# Requirement

Before excute ansible, the following files need to be set appropriate path.

* cuda 10.2 deb file and patch deb files to /ansible/roles/cuda/files.
* gromacs tar gz file from <https://manual.gromacs.org/documentation/> to each gromacs files directory.
* slurm tar bz2 file from <https://download.schedmd.com/slurm/slurm-21.08.1.tar.bz2> to /ansible/roles/slurm/files.
* home directory of ansible shoud be /srv/ansible

# Version info

ansible : 2.13.7

## /srv/ansible/safe\_apply.sh の内容　Ansibleのメインディレクトリに存在し、基本的にこのbash scriptを中心に、適宜Golgiクラスター用Ansible Playbookが安全に実行される。

#!/bin/bash

# /srv/ansible/safe\_apply.sh

# Golgiクラスター用Ansible Playbookを安全に適用するためのスクリプト

#

# 使用方法:

# 1. このスクリプトに実行権限があることを確認: chmod +x /srv/ansible/safe\_apply.sh

# 2. スクリプトを実行: ./safe\_apply.sh

# 3. メニューから実行したい操作を選択

#

# 機能:

# - 親ノード/子ノードへの設定適用

# - GPUタイプ別の設定適用

# - チェックモードでの実行（変更なし）

# - システム状態の検証

# - バックアップの作成

#

# 注意:

# - sudo/root権限が必要です

# - /srv/ansible ディレクトリで実行してください

ANSIBLE\_DIR="/srv/ansible"

DATE\_STAMP=$(date +%Y%m%d\_%H%M%S)

LOG\_DIR="${ANSIBLE\_DIR}/logs"

LOG\_FILE="${LOG\_DIR}/ansible\_${DATE\_STAMP}.log"

# ログディレクトリの作成と古いログの整理

mkdir -p ${LOG\_DIR}

# 30日以上前のログファイルを削除

find ${LOG\_DIR} -name "ansible\_\*.log" -type f -mtime +30 -delete

echo "ログは ${LOG\_FILE} に保存されます"

echo "（古いログファイルは30日後に自動削除されます）"

# 関数定義: 安全な適用

safe\_apply() {

local playbook=$1

local limit=$2

local options=$3

echo "========================================================"

echo "実行するPlaybook: ${playbook}"

echo "対象ホスト: ${limit:-「全ノード」}"

echo "追加オプション: ${options}"

echo "========================================================"

echo "続行しますか？ (yes/no/check)"

echo " yes - 実行します"

echo " no - キャンセルします"

echo " check - チェックモードで実行します（変更なし）"

read -p "> " choice

case $choice in

yes)

echo "Playbookを実行します..."

if [ -n "$limit" ]; then

ansible-playbook -i production ${playbook} --limit=${limit} ${options} --ask-become-pass | tee -a ${LOG\_FILE}

else

ansible-playbook -i production ${playbook} ${options} --ask-become-pass | tee -a ${LOG\_FILE}

fi

;;

check)

echo "チェックモードで実行します（変更は適用されません）..."

if [ -n "$limit" ]; then

ansible-playbook -i production ${playbook} --limit=${limit} ${options} --check --ask-become-pass | tee -a ${LOG\_FILE}

else

ansible-playbook -i production ${playbook} ${options} --check --ask-become-pass | tee -a ${LOG\_FILE}

fi

;;

\*)

echo "キャンセルしました"

return 1

;;

esac

# 変更後の検証（checkモードでなければ）

if [ "$choice" == "yes" ]; then

echo "システム状態を検証しています..."

if [ -n "$limit" ]; then

ansible-playbook -i production verify\_changes.yml --limit=${limit} | tee -a ${LOG\_FILE}

else

ansible-playbook -i production verify\_changes.yml | tee -a ${LOG\_FILE}

fi

fi

return 0

}

# メインメニュー

show\_menu() {

clear

echo "Golgi クラスター管理ツール"

echo "=========================="

echo "1) 親ノード（GolgiAdmin）の設定適用"

echo "2) すべての子ノードに設定適用"

echo "3) 特定の子ノードのみに設定適用"

echo "4) GPUタイプ別に設定適用（GTX 780Ti）"

echo "5) GPUタイプ別に設定適用（RTX 2080）"

echo "6) バックアップの作成のみ"

echo "7) システム状態の検証のみ"

echo "8) 接続可能なノードのみに設定適用" # 新しい選択肢

echo "9) 終了"

echo

read -p "選択してください> " choice

case $choice in

1)

safe\_apply "admin.yml" "" ""

;;

2)

safe\_apply "nodes.yml" "" ""

;;

3)

read -p "対象ノード（カンマ区切り、例: golgi01,golgi02）> " target\_nodes

safe\_apply "nodes.yml" "$target\_nodes" ""

;;

4)

safe\_apply "nodes.yml" "nodes\_gtx780ti" ""

;;

5)

safe\_apply "nodes.yml" "nodes\_rtx2080" ""

;;

6)

ansible-playbook -i production backup\_configs.yml --ask-become-pass | tee -a ${LOG\_FILE}

;;

7)

ansible-playbook -i production verify\_changes.yml | tee -a ${LOG\_FILE}

;;

8)

safe\_apply "nodes.yml" "nodes\_online" ""

;;

9)

echo "終了します"

exit 0

;;

\*)

echo "無効な選択です"

;;

esac

echo

read -p "メインメニューに戻りますか？ (y/n) " back\_to\_menu

if [ "$back\_to\_menu" == "y" ] || [ "$back\_to\_menu" == "Y" ]; then

show\_menu

else

echo "終了します"

exit 0

fi

}

# スクリプト開始

cd ${ANSIBLE\_DIR}

show\_menu

## /srv/ansible/admin.yml の内容

# /srv/ansible/admin.yml

---

# 前段の確認タスク - 親ノードで実行

- hosts: localhost

connection: local

become: yes

gather\_facts: yes

tasks:

- name: Create backup directory

file:

path: /root/ansible\_backups/{{ ansible\_date\_time.date }}

state: directory

mode: '0700'

- name: Backup critical configuration files

copy:

src: "{{ item }}"

dest: "/root/ansible\_backups/{{ ansible\_date\_time.date }}/{{ item | basename }}.bak"

remote\_src: yes

with\_items:

- /etc/defaultdomain

- /etc/exports

- /etc/hosts

- /etc/iptables.rules

- /var/yp/Makefile

- "{{ lookup('env', 'SLURM\_CONF') | default('/opt/slurm/etc/slurm.conf') }}"

ignore\_errors: yes

- name: Check if critical services are running

command: systemctl is-active {{ item }}

register: service\_status\_result

ignore\_errors: yes

changed\_when: false

with\_items:

- rpcbind

- nfs-kernel-server

- slurmctld

- name: Set service status facts

set\_fact:

service\_status: "{{ service\_status|default({}) | combine({item.item: (item.rc == 0)}) }}"

with\_items: "{{ service\_status\_result.results }}"

# 常に安全に適用できる基本設定のみを最初に適用

- hosts: localhost

connection: local

become: yes

roles:

- role: apt

- role: admin-ubuntu

# NFS関連の処理を条件付きで適用

- hosts: localhost

connection: local

become: yes

tasks:

- name: Check if /home is already mounted

shell: mount | grep "on /home"

register: home\_mount

ignore\_errors: yes

changed\_when: false

- name: Apply NFS role only if /home is not properly mounted

include\_role:

name: nfs

when: home\_mount.rc != 0 or home\_mount.stdout == ""

# NIS Server関連の処理を条件付きで適用

- hosts: localhost

connection: local

become: yes

tasks:

- name: Check if NIS server is running

command: systemctl status nis

register: nis\_status

ignore\_errors: yes

changed\_when: false

- name: Check MINGID setting in NIS Makefile

command: grep -E "^MINGID=999" /var/yp/Makefile

register: mingid\_check

ignore\_errors: yes

changed\_when: false

when: nis\_status.rc == 0

- name: Apply NIS server role if not running or MINGID is incorrect

include\_role:

name: nis-server

when: nis\_status.rc != 0 or (mingid\_check is defined and mingid\_check.rc != 0)

# その他のロールを必要に応じて適用

- hosts: localhost

connection: local

become: yes

tasks:

- name: Check if gfortran is installed

command: which gfortran

register: gfortran\_installed

ignore\_errors: yes

changed\_when: false

- name: Apply gfortran role only if not installed

include\_role:

name: gfortran

when: gfortran\_installed.rc != 0

- name: Check if Gromacs 2022.4 is installed

stat:

path: /opt/gromacs-2022.4/bin/gmx

register: gromacs\_2022\_installed

- name: Apply Gromacs 2022.4 role only if not installed

include\_role:

name: gromacs-2022.4

when: not gromacs\_2022\_installed.stat.exists

- name: Check if Slurm is configured and running

command: systemctl status slurmctld

register: slurmctld\_status

ignore\_errors: yes

changed\_when: false

- name: Apply Slurm role only if not properly configured or running

include\_role:

name: slurm

apply:

tags: ["configuration"]

when: slurmctld\_status.rc != 0

vars:

services:

- slurmctld.service

- name: Check if AlphaFold directory is mounted

shell: mount | grep alphafold

register: alphafold\_mounted

ignore\_errors: yes

changed\_when: false

- name: Apply AlphaFold admin role only if not properly mounted

include\_role:

name: admin-alphafold

when: alphafold\_mounted.rc != 0 or alphafold\_mounted.stdout == ""

## /srv/ansible/nodes.yml の内容

# /srv/ansible/nodes.yml

---

# 前段の確認タスク - すべてのノードで実行

- hosts: nodes

become: yes

gather\_facts: yes

tasks:

- name: Check NVIDIA driver status

command: nvidia-smi

register: nvidia\_smi\_result

ignore\_errors: yes

changed\_when: false

- name: Check running jobs

command: squeue -h -o "%i %u" -t running -w {{ ansible\_hostname }}

register: running\_jobs

ignore\_errors: yes

changed\_when: false

- name: Set backup directory path

set\_fact:

backup\_dir: "/root/ansible\_backups/{{ ansible\_date\_time.date | regex\_replace('-', '') }}"

- name: Create backup directory

file:

path: "{{ backup\_dir }}"

state: directory

mode: '0700'

register: dir\_created

- name: Display backup directory path for verification

debug:

msg: "バックアップディレクトリ {{ backup\_dir }} が作成されました"

when: dir\_created.changed

- name: Backup critical configuration files

copy:

src: "{{ item }}"

dest: "{{ backup\_dir }}/{{ item | basename }}.bak"

remote\_src: yes

with\_items:

- /etc/defaultdomain

- /etc/yp.conf

- /etc/nsswitch.conf

- "{{ lookup('env', 'SLURM\_CONF') | default('/opt/slurm/etc/slurm.conf') }}"

ignore\_errors: yes

when: dir\_created.changed or dir\_created.skipped is defined

# 常に安全に適用できる基本設定のみを最初に適用

- hosts: nodes

become: yes

roles:

- role: child-ubuntu

- role: apt

- role: nis

- role: nfs

- role: lm-sensors

# CUDA関連の処理を条件付きで適用（GPUタイプごとに差別化）

- hosts: nodes

become: yes

tasks:

- name: Detect GPU type

shell: nvidia-smi --query-gpu=name --format=csv,noheader | head -n 1

register: gpu\_type

ignore\_errors: yes

changed\_when: false

- name: Set fact for GTX 780Ti nodes

set\_fact:

is\_gtx780ti: true

when: gpu\_type.stdout is defined and 'GTX 780Ti' in gpu\_type.stdout

- name: Set fact for RTX 2080 nodes

set\_fact:

is\_rtx2080: true

when: gpu\_type.stdout is defined and ('RTX 2080' in gpu\_type.stdout or 'RTX 2080 SUPER' in gpu\_type.stdout)

- name: Set fact for GTX 780Ti nodes from inventory (fallback)

set\_fact:

is\_gtx780ti: true

when: (is\_gtx780ti is not defined or not is\_gtx780ti) and inventory\_hostname in groups['nodes\_gtx780ti']

- name: Set fact for RTX 2080 nodes from inventory (fallback)

set\_fact:

is\_rtx2080: true

when: (is\_rtx2080 is not defined or not is\_rtx2080) and inventory\_hostname in groups['nodes\_rtx2080']

- name: Apply CUDA 10.2 only for GTX 780Ti nodes when needed

include\_role:

name: cuda-10.2

when:

- is\_gtx780ti is defined and is\_gtx780ti

- nvidia\_smi\_result.rc != 0 or running\_jobs.stdout == ""

- name: Apply CUDA 11.4 only for RTX 2080 nodes when needed

include\_role:

name: cuda-11.4

when:

- is\_rtx2080 is defined and is\_rtx2080

- nvidia\_smi\_result.rc != 0 or running\_jobs.stdout == ""

# Gromacs関連の処理を条件付きで適用

- hosts: nodes

become: yes

tasks:

- name: Check if Gromacs 2020.3 is installed

stat:

path: /opt/gromacs-2020.3/bin/gmx

register: gromacs\_2020\_installed

- name: Check if Gromacs 2022.4 is installed

stat:

path: /opt/gromacs-2022.4/bin/gmx

register: gromacs\_2022\_installed

- name: Apply Gromacs 2020.3 role only if not installed (for GTX 780Ti nodes)

include\_role:

name: gromacs-2020.3

when:

- is\_gtx780ti is defined and is\_gtx780ti

- not gromacs\_2020\_installed.stat.exists

- name: Apply Gromacs 2022.4 role only if not installed (for RTX 2080 nodes)

include\_role:

name: gromacs-2022.4-child

when:

- is\_rtx2080 is defined and is\_rtx2080

- not gromacs\_2022\_installed.stat.exists

# Tensorflow関連の処理を安全に適用

- hosts: nodes

become: yes

tasks:

- name: Check if Docker is installed

command: which docker

register: docker\_installed

ignore\_errors: yes

changed\_when: false

- name: Check if NVIDIA Docker is configured

command: docker info

register: docker\_info

ignore\_errors: yes

changed\_when: false

- name: Apply Tensorflow role only if Docker or NVIDIA Docker is not properly configured

include\_role:

name: tensorflow

when: docker\_installed.rc != 0 or "NVIDIA" not in docker\_info.stdout

# Slurm関連の処理を最後に、かつ安全に適用

- hosts: nodes

become: yes

tasks:

- name: Check if Slurm is configured and running

command: systemctl status slurmd

register: slurmd\_status

ignore\_errors: yes

changed\_when: false

- name: Apply Slurm role only if not properly configured or running

include\_role:

name: slurm

apply:

tags: ["configuration"]

when: slurmd\_status.rc != 0

vars:

services:

- slurmd.service

# AlphaFold関連の処理を最後に適用

- hosts: nodes

become: yes

tasks:

- name: Check if AlphaFold directory is mounted

shell: mount | grep alphafold

register: alphafold\_mounted

ignore\_errors: yes

changed\_when: false

- name: Apply AlphaFold role only if not properly mounted

include\_role:

name: child-alphafold

when: alphafold\_mounted.rc != 0 or alphafold\_mounted.stdout == ""

## production の内容

# /srv/ansible/production

# Golgiクラスターインベントリファイル

# 最終更新: 2025-02-27

# 全ノードグループ

[nodes]

golgi01

golgi02

golgi03

golgi04

golgi05

golgi06

golgi07

golgi08

golgi09

golgi10

golgi11

golgi12

golgi13

golgi14

golgi15

# 接続可能なノードグループ（ピング成功したノード）

[nodes\_online]

golgi05

golgi06

golgi07

golgi08

golgi09

golgi10

golgi12

golgi13

golgi14

golgi15

# GPUタイプ別グループ - ノードのGPUが変更された場合はここを更新

[nodes\_gtx780ti] # GeForce GTX 780Ti (Compute Capability 3.5)

golgi14

[nodes\_rtx2080] # RTX 2080/2080 SUPER (Compute Capability 7.5)

golgi08

golgi09

golgi10

golgi11

golgi12

golgi13

# CUDAバージョン別グループ - GPUタイプに応じて適切なバージョンが必要

[nodes\_cuda102] # CUDA 10.2 - 旧世代GPU向け (Compute 3.0-6.2)

golgi14

[nodes\_cuda114] # CUDA 11.4 - 新世代GPU向け (Compute 7.0+)

golgi08

golgi09

golgi10

golgi11

golgi12

golgi13

# その他のグループは必要に応じて追加可能

# 例: 特定のメモリ容量、CPU世代、用途など

[nodes:vars]

ansible\_ssh\_user=ansible

ansible\_ssh\_pass=ansible

ansible\_python\_interpreter=/usr/bin/python3

## 以下は、./roles/ ディレクトリ以下にある、YAML形式(.yml)のテキストファイルの内容をそのテキストファイルがある場所と共にすべて書き下したもの。(コマンド”””for FILE in $(find ./roles -type f -name '\*.yml'); do echo "==== $FILE ===="; cat "$FILE"; echo; done”””で出力。)

==== ./roles/gromacs-2022.4-child/meta/main.yml ====

---

dependencies:

- cuda-11.4

==== ./roles/gromacs-2022.4-child/defaults/main.yml ====

version: 2022.4

source\_directory: /opt/src

install\_prefix: /opt/gromacs

==== ./roles/gromacs-2022.4-child/tasks/main.yml ====

- name: install fftw and cmake

apt:

name: ["libfftw3-dev", "cmake"]

state: present

- stat:

path: "{{ install\_prefix }}-{{ version }}/bin/gmx"

register: gmx\_result

- name: ensure a source directory exists

file:

path: "{{ source\_directory }}"

state: directory

- name: unarchive source files

unarchive:

src: "gromacs-{{ version }}.tar.gz"

dest: "{{ source\_directory }}"

when: not gmx\_result.stat.exists

- name: create builddir

tempfile:

state: directory

suffix: gromacs-build

register: builddir

when: not gmx\_result.stat.exists

- name: configure

command: "cmake {{ source\_directory }}/gromacs-{{ version }} -DGMX\_SIMD=AVX2\_256 -DGMX\_GPU=CUDA -DCUDA\_TOOLKIT\_ROOT\_DIR=/usr/local/cuda -DREGRESSIONTEST\_DOWNLOAD=ON -DCMAKE\_INSTALL\_PREFIX={{ install\_prefix }}-{{ version }}"

args:

chdir: "{{ builddir.path }}"

when: not gmx\_result.stat.exists

- name: compile, test and install

make:

target: "{{ item }}"

chdir: "{{ builddir.path }}"

with\_items:

- all

- check

- install

when: not gmx\_result.stat.exists

- name: clear builddir

file:

path: "{{ builddir.path }}"

state: absent

when: not gmx\_result.stat.exists

==== ./roles/gromacs-2019.4/meta/main.yml ====

---

dependencies:

- cuda-11.7

==== ./roles/gromacs-2019.4/defaults/main.yml ====

version: 2019.4

source\_directory: /opt/src

install\_prefix: /opt/gromacs

==== ./roles/gromacs-2019.4/tasks/main.yml ====

- name: install fftw and cmake

apt:

name: ["libfftw3-dev", "cmake"]

state: present

- stat:

path: "{{ install\_prefix }}-{{ version }}/bin/gmx"

register: gmx\_result

- name: ensure a source directory exists

file:

path: "{{ source\_directory }}"

state: directory

- name: unarchive source files

unarchive:

src: "gromacs-{{ version }}.tar.gz"

dest: "{{ source\_directory }}"

when: not gmx\_result.stat.exists

- name: fix cmake

lineinfile:

path: "{{ source\_directory }}/gromacs-{{ version }}/cmake/gmxManageNvccConfig.cmake"

regexp: "{{ item }}"

state: absent

when: not gmx\_result.stat.exists

with\_items:

- '(.\*arch=compute\_20,code=sm\_20.\*)'

- '(.\*arch=compute\_30,code=sm\_30.\*)'

- name: create builddir

tempfile:

state: directory

suffix: gromacs-build

register: builddir

when: not gmx\_result.stat.exists

- name: unarchive regressiontest files

unarchive:

src: "regressiontests-{{ version }}.tar.gz"

dest: "{{ source\_directory }}"

when: not gmx\_result.stat.exists

- name: configure

command: "cmake {{ source\_directory }}/gromacs-{{ version }} -DGMX\_SIMD=AVX2\_256 -DGMX\_GPU=ON -DCUDA\_TOOLKIT\_ROOT\_DIR=/usr/local/cuda -DREGRESSIONTTEST\_PATH={{ source\_directory}}/regressiontests-{{ version }} -DCMAKE\_INSTALL\_PREFIX={{ install\_prefix }}-{{ version }}"

args:

chdir: "{{ builddir.path }}"

when: not gmx\_result.stat.exists

- name: compile, test and install

make:

target: "{{ item }}"

chdir: "{{ builddir.path }}"

with\_items:

- all

- check

- install

when: not gmx\_result.stat.exists

- name: clear builddir

file:

path: "{{ builddir.path }}"

state: absent

when: not gmx\_result.stat.exists

- name: make symbolic link

file:

src: "{{ install\_prefix }}-{{ version }}"

dest: "{{ install\_prefix }}"

state: link

==== ./roles/apt/tasks/main.yml ====

- name: change archive servers to those in Japan

replace:

dest: /etc/apt/sources.list

regexp: 'deb https?://[^(security)]\S\* (.\*)'

replace: 'deb http://jp.archive.ubuntu.com/ubuntu \1'

- command: grep -e "\<universe\>" /etc/apt/sources.list

register: check\_universe\_repo

check\_mode: no

ignore\_errors: yes

changed\_when: no

- name: add the universe repository

command: add-apt-repository universe && apt-get update

when: check\_universe\_repo.rc != 0

- name: disable unattended upgrades

lineinfile:

path: "/etc/apt/apt.conf.d/20auto-upgrades"

regexp: "{{ item.regexp }}"

line: "{{ item.line}}"

with\_items:

- { regexp: 'Update-Package-Lists', line: 'APT::Periodic::Update-Package-Lists "0";' }

- { regexp: 'Unattended-Upgrade', line: 'APT::Periodic::Unattended-Upgrade "0";' }

==== ./roles/tensorflow/handlers/main.yml ====

- name: restart docker daemon

systemd:

name: docker

state: restarted

==== ./roles/tensorflow/tasks/main.yml ====

# /srv/ansible/roles/tensorflow/tasks/main.yml

---

- name: Check if Docker is already installed

command: which docker

register: docker\_check

ignore\_errors: yes

changed\_when: false

- name: Check if NVIDIA Docker is already installed

command: dpkg -l nvidia-docker2

register: nvidia\_docker\_check

ignore\_errors: yes

changed\_when: false

- name: Define preferred Docker version

set\_fact:

docker\_version: "5:20.10.12~3-0~ubuntu-{{ ansible\_distribution\_release }}"

when: docker\_preferred\_version is not defined

- name: install some packages

apt:

name: ["apt-transport-https", "ca-certificates", "software-properties-common"]

state: present

update\_cache: yes

- name: add an apt key

apt\_key:

url: https://download.docker.com/linux/ubuntu/gpg

state: present

when: docker\_check.rc != 0

- name: add a docker stable repository

apt\_repository:

repo: "deb [arch=amd64] https://download.docker.com/linux/ubuntu {{ ansible\_distribution\_release }} stable"

state: present

when: docker\_check.rc != 0

- name: Get available Docker versions

shell: apt-cache madison docker-ce | awk '{print $3}'

register: available\_docker\_versions

changed\_when: false

when: docker\_check.rc != 0

- name: Install specific Docker version if available

apt:

name: "docker-ce={{ docker\_version }}"

update\_cache: yes

when: docker\_check.rc != 0 and docker\_version in available\_docker\_versions.stdout

- name: Install latest Docker version if preferred version not available

apt:

name: docker-ce

update\_cache: yes

when: docker\_check.rc != 0 and (available\_docker\_versions is not defined or docker\_version not in available\_docker\_versions.stdout)

- name: Check GPG key URL validity

command: curl -s -I https://nvidia.github.io/nvidia-docker/gpgkey

register: gpg\_url\_check

ignore\_errors: yes

changed\_when: false

when: nvidia\_docker\_check.rc != 0

tags:

- nvidia-docker

- name: Warn if GPG key URL is not accessible

debug:

msg: "WARNING: The NVIDIA Docker GPG key URL might not be accessible. Please verify the URL."

when: gpg\_url\_check is defined and gpg\_url\_check is not skipped and gpg\_url\_check.rc is defined and gpg\_url\_check.rc != 0

tags:

- nvidia-docker

- name: add an apt key for nvidia-docker

apt\_key:

url: https://nvidia.github.io/nvidia-docker/gpgkey

state: present

tags:

- nvidia-docker

when: nvidia\_docker\_check.rc != 0

- name: add apt repositories

get\_url:

url: "https://nvidia.github.io/nvidia-docker/ubuntu{{ ansible\_distribution\_version }}/nvidia-docker.list"

dest: /etc/apt/sources.list.d/nvidia-docker.list

tags:

- nvidia-docker

when: nvidia\_docker\_check.rc != 0

- name: Ensure Docker is running before NVIDIA Docker installation

systemd:

name: docker

state: started

when: docker\_check.rc == 0 and nvidia\_docker\_check.rc != 0

- name: install nvidia-docker

apt:

name: nvidia-docker2

update\_cache: yes

notify: restart docker daemon

tags:

- nvidia-docker

when: nvidia\_docker\_check.rc != 0

==== ./roles/nfs/tasks/main.yml ====

- name: install nfs

apt:

name: nfs-common

state: present

- name: mount /home

mount:

path: /home

src: "GolgiFS:/volume1/homes"

fstype: nfs

state: mounted

==== ./roles/gromacs-2020.3/meta/main.yml ====

---

dependencies:

- cuda-10.2

==== ./roles/gromacs-2020.3/defaults/main.yml ====

version: 2020.3

source\_directory: /opt/src

install\_prefix: /opt/gromacs

==== ./roles/gromacs-2020.3/tasks/main.yml ====

# /srv/ansible/roles/gromacs-2020.3/tasks/main.yml

---

- name: Check for running jobs

command: squeue -h -o "%i" -t running -w {{ ansible\_hostname }}

register: running\_jobs

ignore\_errors: yes

changed\_when: false

- name: Display warning if running jobs detected

debug:

msg: "WARNING: There are running jobs on this node. Proceeding with Gromacs installation might impact performance."

when: running\_jobs.stdout\_lines | length > 0

- name: install fftw and cmake

apt:

name: ["libfftw3-dev", "cmake"]

state: present

- name: install gcc-8 and g++-8 to use cmake in CUDA10.2

apt:

name: ["gcc-8", "g++-8"]

state: present

- name: Check if Gromacs is already installed

stat:

path: "{{ install\_prefix }}-{{ version }}/bin/gmx"

register: gmx\_result

- name: Check Gromacs version if installed

command: "{{ install\_prefix }}-{{ version }}/bin/gmx --version"

register: gmx\_version

ignore\_errors: yes

changed\_when: false

when: gmx\_result.stat.exists

- name: Check GPU type

shell: nvidia-smi --query-gpu=name --format=csv,noheader | head -n 1

register: gpu\_type

ignore\_errors: yes

changed\_when: false

- name: Set GPU architecture for GTX 780Ti

set\_fact:

cuda\_target: "-DGMX\_CUDA\_TARGET\_SM=35"

when: gpu\_type.stdout is defined and 'GTX 780Ti' in gpu\_type.stdout

- name: Set GPU architecture for RTX 2080

set\_fact:

cuda\_target: "-DGMX\_CUDA\_TARGET\_SM=75"

when: gpu\_type.stdout is defined and 'RTX 2080' in gpu\_type.stdout

- name: Set GPU architecture from inventory (fallback)

set\_fact:

cuda\_target: "-DGMX\_CUDA\_TARGET\_SM=35"

when:

- cuda\_target is not defined

- inventory\_hostname in groups['nodes\_gtx780ti']

- name: Set GPU architecture from inventory for RTX 2080 (fallback)

set\_fact:

cuda\_target: "-DGMX\_CUDA\_TARGET\_SM=75"

when:

- cuda\_target is not defined

- inventory\_hostname in groups['nodes\_rtx2080']

- name: Set default GPU architecture

set\_fact:

cuda\_target: "-DGMX\_CUDA\_TARGET\_SM=35;52;60;75"

when: cuda\_target is not defined

- name: ensure a source directory exists

file:

path: "{{ source\_directory }}"

state: directory

# ここから下は既存インストールがない場合のみ実行

- block:

- name: unarchive source files

unarchive:

src: "gromacs-{{ version }}.tar.gz"

dest: "{{ source\_directory }}"

- name: fix cmake

lineinfile:

path: "{{ source\_directory }}/gromacs-{{ version }}/cmake/gmxManageNvccConfig.cmake"

regexp: "{{ item }}"

state: absent

with\_items:

- '(.\*arch=compute\_20,code=sm\_20.\*)'

- '(.\*arch=compute\_30,code=sm\_30.\*)'

- name: create builddir

tempfile:

state: directory

suffix: gromacs-build

register: builddir

- name: unarchive regressiontest files

unarchive:

src: "regressiontests-{{ version }}.tar.gz"

dest: "{{ source\_directory }}"

- name: configure

command: >

cmake {{ source\_directory }}/gromacs-{{ version }}

-DCMAKE\_C\_COMPILER=gcc-8

-DCMAKE\_CXX\_COMPILER=g++-8

-DGMX\_SIMD=AVX2\_256

-DGMX\_GPU=ON

{{ cuda\_target }}

-DCUDA\_TOOLKIT\_ROOT\_DIR=/usr/local/cuda

-DREGRESSIONTTEST\_PATH={{ source\_directory}}/regressiontests-{{ version }}

-DCMAKE\_INSTALL\_PREFIX={{ install\_prefix }}-{{ version }}

args:

chdir: "{{ builddir.path }}"

- name: compile, test and install

make:

target: "{{ item }}"

chdir: "{{ builddir.path }}"

with\_items:

- all

- check

- install

- name: clear builddir

file:

path: "{{ builddir.path }}"

state: absent

- name: make symbolic link

file:

src: "{{ install\_prefix }}-{{ version }}"

dest: "{{ install\_prefix }}"

state: link

when: not gmx\_result.stat.exists

==== ./roles/slurm/handlers/main.yml ====

- name: update library location

command: "ldconfig -n {{ install\_prefix }}/lib"

listen: update ldconfig

- name: restart SLURM services

systemd:

name: "{{ item }}"

state: restarted

daemon\_reload: yes

loop: "{{ services }}"

listen: restart services

==== ./roles/slurm/meta/main.yml ====

---

dependencies:

- munge

==== ./roles/slurm/defaults/main.yml ====

services: []

source\_directory: /opt/src

install\_prefix: /opt/slurm

slurm\_version: 22.05.7

user: slurm

spool\_dir: /var/spool/slurm.spool

state\_dir: /var/spool/slurm.state

accounting\_file: /var/spool/slurm.accounting

slurm\_logdir: /var/log/slurm

==== ./roles/slurm/tasks/main.yml ====

# /srv/ansible/roles/slurm/tasks/main.yml

---

- name: install build-essential

apt:

name: build-essential

state: present

- name: Check if slurm binary exists

stat:

path: "{{ install\_prefix }}/sbin/slurmctld"

register: slurmctld

- name: Check Slurm functionality

command: "{{ install\_prefix }}/bin/scontrol version"

register: slurm\_version\_check

ignore\_errors: yes

changed\_when: false

when: slurmctld.stat.exists

- name: make a source directory

file:

path: "{{ source\_directory }}"

state: directory

- name: unarchive source files

unarchive:

src: "slurm-{{ slurm\_version }}.tar.bz2"

dest: "{{ source\_directory }}"

when: not slurmctld.stat.exists or (slurm\_version\_check is defined and slurm\_version\_check.rc != 0)

- name: configure SLURM

command: "./configure --prefix={{ install\_prefix }}"

args:

chdir: "{{ source\_directory }}/slurm-{{ slurm\_version }}"

register: configured

when: not slurmctld.stat.exists or (slurm\_version\_check is defined and slurm\_version\_check.rc != 0)

- name: "make {{ install\_prefix }}/etc"

file:

path: "{{ install\_prefix }}/etc"

state: directory

- name: Check if slurm.conf exists

stat:

path: "{{ install\_prefix }}/etc/slurm.conf"

register: slurm\_conf\_exists

- name: Backup existing slurm.conf

copy:

src: "{{ install\_prefix }}/etc/slurm.conf"

dest: "{{ install\_prefix }}/etc/slurm.conf.bak.{{ ansible\_date\_time.iso8601 }}"

when: slurm\_conf\_exists.stat.exists

- name: Check current Slurm configuration

command: grep -c "NodeName=" {{ install\_prefix }}/etc/slurm.conf

register: slurm\_node\_count

ignore\_errors: yes

changed\_when: false

when: slurm\_conf\_exists.stat.exists

- name: Generate expected Slurm configuration to temporary file

template:

src: slurm2.conf.j2

dest: "/tmp/slurm\_new.conf"

check\_mode: yes

when: slurm\_conf\_exists.stat.exists

- name: Count nodes in expected configuration

command: grep -c "NodeName=" /tmp/slurm\_new.conf

register: expected\_node\_count

ignore\_errors: yes

changed\_when: false

when: slurm\_conf\_exists.stat.exists

- name: Update slurm.conf if node count differs or doesn't exist

template:

src: slurm2.conf.j2

dest: "{{ install\_prefix }}/etc/slurm.conf"

register: slurm\_conf

notify: restart services

when: >

not slurm\_conf\_exists.stat.exists or

(slurm\_node\_count is defined and expected\_node\_count is defined and

slurm\_node\_count.stdout != expected\_node\_count.stdout)

- name: compile and install SLURM

make:

target: "{{ item }}"

chdir: "{{ source\_directory }}/slurm-{{ slurm\_version }}"

with\_items:

- all

- install

when: >

not slurmctld.stat.exists or

(slurm\_version\_check is defined and slurm\_version\_check.rc != 0) or

slurm\_conf.changed

- name: Check if ldconf file exists

stat:

path: /etc/ld.so.conf.d/slurm.conf

register: ld\_conf\_exists

- name: add a ldconf file for SLURM

copy:

dest: /etc/ld.so.conf.d/slurm.conf

content: "{{ install\_prefix }}/lib"

notify: update ldconfig

when: not ld\_conf\_exists.stat.exists

- name: make directories for SLURM

file:

path: "{{ item }}"

state: directory

recurse: yes

owner: "{{ user }}"

with\_items:

- "{{ state\_dir }}"

- "{{ spool\_dir }}"

- "{{ slurm\_logdir }}"

- stat:

path: "{{ accounting\_file }}"

register: acct\_result

- name: touch a file for accouting

file:

path: "{{ accounting\_file }}"

owner: "{{ user }}"

state: touch

when: not acct\_result.stat.exists

- name: Check if gres.conf exists

stat:

path: "{{ install\_prefix }}/etc/gres.conf"

register: gres\_conf\_exists

- name: Backup existing gres.conf

copy:

src: "{{ install\_prefix }}/etc/gres.conf"

dest: "{{ install\_prefix }}/etc/gres.conf.bak.{{ ansible\_date\_time.iso8601 }}"

when: gres\_conf\_exists.stat.exists

- name: copy gres.conf only if it doesn't exist

copy:

src: gres.conf

dest: "{{ install\_prefix }}/etc"

when: not gres\_conf\_exists.stat.exists

- name: Check for service files

stat:

path: "/lib/systemd/system/{{ item }}"

register: service\_files

with\_items: "{{ services }}"

loop\_control:

label: "{{ item }}"

- name: copy service files for SLURM

copy:

src: "{{ source\_directory }}/slurm-{{ slurm\_version }}/etc/{{ item }}"

dest: "/lib/systemd/system/{{ item }}"

remote\_src: yes

with\_items: "{{ services }}"

when: not service\_files.results[0].stat.exists

notify: restart services

- name: enable services

systemd:

name: "{{ item }}"

enabled: yes

daemon\_reload: yes

loop: "{{ services }}"

- name: set path setting script for slurm in admin

copy:

src: slurm-bin-path.sh

dest: "/etc/profile.d/slurm-bin-path.sh"

==== ./roles/munge/handlers/main.yml ====

- name: restart munge

systemd:

name: munge.service

state: restarted

==== ./roles/munge/tasks/main.yml ====

- name: install MUNGE

apt:

name: ["munge", "libmunge-dev"]

state: present

notify: restart munge

- name: copy munge.key

copy:

dest: /etc/munge/munge.key

src: munge.key

notify: restart munge

==== ./roles/cuda-12.0/handlers/main.yml ====

- name: reboot the machine

reboot:

==== ./roles/cuda-12.0/defaults/main.yml ====

deb\_version: ubuntu2204-12-0-local\_12.0.0-525.60.13-1

cuda\_version: ubuntu2204-12-0-local

key\_version: 825BBB4F

==== ./roles/cuda-12.0/tasks/main.yml ====

- name: copy a .deb files

copy:

dest: /tmp/cuda.deb

src: "cuda-repo-{{ deb\_version }}\_amd64.deb"

- name: install a .deb package

apt:

deb: "/tmp/cuda.deb"

- name: copy the keyring file to the /usr/share/keyring directory

copy:

src: "{{ item }}"

dest: "/usr/share/keyrings/"

with\_fileglob:

- "/var/cuda-repo-{{ deb\_version }}-local/cuda-\*-keyring.gpg"

- name: install cuda

apt:

name: cuda

state: present

update\_cache: yes

notify: reboot the machine

- name: upgrade cuda

apt:

name: cuda

state: latest

update\_cache: yes

notify: reboot the machine

- meta: flush\_handlers

==== ./roles/cuda-11.4/handlers/main.yml ====

- name: reboot the machine

reboot:

==== ./roles/cuda-11.4/defaults/main.yml ====

deb\_version: ubuntu2004-11-4-local\_11.4.0-470.42.01-1

cuda\_version: ubuntu2004-11-4-local

==== ./roles/cuda-11.4/tasks/main.yml ====

# /srv/ansible/roles/cuda-11.4/tasks/main.yml

---

- name: Check source file existence

stat:

path: "{{ role\_path }}/files/cuda-repo-{{ deb\_version }}\_amd64.deb"

register: cuda\_source\_file

delegate\_to: localhost

- name: Debug source file path

debug:

msg: "Source file exists: {{ cuda\_source\_file.stat.exists }} at {{ role\_path }}/files/cuda-repo-{{ deb\_version }}\_amd64.deb"

- name: Copy CUDA .deb files

copy:

src: "files/cuda-repo-{{ deb\_version }}\_amd64.deb"

dest: "/tmp/cuda.deb"

mode: '0644'

when: cuda\_source\_file.stat.exists

register: copy\_result

- name: Skip CUDA installation if source file missing

debug:

msg: "WARNING: CUDA source file not found. Installation will be skipped."

when: not cuda\_source\_file.stat.exists

- name: Verify .deb file exists after copy

stat:

path: "/tmp/cuda.deb"

register: cuda\_deb\_check

when: cuda\_source\_file.stat.exists

- name: Show copy status

debug:

msg: "CUDA .deb file exists: {{ cuda\_deb\_check.stat.exists | default(false) }}"

when: cuda\_source\_file.stat.exists

- name: Install .deb package

apt:

deb: "/tmp/cuda.deb"

when: cuda\_source\_file.stat.exists and (cuda\_deb\_check.stat.exists | default(false))

register: deb\_installed

- name: Add apt key

apt\_key:

file: "/var/cuda-repo-{{ cuda\_version }}/7fa2af80.pub"

when: deb\_installed is defined and deb\_installed.changed

register: key\_added

- name: Install CUDA

apt:

name: cuda-11-4

state: present

update\_cache: yes

when: key\_added is defined and key\_added.changed

notify: reboot the machine

# Commented out as per original file

#- name: Upgrade CUDA

# apt:

# name: cuda

# state: latest

# update\_cache: yes

# notify: reboot the machine

- meta: flush\_handlers

==== ./roles/gromacs/meta/main.yml ====

---

dependencies:

- cuda

==== ./roles/gromacs/defaults/main.yml ====

version: 5.1.1

source\_directory: /opt/src

install\_prefix: /opt/gromacs

==== ./roles/gromacs/tasks/main.yml ====

- name: install fftw and cmake

apt:

name: ["libfftw3-dev", "cmake"]

state: present

- stat:

path: "{{ install\_prefix }}-{{ version }}/bin/gmx"

register: gmx\_result

- name: ensure a source directory exists

file:

path: "{{ source\_directory }}"

state: directory

- name: unarchive source files

unarchive:

src: "gromacs-{{ version }}.tar.gz"

dest: "{{ source\_directory }}"

when: not gmx\_result.stat.exists

- name: fix cmake

lineinfile:

path: "{{ source\_directory }}/gromacs-{{ version }}/cmake/gmxManageNvccConfig.cmake"

regexp: '(.\*arch=compute\_20,code=sm\_20.\*)'

state: absent

when: not gmx\_result.stat.exists

- name: create builddir

tempfile:

state: directory

suffix: gromacs-build

register: builddir

when: not gmx\_result.stat.exists

- name: configure

command: "cmake {{ source\_directory }}/gromacs-{{ version }} -DGMX\_SIMD=AVX2\_256 -DGMX\_GPU=ON -DCUDA\_TOOLKIT\_ROOT\_DIR=/usr/local/cuda -DREGRESSIONTEST\_DOWNLOAD=ON -DCMAKE\_INSTALL\_PREFIX={{ install\_prefix }}-{{ version }}"

args:

chdir: "{{ builddir.path }}"

when: not gmx\_result.stat.exists

- name: compile, test and install

make:

target: "{{ item }}"

chdir: "{{ builddir.path }}"

with\_items:

- all

- check

- install

when: not gmx\_result.stat.exists

- name: clear builddir

file:

path: "{{ builddir.path }}"

state: absent

when: not gmx\_result.stat.exists

==== ./roles/gromacs-2022.4/meta/main.yml ====

---

dependencies:

- cuda-12.0

==== ./roles/gromacs-2022.4/defaults/main.yml ====

version: 2022.4

source\_directory: /opt/src

install\_prefix: /opt/gromacs

==== ./roles/gromacs-2022.4/tasks/main.yml ====

- name: install fftw and cmake

apt:

name: ["libfftw3-dev", "cmake"]

state: present

- stat:

path: "{{ install\_prefix }}-{{ version }}/bin/gmx"

register: gmx\_result

- name: ensure a source directory exists

file:

path: "{{ source\_directory }}"

state: directory

- name: unarchive source files

unarchive:

src: "gromacs-{{ version }}.tar.gz"

dest: "{{ source\_directory }}"

when: not gmx\_result.stat.exists

- name: create builddir

tempfile:

state: directory

suffix: gromacs-build

register: builddir

when: not gmx\_result.stat.exists

- name: configure

command: "cmake {{ source\_directory }}/gromacs-{{ version }} -DGMX\_SIMD=AVX2\_256 -DGMX\_GPU=CUDA -DCUDA\_TOOLKIT\_ROOT\_DIR=/usr/local/cuda -DREGRESSIONTEST\_DOWNLOAD=ON -DCMAKE\_INSTALL\_PREFIX={{ install\_prefix }}-{{ version }}"

args:

chdir: "{{ builddir.path }}"

when: not gmx\_result.stat.exists

- name: compile, test and install

make:

target: "{{ item }}"

chdir: "{{ builddir.path }}"

with\_items:

- all

- check

- install

when: not gmx\_result.stat.exists

- name: clear builddir

file:

path: "{{ builddir.path }}"

state: absent

when: not gmx\_result.stat.exists

==== ./roles/nis-server/handlers/main.yml ====

- name: restart nis

systemd:

name: "{{ item }}"

state: restarted

loop:

- rpcbind

- nis

==== ./roles/nis-server/defaults/main.yml ====

domain: "GolgiAdmin.golgi"

==== ./roles/nis-server/tasks/main.yml ====

- name: install nis package

apt:

name: nis

state: present

update\_cache: yes

notify: restart nis

- name: set defaultdomain

copy:

dest: /etc/defaultdomain

content: "{{ domain }}"

notify: restart nis

- name: configure /etc/default/nis

replace:

path: /etc/default/nis

regexp: '^NISSERVER=false'

replace: 'NISSERVER=master'

- name: remove all IP accept setting for admin node

replace:

path: /etc/ypserv.securenets

regexp: '^0.0.0.0 0.0.0.0'

replace: '#0.0.0.0 0.0.0.0'

notify: restart nis

- name: add IP range restriction for admin node

lineinfile:

path: /etc/ypserv.securenets

insertafter: '^#0.0.0.0'

line: '255.255.255.0 192.168.2.0'

notify: restart nis

- meta: flush\_handlers

- name: install pip3 to install pexpect

apt:

name: python3-pip

state: present

update\_cache: yes

- name: set domainname

command: "domainname {{ domain }}"

- name: install pexpect for ypinit

pip:

name: pexpect

become: yes

- name: excute ypinit

expect:

command: /usr/lib/yp/ypinit -m

responses:

"^.\*next host to add:": "\x04"

"^Is this correct?.\*[y/n: y]": "y"

- name: modify MINGID in yp Makefile

lineinfile:

path: /var/yp/Makefile

regexp: '^MINGID='

line: 'MINGID=999'

==== ./roles/cuda-11.4-admin/handlers/main.yml ====

- name: reboot the machine

reboot:

==== ./roles/cuda-11.4-admin/defaults/main.yml ====

deb\_version: ubuntu2004-11-4-local\_11.4.0-470.42.01-1

cuda\_version: ubuntu2004-11-4-local

==== ./roles/cuda-11.4-admin/tasks/main.yml ====

- name: copy a .deb files

copy:

dest: /tmp/cuda.deb

src: "cuda-repo-{{ deb\_version }}\_amd64.deb"

- name: install a .deb package

apt:

deb: "/tmp/cuda.deb"

- name: add an apt key

apt\_key:

file: "/var/cuda-repo-{{ cuda\_version }}/7fa2af80.pub"

- name: install cuda

apt:

name: cuda

state: present

update\_cache: yes

notify: reboot the machine

- name: upgrade cuda

apt:

name: cuda

state: latest

update\_cache: yes

notify: reboot the machine

- meta: flush\_handlers

==== ./roles/lm-sensors/tasks/main.yml ====

- name: install lm-sensors

apt:

name: lm-sensors

state: present

update\_cache: yes

==== ./roles/cuda-12.0-child/handlers/main.yml ====

- name: reboot the machine

reboot:

==== ./roles/cuda-12.0-child/defaults/main.yml ====

deb\_version: ubuntu2004-12-0-local\_12.0.0-525.60.13-1

cuda\_version: ubuntu2004-12-0-local

key\_version: 5E22DB91

==== ./roles/cuda-12.0-child/tasks/main.yml ====

- name: copy a .deb files

copy:

dest: /tmp/cuda.deb

src: "cuda-repo-{{ deb\_version }}\_amd64.deb"

- name: install a .deb package

apt:

deb: "/tmp/cuda.deb"

- name: copy the kering file to the /usr/share/keyring directory

copy:

src: "/var/cuda-repo-{{ cuda\_version }}/cuda-{{ key\_version }}-keyring.gpg"

dest: "/usr/share/keyrings/cuda-{{ key\_version }}-keyring.gpg"

remote\_src: yes

- name: install cuda

apt:

name: cuda

state: present

update\_cache: yes

notify: reboot the machine

- name: upgrade cuda

apt:

name: cuda

state: latest

update\_cache: yes

notify: reboot the machine

- meta: flush\_handlers

==== ./roles/child-alphafold/defaults/main.yml ====

install\_prefix: /opt/alphafold

ubuntu\_version: Ubuntu20.04

gpgkey\_path: gpgkey.gpg

ncr\_list: nvidia-container-runtime.list

==== ./roles/child-alphafold/tasks/main.yml ====

# /srv/ansible/roles/child-alphafold/tasks/main.yml

---

- name: Check if AlphaFold directory exists

stat:

path: "{{ install\_prefix }}"

register: alphafold\_dir

- name: Check if AlphaFold is already mounted

shell: mount | grep "{{ install\_prefix }}"

register: alphafold\_mount

ignore\_errors: yes

changed\_when: false

- name: create directory for alphafold

file:

path: "{{ install\_prefix }}"

state: directory

when: not alphafold\_dir.stat.exists

- name: mount admin /opt/alphafold

mount:

path: "{{ install\_prefix }}"

src: "GolgiAdmin:/opt/alphafold"

fstype: nfs

state: mounted

when: alphafold\_mount.rc != 0 or alphafold\_mount.stdout == ""

- name: Check if Docker is installed

command: which docker

register: docker\_check

ignore\_errors: yes

changed\_when: false

- name: install docker for alphafold

apt:

pkg:

- docker-ce

- docker-ce-cli

update\_cache: yes

state: latest

when: docker\_check.rc != 0

- name: Check if CUDA Docker image exists

command: docker images -q nvidia/cuda:11.4.0-base

register: cuda\_image

ignore\_errors: yes

changed\_when: false

- name: pull docker image for nvidia/cuda:11.4-base

command:

cmd: docker pull nvidia/cuda:11.4.0-base

when: cuda\_image.stdout == ""

- name: Verify nvidia-container-runtime repository file exists

stat:

path: /etc/apt/sources.list.d/nvidia-container-runtime.list

register: ncr\_list\_check

- name: copy gpgkey to tmp

copy:

dest: "/tmp/{{ gpgkey\_path }}"

src: "{{ gpgkey\_path }}"

when: not ncr\_list\_check.stat.exists

- name: Check if GPG key is already added

command: apt-key list

register: gpgkey\_check

ignore\_errors: yes

changed\_when: false

- name: add gpg key for nvidia-docker

command:

cmd: "sudo apt-key add /tmp/{{ gpgkey\_path }}"

when: gpgkey\_check.rc != 0 or "NVIDIA CORPORATION" not in gpgkey\_check.stdout

- name: copy nvidia-container-runtime list to /etc/apt/sources.list.d/

copy:

dest: "/etc/apt/sources.list.d/nvidia-container-runtime.list"

src: "{{ ncr\_list }}"

when: not ncr\_list\_check.stat.exists

- name: Verify nvidia-container-runtime repository file

command: cat /etc/apt/sources.list.d/nvidia-container-runtime.list

register: ncr\_content

changed\_when: false

when: ncr\_list\_check.stat.exists

- name: Check if repository is compatible with Ubuntu version

shell: grep -q "ubuntu{{ ansible\_distribution\_version }}" /etc/apt/sources.list.d/nvidia-container-runtime.list

register: ubuntu\_version\_check

ignore\_errors: yes

changed\_when: false

when: ncr\_list\_check.stat.exists

- name: Warn if repository might not be compatible

debug:

msg: "WARNING: The NVIDIA Docker repository might not be compatible with Ubuntu {{ ansible\_distribution\_version }}. Please verify the repository file."

when: ubuntu\_version\_check is defined and ubuntu\_version\_check.get('rc', 0) != 0

- name: update apt

apt:

update\_cache: yes

- name: Check if Docker service is running

command: systemctl status docker

register: docker\_status

ignore\_errors: yes

changed\_when: false

- name: restart docker

service:

name: docker

state: restarted

when: docker\_status.rc != 0

- name: Check if NVIDIA Docker is working

command: docker run --rm --gpus all nvidia/cuda:11.4.0-base nvidia-smi

register: nvidia\_docker\_check

ignore\_errors: yes

changed\_when: false

- name: confirm or build nvidia/cuda:11.4.0-base

command:

cmd: docker run --rm --gpus all nvidia/cuda:11.4.0-base nvidia-smi

when: nvidia\_docker\_check.rc != 0

- name: Check if AlphaFold Docker image exists

command: docker images -q alphafold

register: alphafold\_image

ignore\_errors: yes

changed\_when: false

- name: Build AlphaFold Docker image with timeout extension

block:

- name: Execute Docker build command

command: "docker build -f {{ install\_prefix }}/alphafold/docker/Dockerfile -t alphafold {{ install\_prefix }}/alphafold"

async: 3600

poll: 60

rescue:

- name: Display error message if build fails

debug:

msg: "AlphaFold Docker image build failed. This might be due to network issues or insufficient resources. Please try building manually."

when: alphafold\_image.stdout == "" and not ansible\_check\_mode

- name: install python3-pip

apt:

name: python3-pip

state: present

- name: Check if Python requirements file exists

stat:

path: "{{ install\_prefix }}/alphafold/docker/requirements.txt"

register: req\_file

- name: install docker requirements by pip

command:

cmd: "python3 -m pip install -r {{ install\_prefix }}/alphafold/docker/requirements.txt"

when: req\_file.stat.exists

==== ./roles/cuda-10.2/handlers/main.yml ====

# /srv/ansible/roles/cuda-10.2/handlers/main.yml

---

- name: reboot the machine

block:

- name: Check for running jobs again before reboot

command: squeue -h -o "%i" -t running -w {{ ansible\_hostname }}

register: running\_jobs\_before\_reboot

ignore\_errors: yes

changed\_when: false

- name: Check for critical processes again before reboot

shell: ps aux | grep -v grep | grep -E '(python|gmx|docker|alphafold)' | wc -l

register: process\_count\_before\_reboot

ignore\_errors: yes

changed\_when: false

- name: Set fact for running jobs

set\_fact:

has\_running\_jobs: >

{{ (running\_jobs\_before\_reboot.rc == 0 and running\_jobs\_before\_reboot.stdout\_lines | length > 0) or

(process\_count\_before\_reboot.stdout | int > 5) }}

- name: Abort reboot if jobs are detected

fail:

msg: "Reboot aborted - running jobs detected at reboot time"

when: has\_running\_jobs

- name: Notify users of imminent reboot via wall

command: 'wall "SYSTEM NOTICE: This node will reboot in 2 minutes due to CUDA driver updates. Please save your work."'

ignore\_errors: yes

changed\_when: false

when: not has\_running\_jobs

- name: Wait 2 minutes before reboot

pause:

minutes: 2

when: not has\_running\_jobs

- name: Perform actual reboot

reboot:

reboot\_timeout: 600

pre\_reboot\_delay: 5

post\_reboot\_delay: 30

test\_command: uptime

when: not has\_running\_jobs

==== ./roles/cuda-10.2/defaults/main.yml ====

deb\_version: ubuntu1804-10-2-local-10.2.89-440.33.01\_1.0-1

apt\_key\_version: 10-2-local-10.2.89-440.33.01

patch\_prefix: ubuntu1804-10-2-local\_10.2

==== ./roles/cuda-10.2/tasks/main.yml ====

# /srv/ansible/roles/cuda-10.2/tasks/main.yml

---

- name: Check if CUDA 10.2 is already installed

command: /usr/local/cuda/bin/nvcc --version

register: cuda\_version\_check

ignore\_errors: yes

changed\_when: false

- name: Check CUDA installation directory

stat:

path: /usr/local/cuda-10.2

register: cuda\_10\_2\_dir

- name: Check if GPU drivers are already working

command: nvidia-smi

register: nvidia\_smi\_check

ignore\_errors: yes

changed\_when: false

- name: Check for running jobs

command: squeue -h -o "%i" -t running -w {{ ansible\_hostname }}

register: running\_jobs

ignore\_errors: yes

changed\_when: false

- name: Display warning about running jobs

debug:

msg: "WARNING: There are running jobs. Installation will proceed but reboot will be skipped."

when: running\_jobs.stdout\_lines | length > 0

- name: Proceed with CUDA 10.2 installation only if needed

block:

- name: Check source file existence

stat:

path: "{{ role\_path }}/files/cuda-repo-{{ deb\_version }}\_amd64.deb"

register: cuda\_source\_file

delegate\_to: localhost

- name: Debug source file path

debug:

msg: "Source file exists: {{ cuda\_source\_file.stat.exists }} at {{ role\_path }}/files/cuda-repo-{{ deb\_version }}\_amd64.deb"

- name: Copy CUDA .deb files

copy:

src: "files/cuda-repo-{{ deb\_version }}\_amd64.deb"

dest: "/tmp/cuda.deb"

mode: '0644'

when: cuda\_source\_file.stat.exists

register: copy\_result

- name: Skip CUDA installation if source file missing

debug:

msg: "WARNING: CUDA source file not found. Installation will be skipped."

when: not cuda\_source\_file.stat.exists

- name: Verify .deb file exists after copy

stat:

path: "/tmp/cuda.deb"

register: cuda\_deb\_check

when: cuda\_source\_file.stat.exists

- name: Show copy status

debug:

msg: "CUDA .deb file exists: {{ cuda\_deb\_check.stat.exists | default(false) }}"

when: cuda\_source\_file.stat.exists

- name: Install .deb package

apt:

deb: "/tmp/cuda.deb"

when: cuda\_source\_file.stat.exists and (cuda\_deb\_check.stat.exists | default(false))

register: deb\_installed

- name: Add apt key

apt\_key:

file: "/var/cuda-repo-{{ apt\_key\_version }}/7fa2af80.pub"

when: deb\_installed is defined and deb\_installed.changed

register: key\_added

- name: Install CUDA

apt:

name: cuda

state: present

update\_cache: yes

when: key\_added is defined and key\_added.changed

register: cuda\_installed

- name: Copy patches .deb files

copy:

dest: "/tmp/{{ item }}"

src: "{{ item }}"

with\_items:

- "cuda-repo-{{ patch\_prefix }}.1-1\_amd64.deb"

- "cuda-repo-{{ patch\_prefix }}.2-1\_amd64.deb"

when: cuda\_installed is defined and cuda\_installed.changed

- name: Install patches .deb packages

apt:

deb: "/tmp/{{ item }}"

force: yes

with\_items:

- "cuda-repo-{{ patch\_prefix }}.1-1\_amd64.deb"

- "cuda-repo-{{ patch\_prefix }}.2-1\_amd64.deb"

when: cuda\_installed is defined and cuda\_installed.changed

register: patches\_installed

- name: Upgrade CUDA

apt:

name: cuda

state: latest

update\_cache: yes

when: patches\_installed is defined and patches\_installed.changed

register: cuda\_upgraded

- name: Check for running processes (alternative method)

shell: ps aux | grep -v grep | grep -E '(python|gmx|docker|alphafold)' | wc -l

register: process\_count

ignore\_errors: yes

changed\_when: false

- name: Notify reboot only if CUDA was installed/upgraded and no jobs are running

command: echo "Reboot is needed"

notify: reboot the machine

when:

- (cuda\_installed is defined and cuda\_installed.changed) or (cuda\_upgraded is defined and cuda\_upgraded.changed)

- running\_jobs.stdout\_lines | length == 0

- process\_count.stdout | int < 5

when: cuda\_version\_check.rc != 0 or "V10.2" not in cuda\_version\_check.stdout or nvidia\_smi\_check.rc != 0 or not cuda\_10\_2\_dir.stat.exists

==== ./roles/admin-alphafold/tasks/main.yml ====

- name: create directory for alphafold

file:

path: /opt/alphafold

state: directory

owner: alphafold

- name: mount /dev/sdb1 to /opt/alphafold

mount:

path: "/opt/alphafold"

src: "/dev/sdb1"

fstype: ext4

state: mounted

- name: install nfs-kernel-server

apt:

name: nfs-kernel-server

state: present

update\_cache: yes

- name: set export directory

lineinfile:

dest: /etc/exports

line: '/opt/alphafold 192.168.2.0/255.255.255.0(rw,sync,no\_subtree\_check,no\_root\_squash)'

insertbefore: EOF

state: present

==== ./roles/gfortran/tasks/main.yml ====

- name: install gfortran

apt:

name: gfortran

state: present

update\_cache: yes

==== ./roles/nis/handlers/main.yml ====

- name: restart nis

systemd:

name: "{{ item }}"

state: restarted

loop:

- rpcbind

- nis

==== ./roles/nis/defaults/main.yml ====

domain: "GolgiAdmin.golgi"

server: "GolgiAdmin.golgi"

==== ./roles/nis/tasks/main.yml ====

# /srv/ansible/roles/nis-server/tasks/main.yml

---

- name: Check if NIS server is running

command: systemctl status ypserv

register: ypserv\_status

ignore\_errors: yes

changed\_when: false

- name: Check if NIS database is properly initialized

command: ypcat passwd

register: nis\_db\_status

ignore\_errors: yes

changed\_when: false

- name: install nis package

apt:

name: nis

state: present

update\_cache: yes

notify: restart nis

when: ypserv\_status.rc != 0

- name: Check if defaultdomain is already set

command: cat /etc/defaultdomain

register: default\_domain

ignore\_errors: yes

changed\_when: false

- name: set defaultdomain

copy:

dest: /etc/defaultdomain

content: "{{ domain }}"

notify: restart nis

when: default\_domain.rc != 0 or domain not in default\_domain.stdout

- name: Check NIS server configuration

command: grep -E "^NISSERVER=" /etc/default/nis

register: nis\_server\_config

ignore\_errors: yes

changed\_when: false

- name: configure /etc/default/nis

replace:

path: /etc/default/nis

regexp: '^NISSERVER=false'

replace: 'NISSERVER=master'

when: nis\_server\_config.rc != 0 or "NISSERVER=master" not in nis\_server\_config.stdout

- name: Check securenets configuration

command: grep -E "^#0.0.0.0 0.0.0.0" /etc/ypserv.securenets

register: securenets\_config

ignore\_errors: yes

changed\_when: false

- name: remove all IP accept setting for admin node

replace:

path: /etc/ypserv.securenets

regexp: '^0.0.0.0 0.0.0.0'

replace: '#0.0.0.0 0.0.0.0'

notify: restart nis

when: securenets\_config.rc != 0

- name: Check IP range restriction

command: grep -E "^255.255.255.0 192.168.2.0" /etc/ypserv.securenets

register: ip\_restriction

ignore\_errors: yes

changed\_when: false

- name: add IP range restriction for admin node

lineinfile:

path: /etc/ypserv.securenets

insertafter: '^#0.0.0.0'

line: '255.255.255.0 192.168.2.0'

notify: restart nis

when: ip\_restriction.rc != 0

- meta: flush\_handlers

- name: install pip3 to install pexpect

apt:

name: python3-pip

state: present

update\_cache: yes

- name: Check domainname

command: domainname

register: current\_domainname

ignore\_errors: yes

changed\_when: false

- name: set domainname

command: "domainname {{ domain }}"

when: current\_domainname.rc != 0 or domain not in current\_domainname.stdout

- name: install pexpect for ypinit

pip:

name: pexpect

become: yes

- name: Check if ypinit has been executed

stat:

path: /var/yp/nicknames

register: nicknames\_file

- name: excute ypinit

expect:

command: /usr/lib/yp/ypinit -m

responses:

"^.\*next host to add:": "\x04"

? |

^Is this correct?.\*[y/n: y]

: "y"

when: >

not nicknames\_file.stat.exists or

ypserv\_status.rc != 0 or

nis\_db\_status.rc != 0

- name: Check MINGID setting

command: grep -E "^MINGID=" /var/yp/Makefile

register: mingid

ignore\_errors: yes

changed\_when: false

- name: modify MINGID in yp Makefile

lineinfile:

path: /var/yp/Makefile

regexp: '^MINGID='

line: 'MINGID=999'

when: mingid.rc != 0 or "MINGID=999" not in mingid.stdout

- name: Update NIS database if changes were made

command: /usr/bin/make -C /var/yp

when: >

mingid.changed or

default\_domain.changed or

(nicknames\_file.stat.exists == false)

==== ./roles/child-ubuntu/tasks/main.yml ====

- name: set timezone to Asia/Tokyo

timezone:

name: Asia/Tokyo

- name: configure /etc/hosts

blockinfile:

path: /etc/hosts

block: |

192.168.2.200 GolgiAdmin GolgiAdmin.golgi

192.168.2.201 GolgiFS

- name: link python to python3

file:

src: /usr/bin/python3

dest: /usr/bin/python

state: link

- name: disable suspend mode

command: "sudo systemctl mask sleep.target suspend.target. hibernate.target hybrid-sleep.target"

- name: install zsh

apt:

name: zsh

state: present

==== ./roles/admin-ubuntu/tasks/main.yml ====

- name: set timezone to Asia/Tokyo

timezone:

name: Asia/Tokyo

- name: disable suspend mode

file:

src: /dev/null

dest: "{{ item }}"

state: link

with\_items:

- /etc/systemd/system/sleep.target

- /etc/systemd/system/suspend.target

- /etc/systemd/system/hibernate.target

- /etc/systemd/system/hybrid-sleep.target

- name: configure /etc/hosts

blockinfile:

path: /etc/hosts

block: |

192.168.2.200 GolgiAdmin GolgiAdmin.golgi

192.168.2.201 GolgiFS

192.168.2.1 Golgi01 golgi01

192.168.2.2 Golgi02 golgi02

192.168.2.3 Golgi03 golgi03

192.168.2.4 Golgi04 golgi04

192.168.2.5 Golgi05 golgi05

192.168.2.6 Golgi06 golgi06

192.168.2.7 Golgi07 golgi07

192.168.2.8 Golgi08 golgi08

192.168.2.9 Golgi09 golgi09

192.168.2.10 Golgi10 golgi10

192.168.2.11 Golgi11 golgi11

192.168.2.12 Golgi12 golgi12

192.168.2.13 Golgi13 golgi13

192.168.2.14 Golgi14 golgi14

192.168.2.15 Golgi15 golgi15

- name: copy script for admin

copy:

src: "{{ item }}"

dest: "/root/sbin/{{ item }}"

owner: root

group: root

with\_items:

- do\_all

- shutdown\_all

- name: IP forwarding setup

replace:

path: /etc/sysctl.conf

regexp: "^#net.ipv4.ip\_forward=1"

replace: "net.ipv4.ip\_forward=1"

- name: set iptables.rules

copy:

dest: /etc/iptables.rules

src: iptables.rules

- name: enable to restore iptables setting durint start-up

copy:

dest: /etc/network/if-pre-up.d/iptables-restore

src: iptables-restore

mode: '755'

- name: install ifupdown for iptables setting during start-up

apt:

name: ifupdown

state: present

- name: install zsh

apt:

name: zsh

state: present

- name: install packages for golgi temperature

apt:

name: ["parallel", "moreutils"]

state: present

## verify\_changes.yml の内容(おまけ)

# /srv/ansible/verify\_changes.yml

---

- name: Verify system functionality after changes

hosts: all

gather\_facts: no

tasks:

- name: Check network connectivity

command: ping -c 1 GolgiAdmin

register: ping\_result

ignore\_errors: yes

changed\_when: false

- name: Report network status

debug:

msg: "Network connectivity to GolgiAdmin: {{ 'OK' if ping\_result.rc == 0 else 'FAILED' }}"

- name: Check NIS functionality

command: ypcat passwd

register: nis\_result

ignore\_errors: yes

changed\_when: false

- name: Report NIS status

debug:

msg: "NIS functionality: {{ 'OK' if nis\_result.rc == 0 else 'FAILED' }}"

- name: Check NFS mounts

command: df -h

register: df\_result

ignore\_errors: yes

changed\_when: false

- name: Report NFS status

debug:

msg: "NFS mounts: {{ 'OK' if '/home' in df\_result.stdout else 'FAILED' }}"

- name: Check NVIDIA driver status

command: nvidia-smi

register: nvidia\_result

ignore\_errors: yes

changed\_when: false

- name: Report NVIDIA status

debug:

msg: "NVIDIA driver: {{ 'OK' if nvidia\_result.rc == 0 else 'FAILED' }}"

- name: Check Slurm functionality

command: sinfo

register: slurm\_result

ignore\_errors: yes

changed\_when: false

- name: Report Slurm status

debug:

msg: "Slurm functionality: {{ 'OK' if slurm\_result.rc == 0 else 'FAILED' }}"

- name: Check Gromacs functionality

command: which gmx

register: gromacs\_check

ignore\_errors: yes

changed\_when: false

- name: Check Gromacs version if installed

command: gmx --version

register: gromacs\_version

ignore\_errors: yes

changed\_when: false

when: gromacs\_check.rc == 0

- name: Report Gromacs status

debug:

msg: "Gromacs: {{ 'OK - Version ' + gromacs\_version.stdout.split('\n')[0] if gromacs\_check.rc == 0 else 'NOT FOUND' }}"

when: gromacs\_check.rc == 0 or gromacs\_version is defined

- name: Check Docker and AlphaFold functionality

command: docker images alphafold

register: alphafold\_check

ignore\_errors: yes

changed\_when: false

- name: Report AlphaFold status

debug:

msg: "AlphaFold Docker image: {{ 'AVAILABLE' if 'alphafold' in alphafold\_check.stdout else 'NOT FOUND' }}"

when: alphafold\_check.rc == 0

- name: Collect all verification results

set\_fact:

system\_status: "{{ {

'network': ping\_result.rc == 0,

'nis': nis\_result.rc == 0,

'nfs': '/home' in df\_result.stdout,

'nvidia': nvidia\_result.rc == 0,

'slurm': slurm\_result.rc == 0

} }}"

- name: Generate overall system status report

debug:

msg:

- "==== System Verification Report for {{ inventory\_hostname }} ===="

- "Network Connectivity: {{ 'OK' if system\_status.network else 'FAILED' }}"

- "NIS Functionality: {{ 'OK' if system\_status.nis else 'FAILED' }}"

- "NFS Mounts: {{ 'OK' if system\_status.nfs else 'FAILED' }}"

- "NVIDIA Driver: {{ 'OK' if system\_status.nvidia else 'FAILED' }}"

- "Slurm Functionality: {{ 'OK' if system\_status.slurm else 'FAILED' }}"

- "Overall Status: {{ 'OK' if (system\_status.network and system\_status.nis and system\_status.nfs and system\_status.nvidia and system\_status.slurm) else 'ISSUES DETECTED' }}"

- name: Send alert if verification failed

debug:

msg: "ATTENTION: One or more system components failed verification on {{ inventory\_hostname }}. Please check the detailed report above."

when: not (system\_status.network and system\_status.nis and system\_status.nfs and system\_status.nvidia and system\_status.slurm)

## backup\_configs.yml の内容(おまけ)

# /srv/ansible/backup\_configs.yml

---

- hosts: all

become: yes

gather\_facts: yes

tasks:

- name: Check free disk space on /root

shell: df -h /root | tail -1 | awk '{print $4}'

register: free\_space

changed\_when: false

- name: Warn if disk space is low

debug:

msg: "WARNING: Low disk space for backups. Available: {{ free\_space.stdout }}"

when: free\_space.stdout is defined and ((free\_space.stdout | regex\_replace('G', '') | float) < 1.0)

- name: Create dated backup directory

file:

path: "/root/ansible\_backups/{{ ansible\_date\_time.date }}"

state: directory

mode: '0700'

- name: Backup common configuration files

copy:

src: "{{ item }}"

dest: "/root/ansible\_backups/{{ ansible\_date\_time.date }}/{{ item | basename }}.bak"

remote\_src: yes

with\_items:

- /etc/hosts

- /etc/nsswitch.conf

- /etc/defaultdomain

- /etc/yp.conf

ignore\_errors: yes

- name: Backup SLURM configuration files if they exist

copy:

src: "{{ item }}"

dest: "/root/ansible\_backups/{{ ansible\_date\_time.date }}/{{ item | basename }}.bak"

remote\_src: yes

with\_items:

- "{{ lookup('env', 'SLURM\_CONF') | default('/opt/slurm/etc/slurm.conf') }}"

- "{{ lookup('env', 'SLURM\_CONF') | default('/opt/slurm/etc') }}/gres.conf"

ignore\_errors: yes

- name: Backup NIS configuration files on server

copy:

src: "{{ item }}"

dest: "/root/ansible\_backups/{{ ansible\_date\_time.date }}/{{ item | basename }}.bak"

remote\_src: yes

with\_items:

- /var/yp/Makefile

- /etc/ypserv.securenets

ignore\_errors: yes

when: "'localhost' in inventory\_hostname"

- name: Backup NFS configuration files on server

copy:

src: "{{ item }}"

dest: "/root/ansible\_backups/{{ ansible\_date\_time.date }}/{{ item | basename }}.bak"

remote\_src: yes

with\_items:

- /etc/exports

ignore\_errors: yes

when: "'localhost' in inventory\_hostname"

- name: Backup additional configuration files if they exist

copy:

src: "{{ item }}"

dest: "/root/ansible\_backups/{{ ansible\_date\_time.date }}/{{ item | basename }}.bak"

remote\_src: yes

with\_items:

- /etc/ssh/sshd\_config

- /etc/network/interfaces

- /etc/fstab

- /etc/resolv.conf

- /etc/apt/apt.conf.d/20auto-upgrades

ignore\_errors: yes