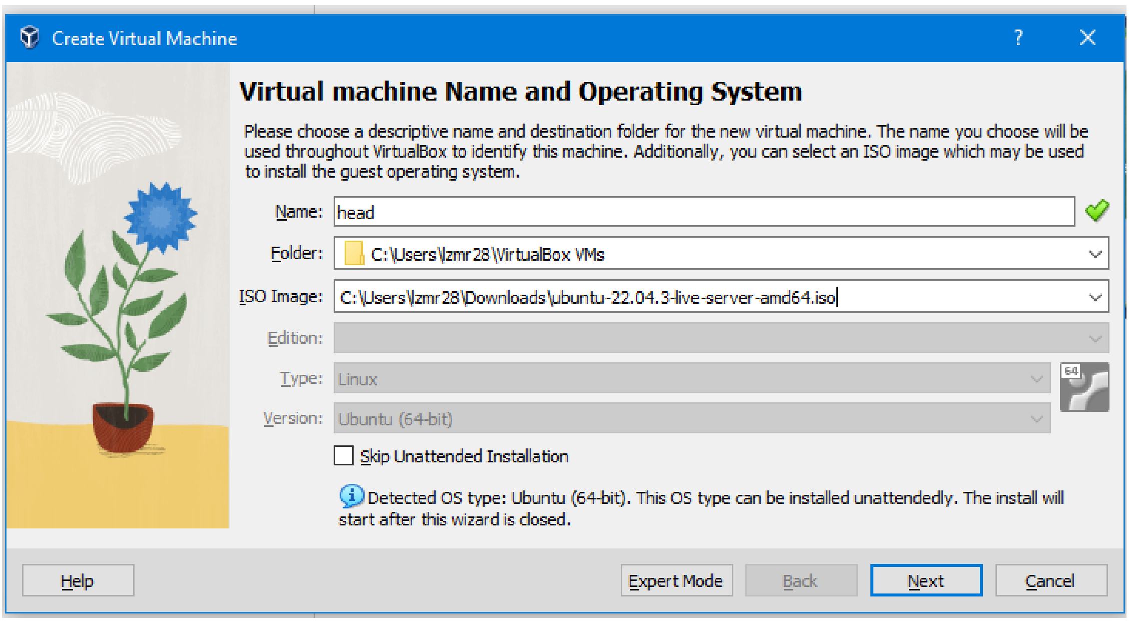
Building a virtual Beowulf cluster

Our goal is to configure a virtual mini cluster that can run distributed applications via MPI. We will start out by installing and configuring an operating system image for the head node, which we will then clone and use for both our compute nodes. Next, we will configure the shared file system NFS to enable access to shared data from any of the nodes. Afterwards, we will enable password-less ssh logins between all nodes such that our MPI processes are able to login to all nodes non-interactively. And finally, you will run your favourite MPI application on your cluster.

On the host system:

1. Download the Ubuntu Server 22.04.4 LTS image from https://ubuntu.com/download/server.
2. In VirtualBox, create a new VM based on the image by clicking "New" and following the guided VM creation to achieve settings similar to the screenshots.

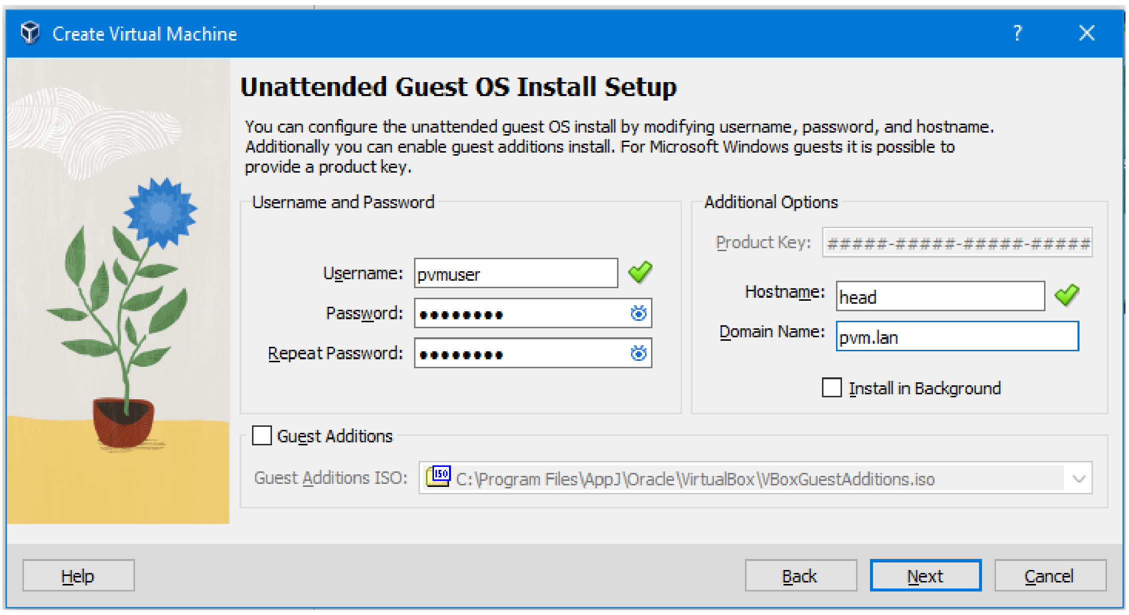


Virtual machine Name and Operating System

Please choose a descriptive name and destination folder for the new virtual machine. The name you choose will be used throughout VirtualBox to identify this machine. Additionally, you can select an ISO image which may be used to install the guest operating system.

Name: head
Folder: C:\Users\lzmr28\VirtualBox VMs
ISO Image: C:\Users\lzmr28\Downloads\ubuntu-22.04.3-live-server-amd64.iso
Edition:
Type: Linux
Version: Ubuntu (64-bit)
Skip Unattended Installation

Detected OS type: Ubuntu (64-bit). This OS type can be installed unattendedly. The install will start after this wizard is closed.



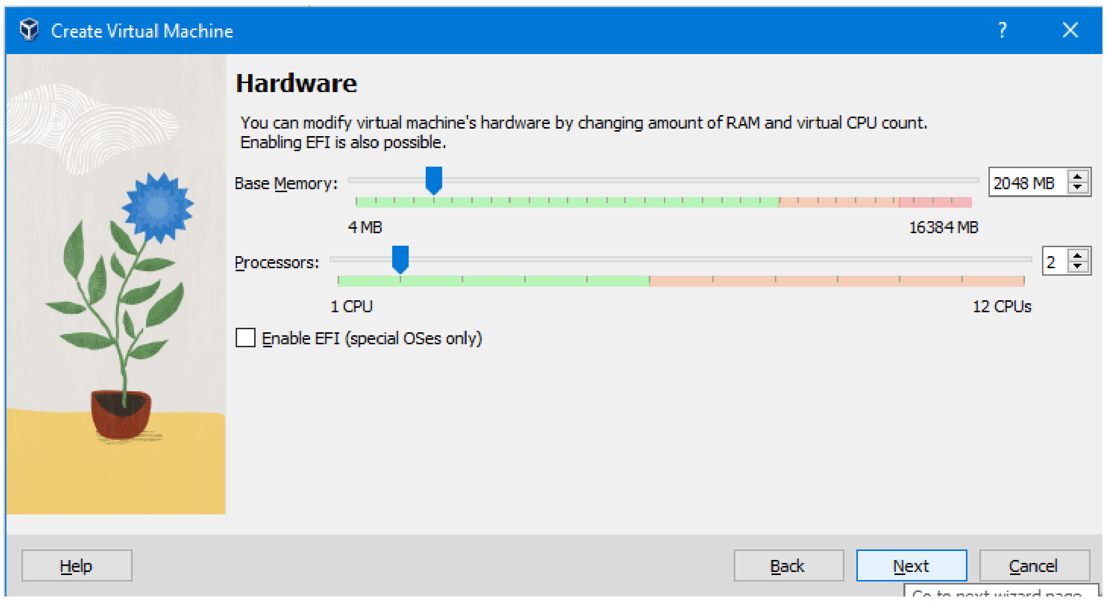
Unattended Guest OS Install Setup

You can configure the unattended guest OS install by modifying username, password, and hostname. Additionally you can enable guest additions install. For Microsoft Windows guests it is possible to provide a product key.

Username and Password
Username: pvmuser
Password: ••••••••
Repeat Password: ••••••••

Additional Options
Product Key: #####-#####-#####-#####
Hostname: head
Domain Name: pvm.lan
Install in Background

Guest Additions
Guest Additions ISO: C:\Program Files\AppJ\Oracle\VirtualBox\VBoxGuestAdditions.iso


Create Virtual Machine

Hardware

You can modify virtual machine's hardware by changing amount of RAM and virtual CPU count. Enabling EFI is also possible.

Base Memory: 2048 MB (4 MB – 16384 MB)

Processors: 2 (1 CPU – 12 CPUs)

☐ Enable EFI (special OSes only)

In the VM:

1. Install the operating system. Ensure to install OpenSSH during this process.
2. Login with your chosen credentials.
3. Install the following:
   ```
   sudo apt update
   sudo apt install net-tools
   sudo apt install nfs-kernel-server
   sudo apt install nfs-common
   sudo apt install build-essential
   sudo apt install openmpi-bin openmpi-doc libopenmpi-dev
   sudo apt install openvswitch-switch
   ```
4. Shutdown the VM in order to allow cloning:
   ```
   shutdown -h now
   ```

In VirtualBox on the host system:

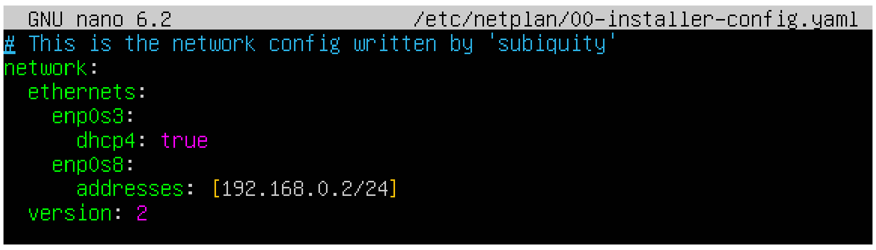
1. Create two clones of the VM: *compute1* and *compute2.*
2. Configure the *Network Adapter 1* for all nodes (head, compute1, compute2) as attached to *NAT*.
3. Configure the *Network Adapter 2* for all nodes (head, compute1, compute2) as attached to *Internal Network* (must be the same for all nodes).
4. Boot up *all* nodes.

On compute1 and compute2:

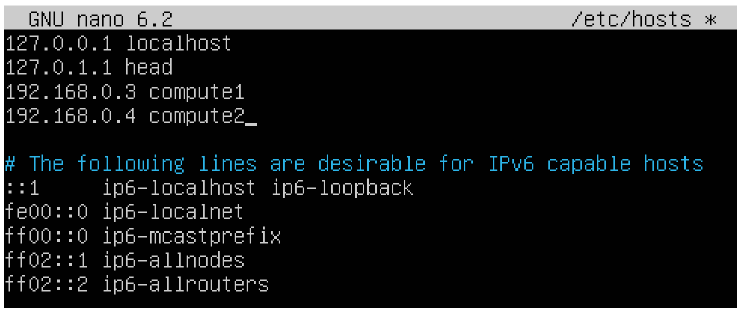
1. Change hostname from head to compute1 or compute2, respectively, in the following files:
   ```
   sudo nano /etc/hostname
   sudo nano /etc/hosts
   ```
2. Reboot for the changes to become effective:
   ```
   reboot
   ```

On all VMs:

1. Determine the name of the second network interface via:
   ```
   ifconfig -a
   ```
2. Assign a unique static IP within the same subnet (e.g. 192.168.0.2/24; 192.168.0.3/24; 192.168.0.4/24) to each node by modifying the network configuration:
   ```
   sudo nano /etc/netplan/00-installer-config.yaml
   sudo netplan apply
   ```
3. Check whether all nodes can mutually reach each other:
   ```
   ping <ip_addr>
   ```
4. Adjust /etc/hosts such that the static IPs match hostnames:
   ```
   sudo nano /etc/hosts
   ```
5. Check whether all nodes can mutually reach each other:
   ```
   ping <hostname>
   ```



```
  GNU nano 6.2                 /etc/netplan/00-installer-config.yaml
# This is the network config written by 'subiquity'
network:
  ethernets:
    enp0s3:
      dhcp4: true
    enp0s8:
      addresses: [192.168.0.2/24]
  version: 2
```



```
  GNU nano 6.2                                    /etc/hosts *
127.0.0.1 localhost
127.0.1.1 head
192.168.0.3 compute1
192.168.0.4 compute2_

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

On head:

1. Start NFS server via command:
   ```
   sudo systemctl start nfs-kernel-server.service
   ```
2. Edit /etc/exports on login by adding line:
   ```
   /home *(rw,async,no_subtree_check,no_root_squash)
   ```
3. Safe and afterwards execute:
   ```
   sudo exportfs -a
   ```

On compute1 and compute2:

1. Mount home directory of head into home directory of node:
   ```
   sudo mount head:/home /home
   cd
   ```
2. Create a file in the home directory on compute1 and check whether it shows up in the home directory on head and compute2.
3. After reboot, the directory would not be mounted anymore. Therefore, we have to add the following line to /etc/fstab:
   ```
    head:/home /home nfs   defaults    0     0
   ```
4. Reboot and check whether the remote home directory is mounted:
   ```
   findmnt
   ```

On any node:

1. Generate ssh keys for password-less logins between all nodes (save to the default location and don't enter a password for the ssh keypair):
   ```
   ssh-keygen
   cp .ssh/id_rsa.pub .ssh/authorized_keys
   ```
2. Disable host key checking by generating a file ~/.ssh/config with the following content:
   ```
   Host *
         StrictHostKeyChecking no
   ```

On any node:

1. Test your MPI installation by running:
   ```
   mpirun -np 3 --host head,compute1,compute2 hostname
   ```
2. Congratulations! You have now manually configured your very own mini cluster – and are now able to run your favourite MPI application.

Ansible+NFS for application deployment

This prepares the cluster for consistent application deployment.

Q0. Installation of Ansible

1. Install Ansible:

```
python3 -m pip install --user ansible
```

2. Check that Ansible has been installed successfully:

```
ansible --version
```

Q1. Create an Ansible Inventory

1. Create directory for Ansible files in $HOME:

```
mkdir .ansible && cd .ansible
```

2. Create a YAML-based Ansible inventory file $HOME/.ansible/hosts that covers all target nodes.

3. Check that the inventory is interpreted correctly:

```
ansible-inventory -i hosts --graph
```

Q2. Create an Ansible configuration file

1. Generate an Ansible config file .ansible.cfg in $HOME:

```
ansible-config init --disabled > $HOME/.ansible.cfg
```

2. In ansible.cfg, comment-in and adjust the inventory variable:

```
inventory=~/.ansible/hosts
```

3. Check that Ansible picks up the configuration file:

```
ansible --version
```

Q3. Ad-hoc commands

Execute your first ad-hoc command to check whether all target nodes are reachable via ping.

Q4. Share application directory via NFS:

1. Create /apps directory:

```
sudo mkdir /apps
```

2. Edit /etc/exports on head node by adding line:

```
/apps *(rw,async,no_subtree_check,no_root_squash)
```

3. Safe and afterwards execute:

```
sudo exportfs -a
```

Q5. Ansible playbook

Write an Ansible playbook `mount_apps.yaml` that mounts the `/apps` directory located on `head` at mount point `/apps` on all compute nodes, adds it to the system-wide `PATH` (`/etc/profile`) on all compute nodes, and reboots all compute nodes. Apply the playbook across the cluster.

Hint: You might find the Ansible documentation of the modules `ansible.posix.mount`, `ansible.builtin.lineinfile`, and `ansible.builtin.reboot` helpful.

Q6. Building and installing additional software

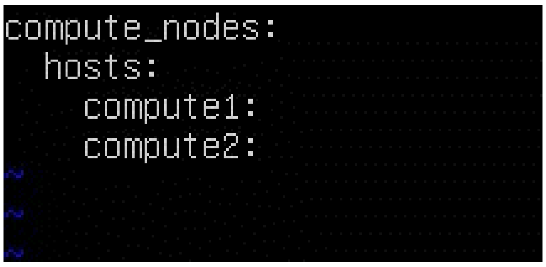
You have now equipped your cluster with a shared directory that allows for the shared access - of users and machines - to additional software packages located in /apps.

Build and install any application you like (e.g. HPL, OSU, a game, etc.) in /apps and check whether it is executable from anywhere or even across multiple nodes of the cluster.

Solutions

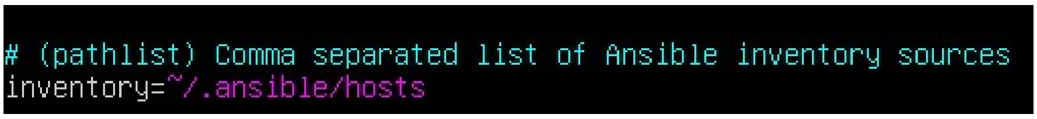
Q0. Instructions provided in task

Q1. Ansible inventory



```
compute_nodes:
  hosts:
    compute1:
    compute2:
~
~
~
```

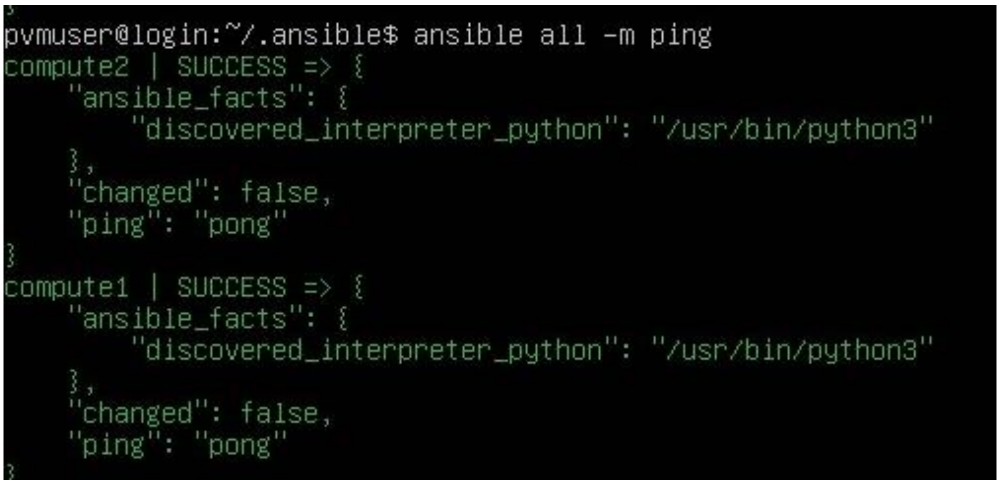
Q2. Ansible configuration



```
# (pathlist) Comma separated list of Ansible inventory sources
inventory=~/.ansible/hosts
```

Q3. Ad-hoc commands

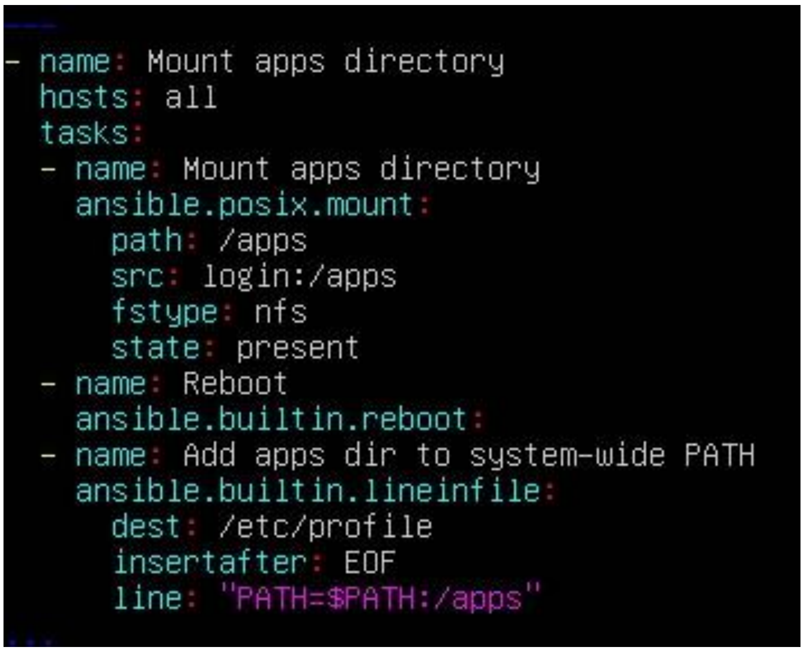
```
ansible all -m ping
```



```
pvmuser@login:~/.ansible$ ansible all -m ping
compute2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
compute1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

Q4. Instructions provided in task.

Q5. Ansible playbook



```
---
- name: Mount apps directory
  hosts: all
  tasks:
  - name: Mount apps directory
    ansible.posix.mount:
      path: /apps
      src: login:/apps
      fstype: nfs
      state: present
  - name: Reboot
    ansible.builtin.reboot:
  - name: Add apps dir to system-wide PATH
    ansible.builtin.lineinfile:
      dest: /etc/profile
      insertafter: EOF
      line: "PATH=$PATH:/apps"
...
```

Issue the playbook across all target nodes via:

```
ansible-playbook mount_apps.yaml --become --ask-become-pass
```