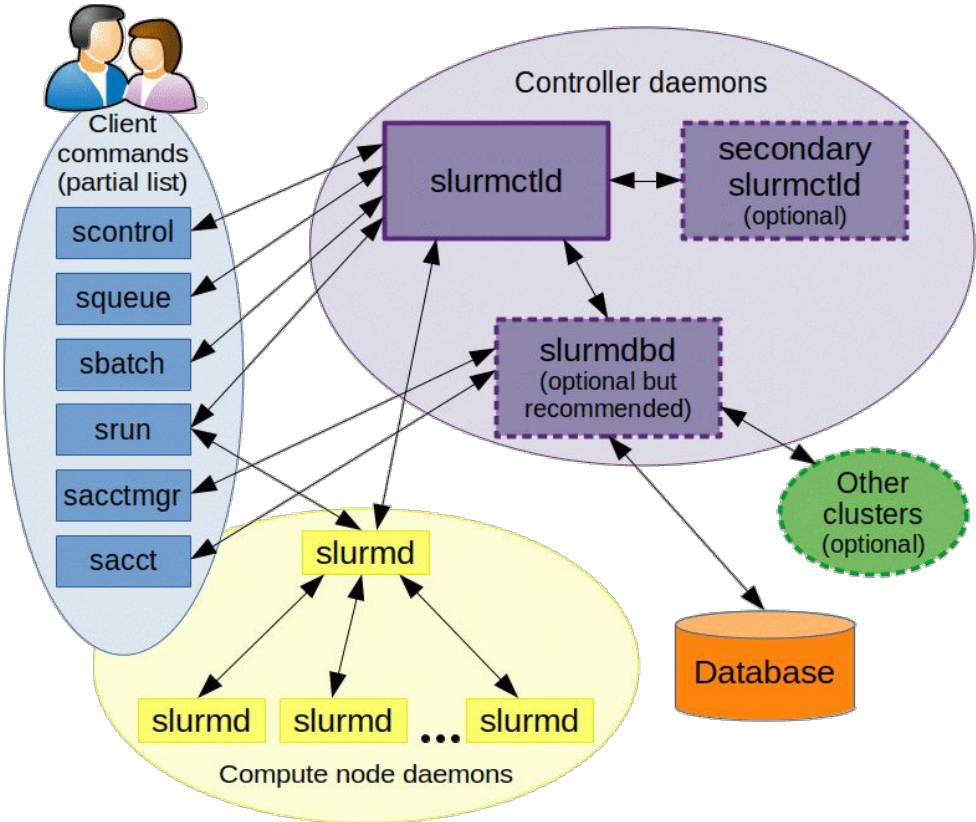
1. 開container : 準備 yolov9 的資料與程式, 並執行 python 單機訓練確認在這個版本的 container可以運作無誤
2. 開gpu vm : 安裝 cuda / docker / singularity 環境, 製作 singularity def -> sif 印象檔, 並簡單在 gpu vm內測試
 - 由於 gpu vm 不會掛載HFS , 但 container & HPC 會, 因此資料程式與 sif印象檔需用 scp/rsync 手動傳輸
3. 到登入節點(ln01.twcc.ai) : 準備 sbatch.sb 與 yolo.sh , 派送sbatch sbatch.sb任務, 觀察log tail -f slurm-xxx.log

```
(base) waue0920@un-ln01:~/slurm/t2yolov9$ tail -f slurm-t2run_segment_train2.log
44[31, 34, 37, 16, 19, 22, 40, 43] 1 24822464 models.yolo.DualDSegment [80, 32, 256, [512, 512, 512, 256, 512, 512, 256, 256]]
yolov9-c-dseg summary: 1204 layers, 57991744 parameters, 57991712 gradients, 372.6 GFLOPs

AMP: checks passed ✓
optimizer: SGD(lr=0.01) with parameter groups 298 weight(decay=0.0), 321 weight(decay=0.0005), 319 bias
augmentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01, num_output_channels=3, method='weighted_average'), CLAHE(p=0.01, clip_l
imit=(1.0, 4.0), tile_grid_size=(8, 8))
train: Scanning /home/waue0920/waue/git/yolov9/dataset/coco/train2017.cache... 117266 images, 1021 backgrounds, 0 corrupt: 100%|██████████| 118287/118287 00:00
train: Scanning /home/waue0920/waue/git/yolov9/dataset/coco/train2017.cache... 117266 images, 1021 backgrounds, 0 corrupt: 100%|██████████| 118287/118287 00:00
val: Scanning /home/waue0920/waue/git/yolov9/dataset/coco/val2017.cache... 4952 images, 48 backgrounds, 0 corrupt: 100%|██████████| 5000/5000 00:00
Plotting labels to runs/train-seg/gelan-c-seg/labels.jpg...
Image sizes 640 train, 640 val
Using 64 dataloader workers
Logging results to runs/train-seg/gelan-c-seg
Starting training for 100 epochs...

Epoch  GPU_mem  box_loss  seg_loss  cls_loss  dfl_loss  Instances  Size
0/99    8.03G    4.684    6.039    7.233    5.336      53         640:  5%| 87/1849 00:41
```

補充: 其他 Slurm 指令



功能	sbatch	salloc + srun
適合用途	非互動 batch 工作	測試、除錯、開發等互動式作業
是否互動	否 (非同步)	是 (進入互動 shell)
資源分配	自動排程後分配	立刻分配 (若有空間資源)
指令執行方式	在 script 裡寫指令	手動輸入 <code>srun</code> 指令

變數名稱	描述
SLURM_JOB_ID	Job ID
SLURM_JOB_NAME	Job name
SLURM_JOB_ACCOUNT	Project id
SLURM_JOB_NUM_NODES SLURM_NNODES	Number of nodes allocated to job
SLURM_JOB_NODELIST SLURM_NODELIST	Nodes assigned to job
SLURM_NTASKS_PER_NODE	Number of tasks requested per node.

<https://slurm.schedmd.com/quickstart.html#arch>

	使用手冊	Partition資訊與限制	收費
T2 (TWCC)	https://man.twcc.ai/@twccdocs/doc-twnia2-main-zh/https%3A%2F%2Fman.twcc.ai%2F%40twccdocs%2Fgetstarted-twnia2-submit-job-zh	https://man.twcc.ai/@twccdocs/doc-twnia2-main-zh/https%3A%2F%2Fman.twcc.ai%2F%40twccdocs%2Fguide-twnia2-queue-zh	<div> <div>Q 臺灣AI雲服務(TWCC)計價 </div> <div>Q 台灣杉三號非保留佇列(NRQ)費率 </div> <div>Q 創進一號 x-86 非保留佇列(NRQ)費率 </div> <div>Q 晶創25 (NANO 5) 非保留佇列(NRQ)費率 </div> <div>Q 台灣杉三號計畫目錄 (/project) 儲存空間計價 </div> </div>
N5 (晶創主機)	https://man.twcc.ai/@AI-Pilot/manual	https://man.twcc.ai/aPiCU8VXS7SZqFJBOoSqBQ	
T3 (Taiwania3)	https://man.twcc.ai/@TWCC-III-manual/H1bEXeGcu	https://man.twcc.ai/@TWCC-III-manual/ryyo0tsuu	https://iservice.nchc.org.tw/nchc_service/nchc_service_qa.php?target=54
F1 (創進一號)	https://man.twcc.ai/@f1-manual/manual	https://man.twcc.ai/@f1-manual/partition	



NARLabs 財團法人國家實驗研究院

國家高速網路與計算中心

National Center for High-performance Computing

Backup

- 方法一: 直接拉下 docker hub 上的印象檔 -> 轉成 singularity 格式的印象檔
 - 直接從 Docker Hub 或 Singularity Hub 下載預建的映像檔, 然後使用 `singularity build` 指令來生成 Singularity 格式的映像檔

```
$ singularity pull <my_image>.sif docker://<url>/<repo_name>
```

```
~ singularity pull docker://nvcv.io/nvidia/pytorch:23.06-py3
```

- 原則上, 所有 docker hub 上的 image 都可以用這個方法拉下來轉換成 sif
 - <https://hub.docker.com/r/xxxx/yolov9-gpu>
- 由於需要 gpu 的支援, 最好去 nvidia 的 docker hub 上下載, 有較好的 gpu 對應
 - <https://docs.nvidia.com/deeplearning/frameworks/pytorch-release-notes/running.html>
- 若專案有提到使用哪個 docker image, 就可以直接選擇他了

Installation

Docker environment (recommended)

▼ Expand

```
# create the docker container, you can change the share memory size if you have more.
nvidia-docker run --name yolov9 -it -v your_coco_path:/coco/ -v your_code_path:/yolov9 --shm
```

9 --shm-size=64g nvcv.io/nvidia/pytorch:21.11-py3