Оглавление

1. Включение Hyper-V

В Windows 10/11 роль Hyper-V можно включить через компоненты Windows.

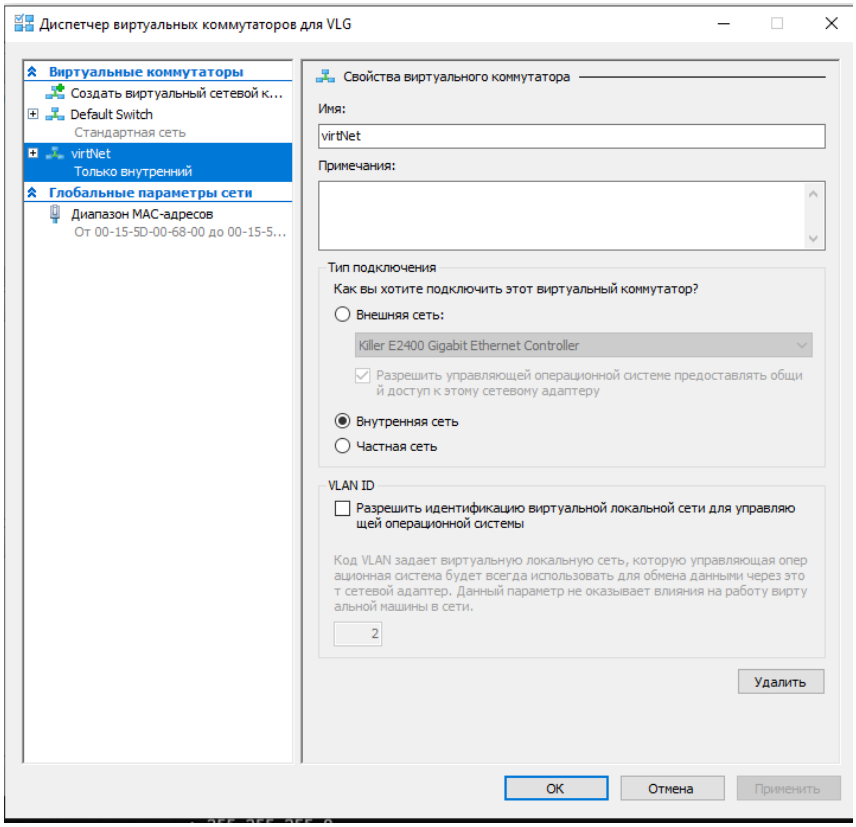
1. Откройте **Параметры → Приложения → Приложения и возможности**.
2. Перейдите по ссылке **Программы и компоненты**.
3. В левом меню выберите **Включение или отключение компонентов Windows**.
4. Найдите пункт **Hyper-V** и поставьте галочку (можно включить и подкомпоненты: Hyper-V Management Tools, Hyper-V Platform).
5. Нажмите **ОК** и перезагрузите компьютер по запросу.

2. Настройка сети

Цель — создать внутреннюю сеть для виртуальных машин и настроить NAT, чтобы они имели доступ к внешней сети через хост.

2.1. Создание внутреннего виртуального коммутатора

1. Запустите **Hyper-V Manager**.
2. В левой панели выберите нужный сервер (локальный компьютер).
3. В правой панели откройте **Virtual Switch Manager…**.
4. Создайте **Internal** виртуальный коммутатор, например с именем `virtNet`



2.2. Определение сетевого адаптера и назначение IP

Откройте PowerShell **от имени администратора** и выполните: `ipconfig`

Найдите используемый адаптер Ethernet, к которому будет привязана внутренняя сеть:

Нужно выбрать IP-адрес, который будет выступать **адресом маршрутизатора/NAT-шлюза**. Обычно это:

- первый адрес: `10.200.X.1`,
  или
- последний адрес: `10.200.X.254`.

Просмотр и управление IP-адресами интерфейса

Посмотреть текущие IP-адреса: `Get-NetIPAddress`

Удалить IP-адрес: `Remove-NetIPAddress -InterfaceIndex <InterfaceIndex> -IPAddress <IP-адрес>`

Добавить новый IP-адрес к виртуальному адаптеру (например, `vEthernet (virtNet)`):

```
New-NetIPAddress `
  -InterfaceAlias "vEthernet (virtNet)" `
  -IPAddress 10.200.X.254 `
  -PrefixLength 24
```

Здесь:

- `InterfaceAlias` — имя виртуального адаптера (в скобках — имя вашего коммутатора),
- `IPAddress` — выбранный адрес шлюза,
- `PrefixLength` — длина префикса (для /24 — это 24).

2.3. Настройка NAT

Теперь нужно настроить трансляцию адресов (NAT), чтобы ВМ могли выходить в сеть через хост.

Просмотр текущих NAT-правил:

```
Get-NetNat
Get-NetNatStaticMapping
```

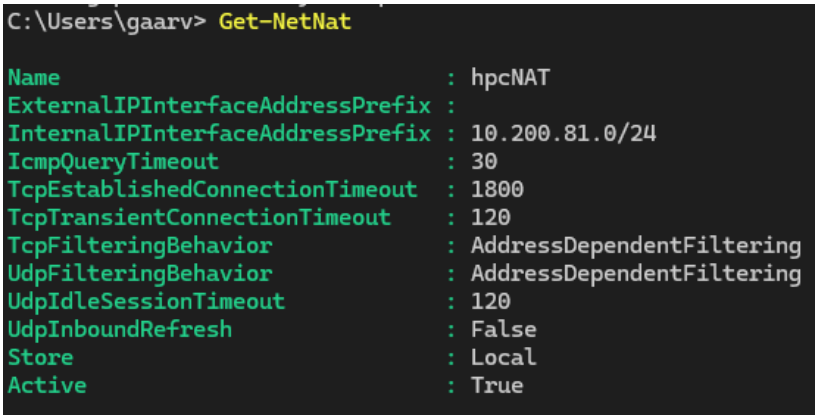
Удаление существующего объекта NAT (если что-то настроено неправильно):

```
Remove-NetNat -Name <старое_имя_NAT>
```

Создание нового NAT для подсети `10.200.X.0/24`:

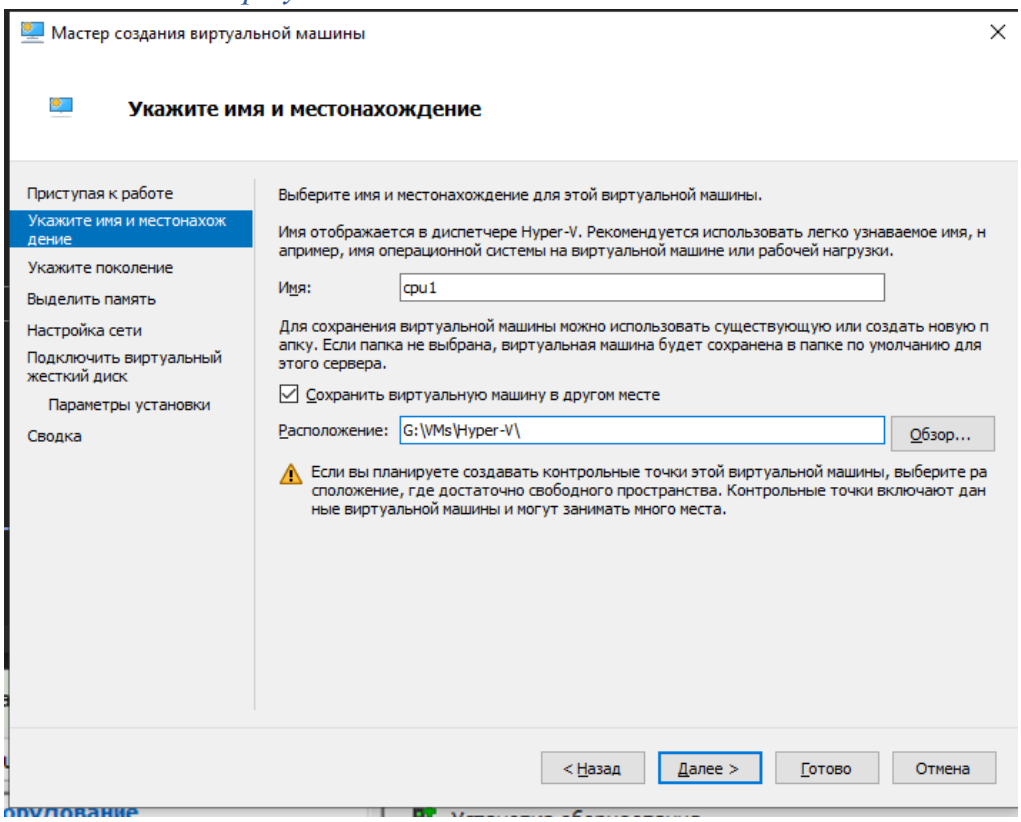
```
New-NetNat -Name "hpcNAT" -InternalIPInterfaceAddressPrefix
10.200.X.0/24
```

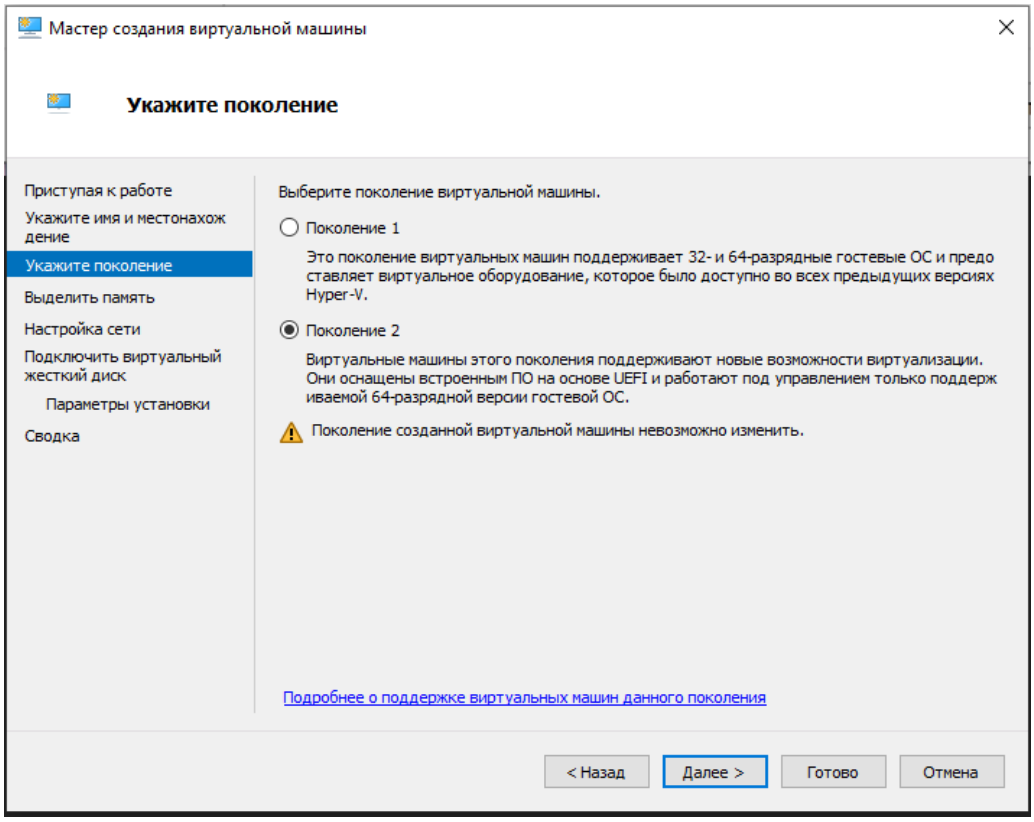
После этого вывод `Get-NetNat` должен показать один объект NAT с указанной подсетью.

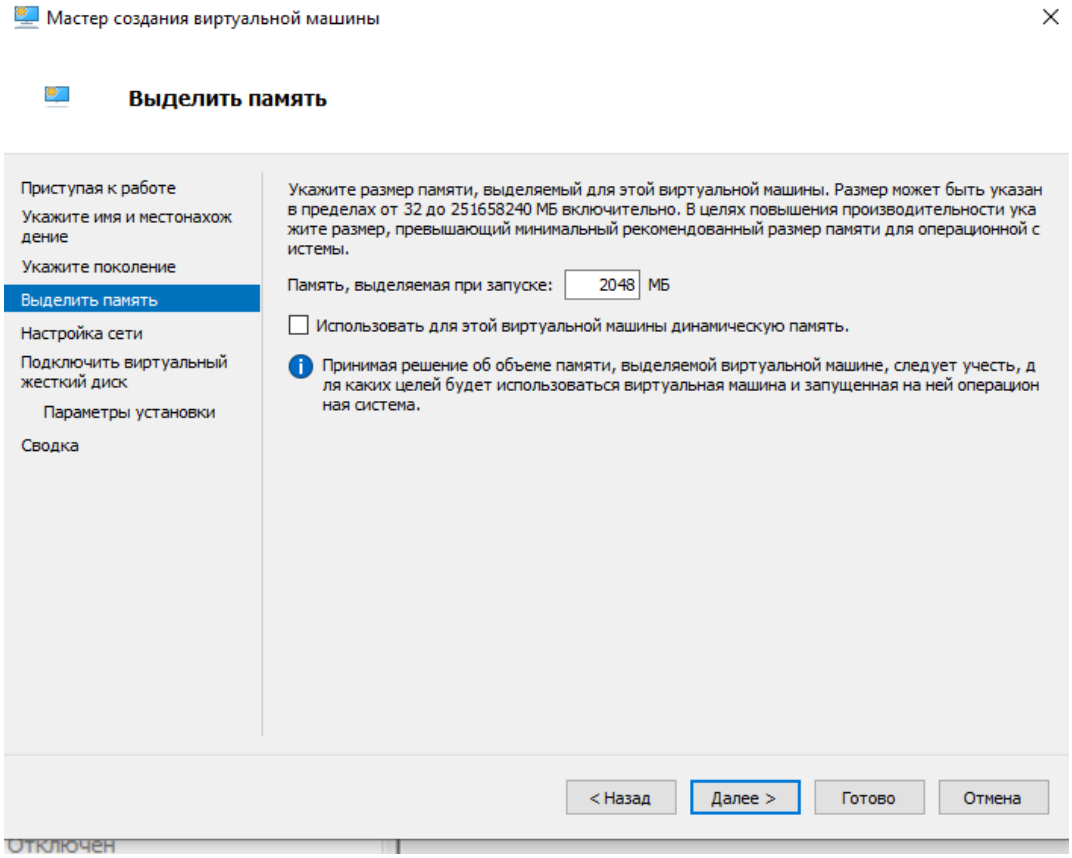


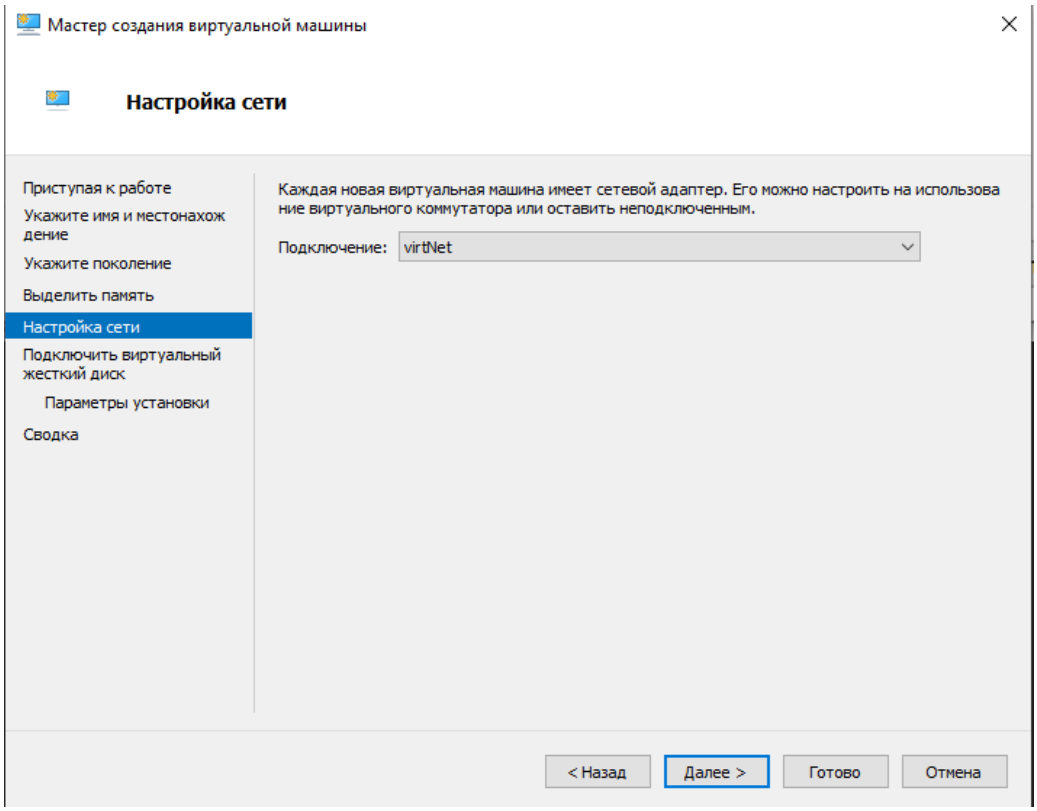
3. Создание и предварительная настройка виртуальной машины

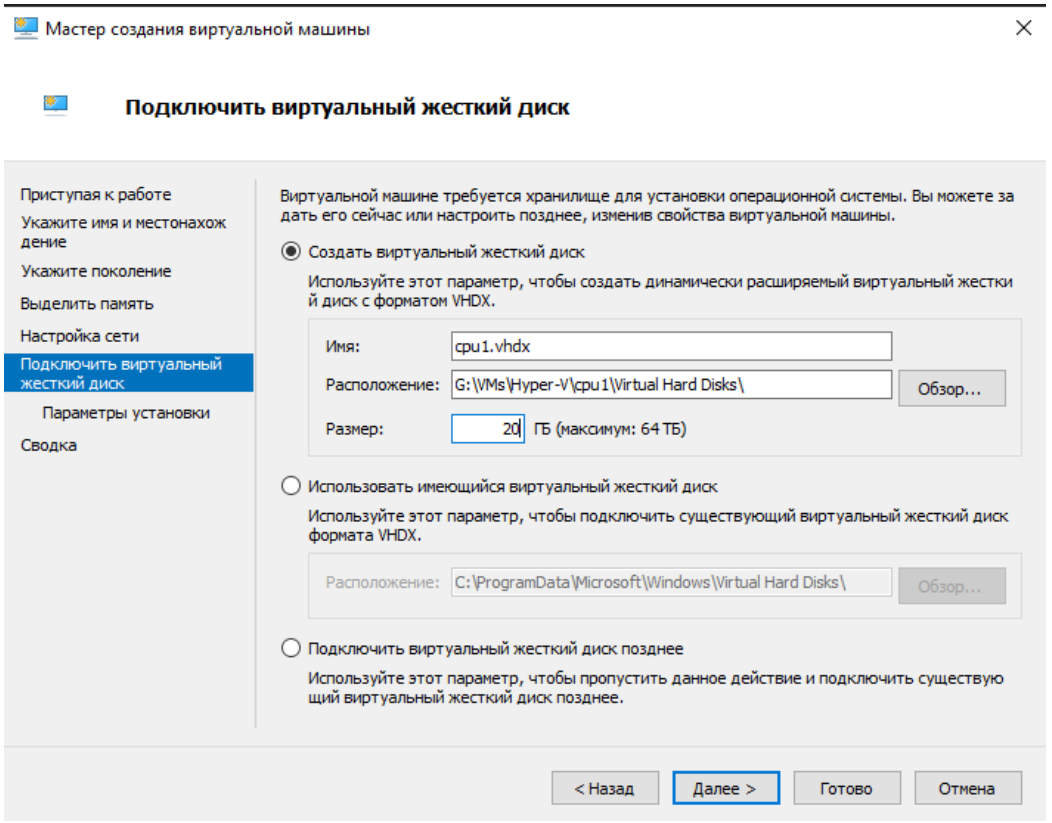
3.1. Создание виртуальной машины

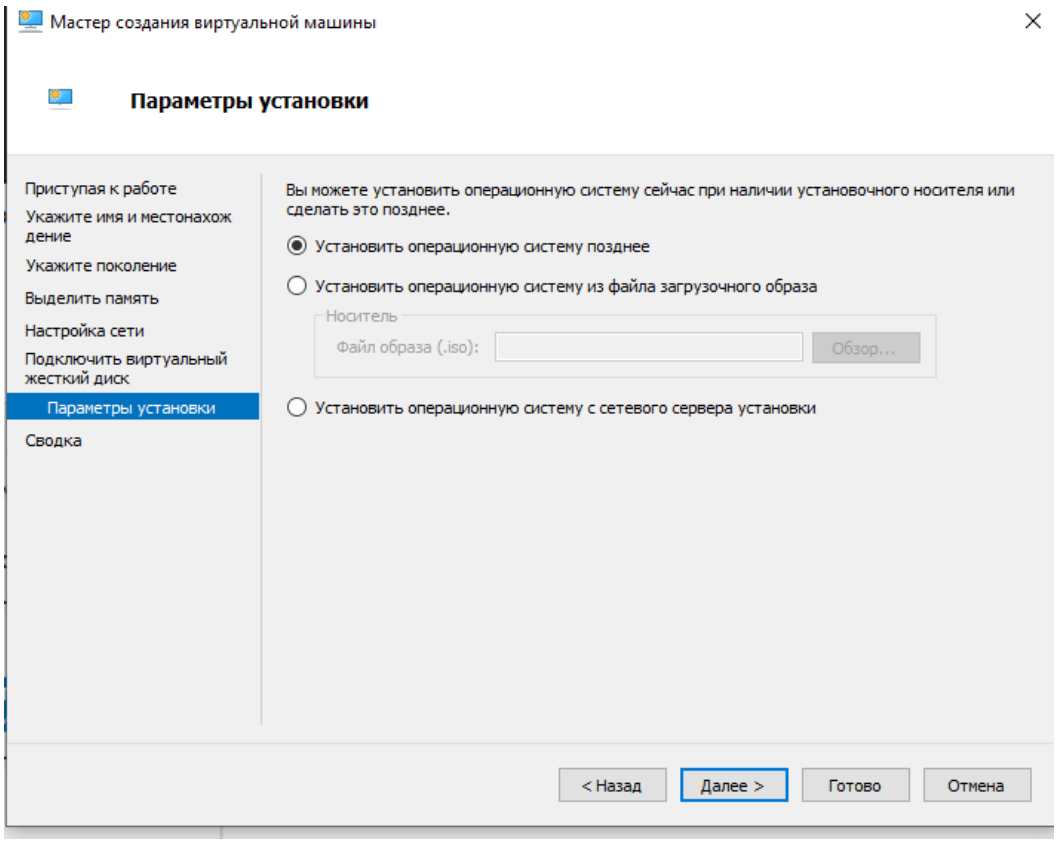




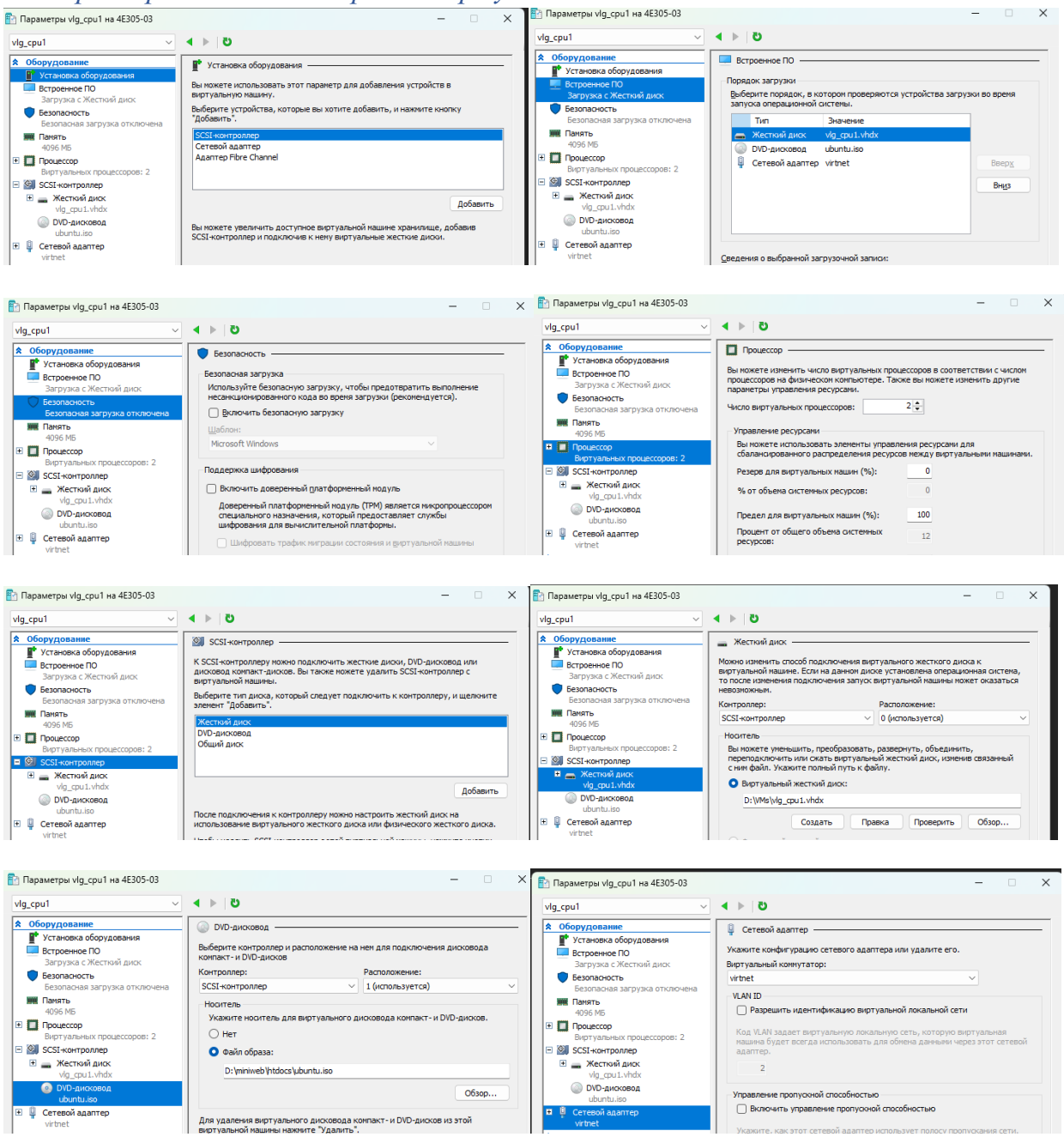

Мастер создания виртуальной машины
Выделить память
Приступая к работе
Укажите имя и местонахождение
Укажите поколение
Выделить память
Настройка сети
Подключить виртуальный жесткий диск
Параметры установки
Сводка
Укажите размер памяти, выделяемый для этой виртуальной машины. Размер может быть указан в пределах от 32 до 251658240 МБ включительно. В целях повышения производительности укажите размер, превышающий минимальный рекомендованный размер памяти для операционной системы.
Память, выделяемая при запуске: 2048 МБ
Использовать для этой виртуальной машины динамическую память.
Принимая решение об объеме памяти, выделяемой виртуальной машине, следует учесть, для каких целей будет использоваться виртуальная машина и запущенная на ней операционная система.
< Назад
Далее >
Готово
Отмена

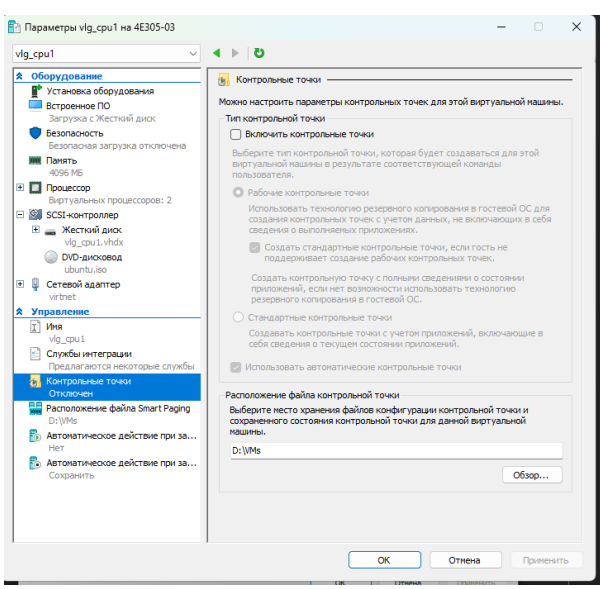

Мастер создания виртуальной машины
Настройка сети
Приступая к работе
Укажите имя и местонахождение
Укажите поколение
Выделить память
Настройка сети
Подключить виртуальный жесткий диск
Параметры установки
Сводка
Каждая новая виртуальная машина имеет сетевой адаптер. Его можно настроить на использование виртуального коммутатора или оставить неподключенным.
Подключение: virtNet
< Назад
Далее >
Готово
Отмена

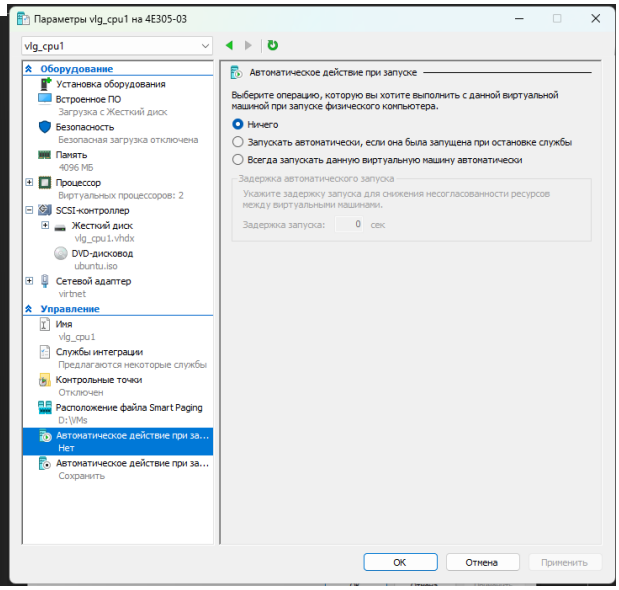

Мастер создания виртуальной машины
Подключить виртуальный жесткий диск
Приступая к работе
Укажите имя и местонахождение
Укажите поколение
Выделить память
Настройка сети
Подключить виртуальный жесткий диск
Параметры установки
Сводка
Виртуальной машине требуется хранилище для установки операционной системы. Вы можете задать его сейчас или настроить позднее, изменив свойства виртуальной машины.
Создать виртуальный жесткий диск
Используйте этот параметр, чтобы создать динамически расширяемый виртуальный жесткий диск с форматом VHDX.
Имя: cpu1.vhdx
Расположение: G:\VMs\Hyper-V\cpu1\Virtual Hard Disks\
Обзор...
Размер: 20 ГБ (максимум: 64 ТБ)
Использовать имеющийся виртуальный жесткий диск
Используйте этот параметр, чтобы подключить существующий виртуальный жесткий диск формата VHDX.
Расположение: C:\ProgramData\Microsoft\Windows\Virtual Hard Disks\
Обзор...
Подключить виртуальный жесткий диск позднее
Используйте этот параметр, чтобы пропустить данное действие и подключить существующий виртуальный жесткий диск позднее.
< Назад
Далее >
Готово
Отмена

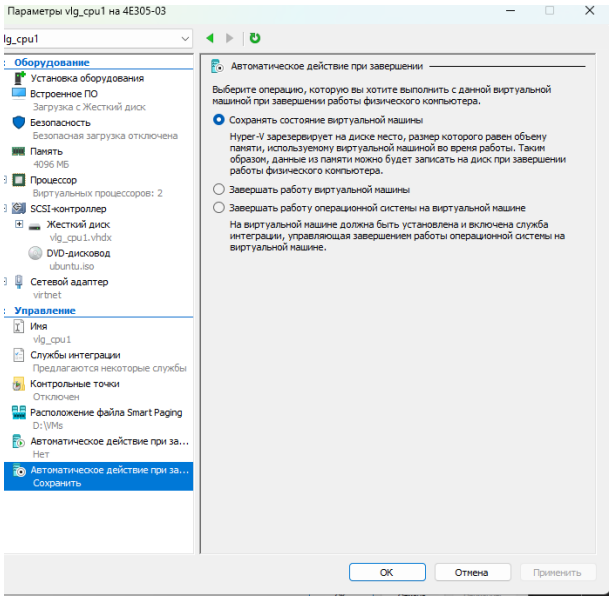

Мастер создания виртуальной машины
Параметры установки
Приступая к работе
Укажите имя и местонахождение
Укажите поколение
Выделить память
Настройка сети
Подключить виртуальный жесткий диск
Параметры установки
Сводка
Вы можете установить операционную систему сейчас при наличии установочного носителя или сделать это позднее.
Установить операционную систему позднее
Установить операционную систему из файла загрузочного образа
Носитель
Файл образа (.iso):
Обзор...
Установить операционную систему с сетевого сервера установки
< Назад
Далее >
Готово
Отмена

3.2. Предварительная настройка виртуальной машины

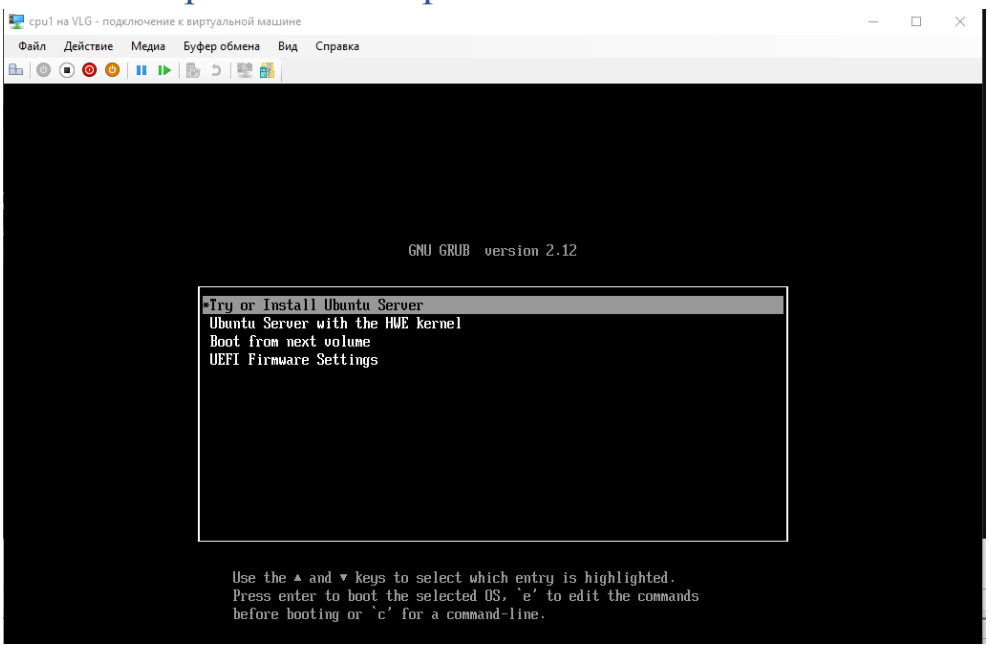


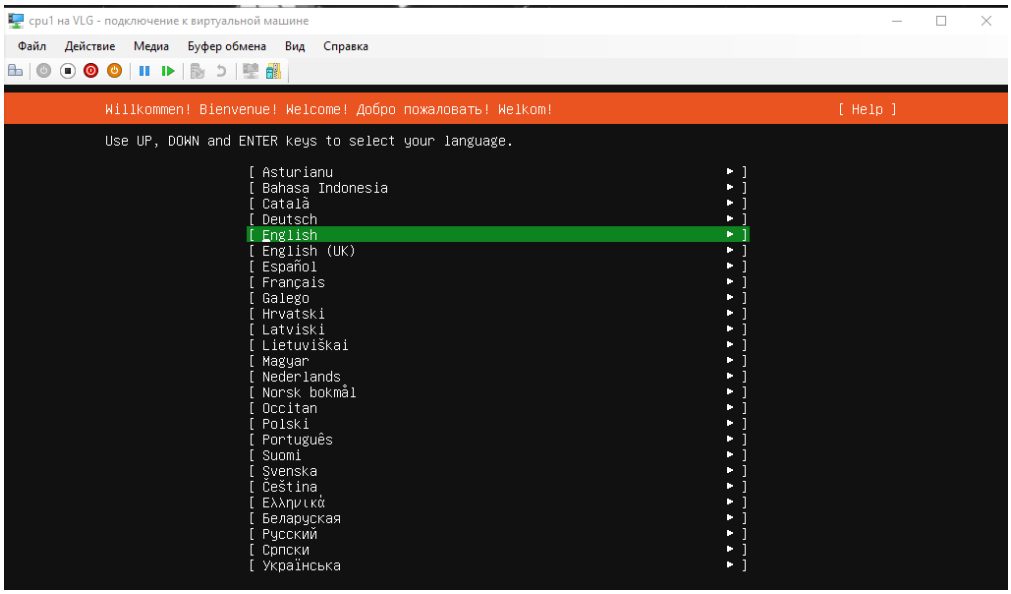

Параметры vlg_cpu1 на 4E305-03
vlg_cpu1
Оборудование
Установка оборудования
Встроенное ПО
Загрузка с Жесткий диск
Безопасность
Безопасная загрузка отключена
Память
4096 МБ
Процессор
Виртуальных процессоров: 2
SCSI-контроллер
Жесткий диск
vlg_cpu1.vhdx
DVD-дисковод
ubuntu.iso
Сетевой адаптер
virtnet
Управление
Имя
vlg_cpu1
Службы интеграции
Предлагаются некоторые службы
Контрольные точки
Отключен
Расположение файла Smart Paging
D:\VMs
Автоматическое действие при за...
Нет
Автоматическое действие при за...
Сохранить
Контрольные точки
Можно настроить параметры контрольных точек для этой виртуальной машины.
Тип контрольной точки
Включить контрольные точки
Выберите тип контрольной точки, которая будет создаваться для этой виртуальной машины в результате соответствующей команды пользователя.
Рабочие контрольные точки
Использовать технологию резервного копирования в гостевой ОС для создания контрольных точек с учетом данных, не включающих в себя сведения о выполняемых приложениях.
Создать стандартные контрольные точки, если гость не поддерживает создание рабочих контрольных точек.
Создать контрольную точку с полными сведениями о состоянии приложений, если нет возможности использовать технологию резервного копирования в гостевой ОС.
Стандартные контрольные точки
Создавать контрольные точки с учетом приложений, включающие в себя сведения о текущем состоянии приложений.
Использовать автоматические контрольные точки
Расположение файла контрольной точки
Выберите место хранения файлов конфигурации контрольной точки и сохраненного состояния контрольной точки для данной виртуальной машины.
D:\VMs
Обзор...
OK
Отмена
Применить

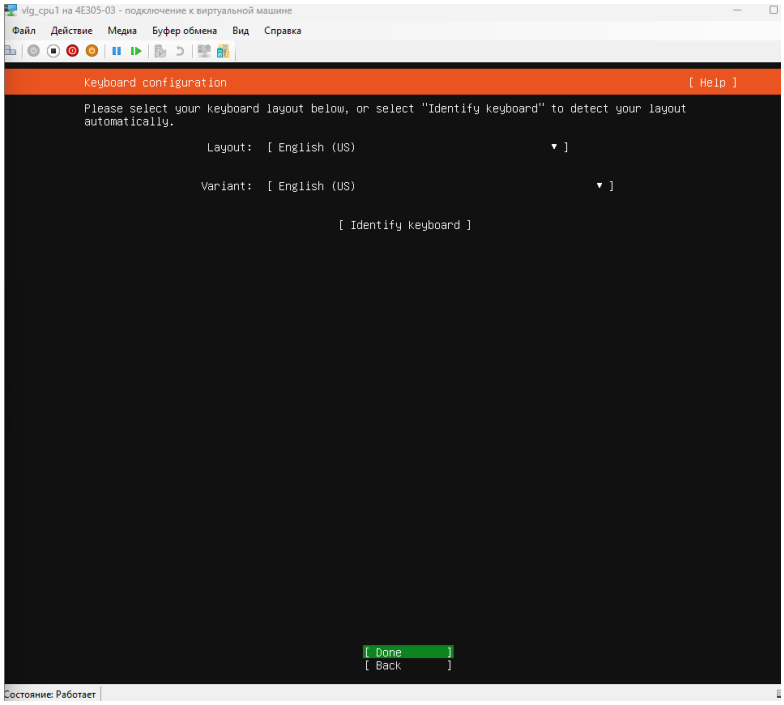

Параметры vlg_cpu1 на 4E305-03
vlg_cpu1
Оборудование
Установка оборудования
Встроенное ПО
Загрузка с Жесткий диск
Безопасность
Безопасная загрузка отключена
Память
4096 МБ
Процессор
Виртуальных процессоров: 2
SCSI-контроллер
Жесткий диск
vlg_cpu1.vhdx
DVD-дисковод
ubuntu.iso
Сетевой адаптер
virtnet
Управление
Имя
vlg_cpu1
Службы интеграции
Предлагаются некоторые службы
Контрольные точки
Отключен
Расположение файла Smart Paging
D:\VMs
Автоматическое действие при за...
Нет
Автоматическое действие при за...
Сохранить
Автоматическое действие при запуске
Выберите операцию, которую вы хотите выполнить с данной виртуальной машиной при запуске физического компьютера.
Ничего
Запускать автоматически, если она была запущена при остановке службы
Всегда запускать данную виртуальную машину автоматически
Задержка автоматического запуска
Укажите задержку запуска для снижения несогласованности ресурсов между виртуальными машинами.
Задержка запуска:
0
сек
OK
Отмена
Применить

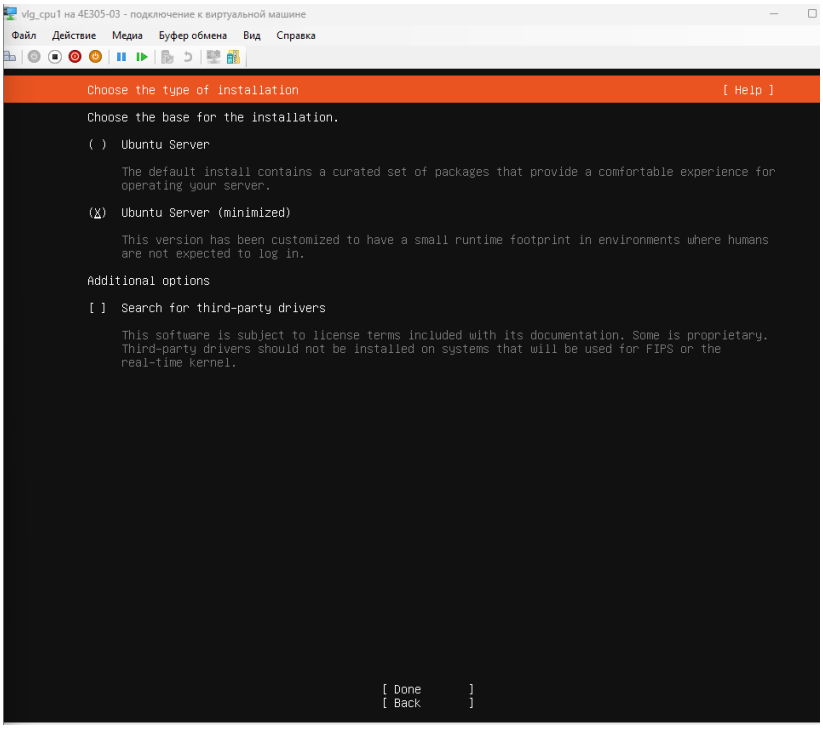

Параметры vlg_cpu1 на 4E305-03
lg_cpu1
Оборудование
Установка оборудования
Встроенное ПО
Загрузка с Жесткий диск
Безопасность
Безопасная загрузка отключена
Память
4096 МБ
Процессор
Виртуальных процессоров: 2
SCSI-контроллер
Жесткий диск
vlg_cpu1.vhdx
DVD-дисковод
ubuntu.iso
Сетевой адаптер
virtnet
Управление
Имя
vlg_cpu1
Службы интеграции
Предлагаются некоторые службы
Контрольные точки
Отключен
Расположение файла Smart Paging
D:\VMs
Автоматическое действие при за...
Нет
Автоматическое действие при за...
Сохранить
Автоматическое действие при завершении
Выберите операцию, которую вы хотите выполнить с данной виртуальной машиной при завершении работы физического компьютера.
Сохранять состояние виртуальной машины
Hyper-V зарезервирует на диске место, размер которого равен объему памяти, используемому виртуальной машиной во время работы. Таким образом, данные из памяти можно будет записать на диск при завершении работы физического компьютера.
Завершать работу виртуальной машины
Завершать работу операционной системы на виртуальной машине
На виртуальной машине должна быть установлена и включена служба интеграции, управляющая завершением работы операционной системы на виртуальной машине.
OK
Отмена
Применить

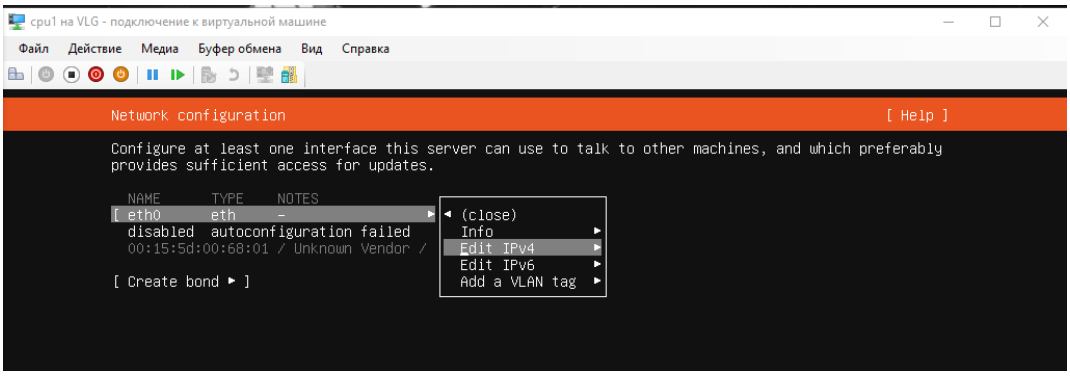
4. Установка и первичная настройка ОС

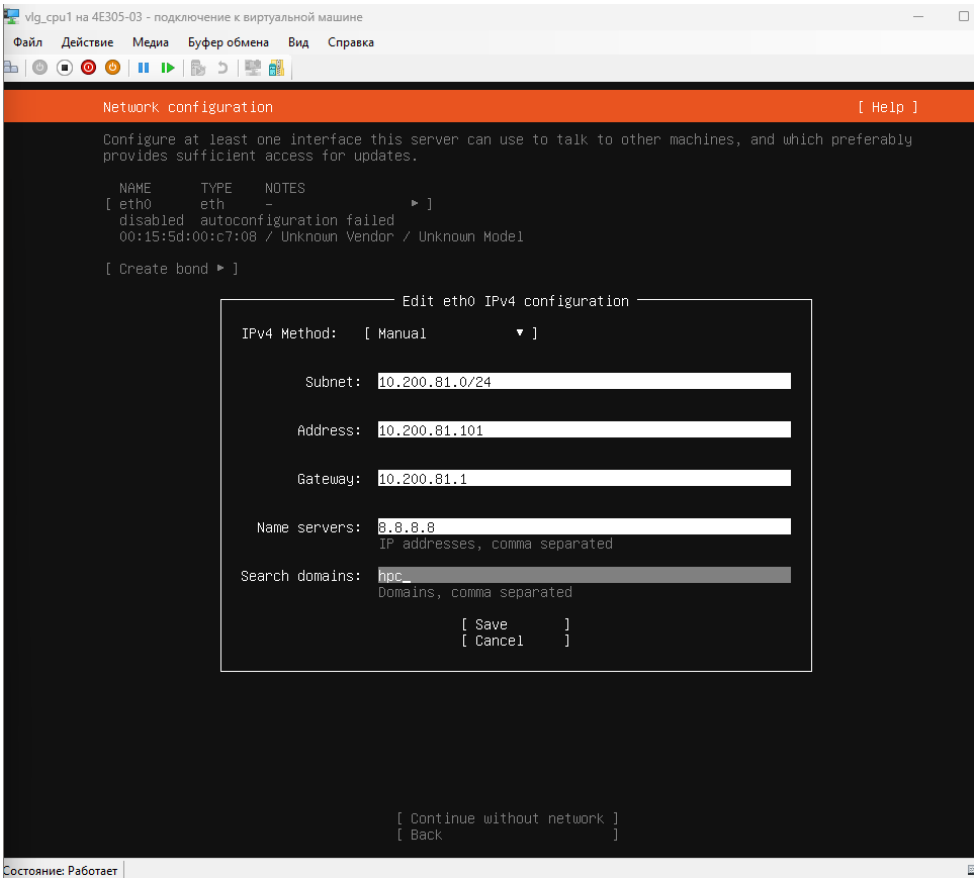


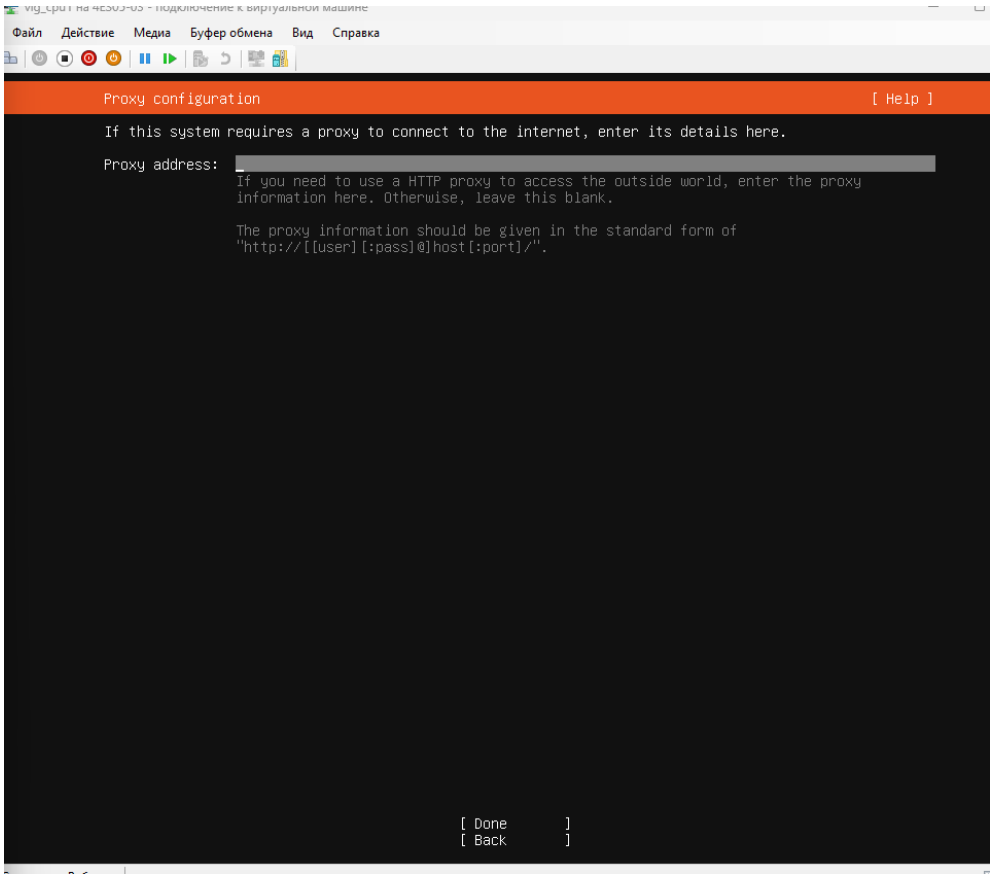


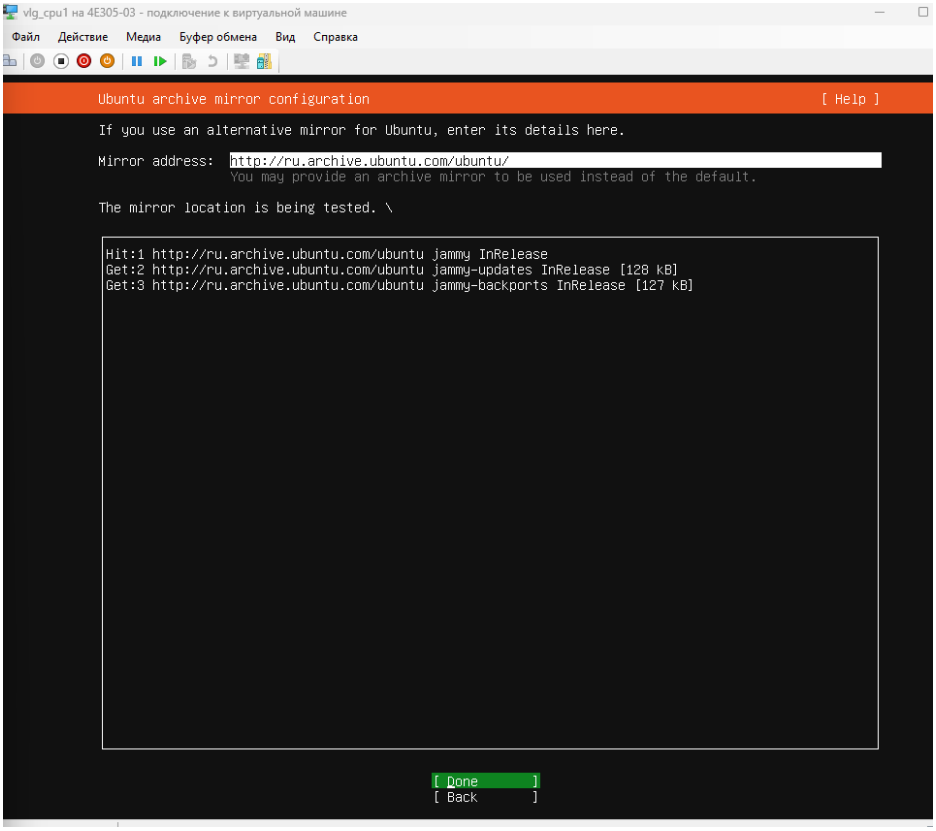

vlg_cpu1 на 4E305-03 - подключение к виртуальной машине
Файл Действие Медиа Буфер обмена Вид Справка
Keyboard configuration [Help]
Please select your keyboard layout below, or select "Identify keyboard" to detect your layout automatically.
Layout: [English (US) ▾]
Variant: [English (US) ▾]
[Identify keyboard]
[Done]
[Back]
Состояние: Работает

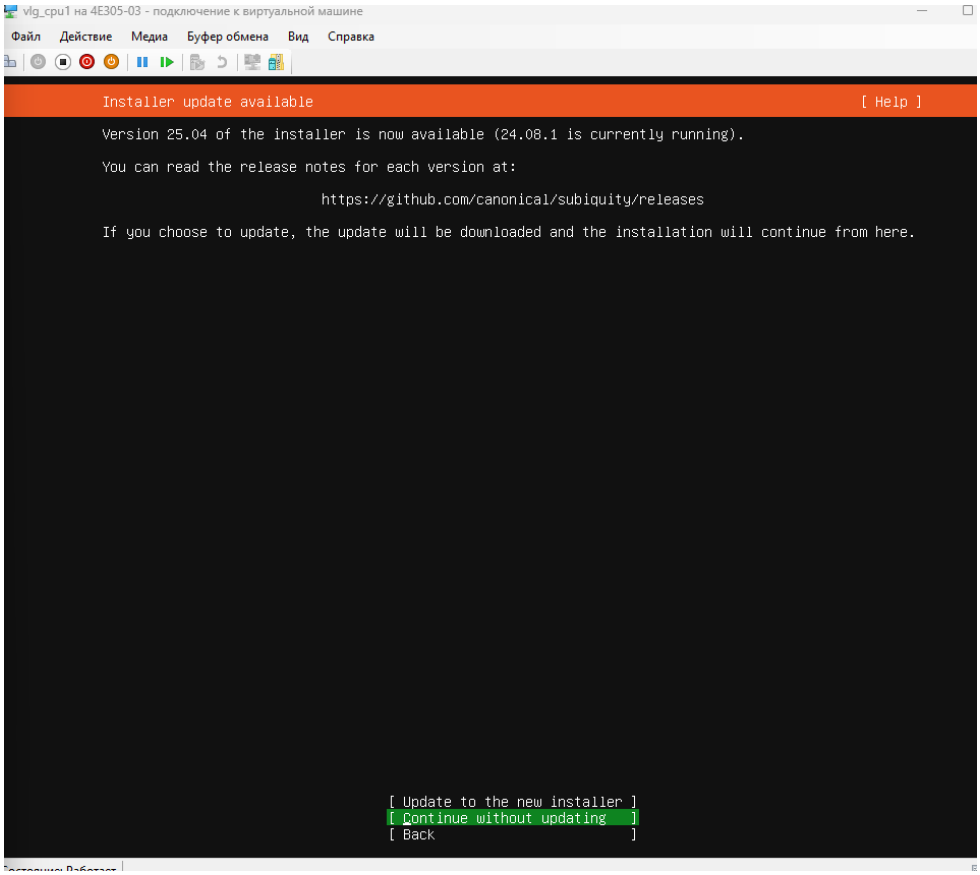

vlg_cpu1 на 4E305-03 - подключение к виртуальной машине
Файл Действие Медиа Буфер обмена Вид Справка
Choose the type of installation [Help]
Choose the base for the installation.
( ) Ubuntu Server
The default install contains a curated set of packages that provide a comfortable experience for operating your server.
(X) Ubuntu Server (minimized)
This version has been customized to have a small runtime footprint in environments where humans are not expected to log in.
Additional options
[ ] Search for third-party drivers
This software is subject to license terms included with its documentation. Some is proprietary. Third-party drivers should not be installed on systems that will be used for FIPS or the real-time kernel.
[Done]
[Back]

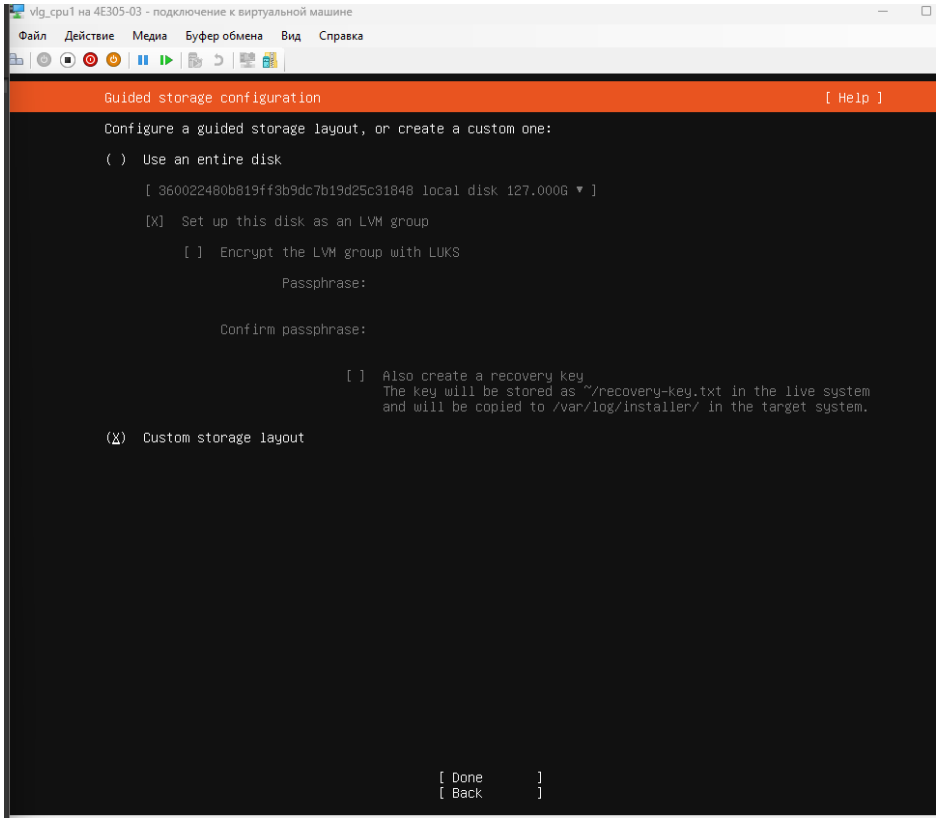

cpu1 на VLG - подключение к виртуальной машине
Файл Действие Медиа Буфер обмена Вид Справка
Network configuration [Help]
Configure at least one interface this server can use to talk to other machines, and which preferably provides sufficient access for updates.
NAME TYPE NOTES
[eth0 eth -
disabled autoconfiguration failed
00:15:5d:00:68:01 / Unknown Vendor /
[Create bond ▸]
(close)
Info
Edit IPv4
Edit IPv6
Add a VLAN tag

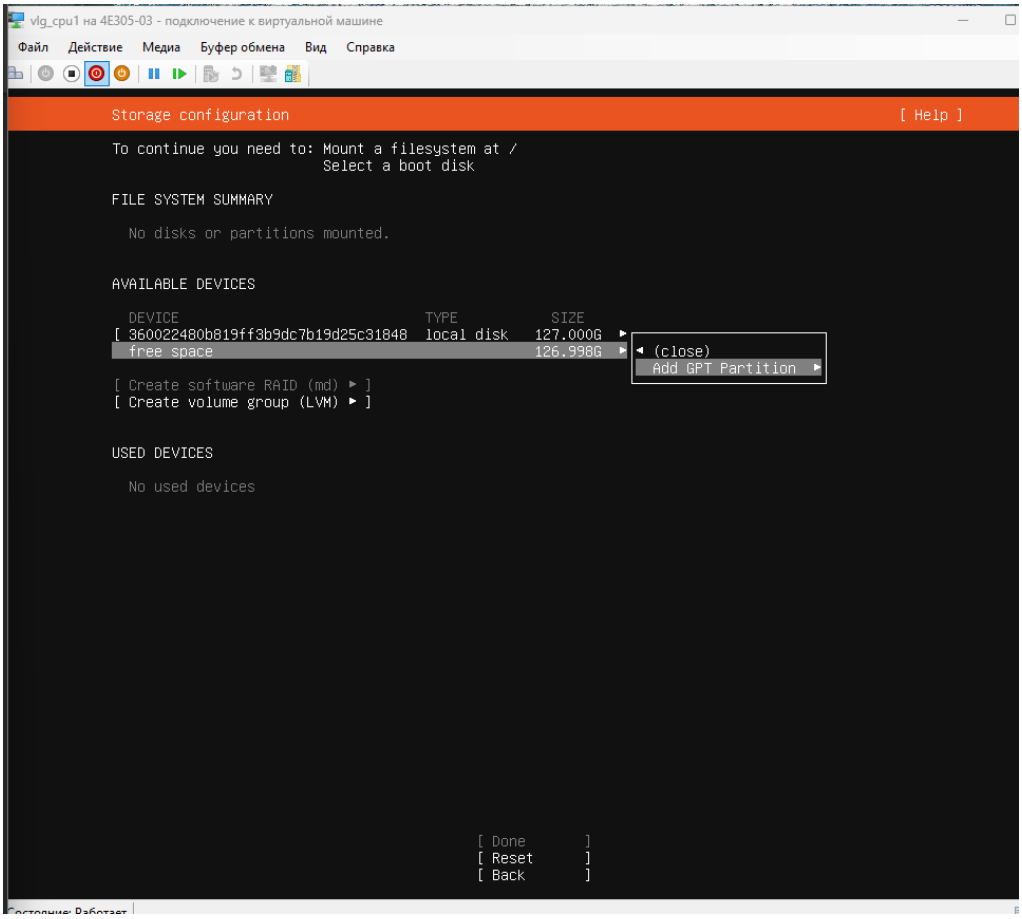

vlg_cpu1 на 4E305-03 - подключение к виртуальной машине
Файл Действие Медиа Буфер обмена Вид Справка
Network configuration [Help]
Configure at least one interface this server can use to talk to other machines, and which preferably provides sufficient access for updates.
NAME TYPE NOTES
[eth0 eth - ▸]
disabled autoconfiguration failed
00:15:5d:00:c7:08 / Unknown Vendor / Unknown Model
[Create bond ▸]
Edit eth0 IPv4 configuration
IPv4 Method: [Manual ▾]
Subnet: 10.200.81.0/24
Address: 10.200.81.101
Gateway: 10.200.81.1
Name servers: 8.8.8.8
IP addresses, comma separated
Search domains: hpc_
Domains, comma separated
[Save]
[Cancel]
[Continue without network]
[Back]
Состояние: Работает

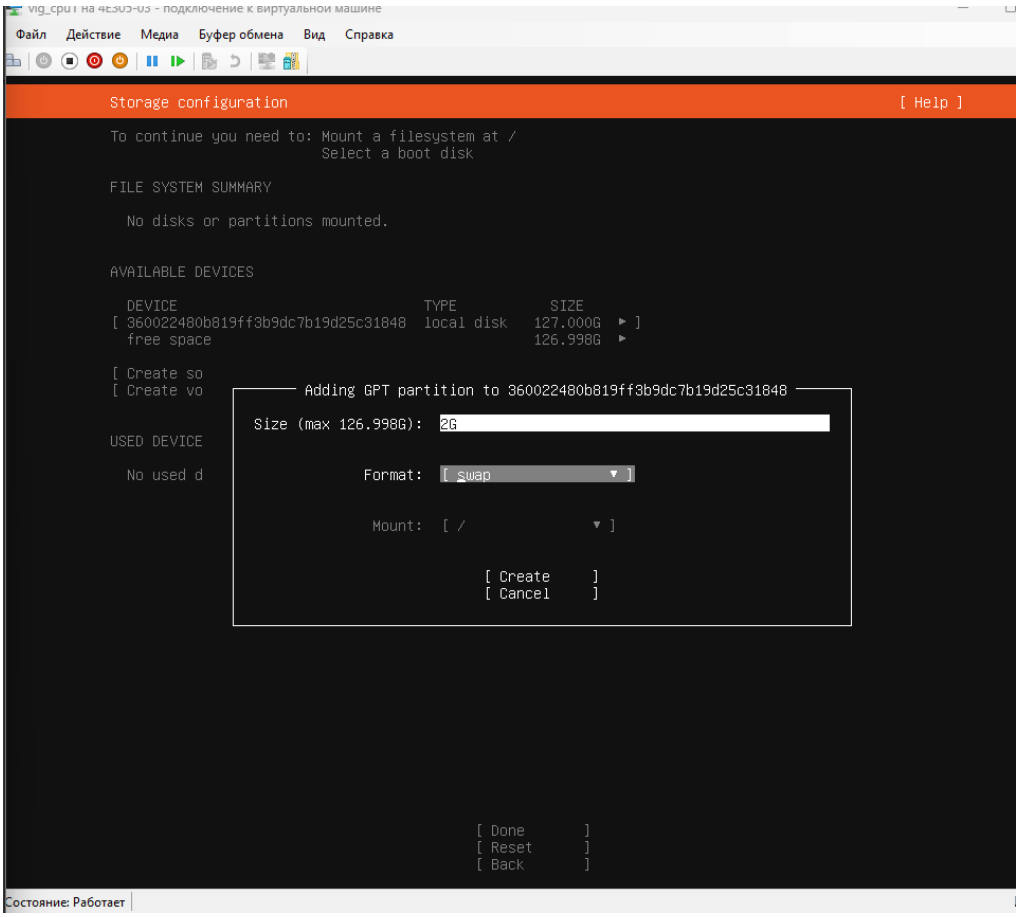

Файл Действие Медиа Буфер обмена Вид Справка
Proxy configuration [Help]
If this system requires a proxy to connect to the internet, enter its details here.
Proxy address:
If you need to use a HTTP proxy to access the outside world, enter the proxy information here. Otherwise, leave this blank.
The proxy information should be given in the standard form of "http://[[user][:pass]@]host[:port]/".
[Done]
[Back]


vlg_cpu1 на 4E305-03 - подключение к виртуальной машине
Файл Действие Медиа Буфер обмена Вид Справка
Ubuntu archive mirror configuration [Help]
If you use an alternative mirror for Ubuntu, enter its details here.
Mirror address: http://ru.archive.ubuntu.com/ubuntu/
You may provide an archive mirror to be used instead of the default.
The mirror location is being tested. \
Hit:1 http://ru.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://ru.archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:3 http://ru.archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
[Done]
[Back]

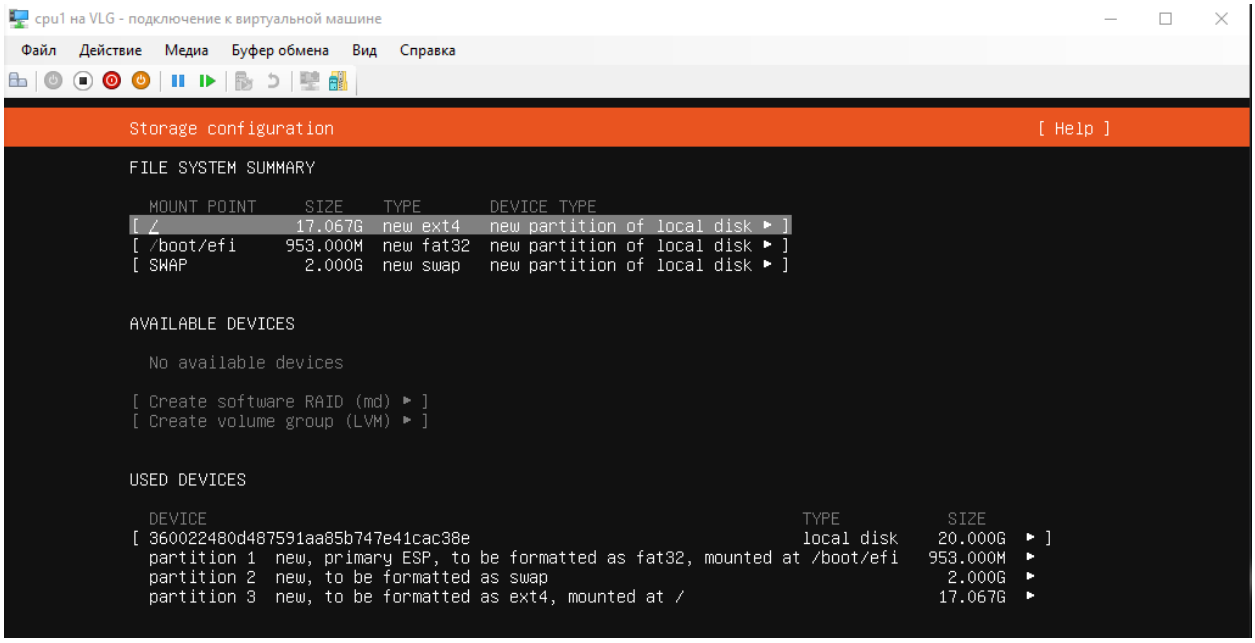

vlg_cpu1 на 4E305-03 - подключение к виртуальной машине
Файл Действие Медиа Буфер обмена Вид Справка
Installer update available [Help]
Version 25.04 of the installer is now available (24.08.1 is currently running).
You can read the release notes for each version at:
https://github.com/canonical/subiquity/releases
If you choose to update, the update will be downloaded and the installation will continue from here.
[Update to the new installer]
[Continue without updating]
[Back]

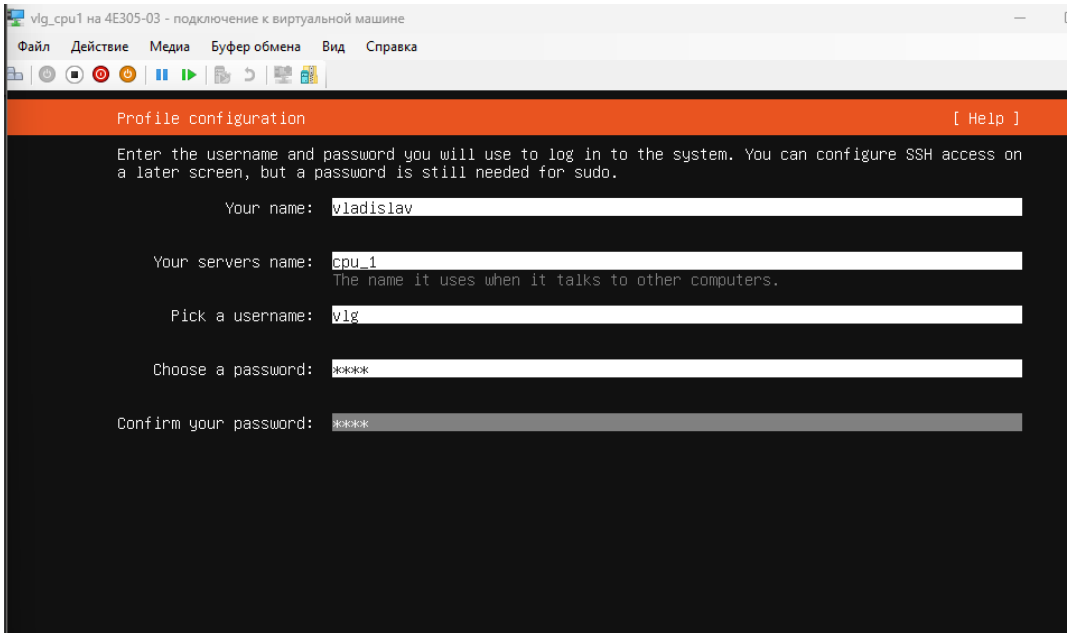

vlg_cpu1 на 4E305-03 - подключение к виртуальной машине
Файл Действие Медиа Буфер обмена Вид Справка
Guided storage configuration [Help]
Configure a guided storage layout, or create a custom one:
( ) Use an entire disk
[360022480b819ff3b9dc7b19d25c31848 local disk 127.000G ▾]
[X] Set up this disk as an LVM group
[ ] Encrypt the LVM group with LUKS
Passphrase:
Confirm passphrase:
[ ] Also create a recovery key
The key will be stored as ~/recovery-key.txt in the live system
and will be copied to /var/log/installer/ in the target system.
(X) Custom storage layout
[Done]
[Back]

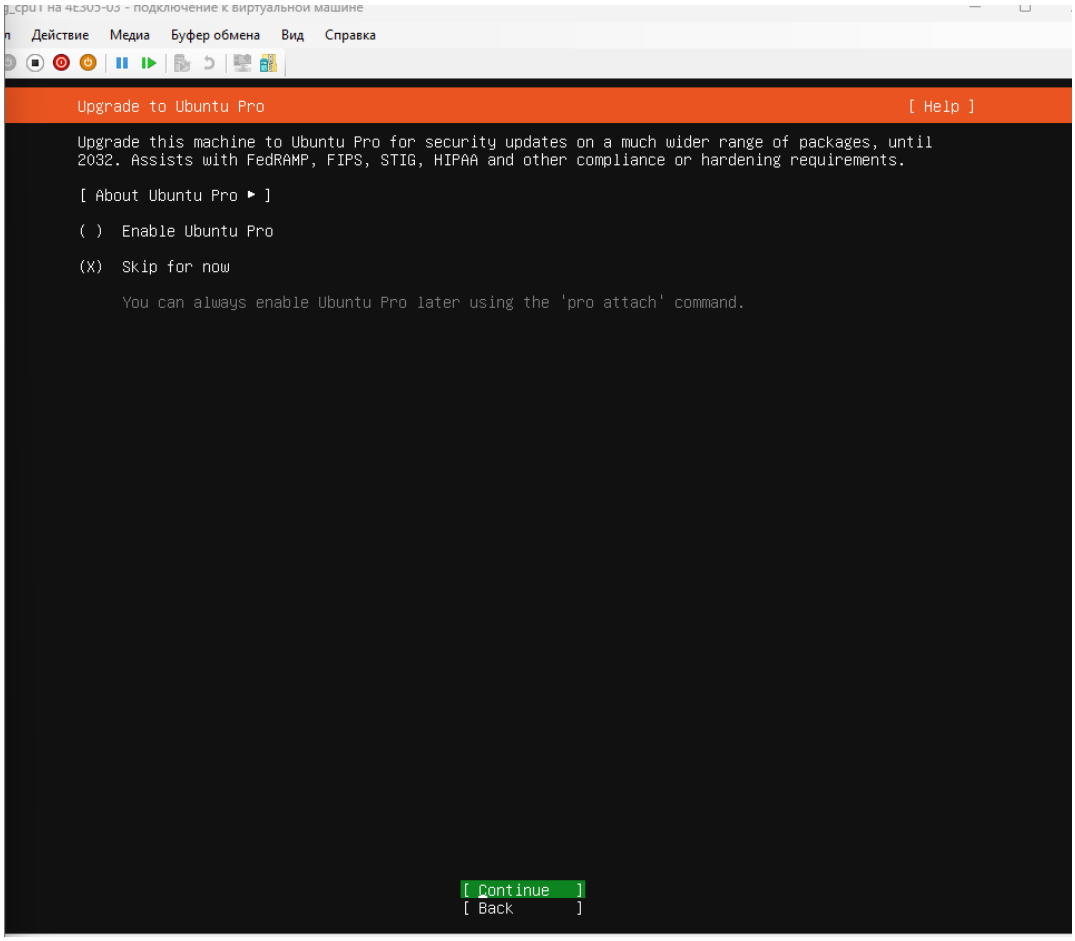

vlg_cpu1 на 4E305-03 - подключение к виртуальной машине
Файл Действие Медиа Буфер обмена Вид Справка
Storage configuration [Help]
To continue you need to: Mount a filesystem at /
Select a boot disk
FILE SYSTEM SUMMARY
No disks or partitions mounted.
AVAILABLE DEVICES
DEVICE TYPE SIZE
[360022480b819ff3b9dc7b19d25c31848 local disk 127.000G ▸
free space 126.998G ▸
◂ (close)
Add GPT Partition ▸
[Create software RAID (md) ▸]
[Create volume group (LVM) ▸]
USED DEVICES
No used devices
[Done]
[Reset]
[Back]



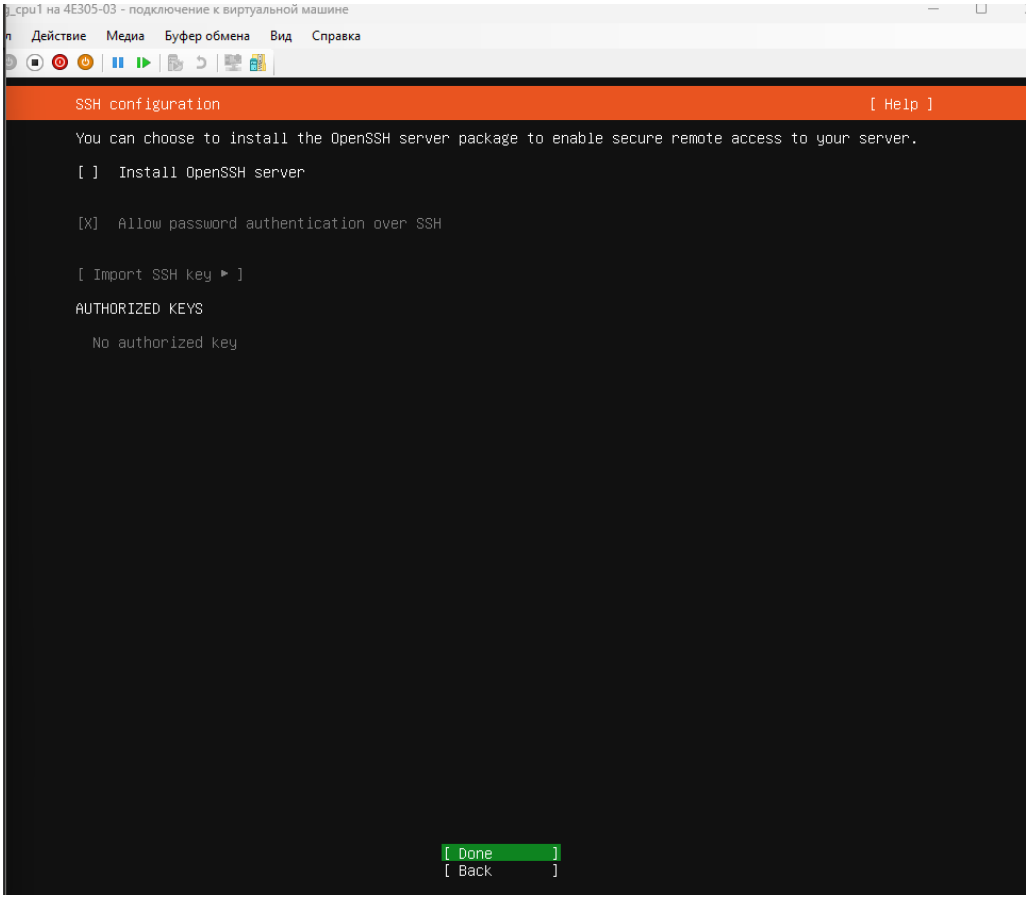
| Под корневую директорию (/) выделяем **весь оставшийся объём** диска.

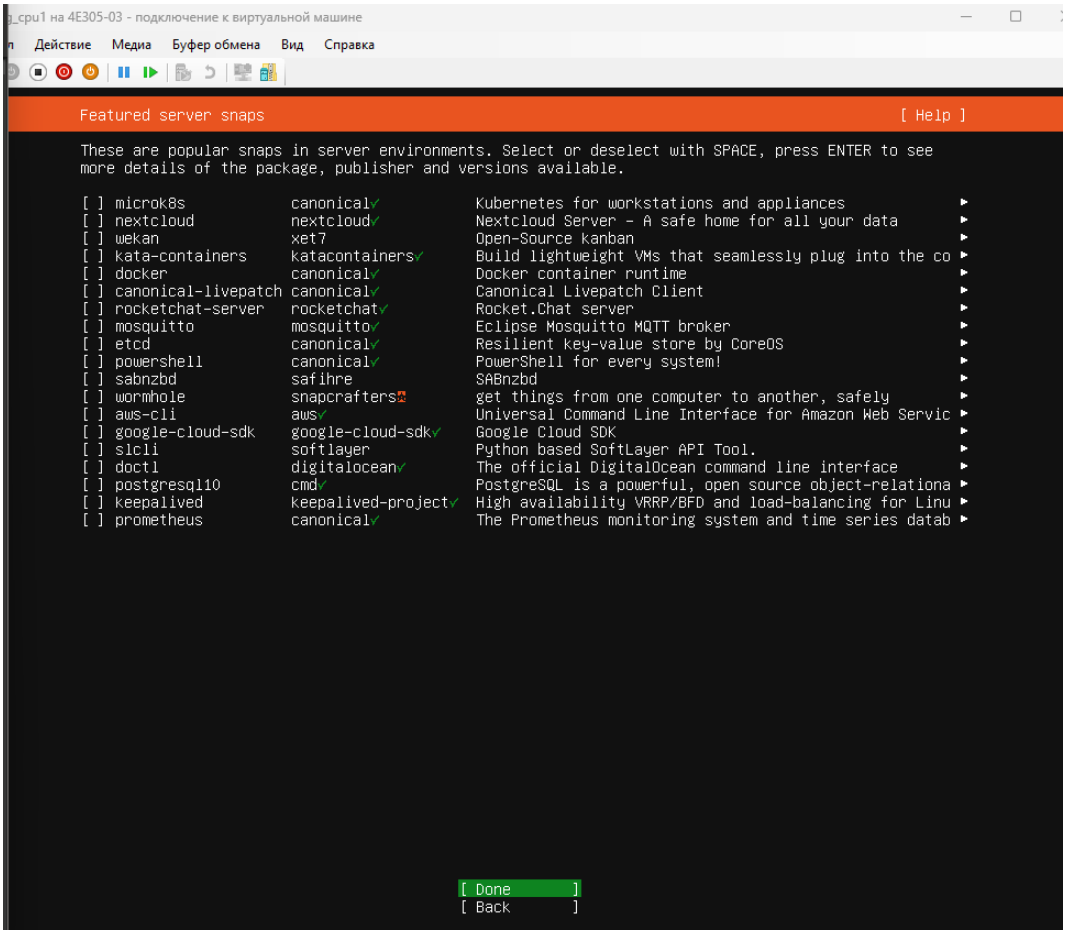



vlg_cpu1 на 4E305-03 - подключение к виртуальной машине
Файл Действие Медиа Буфер обмена Вид Справка
Profile configuration [Help]
Enter the username and password you will use to log in to the system. You can configure SSH access on a later screen, but a password is still needed for sudo.
Your name: vladislav
Your servers name: cpu_1
The name it uses when it talks to other computers.
Pick a username: vlg
Choose a password: ****
Confirm your password: ****


Действие Медиа Буфер обмена Вид Справка
Upgrade to Ubuntu Pro [Help]
Upgrade this machine to Ubuntu Pro for security updates on a much wider range of packages, until 2032. Assists with FedRAMP, FIPS, STIG, HIPAA and other compliance or hardening requirements.
[About Ubuntu Pro ▸]
( ) Enable Ubuntu Pro
(X) Skip for now
You can always enable Ubuntu Pro later using the 'pro attach' command.
[Continue]
[Back]

Отмечаем галочкой Install OpenSSH server (стоит обновить скриншот):





После установки войдите в систему по консоли Hyper-V и обновите пароль root и пакеты:

```
sudo -
passwd
```

```
sudo apt update
sudo apt upgrade -y
```

Установите необходимый софт:

```
sudo apt install -y \
  nano slurmd \
  openmpi-bin openmpi-doc libopenmpi-dev \
  iputils-ping \
  build-essential
```

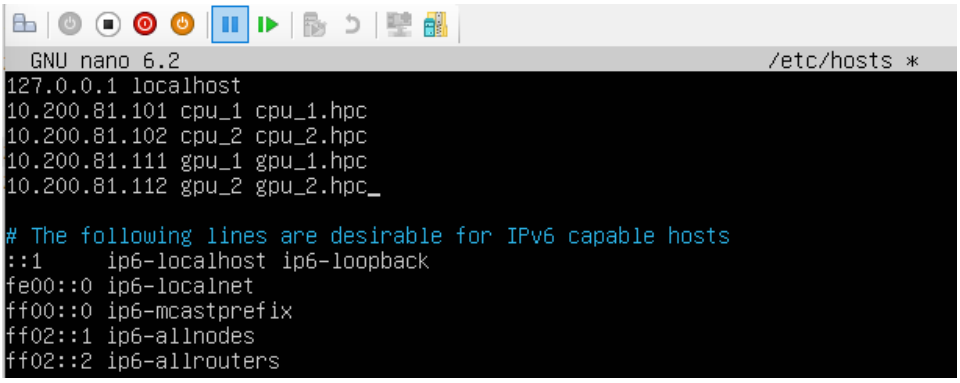
Отключите автоматическую конфигурацию сети cloud-init (если она мешает вашей ручной настройке netplan):

```
sudo nano /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg
```

Содержимое файла:

```
network: {config: disabled}
```

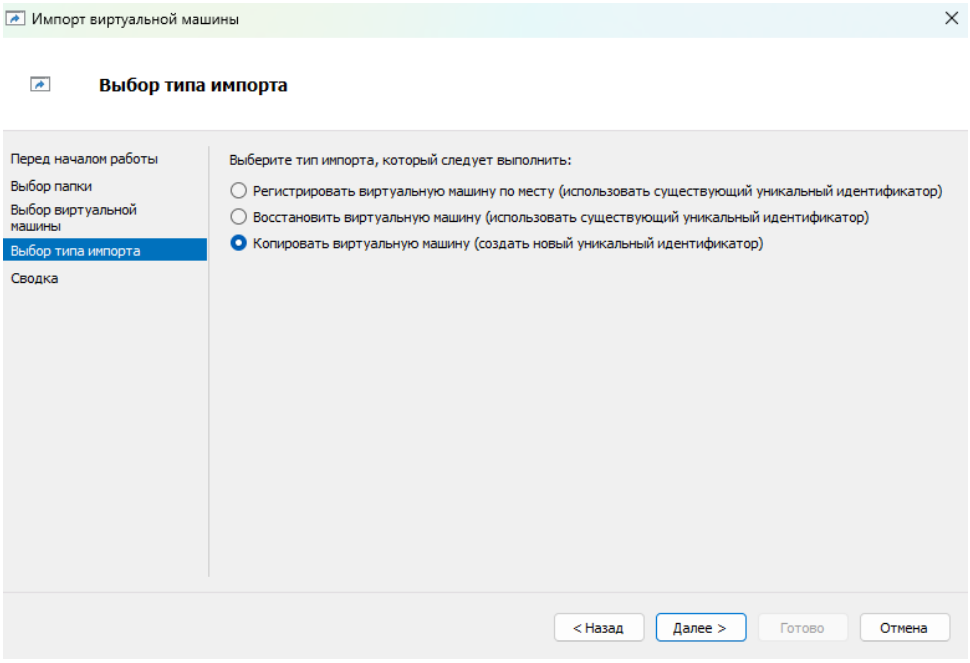
В файле `/etc/hosts` укажите адреса и имена ваших будущих виртуальных машин:

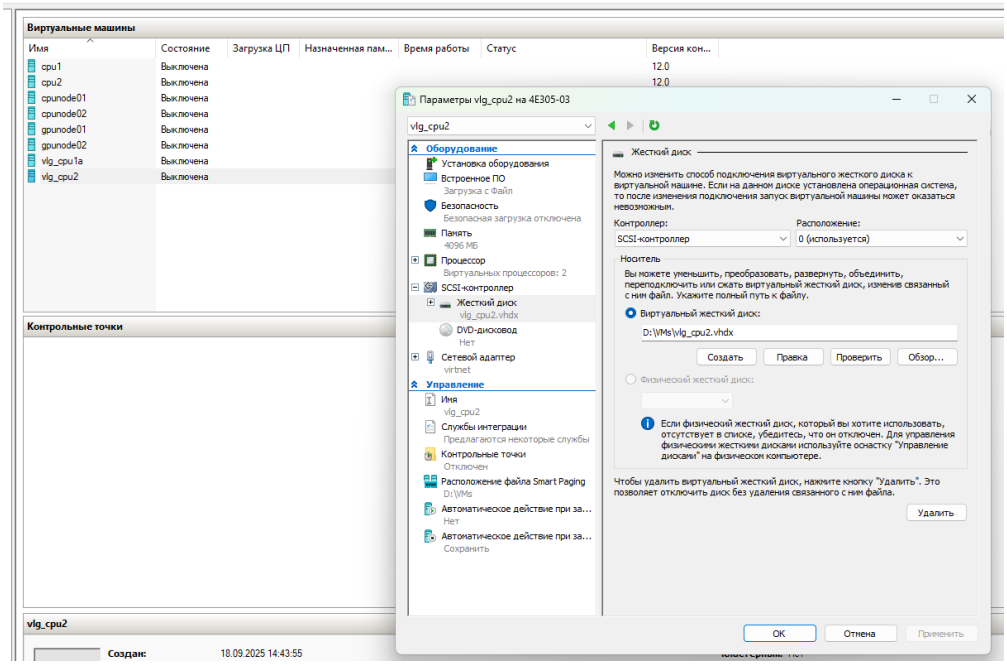


5. Клонирование машины

Цель — быстро развернуть несколько одинаковых узлов (например, `cpu1`, `cpu2`, `gpu1`) на основе готовой эталонной ВМ.

1. **Обязательно выключите исходную ВМ**.
2. В Hyper-V Manager нажмите правой кнопкой по машине → **Export…** и выберите временную папку (например, `C:\tmp\export`).
3. После завершения экспорта при необходимости переименуйте текущую машину и её диск, добавив к имени букву, например:
    - `vlg_cpu01` → `vlg_cpu01a`.
4. Выполните **Import Virtual Machine…** и импортируйте ВМ из папки экспорта как **новую** машину (новый идентификатор).



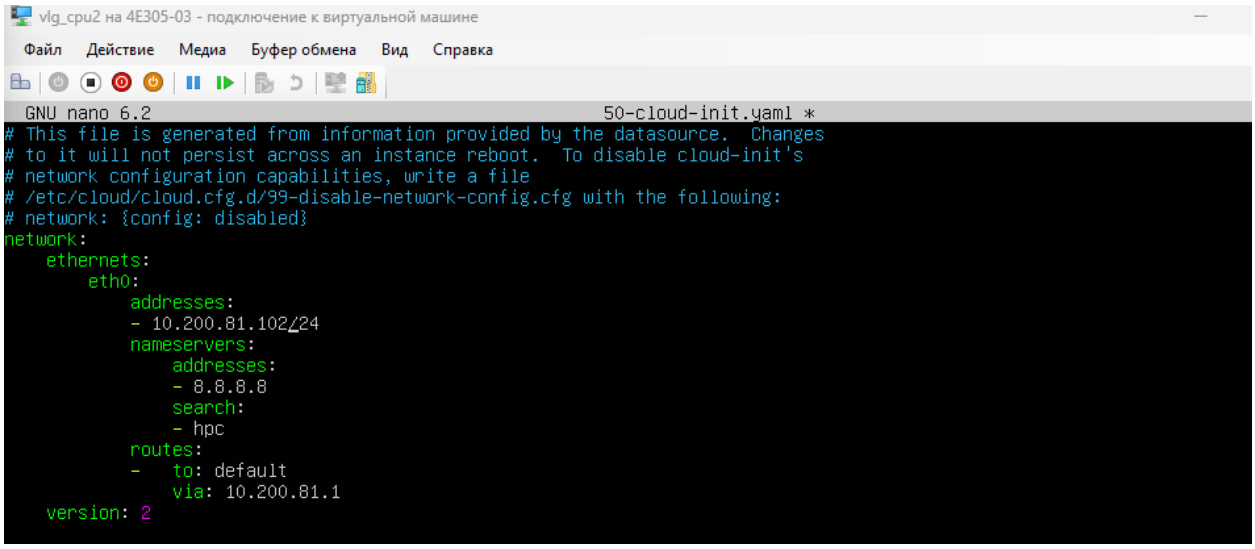


После импорта:

- Переименуйте новую ВМ и её виртуальный диск в нужный вариант (например, `vlg_cpu02`).
- Старую ВМ можно вернуть к исходному имени, если это удобно.

На клоне измените IP-адрес в конфигурации сети, например в файле `/etc/netplan/50-*.yaml`:

```
sudo nano /etc/netplan/50-cloud-init.yaml
```



Задайте уникальный IP для каждой машины в подсети `10.200.X.0/24`.

Измените имя хоста на новое:

```
hostnamectl set-hostname cpu_2
```

Примените изменения:

```
sudo netplan apply
```

```
sudo reboot
```

6. Подключение между машинами

На хосте (Windows) можно проверить IP-адреса:

```
Get-NetIPAddress
```

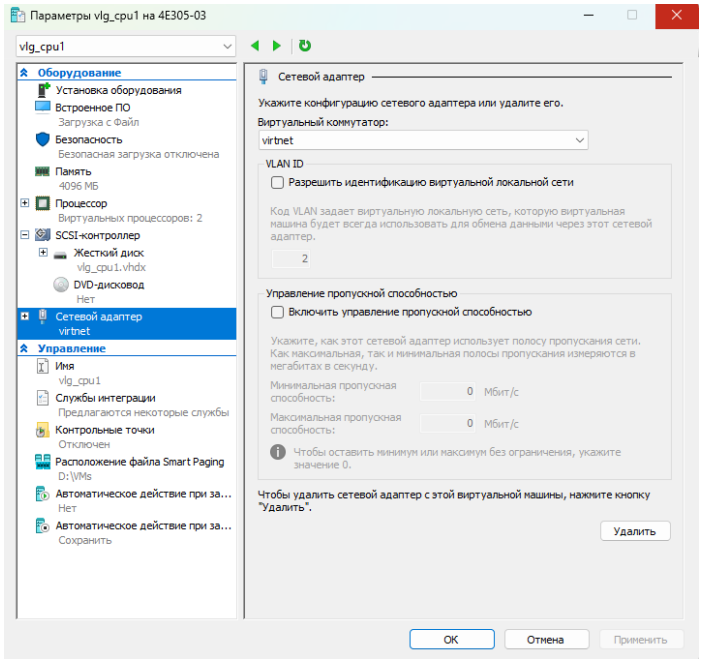
```
IPAddress         : 10.200.81.1
InterfaceIndex    : 4
InterfaceAlias    : vEthernet (virtnet)
AddressFamily     : IPv4
Type              : Unicast
PrefixLength      : 24
PrefixOrigin      : Manual
SuffixOrigin      : Manual
AddressState      : Preferred
ValidLifetime     :
PreferredLifetime :
SkipAsSource      : False
PolicyStore       : ActiveStore
```

Убедитесь, что:

- Все ВМ находятся в одной подсети (`10.200.X.0/24`).
- NAT настроен корректно, в `Get-NetNat` виден **один** объект NAT, остальные, если есть, лучше удалить:

  ```
  Remove-NetNat -Name <лишний_NAT>
  ```

- На всех ВМ выбран верный сетевой адаптер:



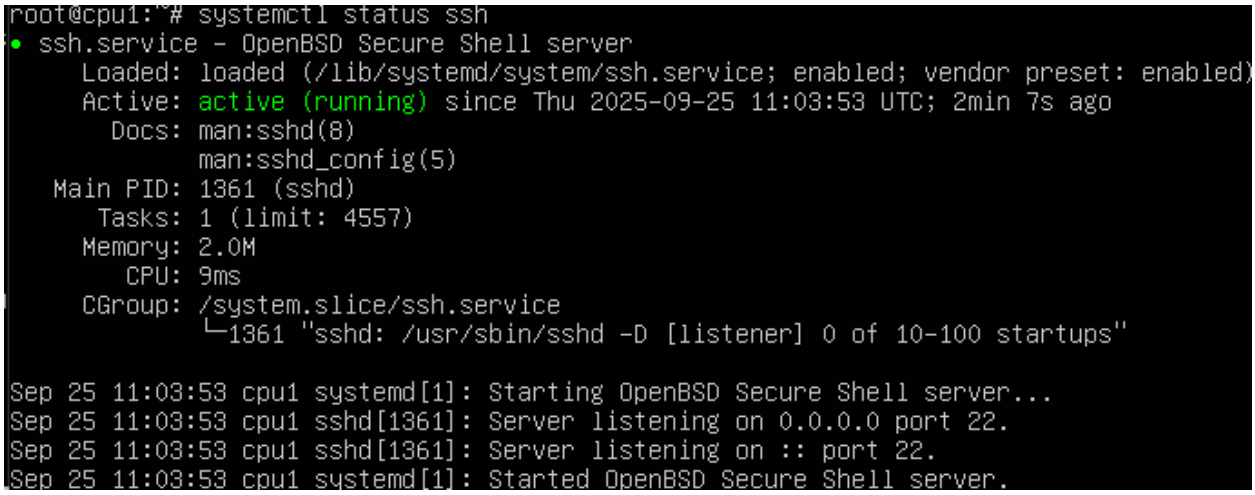
На самих ВМ проверьте связь:

```
ping 10.200.X.101   # с cpu1 на cpu2
ping 10.200.X.102   # и т.д.
```

7. Подключение по SSH

Проверяем статус службы SSH:

```
systemctl status ssh
```



Для подключения к ВМ по SSH через хост удобно настроить проброс портов в NAT.

На хосте (PowerShell от администратора) посмотрите текущие правила проброса и удалите лишние:

```
Get-NetNatStaticMapping
```

Придумайте и сохраните у себя ваше правило проброса портов:

```
10.200.81.101 cpu_1 40101
10.200.81.102 cpu_2 40102
10.200.81.111 gpu_1 40111
10.200.81.112 gpu_2 40112
```

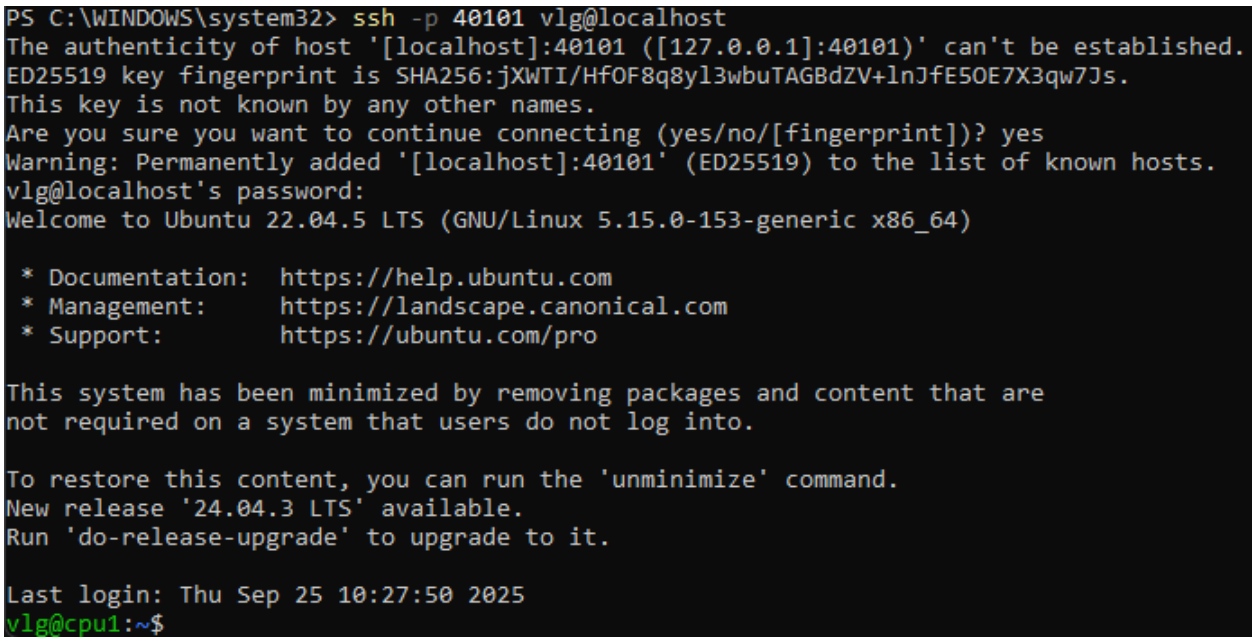
На хосте (PowerShell от администратора) добавьте правило проброса:

```
Add-NetNatStaticMapping `
  -NatName "vlgNAT" `
  -ExternalIPAddress 0.0.0.0/0 `
  -ExternalPort 40101 `
  -InternalIPAddress 10.200.X.101 `
  -InternalPort 22 `
  -Protocol TCP
```

Повторите для каждого узла, меняя `ExternalPort` и `InternalIPAddress`.

Проверка из PowerShell:

```
ssh -p 40101 vlg@localhost
```



Если подключение проходит — SSH настроен корректно.

8. Настройка GPU-узла

GPU-узел создаётся клонированием CPU-узла и добавлением GPU-партиционирования.

1. Клонируйте `cpu1` в новую машину `gpu1` (по шагам из раздела 5).
2. Запустите обе машины и проверьте, что они **пингуют** друг друга.

8.1. Получение информации о GPU на хосте

На хосте:

- Для Windows 11:

```
Get-VMHostPartitionableGpu
```

- Для Windows 10:

```
Get-VMPartitionableGpu
```

Очень важно выбрать именно **NVIDIA-ускоритель**: в строке `InstancePath` будет `VEN_10DE` (код производителя NVIDIA).

8.2. Добавление GPU-адаптера в ВМ

Пример команды:

```
Add-VMGpuPartitionAdapter `
  -VMName "vlg_gpu01" `
  -InstancePath "\\?
\PCI#VEN_10DE&DEV_2487&SUBSYS_40741458&REV_A1#4&d0467e6&0&0008#{064092
b3-625e-43bf-9eb5-dc845897dd59}\GPUPARAV" `
  -MinPartitionVRAM 0 -MaxPartitionVRAM 1000000000
-OptimalPartitionVRAM 1000000000 `
  -MinPartitionCompute 0 -MaxPartitionCompute 1000000000
-OptimalPartitionCompute 1000000000
```

Если у вас только одна видеокарта и команда ругается на `-InstancePath`, можно пропустить этот параметр — Hyper-V возьмёт доступный GPU по умолчанию.

В PowerShell (от имени администратора) настраиваем узлы с CUDA:

```
PS C:\Users\gaar.vs> Set-VM -VMName vlg_gpu01 -GuestControlledCacheTypes $true -LowMemoryMappedIoSpace
 3GB -HighMemoryMappedIoSpace 32GB
```

8.3. Подготовка драйверов CUDA через WSL

На хосте нужно получить драйверы, которые использует WSL.

1. Найдите путь к драйверам видеокарты, например:

```
PS C:\Users\gaar.vs> Get-CimInstance -ClassName Win32_VideoController -Property *
```

```
InstalledDisplayDrivers     : C:\WINDOWS\System32\DriverStore\FileRepository\nv_dispig.inf_amd64_0af
                              ec3f2050014a0\nvldumdx.dll,C:\WINDOWS\System32\DriverStore\FileReposit
                              ory\nv_dispig.inf_amd64_0afec3f2050014a0\nvldumdx.dll,C:\WINDOWS\Syste
                              m32\DriverStore\FileRepository\nv_dispig.inf_amd64_0afec3f2050014a0\nv
                              ldumdx.dll,C:\WINDOWS\System32\DriverStore\FileRepository\nv_dispig.in
                              f_amd64_0afec3f2050014a0\nvldumdx.dll
```

В данном случае путь:

```
C:
\WINDOWS\System32\DriverStore\FileRepository\nv_dispig.inf_amd64
_0afec3f2050014a0\
```

2. Убедитесь, что в каталоге `C:\Windows\System32\lxss\lib` есть файлы:

- `libcuda.so`
- `libd3d12core.so`
- `libnvoptix.so.1`

Если их нет:

```
wsl --update
```

```
wsl --install
```

```
wsl –shutdown
```

При установке создайте пользователя WSL, затем зайдите в него:

```
wsl
```

Внутри WSL:

```
cd /usr/lib/wsl/lib
```

```
ls
```

тут должны быть все файлы.

Далее смонтируйте диск C:

```
sudo mkdir -p /mnt/c
```

```
sudo mount -t drvfs C: /mnt/c
```

```
ls /mnt/c/Users   # проверка
```

Скопируйте библиотеку в удобное место на диске C:

```
sudo cp -r /usr/lib/wsl/lib /mnt/c/lib
```

8.4. Копирование драйверов на GPU-узел

Включите ВМ `gpu1`. С хоста скопируйте каталоги с драйверами (через проброшенный SSH-порт, к примеру 40111):

```
scp -r -P 40111 "C:
\Windows\System32\DriverStore\FileRepository\nv_dispig.inf_amd64_0afec
3f2050014a0" vlg@localhost:/tmp/
```

```
scp -r -P 40111 "C:\Windows\System32\lxss\lib" vlg@localhost:/tmp/
```

```
# или путь /mnt/c/lib, если копировали из WSL туда
```

На ВМ (от root):

```
cd /usr/lib
```

```
mkdir -p wsl/drivers
```

```
mv /tmp/nv_dispig.inf_amd64_0afec3f2050014a0 /usr/lib/wsl/drivers/
mv /tmp/lib /usr/lib/wsl/


chown -R root:root /usr/lib/wsl/*
chmod -R 755 /usr/lib/wsl/*
```

Создайте конфиги:

```
cd /etc/ld.so.conf.d/
nano ld.wsl.conf
```

Содержимое, например:

Затем:

```
cd /etc/profile.d/
nano wsl.sh
```

Пример содержимого:

Сделайте скрипт исполняемым:

```
chmod +x /etc/profile.d/wsl.sh
```

Создайте необходимые симлинки:

```
ln -sf /usr/lib/wsl/lib/libd3d12core.so /usr/lib/wsl/lib/
libD3D12Core.so
ln -sf /usr/lib/wsl/lib/libnvoptix.so.1      /usr/lib/wsl/lib/
libnvoptix_loader.so.1
ln -sf /usr/lib/wsl/lib/libcuda.so           /usr/lib/wsl/lib/
libcuda.so.1
```

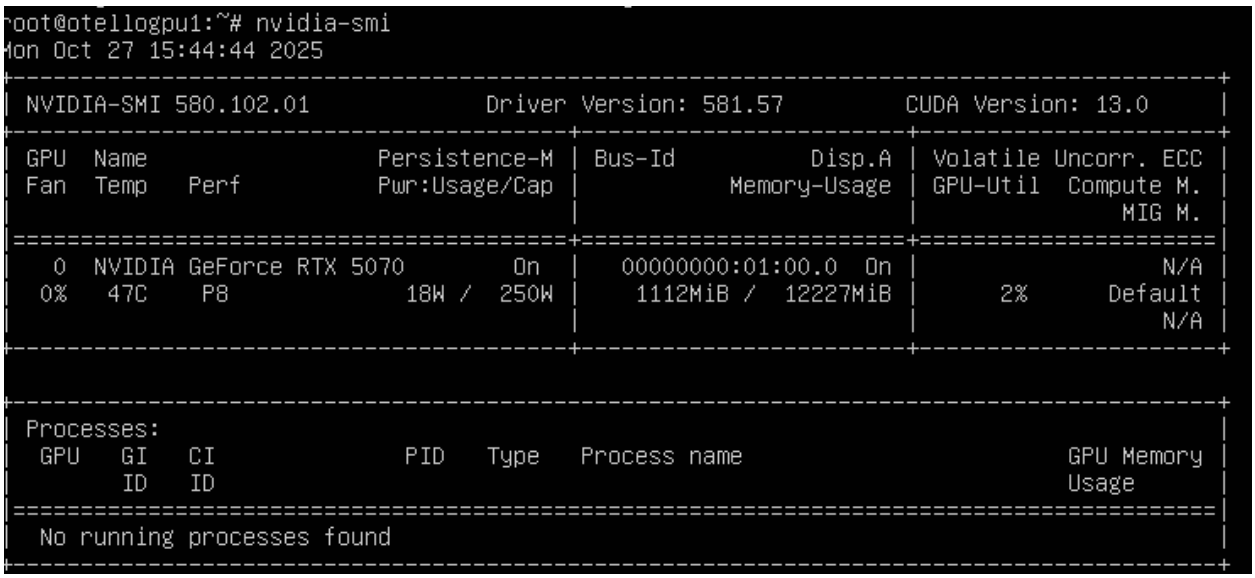
Соберите ядро для dxgkrnl:

```
curl -fsSL https://content.staralt.dev/dxgkrnl-dkms/main/install.sh |
sudo bash -es
```

Перезапустите ВМ и проверьте:

```
nvidia-smi
```

Если всё прошло успешно, вы увидите информацию об ускорителе:



Ошибка вида:

```
NVIDIA-SMI has failed because it couldn't communicate with the
NVIDIA driver. Make sure that the latest NVIDIA driver is
installed and running. Failed to properly shut down NVML: Driver
Not Loaded
```

означает, что драйвер вашей видеокарты отличается от драйвера на узле. Необходимо переустановить драйвер.

При ошибке вида:

```
Failed to initialize NVML: GPU access blocked by the operating
system
```

проверьте, перезагружали ли ВМ, и при необходимости повторите шаги настройки.

Удаление собранного драйвера (при необходимости)

Если после нескольких попыток проблема не решается, можно полностью удалить собранный драйвер `dxgkrnl` и выполнить установку заново. Для удаления используйте следующие команды:

```
sudo modprobe -r dxgkrnl 2>/dev/null || true

for v in $(dkms status dxgkrnl 2>/dev/null | sed -n 's/dxgkrnl\/
\([0-9a-f]\+\).*/\1/p' | sort -u); do

  sudo dkms remove -m dxgkrnl -v "$v" --all || true

done

sudo rm -rf /usr/src/dxgkrnl-* 2>/dev/null

sudo rm -rf /var/lib/dkms/dxgkrnl 2>/dev/null

sudo find /lib/modules -type f -name 'dxgkrnl.ko*' -delete

sudo find /lib/modules -type d -path '*/drivers/hv/dxgkrnl'
-empty -delete
```

```
sudo depmod -a

sudo rm -rf /tmp/WSL2-Linux-Kernel 2>/dev/nul

sudo reboot
```

После успешной настройки можно клонировать GPU-машину по аналогии с CPU-узлами, не забывая снова пробрасывать GPU через `Add-VMGpuPartitionAdapter` и настраивать ВМ через `Set-VM`.

9. Установка slurm

9.1. Общая схема

- На **главном узле** устанавливается служба `slurmctld`.
- На **рабочих узлах** — `slurmd`.
- Для аутентификации используется `munge`.

Убедитесь, что библиотека `munge` установлена на всех узлах.

9.2. Установка пакетов

На всех узлах (от root):

```
apt install -y slurm-wlm munge
```

Проверьте каталог ключей `munge` (обычно `/etc/munge/`):

- На всех узлах ключ должен быть **одинаковый** (значения `md5sum` совпадают).
- Если хэши различаются, скопируйте ключи с основного узла на остальные и исправьте владельца:

```
chown -R munge:munge /etc/munge
chmod 700 /etc/munge
systemctl restart munge
```

9.3. Подготовка конфигурации Slurm

Конфигурация по умолчанию — `/etc/slurm/slurm.conf`.

Сгенерировать конфиг удобнее через Slurm System Configuration Tool (веб-форма):

Находим используемый домен:

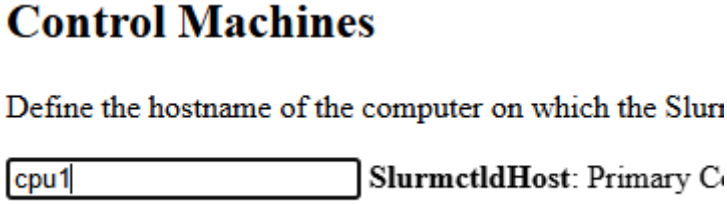
```
cat /etc/resolv.conf
```

В конце файла указано:

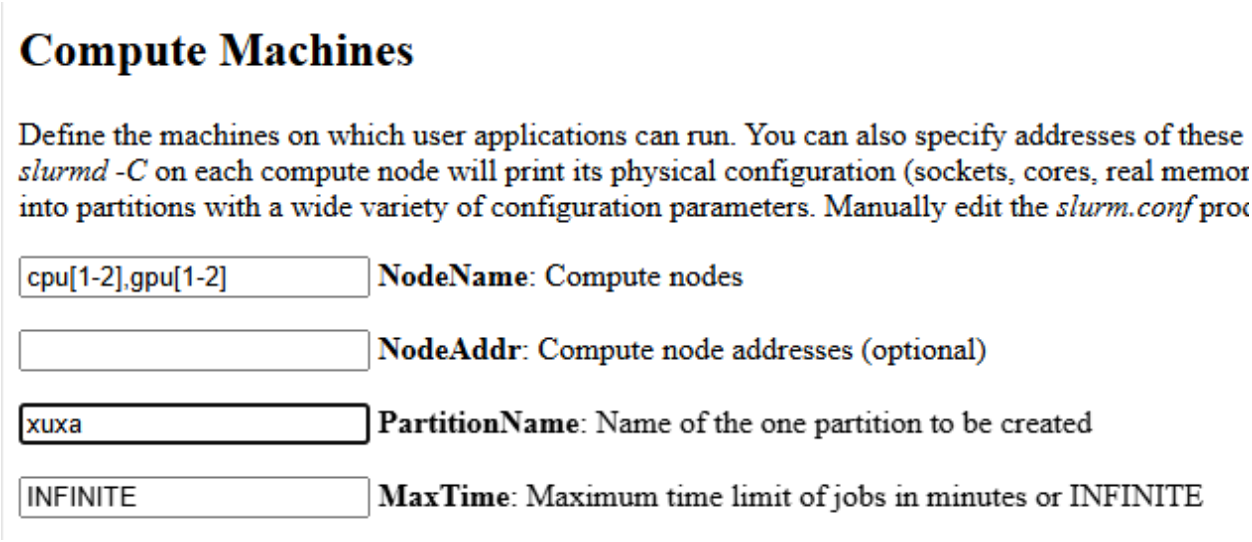
```
search hpc
```

Укажите домен и имя кластера:

В поле контроллера укажите **главный узел:**

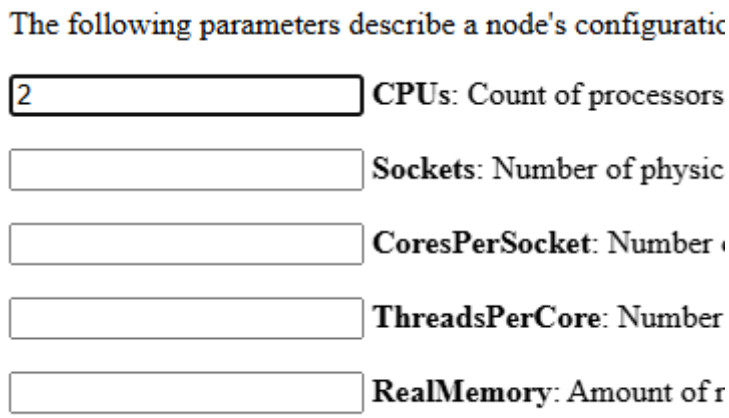


В разделе узлов перечислите все узлы кластера (можно использовать маски / регулярки):

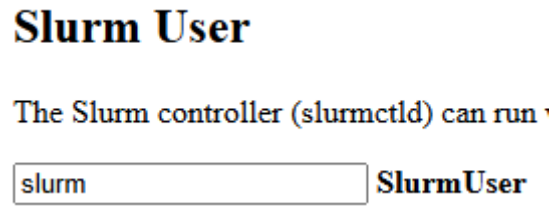


- Адреса оставляем пустыми
- В третьем поле указываем любое название
- В четвёртом поле можем указать максимально время выполнение задачи

В разделе ресурсов укажите количество CPU/ядер:



Пользователя демона Slurm оставьте `slurm`:



Остальные параметры (очереди, интерконнект, prolog/epilog и т.п.) можно оставить по умолчанию:

State Preservation

Define the location of a directory where the slurmctld daemon
should also be defined. This must be a unique directory on eac

/var/spool/slurmctld **StateSaveLocation**: Slurmctld sta

/var/spool/slurmd **SlurmdSpoolDir**: Slurmd state sa

Define when a non-responding (DOWN) node is returned to se
Select one value for **ReturnToService**:

- ○ **0**: When explicitly restored to service by an administrator.
- ○ **1**:Upon registration with a valid configuration only if it wa
- ◉ **2**:Upon registration with a valid configuration.

Scheduling

Define the mechanism to be used for controlling job ordering.
Select one value for **SchedulerType**:

- ◉ **Backfill**: FIFO with backfill
- ○ **Builtin**: First-In First-Out (FIFO)

Interconnect

Define the node interconnect used.
Select one value for **SwitchType**:

- ○ **HPE Slingshot**: HPE Slingshot proprietary interconnect
- ◉ **None**: No special handling required (InfiniBand, Myrinet, Ethernet, etc.)

Default MPI Type

Specify the type of MPI to be used by default. Slurm will configu
Select one value for **MpiDefault**:

- ○ **MPI-PMI2** (For PMI2-supporting MPI implementations)
- ○ **MPI-PMIx** (Exascale PMI implementation)
- ◉ **None**: This works for most other MPI types.

Process Tracking

Define the algorithm used to iden
Select one value for **ProctrackTy**

- (●) **Cgroup**: Use Linux *cgroup* to
- ( ) **LinuxProc**: Use parent proce
- ( ) **Pgid**: Use Unix Process Grou

Resource Selection

Define resource (node) selection algorithm to be used.
Select one value for **SelectType**:

- (●) **cons_tres**: Allocate individual processors, memory, GPUs, and other trackable resources
- ( ) **Linear**: Node-base resource allocation, does not manage individual processor allocation

Task Launch

Define a task launch plugin. This may

- [ ] **None**: No task launch actions
- [x] **Affinity**: CPU affinity support (see
- [x] **Cgroup**: Allocated resources const

Event Logging

Slurmctld and slurmd daemons can each be con

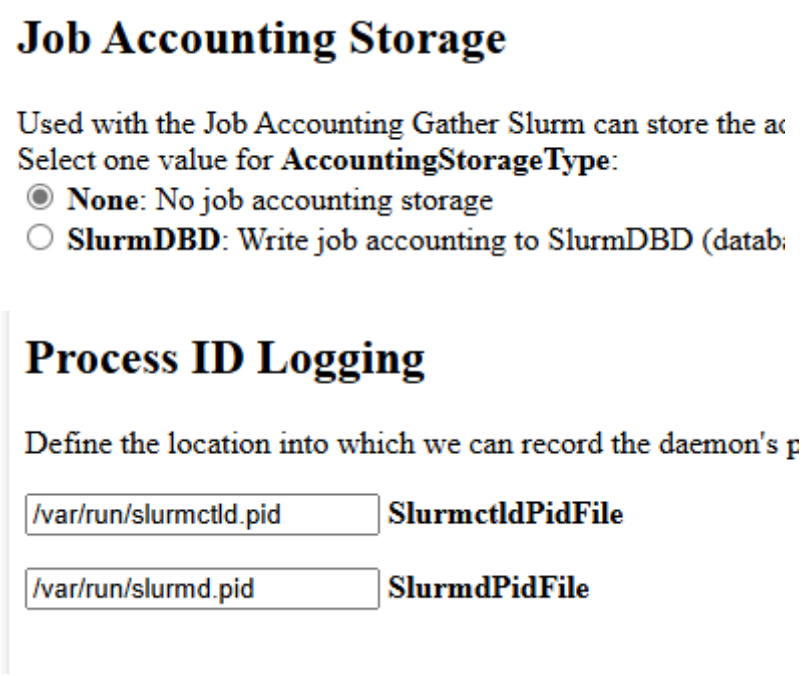
/var/log/slurmctld.log **SlurmctldLogFile**

/var/log/slurmd.log **SlurmdLogFile** (if

Job Accounting Gather

Slurm accounts for resource use per job. System sp
Select one value for **JobAcctGatherType**:

- (●) **None**: No job accounting
- ( ) **cgroup**: Specific Linux cgroup information gat
- ( ) **Linux**: Specific Linux process table informatio



Сохраните сгенерированный `slurm.conf` и скопируйте его на главный узел в `/etc/slurm/slurm.conf`.

Проверьте, что файл читается всеми:

```
chmod 644 /etc/slurm/slurm.conf
```

Скопируйте конфиг на остальные узлы (scp, rsync) в тот же путь.

Создайте конфиг `cgroup.conf`:

```
sudo nano /etc/slurm/cgroup.conf
```

Минимальное содержимое:

```
CgroupAutomount=true

ConstrainCores=no

ConstrainRAMSpace=no
```

Скопируйте этот файл на все узлы и убедитесь, что у него права на чтение (`644`).

9.4. SSH-ключи между узлами

От обычного пользователя (не root) на главном узле:

```
ssh-keygen      # параметры по умолчанию

ssh-copy-id <имя_узла_1>

ssh-copy-id <имя_узла_2>

...
```

Это нужно для запуска задач и администрирования без постоянного ввода пароля.

9.5. Запуск служб

На рабочих узлах:

```
sudo systemctl restart slurmd
```

На главном узле:

```
sudo systemctl restart slurmctld
```

Проверьте состояние:

```
systemctl status slurmd
systemctl status slurmctld
```

Убедитесь, что все файлы и директории, указанные в `slurm.conf`, существуют и принадлежат пользователю `slurm`:

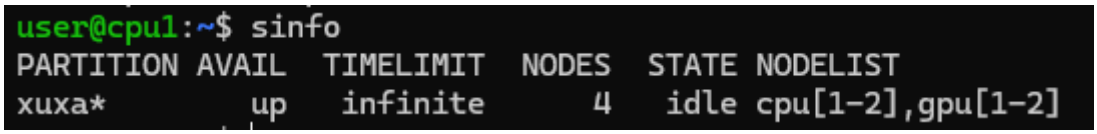
```
chown -R slurm:slurm /var/spool/slurm*
```

Если их нет, то создаем их. Важно помнить, что там могут быть не только файлы, но и директории.

Проверьте видимость узлов:

```
sinfo
```

```
scontrol show nodes
```



10. Настройка серверной части (NFS и CUDA)

10.1. NFS-шеринг

На **главном узле** (сервер):

```
sudo apt install -y nfs-kernel-server
```

На остальных узлах (клиенты):

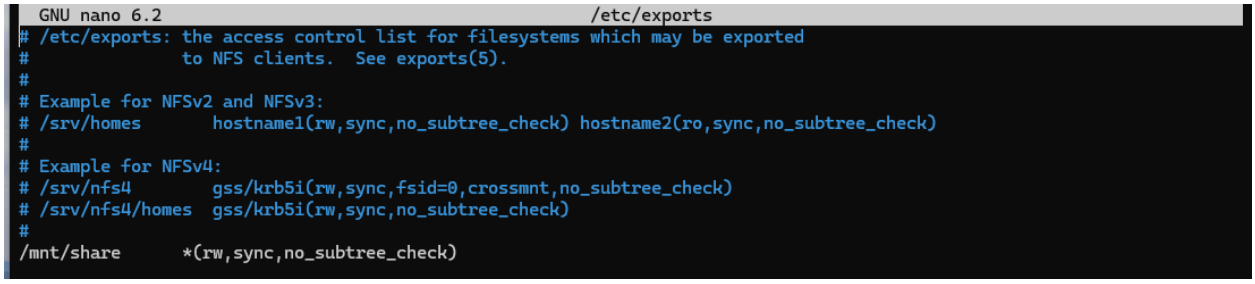
```
sudo apt install -y nfs-common
```

Создайте каталог на всех узлах:

```
sudo mkdir -p /mnt/share
```

Добавьте строку на **главном узле**:

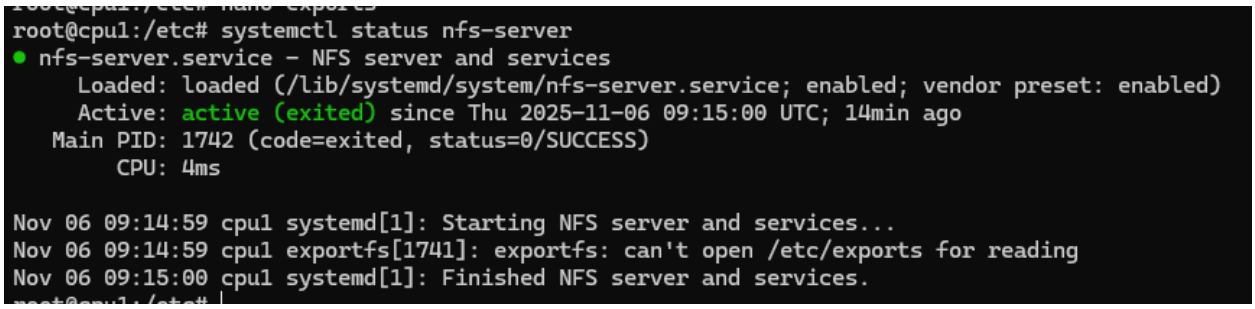
```
/mnt/share     *(rw,sync,no_subtree_check)
```



Примените настройки:

```
sudo exportfs -ra
```

```
sudo systemctl status nfs-server
```



На клиентских узлах смонтируйте ресурс:

1. Откройте /etc/fstab и добавьте строку

   ```
   cpu1:/mnt/share   /mnt/share   nfs   defaults   0 0
   ```

2. Примените настройку:

   ```
   sudo mount -a
   ```

Проверьте, что файл, созданный на главном узле в `/mnt/share`, виден на всех клиентах.

10.2. Проверка пакетов CUDA

На **основном узле** (сервере, где выполняется компиляция CUDA-кода) необходимо:

1. Проверить наличие и версию `nvcc`.
2. При необходимости установить актуальную версию CUDA Toolkit.
3. Собрать тестовый CUDA-пример.

4. Запустить его на GPU-узле через Slurm.

10.2.1. Проверка установленной версии CUDA

Проверьте, установлен ли компилятор `nvcc` и какую версию он использует:

```
nvcc --version
```

`nvcc` — это обёртка над `gcc`, которая компилирует CUDA-код.

Если команда не найдена или версия слишком старая, имеет смысл обновить CUDA Toolkit.

10.2.2. Обновление и установка CUDA Toolkit

При наличии старого пакета из репозитория дистрибутива (например, `nvidia-cuda-toolkit` в Ubuntu) его лучше удалить, чтобы он не конфликтовал с «официальной» версией CUDA от NVIDIA:

```
sudo apt remove --purge -y nvidia-cuda-toolkit
```

Далее добавьте официальный репозиторий CUDA.
Подставьте свою версию Ubuntu:

1. `ubuntu2204` для Ubuntu 22.04
2. `ubuntu2404` для Ubuntu 24.04
   и т.п.

```
wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64/cuda-keyring_1.1-1_all.deb
```

```
sudo dpkg -i cuda-keyring_1.1-1_all.deb
```

```
sudo apt update
```

Установите нужную ветку CUDA Toolkit (например, 12.8):

```
sudo apt install -y cuda-toolkit-12-8
```

Добавьте CUDA в `PATH` и `LD_LIBRARY_PATH` (пример для bash/WSL):

```
echo 'export PATH=/usr/local/cuda-12.8/bin:$PATH' >> ~/.bashrc
```

```
echo 'export LD_LIBRARY_PATH=/usr/local/cuda-12.8/lib64:$LD_LIBRARY_PATH' >> ~/.bashrc
```

```
source ~/.bashrc
```

Проверьте, что теперь используется нужная версия:

```
nvcc –version
```

11. Проверка CUDA

Сначала узнаём вычислительную способность видеокарты на CUDA-узле:

```
nvidia-smi --query-gpu=compute_cap --format=csv,noheader
```

Например, если вывод:

```
8.9
```

то архитектура для `nvcc` будет `sm_89` (число без точки).

Далее на **основном узле** компилируем тестовый файл, например `hello.cu`:

```
nvcc -arch=sm_89 hello.cu -o hello
```

После успешной компиляции перенесите бинарник в общий каталог, доступный всем узлам (NFS):

```
mv hello /mnt/share
```

11.1. Запуск теста на CUDA-узле через Slurm

Запускаем тестовый бинарник на конкретном узле с GPU (имя узла подставьте своё):

```
srun --nodelist=<название_CUDA-узла> --time=00:10:00 /mnt/share/hello
```

Если всё настроено корректно:

- задание в Slurm успешно стартует,
- бинарник выполняется без ошибок,
- при этом `nvidia-smi` на CUDA-узле должен показывать загрузку GPU процессом Slurm.

11.2. Пример 1 тестовой программы на CUDA

Ниже приведён пример простой программы на CUDA, которая не столько считает, сколько выводит информацию об устройстве. Её можно использовать как тест для проверки корректности настроек:

```
#include "cuda_runtime.h"
#include <stdio.h>
#include "device_launch_parameters.h"

#define DIM 1000
#define N 50000

__global__ void add(int* a, int* b, int* c);
void infoAboutDevices(void);

int main(void) {
    infoAboutDevices();
    return 0;
}
```

```
__global__ void add(int* a, int* b, int* c) {
    int tid = blockIdx.x;    // this thread handles the data at its thread
id
    if (tid < N)
        c[tid] = a[tid] + b[tid];
}

void infoAboutDevices(void) {
    cudaDeviceProp prop;
    int count;
    cudaGetDeviceCount(&count);
    cudaGetDeviceProperties(&prop, 0);
    printf("   --- General Information for device %d ---\n", 0);
    printf("Name:  %s\n", prop.name);
    printf("Compute capability:  %d.%d\n", prop.major, prop.minor);
    printf("Clock rate:  %d\n", prop.clockRate);
    printf("Device copy overlap:  ");
    if (prop.deviceOverlap)
        printf("Enabled\n");
    else
        printf("Disabled\n");
    printf("Kernel execution timeout :  ");
    if (prop.kernelExecTimeoutEnabled)
        printf("Enabled\n");
    else
        printf("Disabled\n");

    printf("   --- Memory Information for device %d ---\n", 0);
    printf("Total global mem:  %ld\n", prop.totalGlobalMem);
    printf("Total constant Mem:  %ld\n", prop.totalConstMem);
    printf("Max mem pitch:  %ld\n", prop.memPitch);
    printf("Texture Alignment:  %ld\n", prop.textureAlignment);

    printf("   --- MP Information for device %d ---\n", 0);
    printf("Multiprocessor count:  %d\n",
        prop.multiProcessorCount);
```

```
        printf("Shared mem per mp:  %ld\n", prop.sharedMemPerBlock);
        printf("Registers per mp:  %d\n", prop.regsPerBlock);
        printf("Threads in warp:  %d\n", prop.warpSize);
        printf("Max threads per block:  %d\n",
            prop.maxThreadsPerBlock);
        printf("Max thread dimensions:  (%d, %d, %d)\n",
            prop.maxThreadsDim[0], prop.maxThreadsDim[1],
            prop.maxThreadsDim[2]);
        printf("Max grid dimensions:  (%d, %d, %d)\n",
            prop.maxGridSize[0], prop.maxGridSize[1],
            prop.maxGridSize[2]);
        printf("\n");
    }
```

Рекомендуемый рабочий процесс:

1. Сохранить этот код в файл `hello.cu` на **основном узле**.
2. Скомпилировать с нужной архитектурой:
   `nvcc -arch=sm_89 hello.cu -o hello`
3. Переместить бинарник в общий каталог `/mnt/share`.
4. Запустить через Slurm на узле с GPU:
   `srun --nodelist=<название_CUDA-узла> --time=00:10:00 /mnt/share/hello`

После запуска должна вывестись некоторая информация о видеокарте. Например:

```
   --- General Information for device 0 ---
Name:  NVIDIA GeForce RTX 5070
Compute capability:  12.0
Clock rate:  2512000
Device copy overlap:  Enabled
Kernel execution timeout :  Enabled
   --- Memory Information for device 0 ---
Total global mem:  12820480000
Total constant Mem:  65536
Max mem pitch:  2147483647
Texture Alignment:  512
   --- MP Information for device 0 ---
Multiprocessor count:  48
Shared mem per mp:  49152
Registers per mp:  65536
```

```
  Threads in warp:  32
  Max threads per block:  1024
  Max thread dimensions:  (1024, 1024, 64)
  Max grid dimensions:  (2147483647, 65535, 65535)
```

11.3. Пример 2 тестовой программы на CUDA

Ниже приведён простой пример программы на CUDA, которая проверяет корректность выделения памяти узлам.

```
// file: simple_cuda_malloc.cu

#include <cuda_runtime.h>
#include <stdio.h>

#define CUDA_CHECK(call)                                    \
    do {                                                    \
        cudaError_t err = (call);                           \
        if (err != cudaSuccess) {                           \
            fprintf(stderr, "CUDA error %s:%d: %s\n",       \
                    __FILE__, __LINE__,                     \
                    cudaGetErrorString(err));               \
            return 1;                                       \
        }                                                   \
    } while (0)

int main() {
    int N = 10;
    size_t bytes = N * sizeof(int);

    // --- хостовая (CPU) память ---
    int *h_data = (int*)malloc(bytes);
    if (!h_data) {
        fprintf(stderr, "Host malloc failed\n");
        return 1;
    }

    // инициализация данных на хосте
    for (int i = 0; i < N; ++i) {
        h_data[i] = i;
```

```
    }

    // --- память на устройстве (GPU) ---
    int *d_data = NULL;

    // выделяем память на GPU
    CUDA_CHECK(cudaMalloc((void**)&d_data, bytes));

    // копируем данные с хоста на устройство
    CUDA_CHECK(cudaMemcpy(d_data, h_data, bytes, cudaMemcpyHostToDevice));

    // для примера обнулим данные на устройстве
    CUDA_CHECK(cudaMemset(d_data, 0, bytes));

    // копируем данные обратно на хост
    CUDA_CHECK(cudaMemcpy(h_data, d_data, bytes, cudaMemcpyDeviceToHost));

    // выводим результат
    printf("Данные после cudaMemset (должны быть нули):\n");
    for (int i = 0; i < N; ++i) {
        printf("%d ", h_data[i]);
    }
    printf("\n");

    // освобождаем память
    CUDA_CHECK(cudaFree(d_data));
    free(h_data);

    // корректно завершаем работу с CUDA
    CUDA_CHECK(cudaDeviceReset());

    return 0;
}
```

Ожидаемый вывод: 0 0 0 0 0 0 0 0 0 0

Если вы получили ошибку «`CUDA error /home/otello/test.cu:37: out of memory`», значит, система получает данные о карте, но не может выделить ресурсы.

Одна из причин – минимальный показатель загрузки и памяти при выделении доступа к ускорителю были нулевые. Попробуйте их увеличить до 10% от максимальной:

```
  Add-VMGpuPartitionAdapter -VMName <название_ВМ> -InstancePath
"\\?
\PCI#VEN_10DE&DEV_2F04&SUBSYS_F3261569&REV_A1#740E0A73882DB04800
#{064092b3-625e-43bf-9eb5-dc845897dd59}\GPUPARAV"
-MinPartitionVRAM 100000000 -MaxPartitionVRAM 1000000000
-OptimalPartitionVRAM 1000000000 -MinPartitionCompute 100000000
-MaxPartitionCompute 1000000000 -OptimalPartitionCompute
1000000000
```

12. Проверка MPI

mpi_test.cpp:

```
#include <mpi.h>
#include <iostream>

int main(int argc, char** argv) {
    // 1. Инициализируем MPI
    MPI_Init(&argc, &argv);

    // 2. Получаем число процессов
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // 3. Получаем номер текущего процесса
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // 4. Узнаём имя узла
    char node_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(node_name, &name_len);

    // 5. Печатаем информацию
    std::cout << "Hello from rank " << rank
              << " of " << world_size
              << " on node " << node_name << std::endl;

    // 6. Завершаем MPI
    MPI_Finalize();
    return 0;
}
```

Компилируем программу на главном узле:

```
mpicxx mpi_test.cpp -o mpi_test
```

Переносим исполняемый файл в `/mnt/share`.

В /mnt/share создаём файл `mpi_test.slurm` с вашими значениями параметров `--partition` и `--nodelist`:

```
#!/bin/bash
#SBATCH --job-name=mpi_test
#SBATCH --partition=hpc_part
#SBATCH --nodelist=cpu1,cpu2,gpu1,gpu2
#SBATCH --ntasks-per-node=2
#SBATCH --output=mpi_test.out
#SBATCH --error=mpi_test.err

mpirun /mnt/share/mpi_test
```

Если файл создавался в Windows, необходимо конвертировать переносы из CRLF в LF:

```
dos2unix mpi_test.slurm
```

Отправить файл в работу:

```
sbatch mpi_test.slurm
```

Открыть вывод:

```
cat mpi_test.out
```

Пример вывода:

```
Hello from rank 0 of 8 on node cpu1
Hello from rank 1 of 8 on node cpu1
Hello from rank 4 of 8 on node gpu1
Hello from rank 5 of 8 on node gpu1
Hello from rank 2 of 8 on node cpu2
Hello from rank 3 of 8 on node cpu2
Hello from rank 6 of 8 on node gpu2
Hello from rank 7 of 8 on node gpu2
```