

010123131

# Software Development Practice I

## Handout #3

<rawat.s@eng.kmutnb.ac.th>

Last Update: 2024-07-05

# Agenda

- Access to remote servers using SSH
- Installation of OpenSSH server & client tools
- SSH authentication methods
- Remote file copy and remote program execution on Ubuntu servers using SSH
- Remote software development with VS Code IDE

# SSH

- **SSH (Secure Shell)** is a **network protocol** used for secure remote communication between two devices.
- It provides a **secure way** to connect to a remote device over an unsecured network by encrypting all data transmitted between the devices.
- It supports **public key cryptography** for authenticating the devices and protecting the integrity of the data.

# SSH

- **SSH** is based on a **client-server architecture**.
- The **SSH server** runs on the remote machine, while the **SSH client** runs on the local machine.
- **OpenSSH** is an open source implementation of the **SSH protocol**.

**OpenSSH**

<https://www.openssh.com/>

# SSH

- The primary use of **SSH** is for **secure remote login** to a remote computer or server.
- Once connected, users can execute commands on the remote system as if they were sitting in front of it.
- **SSH** also supports **secure file transfer** (e.g. scp), used to securely copy files between devices.

# SSH

- When a user (client) attempts to connect to a remote system using SSH, the server sends its **public key** to the client, which encrypts its **session key** with the server's **public key** and sends it back to the server.
- The server decrypts the **session key** using its **private key** and uses this **session key** to encrypt all data transmitted between the client and the server.

# OpenSSH for Ubuntu

- Installation of the **OpenSSH client and server**:

```
$ sudo apt install openssh-client
```

```
$ sudo apt install openssh-server
```

```
$ sudo systemctl enable ssh
```

```
$ sudo systemctl start ssh
```

```
$ sudo systemctl status ssh
```

Note: **sshd** (OpenSSH server) is the daemon program for ssh client.

To login to a **remote SSH server** using the **SSH client** program:

```
$ ssh <username@remote_server> -p <port_number>
```

# SSH Client

Use a SSH client in Ubuntu to access a remote server (Raspberry Pi) in a LAN.

```
# Remove all existing entries for the host named "raspberrypi"
# from the ~/.ssh/known_hosts file.
$ ssh-keygen -f ~/.ssh/known_hosts -R "raspberrypi"

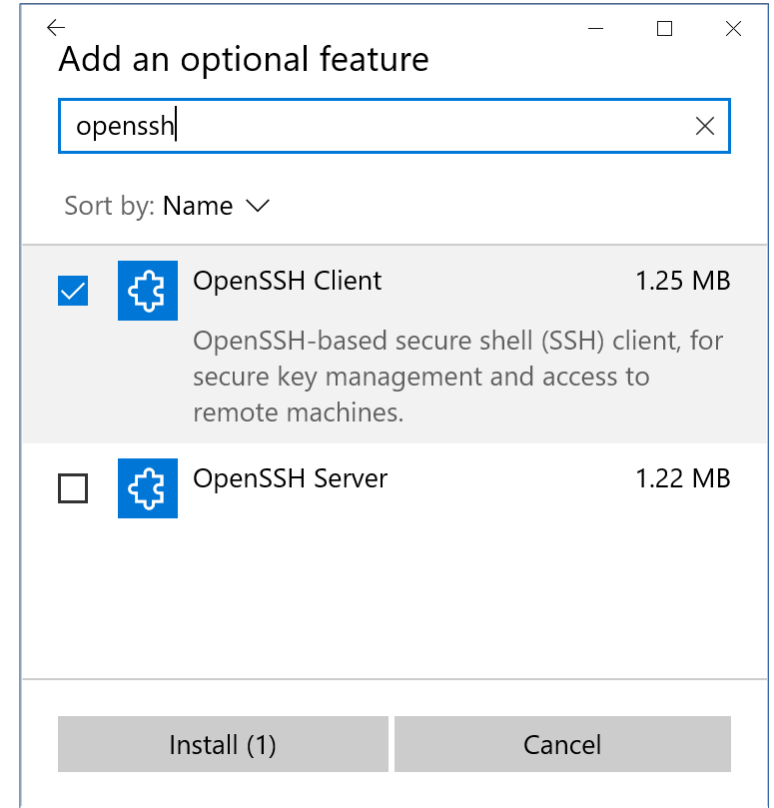
# Use SSH client to remote access and login
# to the RPi SBC (hostname: "raspberrypi").
$ ssh pi@raspberrypi
```

Note: The default port number for SSH is 22.



# Installation of the OpenSSH Client for Windows

1. Open **Settings**, select **Apps**, then select **Optional Features**.
2. Scan the list to see if the OpenSSH is already installed. If not, at the top of the page, select **Add a feature**, then:
  - Find **OpenSSH Client**, then select **Install**
  - Find **OpenSSH Server**, then select **Install**
3. Once setup completes, return to **Apps** and **Optional Features** and confirm OpenSSH is listed.
4. Open the **Services** desktop app. (Select **Start**, type *services.msc* in the search box, and then select the **Service** app or press **ENTER**.)
5. In the details pane, double-click **OpenSSH SSH Server**.
6. On the **General** tab, from the **Startup type** drop-down menu, select **Automatic**.
7. To start the service, select **Start**.



# OpenSSH for Windows

OpenSSH for Windows has the below commands built in.

- `ssh` is the SSH client component that runs on the user's local system
- `sshd` is the SSH server component that must be running on the system being managed remotely
- `ssh-keygen` generates, manages and converts authentication keys for SSH
- `ssh-agent` stores private keys used for public key authentication
- `ssh-add` adds private keys to the list allowed by the server
- `ssh-keyscan` aids in collecting the public SSH host keys from hosts
- `sftp` is the service that provides the Secure File Transfer Protocol, and runs over SSH
- `scp` is a file copy utility that runs on SSH

# Ubuntu VM Settings

Oracle VM VirtualBox Manager

File Machine Help

Tools

New Add Settings Discard Show

64 **Ubuntu Desktop 22.04 LTS** Running

### General

Name: Ubuntu Desktop 22.04 LTS  
Operating System: Ubuntu (64-bit)

### System

Base Memory: 2048 MB  
Processors: 2  
Boot Order: Hard Disk, Optical, Floppy  
Acceleration: Nested Paging, KVM Paravirtualization

### Display

Video Memory: 16 MB  
Graphics Controller: VMSVGA  
Remote Desktop Server: Disabled  
Recording: Disabled

### Storage

Controller: IDE  
IDE Secondary Device 0: [Optical Drive] Empty  
Controller: SATA  
SATA Port 0: Ubuntu Desktop 22.04 LTS.vdi (Normal, 20.00 GB)

### Audio

Host Driver: Default  
Controller: ICH AC97

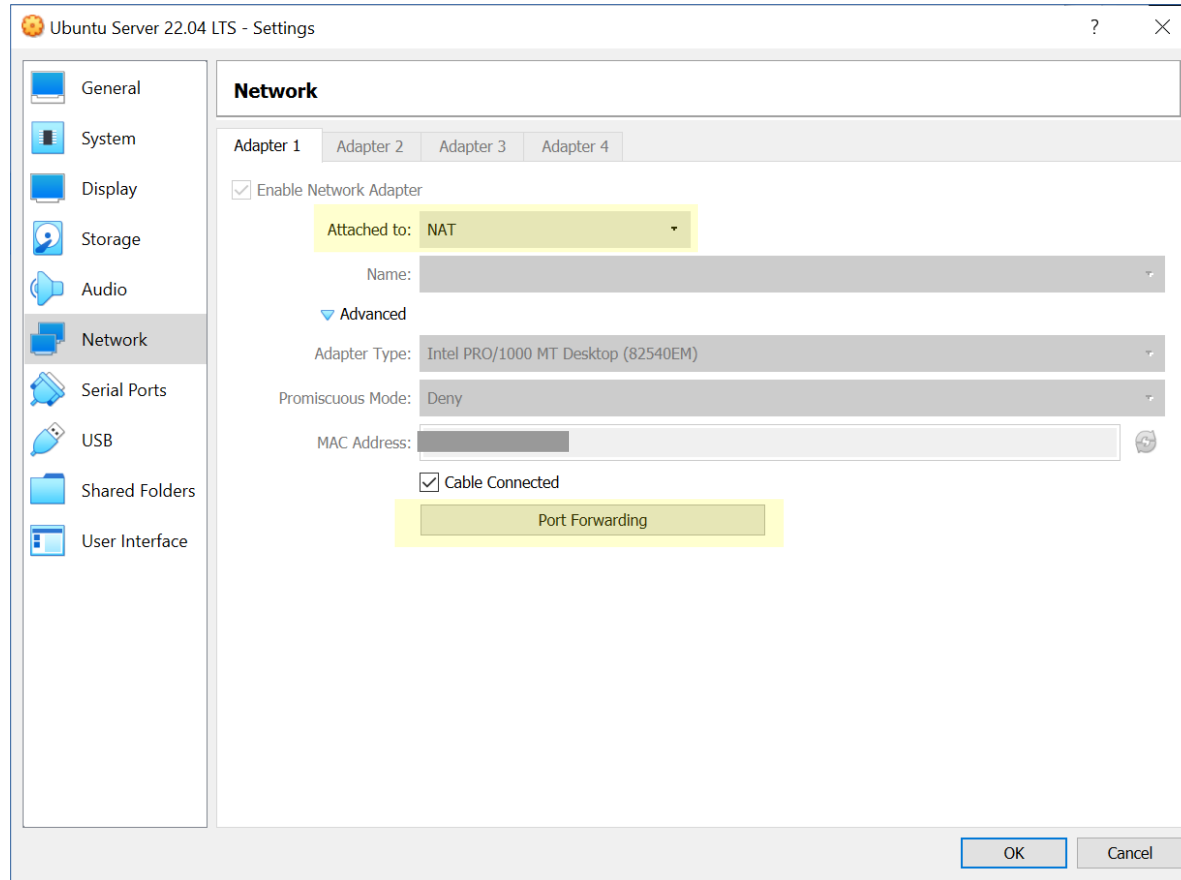
### Network

Adapter 1: Intel PRO/1000 MT Desktop (NAT)

### Preview

Ubuntu Desktop 22.04 LTS

# VirtualBox VM Settings: NAT (Network Access Translation)



# Ubuntu VM Settings: SSH Port Forwarding

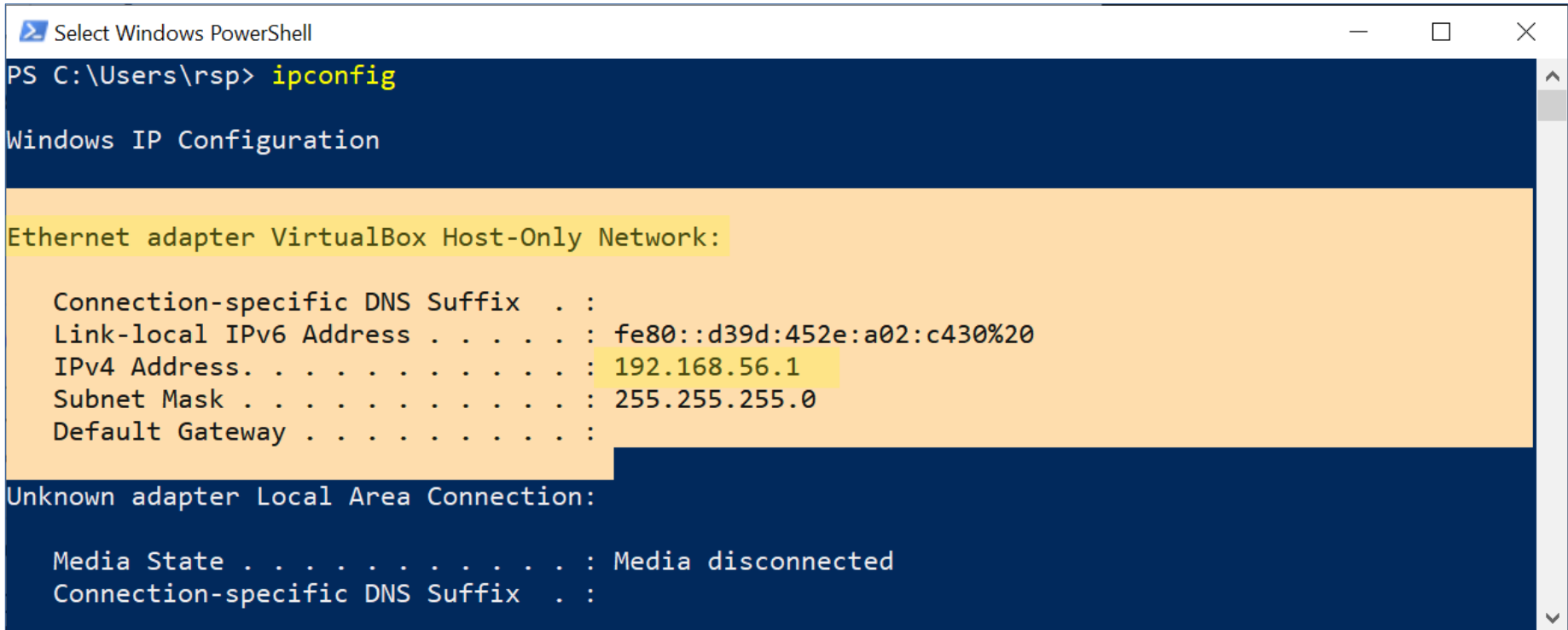
The screenshot shows the Oracle VM VirtualBox settings window for an Ubuntu Server 22.04 LTS VM. The 'Network' tab is selected in the left sidebar. The 'Network' section is expanded to show 'Adapter 1'. A 'Port Forwarding Rules' dialog box is open, displaying a table with one rule for SSH.

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
ssh	TCP		2222		22

Enable port forwarding to allow SSH by forwarding the host port (e.g. 2222) to the guest port 22.

OK Cancel

# SSH from Windows PowerShell to Ubuntu VM

A screenshot of a Windows PowerShell terminal window titled "Select Windows PowerShell". The terminal shows the command "ipconfig" being executed. The output displays network configuration for two adapters. The first adapter is "Ethernet adapter VirtualBox Host-Only Network:", which is highlighted in yellow. Its IPv4 address is "192.168.56.1", also highlighted in yellow. The second adapter is "Unknown adapter Local Area Connection:", which is not highlighted. Its media state is "Media disconnected".

```
PS C:\Users\rsp> ipconfig

Windows IP Configuration

Ethernet adapter VirtualBox Host-Only Network:

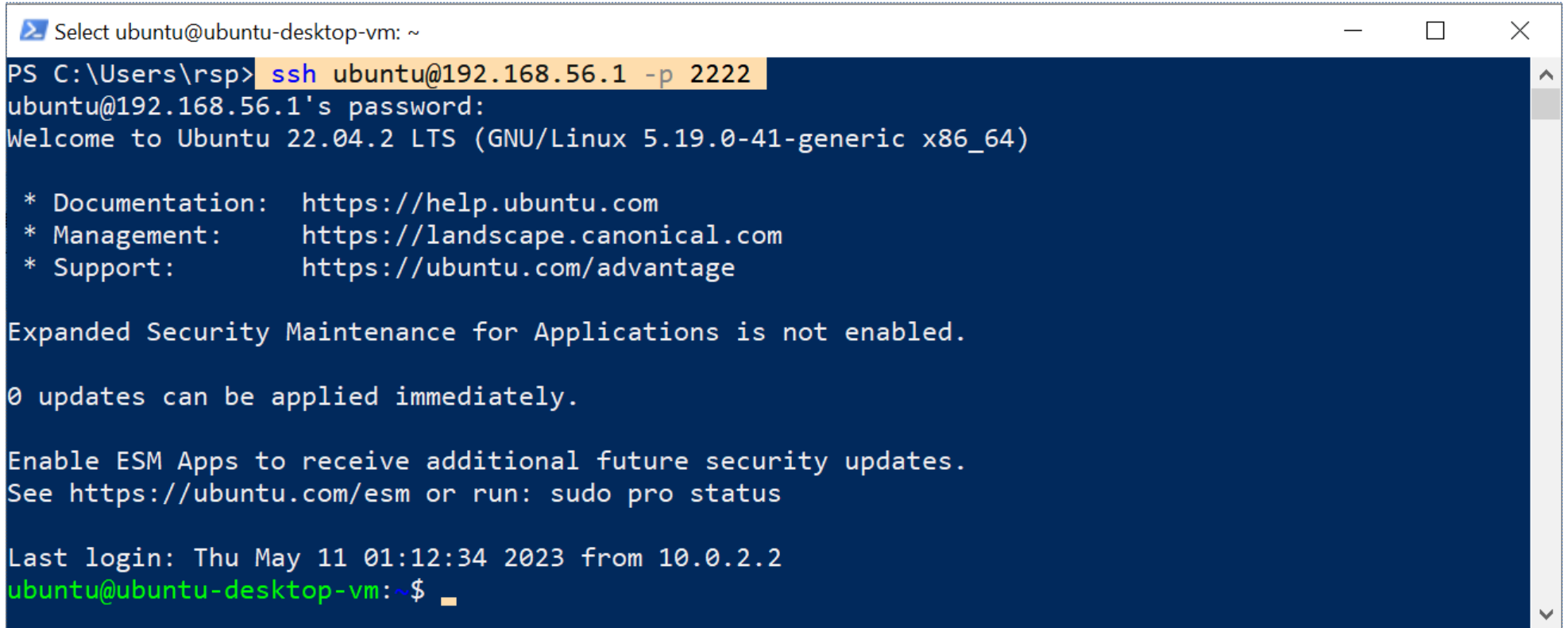
    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::d39d:452e:a02:c430%20
    IPv4 Address. . . . . : 192.168.56.1
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 

Unknown adapter Local Area Connection:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :
```

Open the Windows PowerShell and run the `ipconfig` command.  
Search for the IP address of the **Ethernet adapter for Virtual-Box (Host-Only)**.

# SSH from Windows to Ubuntu VM (Guest OS)

A screenshot of a Windows terminal window titled "Select ubuntu@ubuntu-desktop-vm: ~". The terminal shows a Windows command prompt with the command "ssh ubuntu@192.168.56.1 -p 2222" entered. The terminal then displays the Ubuntu login prompt, the password prompt, and the Ubuntu login banner. The banner includes the Ubuntu version (22.04.2 LTS), the kernel version (5.19.0-41-generic x86\_64), and links for documentation, management, and support. It also displays a message about Expanded Security Maintenance for Applications (ESM) not being enabled and that 0 updates can be applied immediately. The terminal ends with the prompt "ubuntu@ubuntu-desktop-vm:~\$".

```
Select ubuntu@ubuntu-desktop-vm: ~
PS C:\Users\rsp> ssh ubuntu@192.168.56.1 -p 2222
ubuntu@192.168.56.1's password:
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.19.0-41-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

Expanded Security Maintenance for Applications is not enabled.

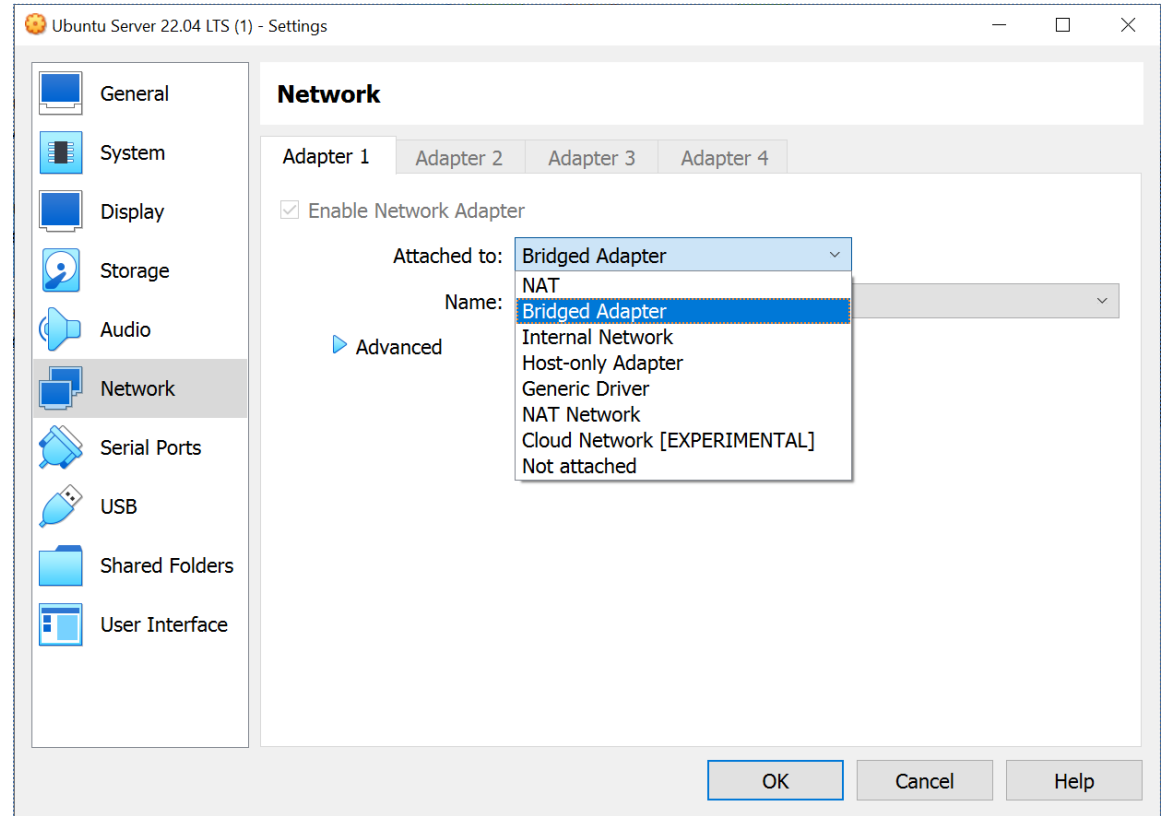
0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Last login: Thu May 11 01:12:34 2023 from 10.0.2.2
ubuntu@ubuntu-desktop-vm:~$
```

# VirtualBox Networking Modes

- Host-only Adapter
- Bridged Adapter
- NAT Network
- Internal Network





# Host-only Networking

- This mode allows communication between VMs and the host machine while isolating them from the external network.
  - **Communication between host and VMs:** The VMs can communicate with the host machine and with each other through the Host-only Adapter.
  - **Isolation from external network:** The VMs are isolated from the external network, such as the Internet or other physical computers.
- A virtual network interface, known as the "**Host-only Adapter**", is created on the host machine.

# NAT Networking

- In **NAT (Network Address Translation)** mode, VirtualBox acts as a router between VMs and the external network.
- This allows VMs to access the Internet while maintaining isolation from the host machine and other physical computers on the local network.

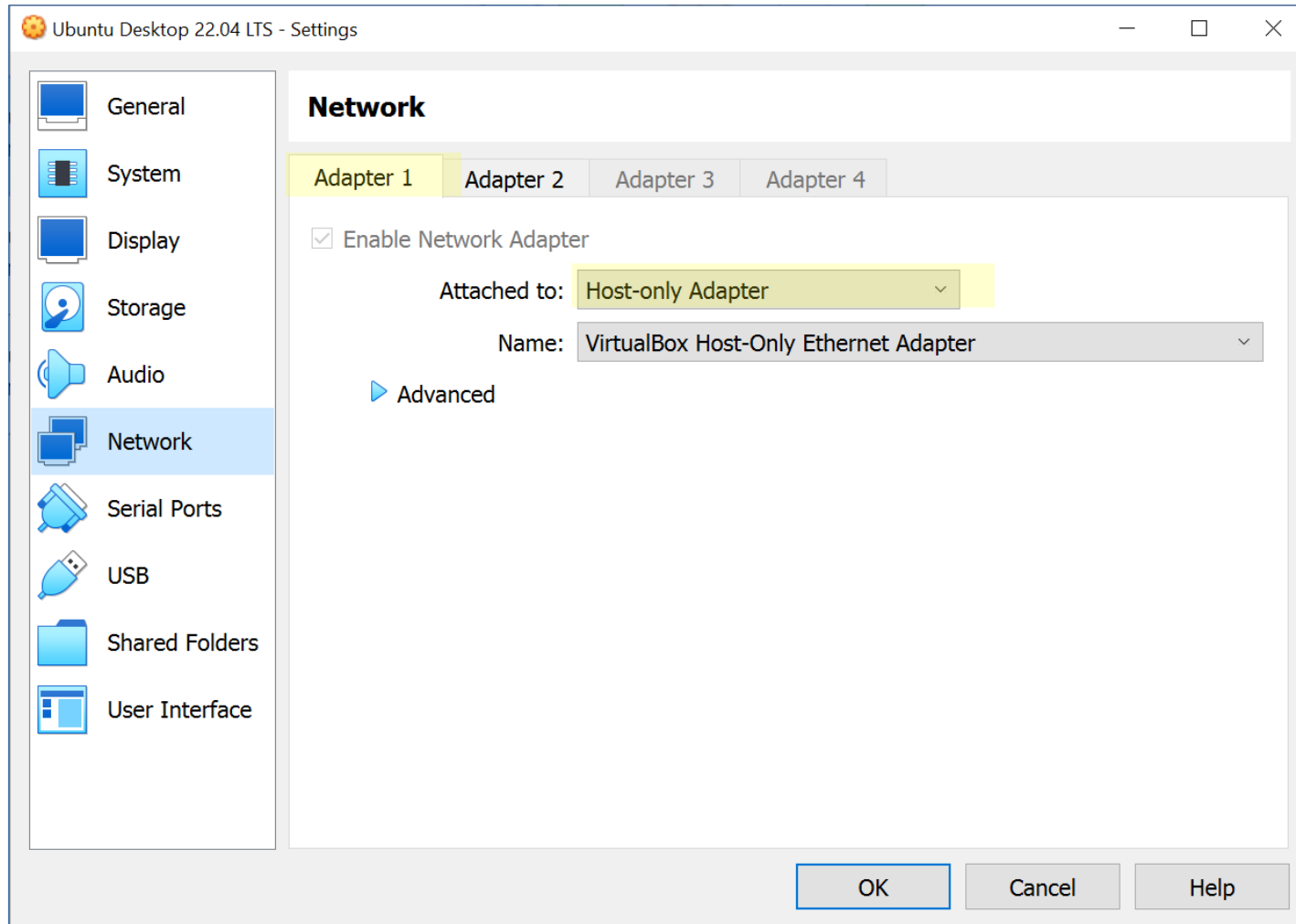
# Internal Networking

- **Isolated Virtual Network:** The VMs connected to this internal network can communicate with each other but remain isolated from the external network and other physical machines.
- **Inter-VM Communication:** VMs can communicate with each other over the internal network.
- **No Access to External Network:** The VMs do not have direct access to the external network or the internet.

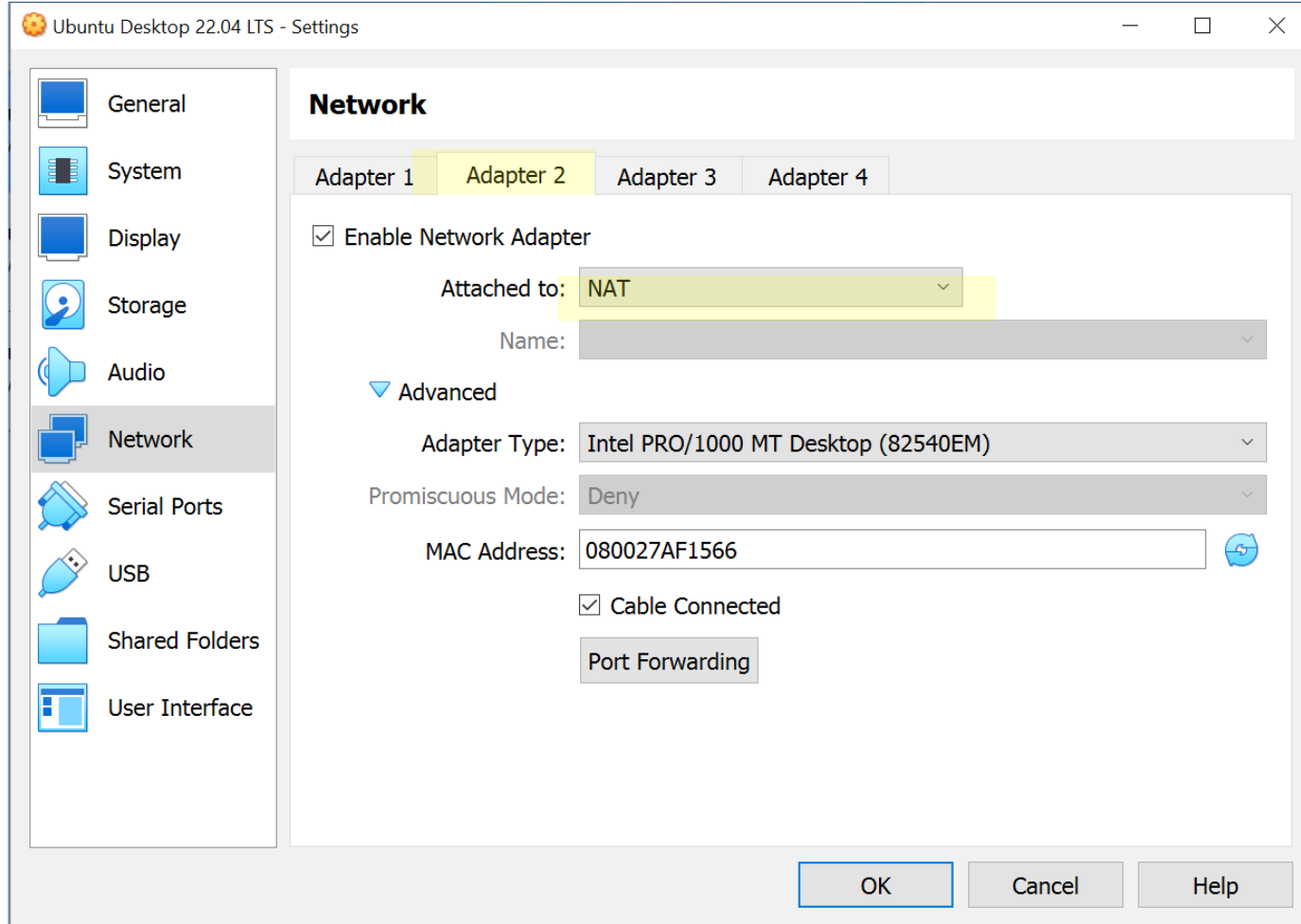
# Host-only Adapter with NAT

- It is possible to create two network adapters and add them to a VM.
  - The **Host-only Adapter** will provide communication within the private network.
  - The **NAT adapter** will handle internet connectivity.
- Ubuntu VM: Use Linux commands to show the network adapters and IP addresses:
  - ```
$ ip a
```
  - Two network adapters such as `enp0s3` and `enp0s8`.

# Use Host-only Adapter with NAT



# Use Host-only Adapter with NAT



```
ubuntu@ubuntu-desktop-vm: ~  
ubuntu@ubuntu-desktop-vm:~$ ip a  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000  
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
    inet 127.0.0.1/8 scope host lo  
        valid_lft forever preferred_lft forever  
    inet6 ::1/128 scope host  
        valid_lft forever preferred_lft forever  
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000  
    link/ether 08:00:27:7e:b9:96 brd ff:ff:ff:ff:ff:ff  
    inet 192.168.56.101/24 brd 192.168.56.255 scope global dynamic noprefixroute enp0s3  
        valid_lft 509sec preferred_lft 509sec  
    inet6 fe80::de06:2c46:4f8d:a3a/64 scope link noprefixroute  
        valid_lft forever preferred_lft forever  
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000  
    link/ether 08:00:27:af:15:66 brd ff:ff:ff:ff:ff:ff  
    inet 10.0.3.15/24 brd 10.0.3.255 scope global dynamic noprefixroute enp0s8  
        valid_lft 86009sec preferred_lft 86009sec  
    inet6 fe80::4206:9a4d:5181:1e6e/64 scope link noprefixroute  
        valid_lft forever preferred_lft forever  
ubuntu@ubuntu-desktop-vm:~$
```

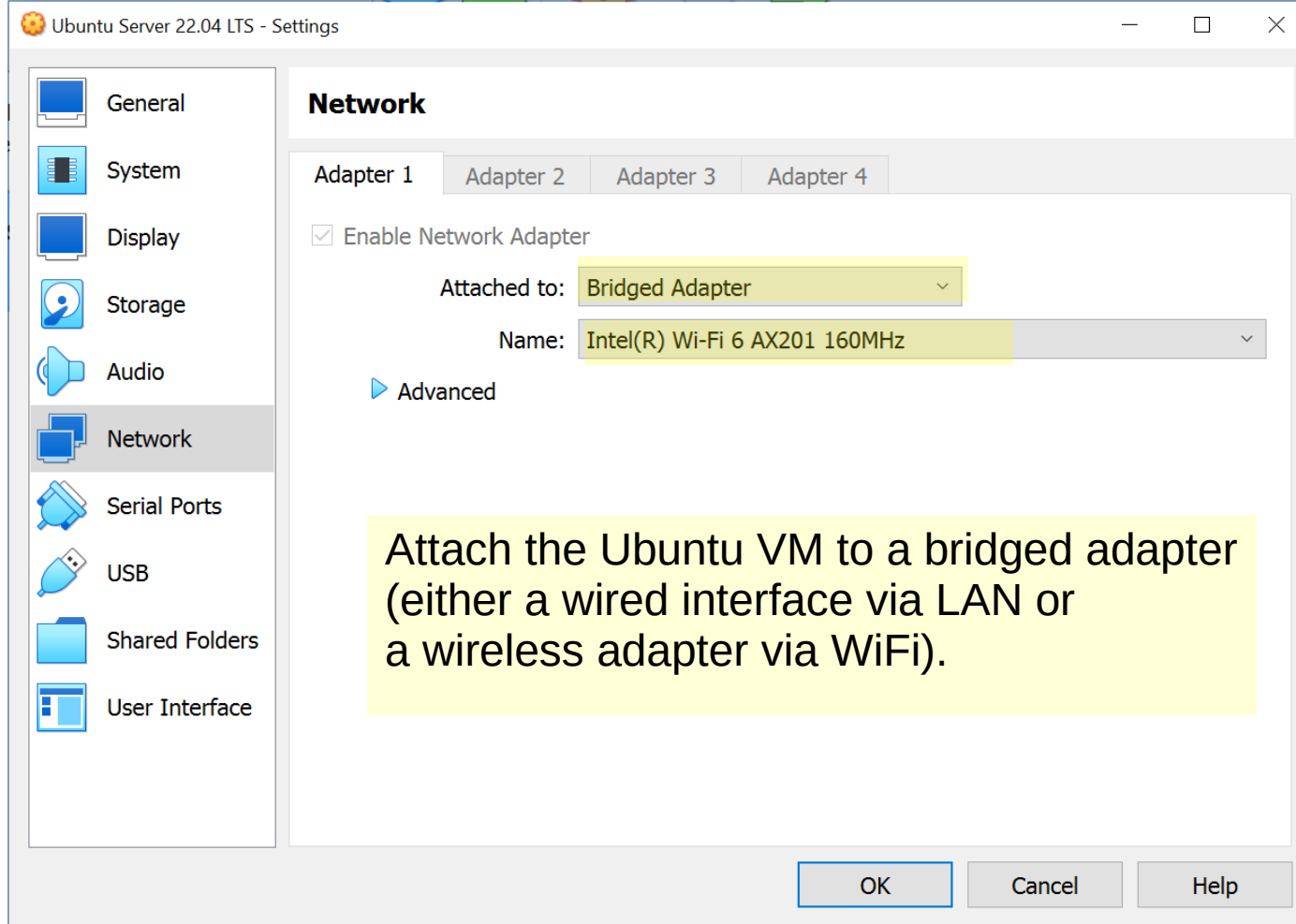
```
ubuntu@ubuntu-desktop-vm: ~  
ubuntu@ubuntu-desktop-vm:~$ ip route  
default via 10.0.3.2 dev enp0s8 proto dhcp metric 101  
10.0.3.0/24 dev enp0s8 proto kernel scope link src 10.0.3.15 metric 101  
169.254.0.0/16 dev enp0s3 scope link metric 1000  
192.168.56.0/24 dev enp0s3 proto kernel scope link src 192.168.56.101 metric 100  
ubuntu@ubuntu-desktop-vm:~$ ping -c 3 8.8.8.8  
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.  
64 bytes from 8.8.8.8: icmp_seq=1 ttl=53 time=134 ms  
64 bytes from 8.8.8.8: icmp_seq=2 ttl=53 time=70.1 ms  
64 bytes from 8.8.8.8: icmp_seq=3 ttl=53 time=64.9 ms  
  
--- 8.8.8.8 ping statistics ---  
3 packets transmitted, 3 received, 0% packet loss, time 2008ms  
rtt min/avg/max/mdev = 64.949/89.742/134.221/31.520 ms  
ubuntu@ubuntu-desktop-vm:~$
```



# Bridged Networking

- With bridged networking, VirtualBox uses a device driver on the host system that filters data from the physical network adapter (wired or wireless).
- This enables the VirtualBox to intercept data from the physical network and inject data into it, effectively creating a software-based network interface.
- Data can be sent from the Host to the virtual machine using this interface.

# To enable Bridged networking mode in VirtualBox



Ubuntu Server 22.04 LTS (1) - Settings

**Network**

Adapter 1 | Adapter 2 | Adapter 3 | Adapter 4

Enable Network Adapter

Attached to: Bridged Adapter

Name: Intel(R) Wi-Fi 6 AX201 160MHz

Advanced

Adapter Type: Intel PRO/1000 MT Desktop (82540EM)

Promiscuous Mode: Allow All

MAC Address: 080027877C80

Cable Connected

OK Cancel Help

# Bridged Networking Mode

- Bridged mode has several advantages:
  - Virtual machines can be easily accessed over a LAN without the need for NAT or Port Forwarding configuration.
  - In Bridged mode, the VM will receive its own IP address from the DHCP server. This makes Bridged mode a suitable option for production environments.

# Bridged Networking Mode

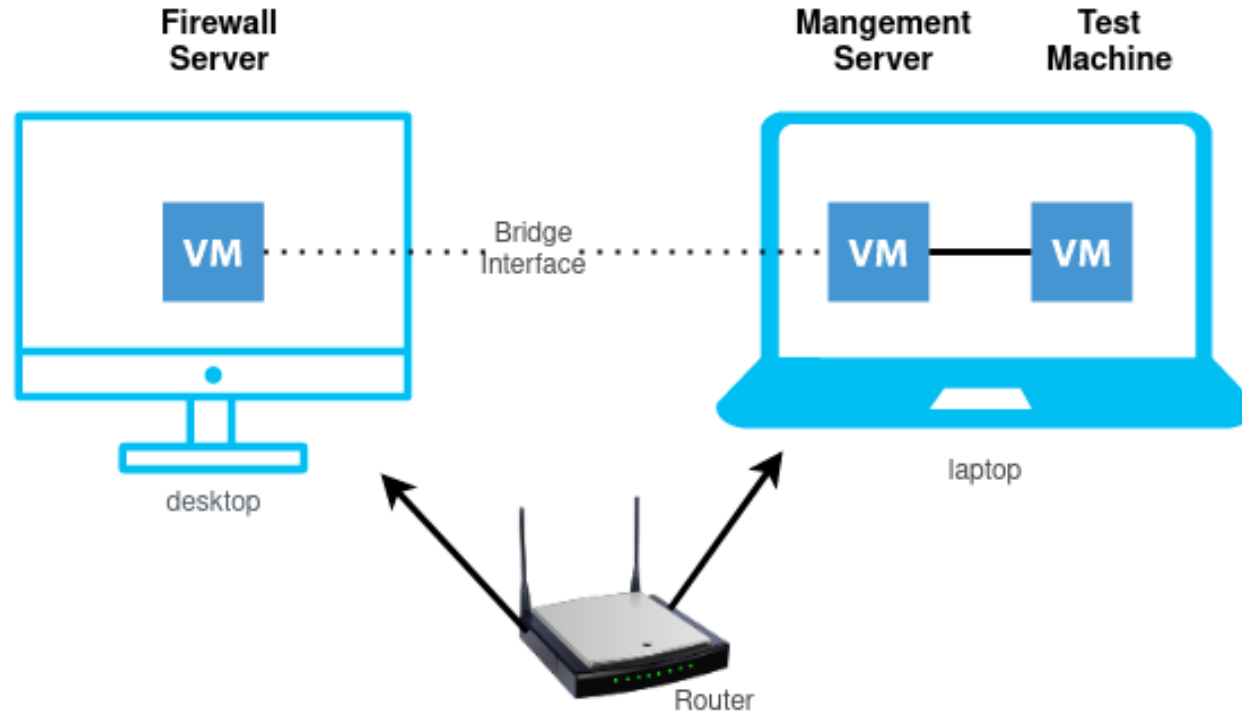


Image source: <https://linuxhint.com/use-virtualbox-bridged-adapter/>

# Bridged Networking Mode

- Disadvantages when using Bridged mode:
  - If too many virtual machines or devices are connected to the network, the DHCP server may run out of IP addresses, or at least not be able to allocate IP addresses.

# SSH Authentication Methods

- **Password authentication (default method):**  
The user provides his / her username and password to authenticate to the remote server.
- **Public key authentication:** This method uses public key cryptography to authenticate the user on the remote server. The client's public key file must be copied to the remote SSH server.

# SSH Authentication

- **Linux environments** commonly use **public-key and private-key pairs** to drive authentication that doesn't require the use of passwords.
- **OpenSSH** includes open source tools to help support **key-based authentication**, specifically:
  - **ssh-keygen** for generating secure keys
  - **ssh-agent** and **ssh-add** for securely storing private keys
  - **scp** and **sftp** to securely copy public key files during initial use of a server



# SSH Authentication Methods

## Ubuntu:

1) Create a new SSH key pair:

```
$ ssh-keygen -t rsa -b 4096
```

2) Set the permissions on the private key file:

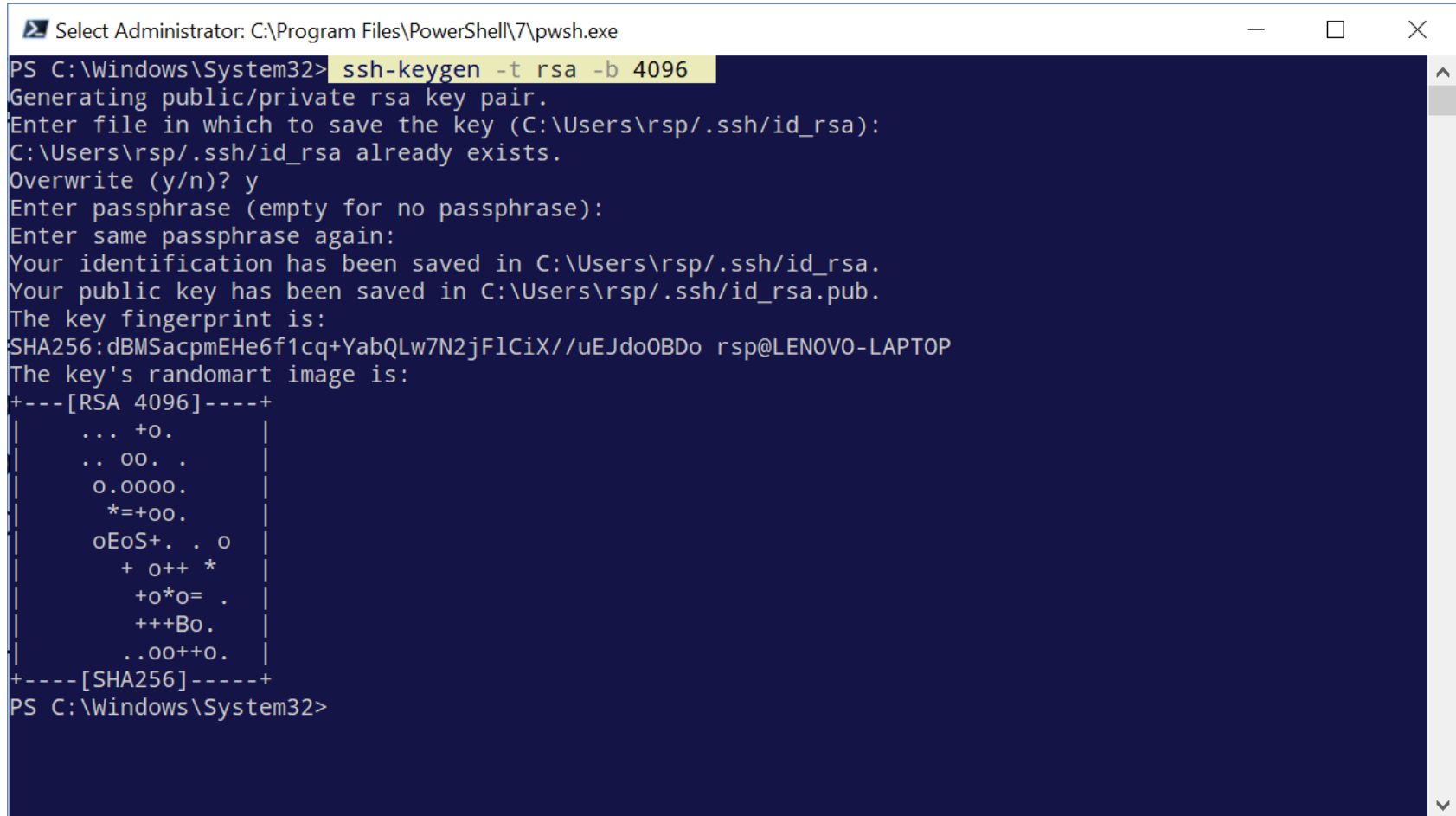
```
$ chmod 600 $HOME/.ssh/id_rsa.pub
```

3) Copy the public key to the remote SSH server:

```
$ ssh-copy-id <username@remote_server>
```

Now you should be able to log in **from Ubuntu to the remote SSH server** without entering a password.

# Create Public Key / Private Key File for Windows.



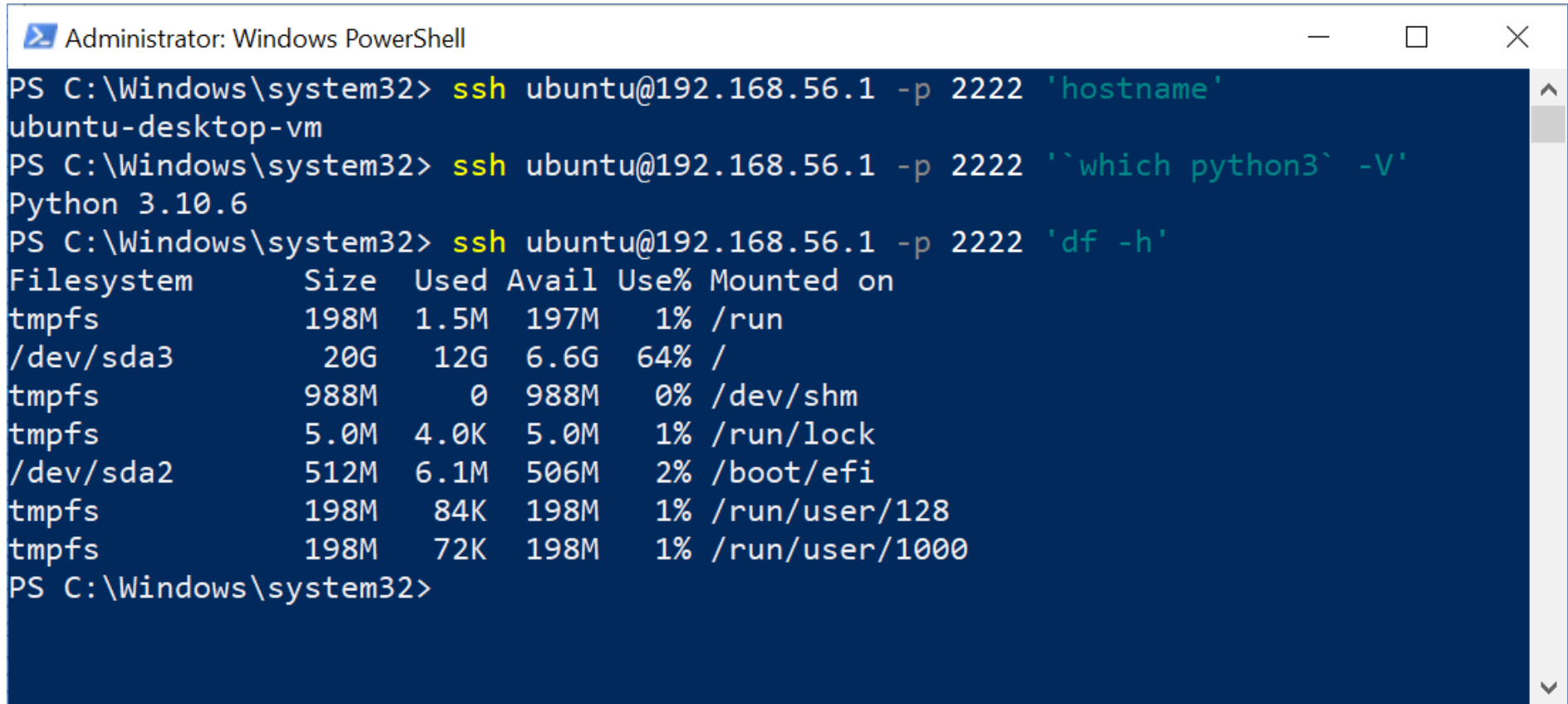
```
Select Administrator: C:\Program Files\PowerShell\7\pwsh.exe
PS C:\Windows\System32> ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\rsp/.ssh/id_rsa):
C:\Users\rsp/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\rsp/.ssh/id_rsa.
Your public key has been saved in C:\Users\rsp/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:dbMSacpmEHe6f1cq+YabQLw7N2jF1CiX//uEJdoOBDo rsp@LENOVO-LAPTOP
The key's randomart image is:
+---[RSA 4096]-----+
|
|... +o.
|.. oo. .
|o.oooo.
| *+=+oo.
|oEoS+. . o
| + o++ *
| +o*o= .
| +++Bo.
|..oo++o.
|-----[SHA256]-----+
PS C:\Windows\System32>
```

## Copy the Public Key File from Windows PowerShell to Ubuntu VM.

```
ubuntu@ubuntu-desktop-vm: ~  
PS C:\Windows\system32> type $env:USERPROFILE\.ssh\id_rsa.pub `^  
>> | ssh ubuntu@192.168.56.1 -p 2222 "cat >> .ssh/authorized_keys"  
PS C:\Windows\system32> ssh ubuntu@192.168.56.1 -p 2222  
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.19.0-41-generic x86_64)  
  
* Documentation:  https://help.ubuntu.com  
* Management:    https://landscape.canonical.com  
* Support:       https://ubuntu.com/advantage  
  
Expanded Security Maintenance for Applications is not enabled.  
  
0 updates can be applied immediately.  
  
Enable ESM Apps to receive additional future security updates.  
See https://ubuntu.com/esm or run: sudo pro status  
  
Last login: Thu May 11 21:55:44 2023 from 10.0.2.2  
ubuntu@ubuntu-desktop-vm:~$
```

```
type $env:USERPROFILE\.ssh\id_rsa.pub `^  
| ssh ubuntu@192.168.56.1 -p 2222 "cat >> .ssh/authorized_keys"
```

# Remote execution of commands in Ubuntu VM using SSH



```
Administrator: Windows PowerShell
PS C:\Windows\system32> ssh ubuntu@192.168.56.1 -p 2222 'hostname'
ubuntu-desktop-vm
PS C:\Windows\system32> ssh ubuntu@192.168.56.1 -p 2222 '`which python3` -V'
Python 3.10.6
PS C:\Windows\system32> ssh ubuntu@192.168.56.1 -p 2222 'df -h'
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           198M  1.5M  197M   1% /run
/dev/sda3       20G   12G   6.6G  64% /
tmpfs           988M    0   988M   0% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
/dev/sda2       512M  6.1M  506M   2% /boot/efi
tmpfs           198M   84K  198M   1% /run/user/128
tmpfs           198M   72K  198M   1% /run/user/1000
PS C:\Windows\system32>
```

Now you should be able to log in **from Windows PowerShell to Ubuntu VM** without entering a password.

# Assignments

- 1) Download the image file (.iso) for **Ubuntu Server** and use the **22.04.x LTS** version.

Download site: <https://ubuntu.com/download/server>.

- 2) Use the VirtualBox VM Manager to create a Ubuntu Server VM (with minimal installation).
  - Network Setting: **Host-Only Adapter + NAT**.
  - Start the VM in Normal mode (non-headless).
  - Login into the system via the Linux console.

## 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:  

Folder:


ISO Image:

Edition:

Type:

Version:

Skip Unattended Installation

 You have selected to skip unattended guest OS install, the guest OS will need to be installed manually.

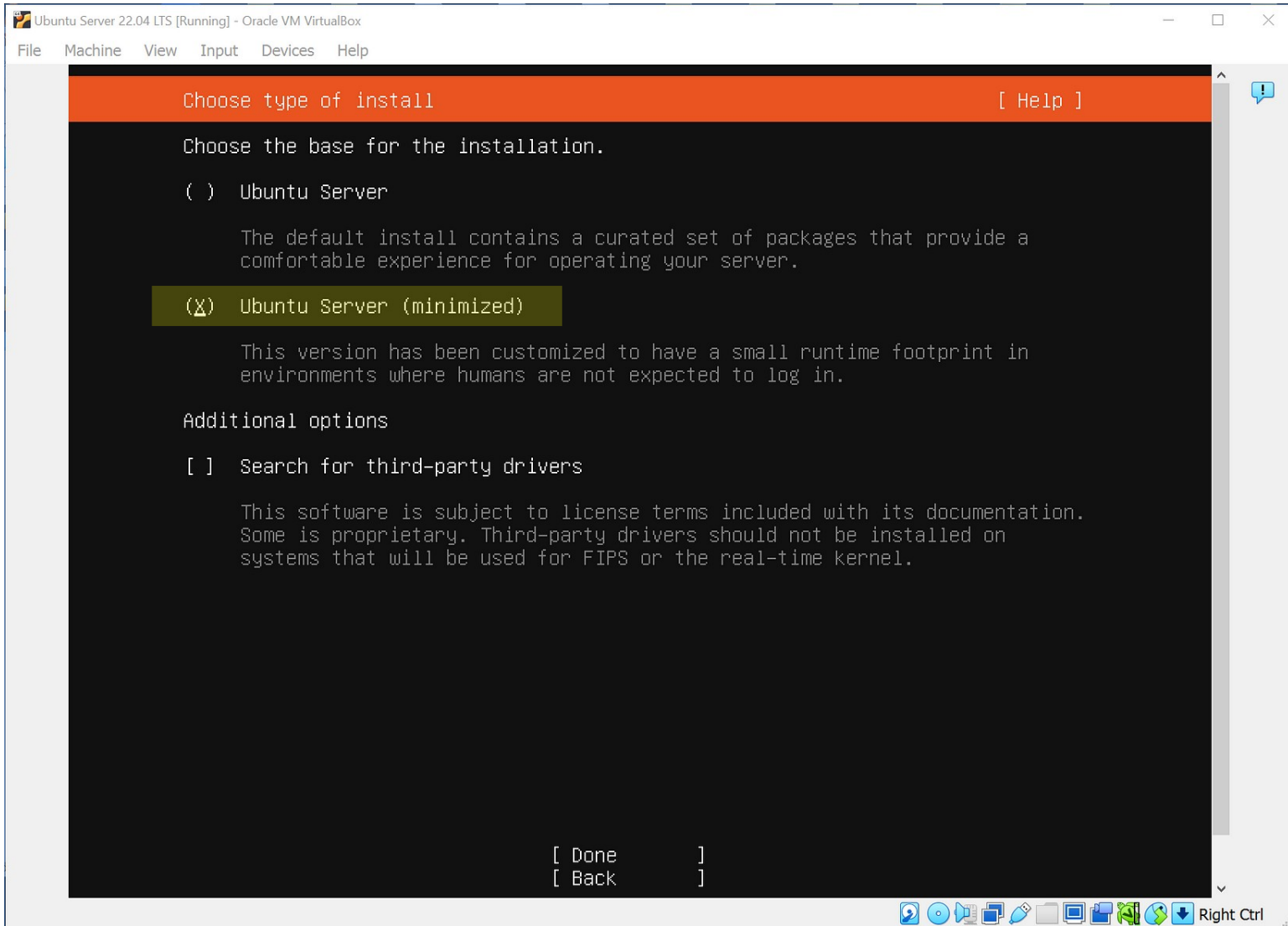
Help

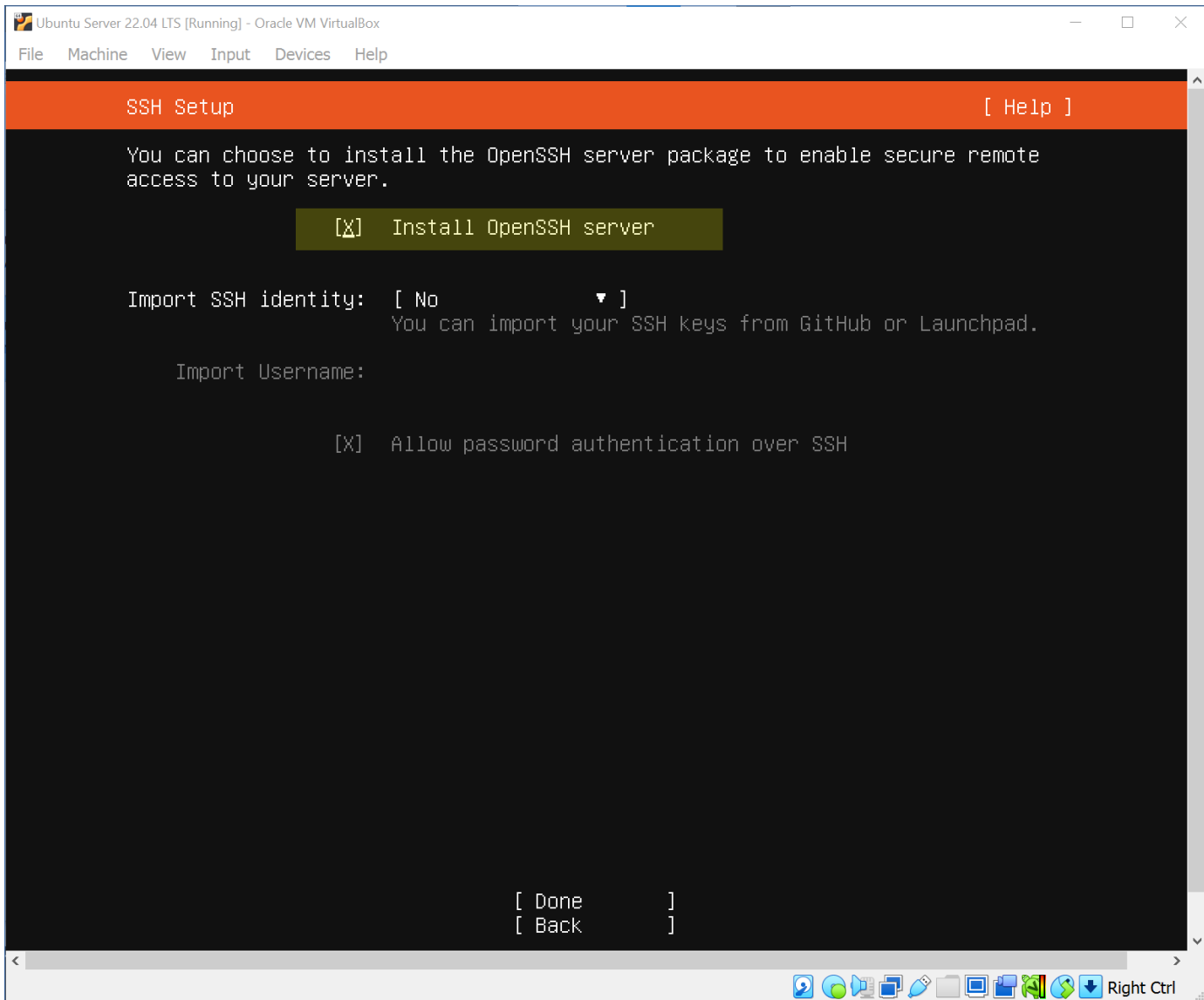
Expert Mode

Back

Next

Cancel







# Tasks: Ubuntu Server 1

- Request an IP from the DHCP server for the enp0s8 interface.

```
$ sudo dhclient -1 enp0s8
```

- Check network interfaces:

```
$ ip link
```

- Check the IP address of the machine:

```
$ ip a
```

- Check routing tables:

```
$ ip route
```

# Tasks: Ubuntu Server 1

- Check the Internet connectivity using the ping command.

```
$ ping 8.8.8.8 -c 5
```

```
$ ping google.com -c 5
```

- Update Ubuntu software packages.

```
$ sudo apt update
```

```
$ sudo apt upgrade -y && sudo apt dist-upgrade
```

- Install additional packages (if not already installed).

```
$ sudo apt install nano less wget curl iputils-ping
```

# Tasks: Ubuntu Server 1

```
$ sudo nano /etc/netplan/00-installer-config.yaml
```

```
# This is a network configuration file for netplan.  
network:  
  ethernets:  
    enp0s3:  
      dhcp4: true  
    enp0s8:  
      dhcp4: true  
  version: 2
```

Enable the DHCP client to request IP addresses for both network interfaces.

# Tasks: Ubuntu Server 1

- Change the host name of the server to "ubuntu-server-vm1".
- Edit the files using nano: "/etc/hostname" and "/etc/hosts".
- Check the SSH server status (it should be active/running).  

```
$ systemctl status ssh
```
- Install the avahi-daemon service for mDNS and check its status.  

```
$ sudo apt install avahi-daemon
```

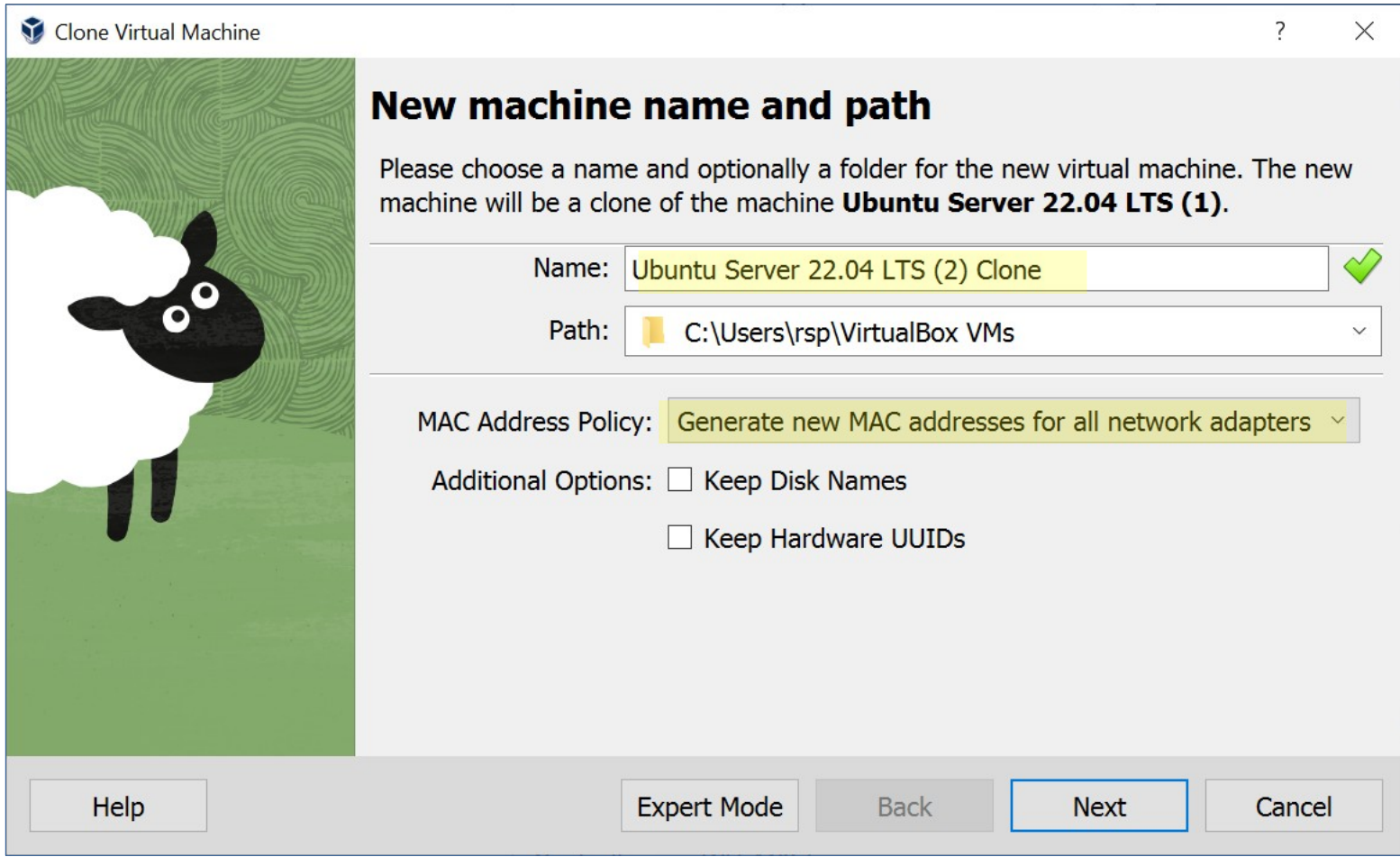
```
$ systemctl status avahi-daemon
```
- Use the SSH client (in Windows PowerShell terminal) to connect to the server.

# Tasks: Ubuntu Server 2

- Create a new VM by full-cloning the first Ubuntu Server VM.
  - Don't forget to re-generate a new MAC address.
- Start the VM in Normal mode (non-headless).
- Login into the system via the Linux console.
- Change the host name of the server to "ubuntu-server-vm2" by using the `hostnamectl` command.

```
$ sudo hostnamectl set-hostname "ubuntu-server-vm2"
```

```
$ hostname
```



# Tasks: Ubuntu Server 2

- Recreate the machine ID hexstring in the following files:  
"/etc/machine-id" and "/var/lib/dbus/machine-id".

```
$ sudo rm -f /etc/machine-id
```

```
$ sudo dbus-uuidgen --ensure=/etc/machine-id
```

```
$ sudo rm /var/lib/dbus/machine-id
```

```
$ sudo dbus-uuidgen --ensure
```

- Reboot the server.

```
$ sudo reboot
```

# Tasks: Ubuntu Server 2

- Use the SSH client (in Windows PowerShell terminal) to connect to the second Ubuntu server.
- Check the IP address of the server.

```
$ ip a
```

- Ping the first server by specifying its hostname.

```
$ ping "ubunt-server-vm1.local" -c 5
```

- Note: The first Ubuntu Server VM must be running.



# Task Ubuntu Server 2

- Check the OS release.

```
$ cat /etc/os-release | head -n 4
```

- Check the current disk usage.

```
$ df -h
```

- Check the current memory usage (in MB).

```
$ free -m
```

- Check the Linux kernel version:

```
$ uname -pro
```

# VS Code IDE

- **Visual Studio Code** (also called **VS Code**) is an **open-source IDE** developed by Microsoft.
- It is designed to be lightweight and extensible, while also providing powerful features for coding and debugging.
- VS Code supports a wide range of programming languages and frameworks, including JavaScript, Python, C++, and many others.

<https://github.com/microsoft/vscode>

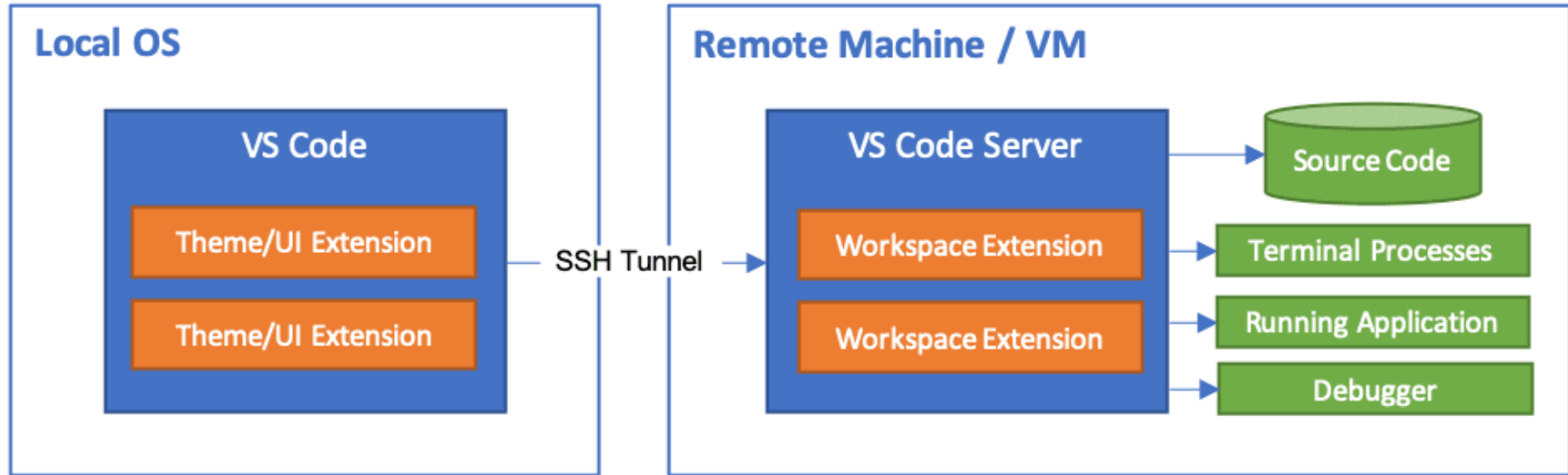
# Remote Development

- The **VS Code Remote Development** feature allows developers to work on their code in a **remote environment**, such as a virtual machine (e.g. **WSL2**), a docker container, or a remote server (via SSH).
- This allows developers to use their **local VS Code editor** to write & debug code on a **remote system**, without having to switch between different tools or environments.

# Remote Development

- The **VS Code Remote Development Pack** is a collection of extensions and tools for the VS Code IDE that enable developers to use the **Remote Development feature** in VS Code.
- It includes the **Remote Development Extension**, which provides the core functionality for connecting to and working in remote environments, as well as several other extensions.

# Visual Studio Code Remote Development using SSH



<https://code.visualstudio.com/docs/remote/remote-overview>

<https://code.visualstudio.com/docs/remote/ssh>

# Installation of Remote Development Pack in VS Code

The screenshot displays the Visual Studio Code interface with the Remote Development extension pack installed. The left sidebar shows the Extensions view with a search for 'remote development'. The main area shows the details for the 'Remote Development' extension pack by Microsoft, version v0.24.0 (Preview). The extension is enabled globally, and the 'Visual Studio Code Remote' logo is visible at the bottom. The status bar at the bottom indicates 'Restricted Mode'.

EXTENSIONS: ... remote development

- Remote - SSH**  
Open any folder on a rem...  
Reload Required
- Remote Explorer**  
View remote machines fo...  
Microsoft
- Remote Development**  
An extension pack that le...  
Microsoft
- Remote - SSH: Editing ...**  
Edit SSH configuration files  
M... Reload Required
- Remote - Tunnels**  
Connect to a remote mac...  
Microsoft
- Azure Mac...** 1.5M ★ 3.5  
This extension is used by ...  
Microsoft **Install**
- Remote Rep...** 884K ★ 5

Extension: Remote Development - Visual Studio Code

Extension: Remote Development X

## Remote Development

 v0.24.0 **Preview**  
Microsoft microsoft.com | 3,948,329 | ★★★★★ (107)  
An extension pack that lets you open any folder in a container, on a re...  
**Disable** **Uninstall** ⚙️  
This extension is enabled globally.

DETAILS | **RUNTIME STATUS**

### Extension Pack (4)

- WSL** 25ms  
Open any folder in the Windows Subsystem ...  
Microsoft
- Dev Containers** 79ms  
Open any folder or repository inside a Docke...  
Microsoft

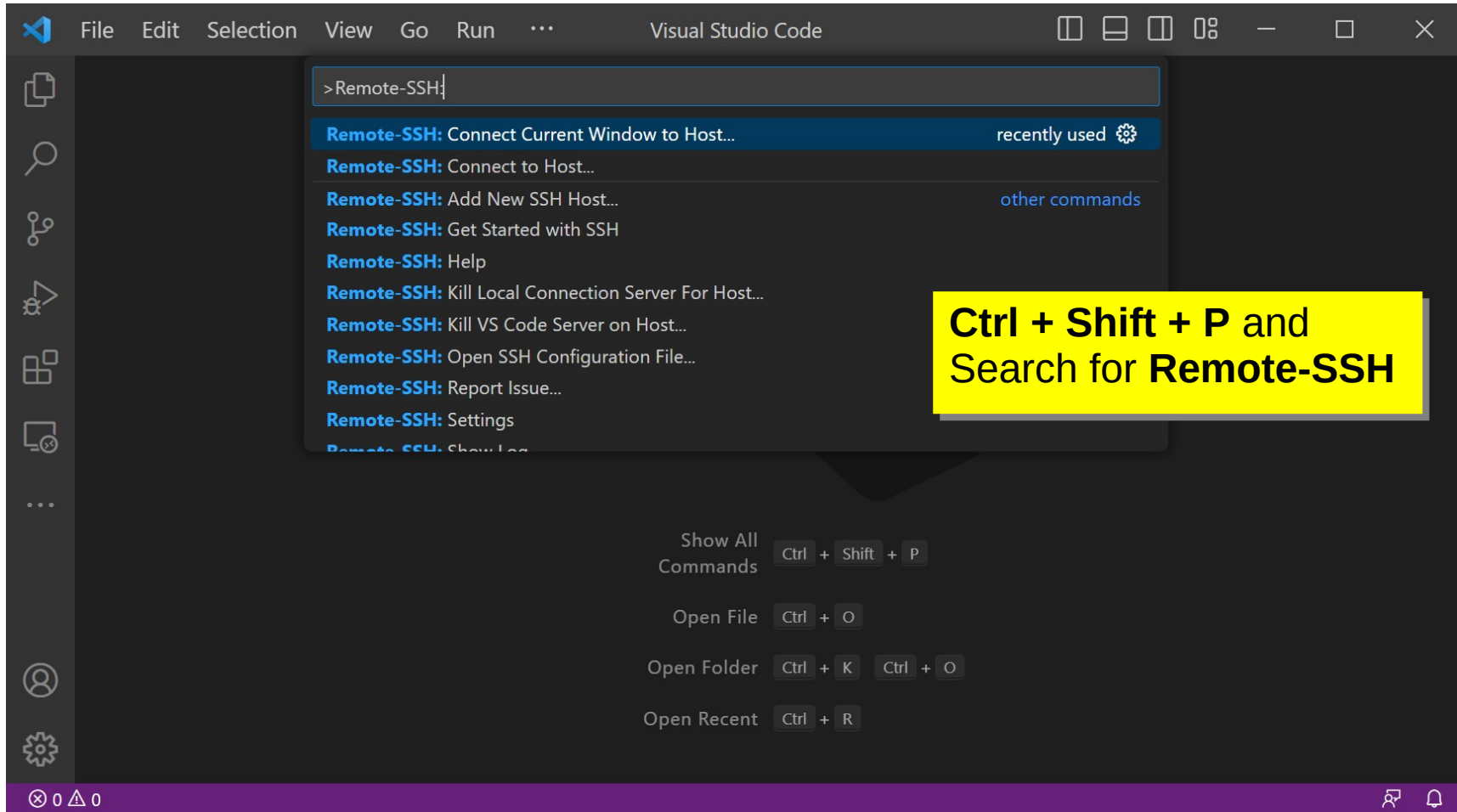
Categories  
Other  
Extension Packs

Extension Resources  
Marketplace  
Repository  
License  
Microsoft

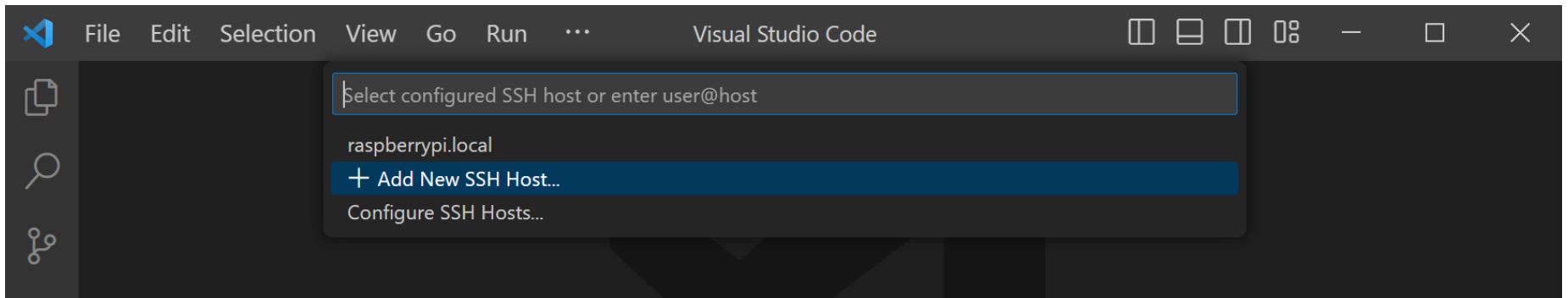
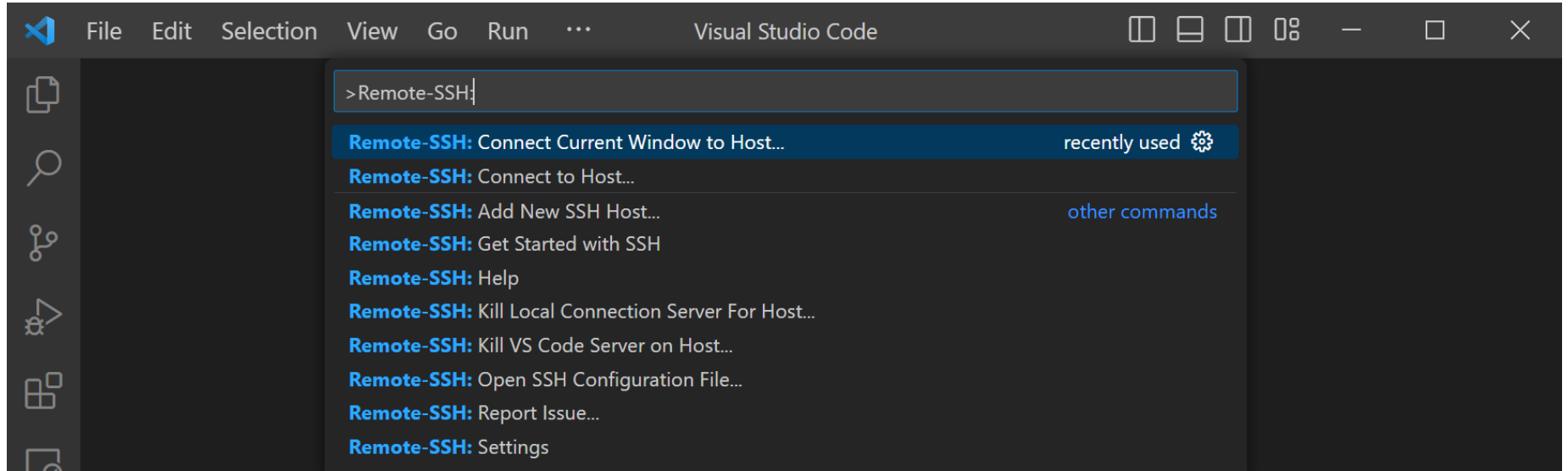
Visual Studio Code Remote

Restricted Mode 0 0 0

# Remote-SSH Commands

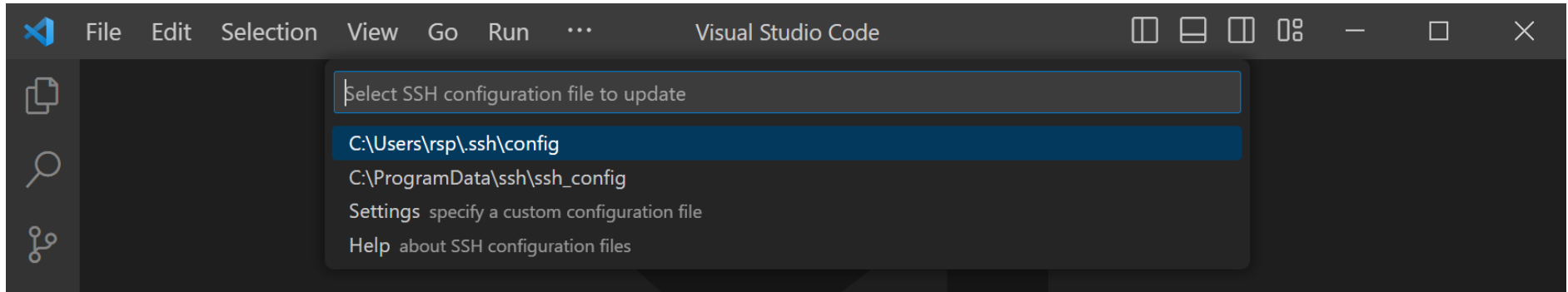
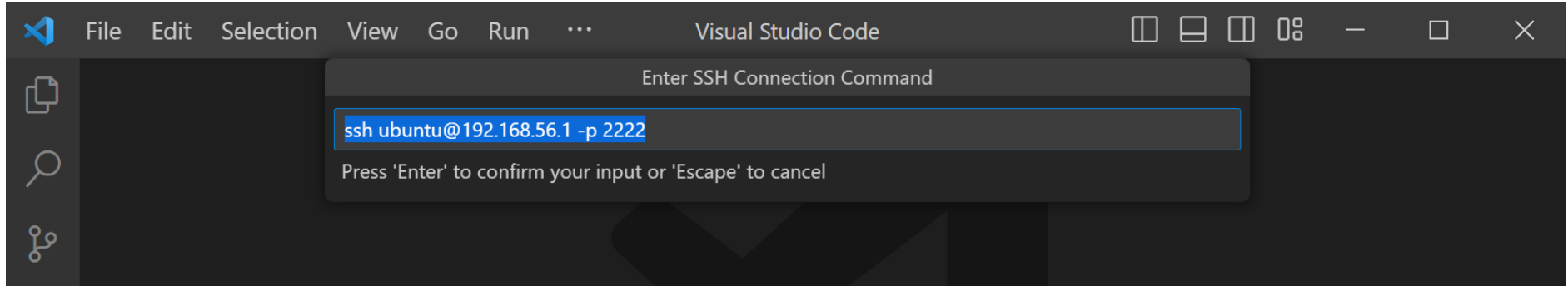


# Connect to a remote-SSH host.

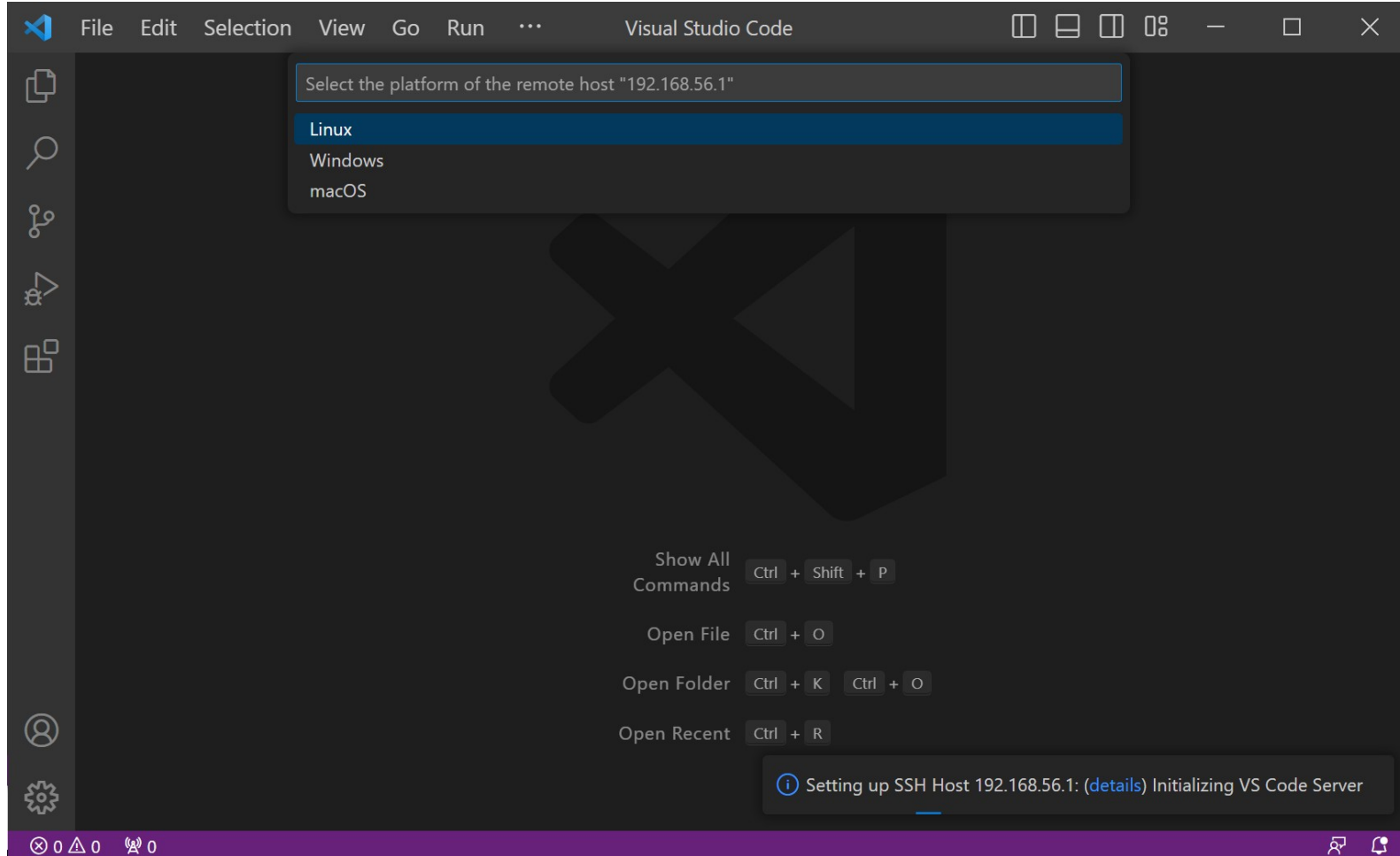




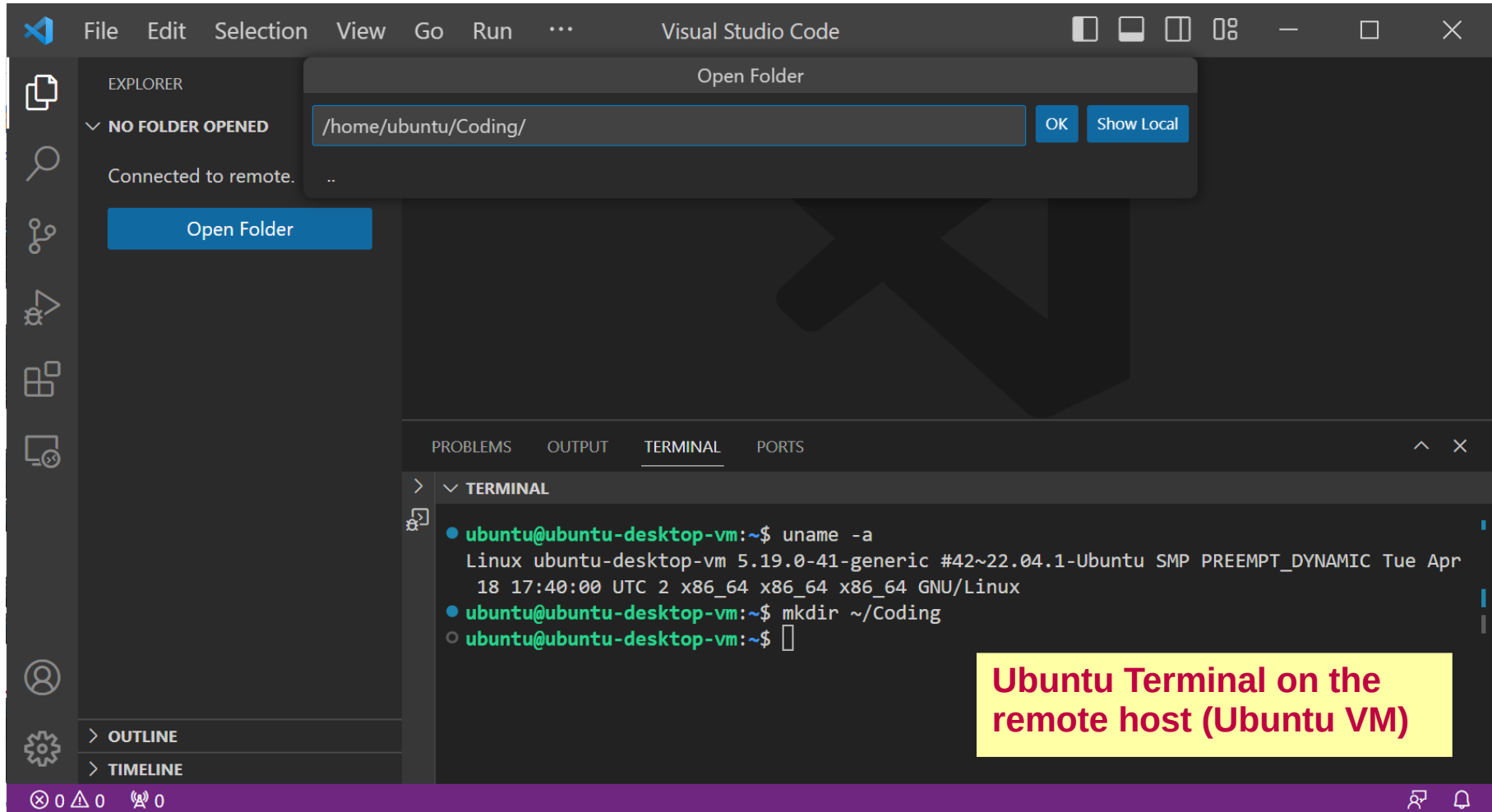
# Connect to a new SSH host (Ubuntu VM).



# Automatic Installation of VS Code Server on the host (Ubuntu VM)



# Open a folder on the remote host (Ubuntu VM).

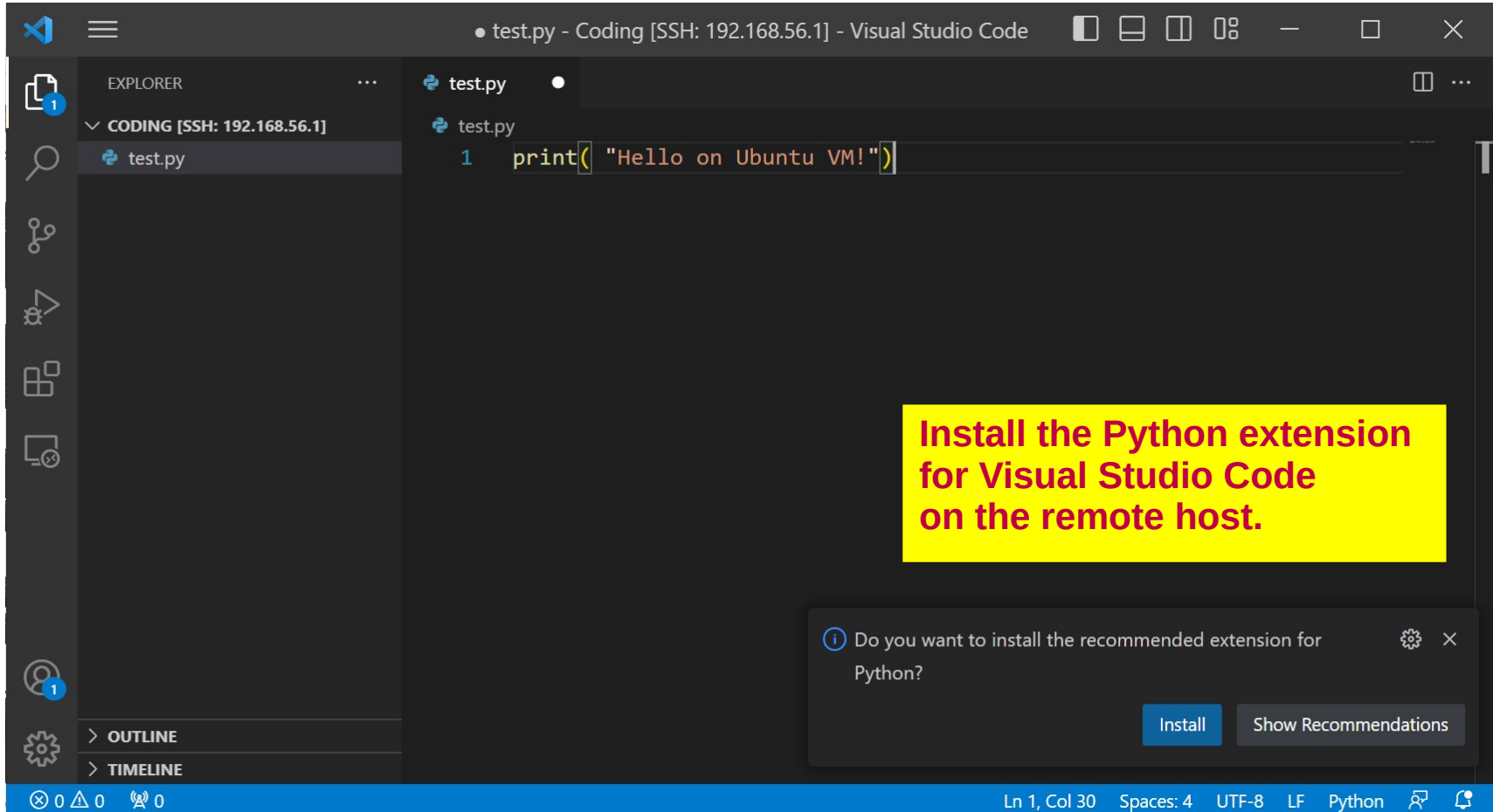


The screenshot displays the Visual Studio Code interface with a remote connection to an Ubuntu VM. The Explorer sidebar on the left shows "NO FOLDER OPENED" and "Connected to remote." with an "Open Folder" button. An "Open Folder" dialog box is open, showing the path `/home/ubuntu/Coding/` and buttons for "OK" and "Show Local". The Terminal window at the bottom shows the following commands and output:

```
ubuntu@ubuntu-desktop-vm:~$ uname -a
Linux ubuntu-desktop-vm 5.19.0-41-generic #42~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Tue Apr 18 17:40:00 UTC 2 x86_64 x86_64 x86_64 GNU/Linux
ubuntu@ubuntu-desktop-vm:~$ mkdir ~/Coding
ubuntu@ubuntu-desktop-vm:~$
```

A yellow callout box in the bottom right of the terminal area contains the text: **Ubuntu Terminal on the remote host (Ubuntu VM)**

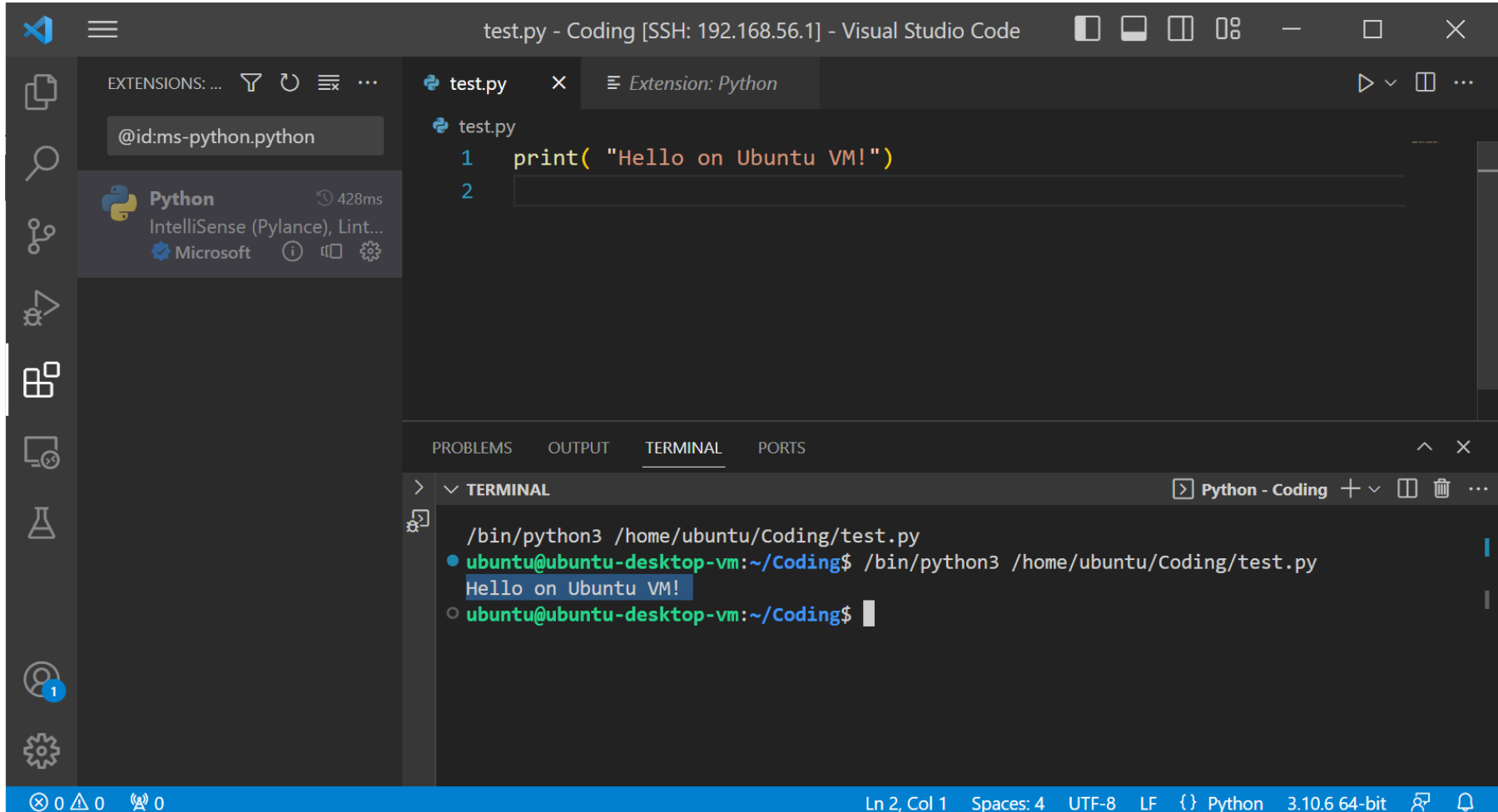
# Create a Python script file.



# Installation of the Python extension on the remote host (Ubuntu VM)

The screenshot displays the Visual Studio Code interface on a remote host. The top bar shows the window title: "Extension: Python - Coding [SSH: 192.168.56.1] - Visual Studio Code". The left sidebar contains the "EXTENSIONS: ..." view, where the search filter "@id:ms-python.python" is applied. The "Python" extension by Microsoft is highlighted, showing its icon and name. The main editor area displays the details for the "Python" extension (version v2023.6.1) by Microsoft. The extension is currently in the "Installing" state, as indicated by the blue "Installing" button. The extension description includes: "IntelliSense (Pylance), Linting, Debugging (multi-threaded, rem...". Below the description, there are tabs for "DETAILS", "FEATURE CONTRIBUTIONS", "CHANGELOG", and "EXTENSION PACK". The "Python extension for Visual Studio Code" section provides a detailed description: "A Visual Studio Code extension with rich support for the Python language (for all actively supported versions of the language: >=3.7), including features such as IntelliSense (Pylance), linting, debugging, code navigation, code formatting, refactoring, variable explorer, test explorer, and more!". On the right side, there is a "Categories" section with several tags: "Programming Languages", "Linters", "Debuggers", "Formatters", "Data Science", "Machine Learning", and "Notebooks". The bottom status bar shows the system tray with icons for window management and a notification bell.

# Execution of the Python script on the remote host (Ubuntu VM)



The screenshot displays the Visual Studio Code interface connected to a remote host via SSH. The main editor window shows a file named `test.py` with the following Python code:

```
1 print( "Hello on Ubuntu VM!")  
2
```

The left sidebar shows the Extensions view with the Python extension installed. The bottom panel shows the Terminal window with the following output:

```
> ✓ TERMINAL Python - Coding + ▾ □ □ □ ...  
/bin/python3 /home/ubuntu/Coding/test.py  
• ubuntu@ubuntu-desktop-vm:~/Coding$ /bin/python3 /home/ubuntu/Coding/test.py  
Hello on Ubuntu VM!  
○ ubuntu@ubuntu-desktop-vm:~/Coding$
```

The status bar at the bottom indicates the current cursor position: `Ln 2, Col 1`, `Spaces: 4`, `UTF-8`, `LF`, `{ }`, `Python`, `3.10.6 64-bit`.

# Create a new C source code file (main.c).

The screenshot shows the Visual Studio Code interface with a C source code file named `main.c` open. The code in the editor is as follows:

```
1 #include <stdio.h>
2 #include <time.h>
3
4 int main(int argc, char *argv[]) {
5     time_t current_time;
6     struct tm *local_time;
7
8     // Get current time
9     current_time = time(NULL);
10
11    // Convert current time to local time
12    local_time = localtime(&current_time);
```

A yellow callout box on the right side of the editor contains the text: **Install the C/C++ Extension Pack for Visual Studio Code on the remote host.**

At the bottom of the terminal, a notification asks: "Do you want to install the recommended extension for C?" with "Install" and "Show Recommendations" buttons.

# Installation of the C/C++ Extension Pack

The screenshot shows the Visual Studio Code interface with the C/C++ Extension Pack being installed. The top bar indicates the current extension is active: "Extension: C/C++ Extension Pack - Coding [SSH: 192.168.56.1] - Visual Studio C...". The left sidebar shows the "EXTENSIONS: ..." view with a search filter "@id:ms-vscode.cpptools-exten" and a list of extensions, including "C/C++ Ex..." with 16.5M downloads and a 4.5 star rating. The main area displays the "C/C++ Extension Pack" details, including the Microsoft logo, version "v1.3.0", and a rating of five stars. A blue "Installing" button is visible, along with a note that the extension is enabled in the Remote Extension Host. Below this, the "Extension Pack (3)" section lists the included extensions: "C/C++" (IntelliSense, debugging, and code br...) with a 242ms load time and a checkmark indicating it is installed. The bottom of the screen shows the status bar with "0" errors, "0" warnings, and "0" messages.

Extension: C/C++ Extension Pack - Coding [SSH: 192.168.56.1] - Visual Studio C...

EXTENSIONS: ... @id:ms-vscode.cpptools-exten

C/C++ Ex... 16.5M ★ 4.5  
Popular extensions for C+...  
Microsoft Installing ⓘ

**C/C++ Extension Pack** v1.3.0  
Microsoft microsoft.com | 16,549,748 | ★★★★★  
Popular extensions for C++ development in Visual Studio Code.  
Installing ⓘ This extension is enabled in the Remote Extension Host because it prefers to run there. [Learn More](#)

DETAILS CHANGELOG

**Extension Pack (3)**

**C/C++** 242ms  
C/C++ IntelliSense, debugging, and code br...  
Microsoft ✓ Installed ⚙️

**C/C++ Extension Pack**  
This extension pack includes a set of popular extensions  
C++ development in Visual Studio Code:

Categories  
Extension Packs

Extension Resources  
[Marketplace Repository](#)  
[License](#)  
[Microsoft](#)

0 0 0



## Demo C code (file: main.c)

```
#include <stdio.h>
#include <time.h>

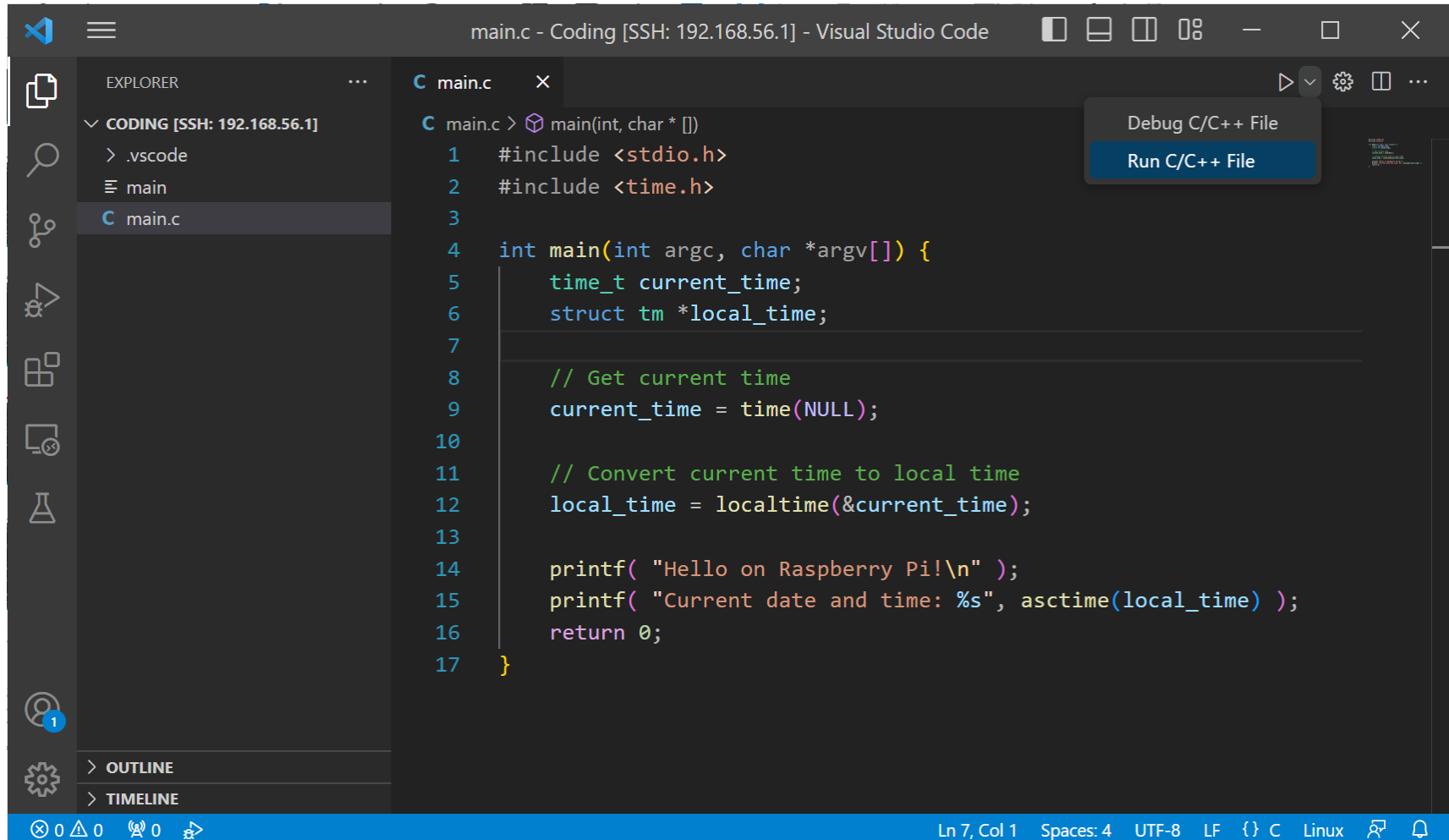
int main(int argc, char *argv[]) {
    time_t current_time;
    struct tm *local_time;

    // Get current time
    current_time = time(NULL);

    // Convert current time to local time
    local_time = localtime(&current_time);

    printf( "Hello on Raspberry Pi!\n" );
    printf( "Current date and time: %s", asctime(local_time) );
    return 0;
}
```

# Compile and Run the C Program

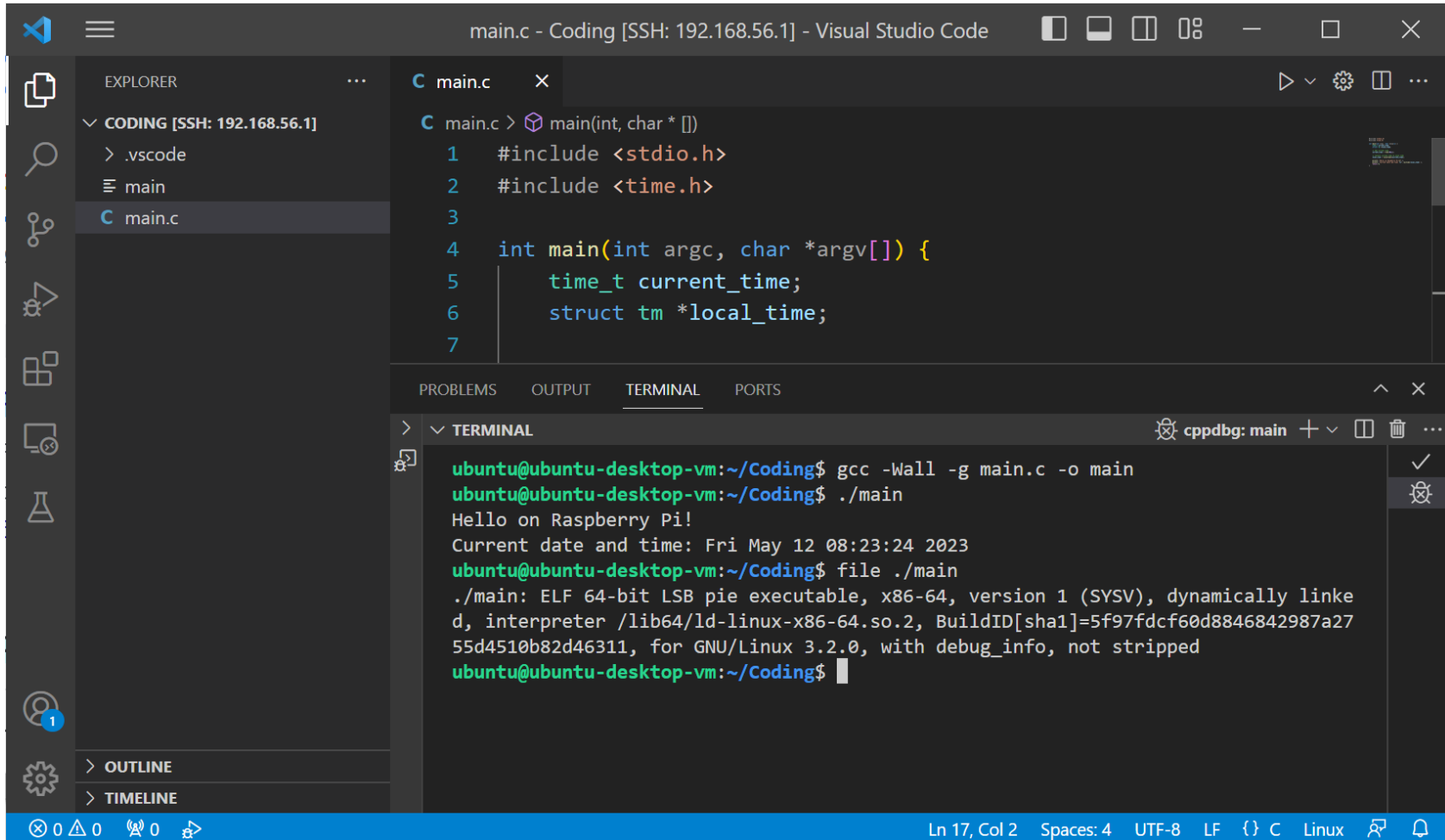


```
main.c - Coding [SSH: 192.168.56.1] - Visual Studio Code
EXPLORER
CODING [SSH: 192.168.56.1]
  .vscode
  main
  C main.c
C main.c
main(int, char * [])
1 #include <stdio.h>
2 #include <time.h>
3
4 int main(int argc, char *argv[]) {
5     time_t current_time;
6     struct tm *local_time;
7
8     // Get current time
9     current_time = time(NULL);
10
11    // Convert current time to local time
12    local_time = localtime(&current_time);
13
14    printf( "Hello on Raspberry Pi!\n" );
15    printf( "Current date and time: %s", asctime(local_time) );
16    return 0;
17 }
```

Debug C/C++ File  
Run C/C++ File

Ln 7, Col 1 Spaces: 4 UTF-8 LF {} C Linux

# Run Linux commands in the remote terminal (Ubuntu VM)



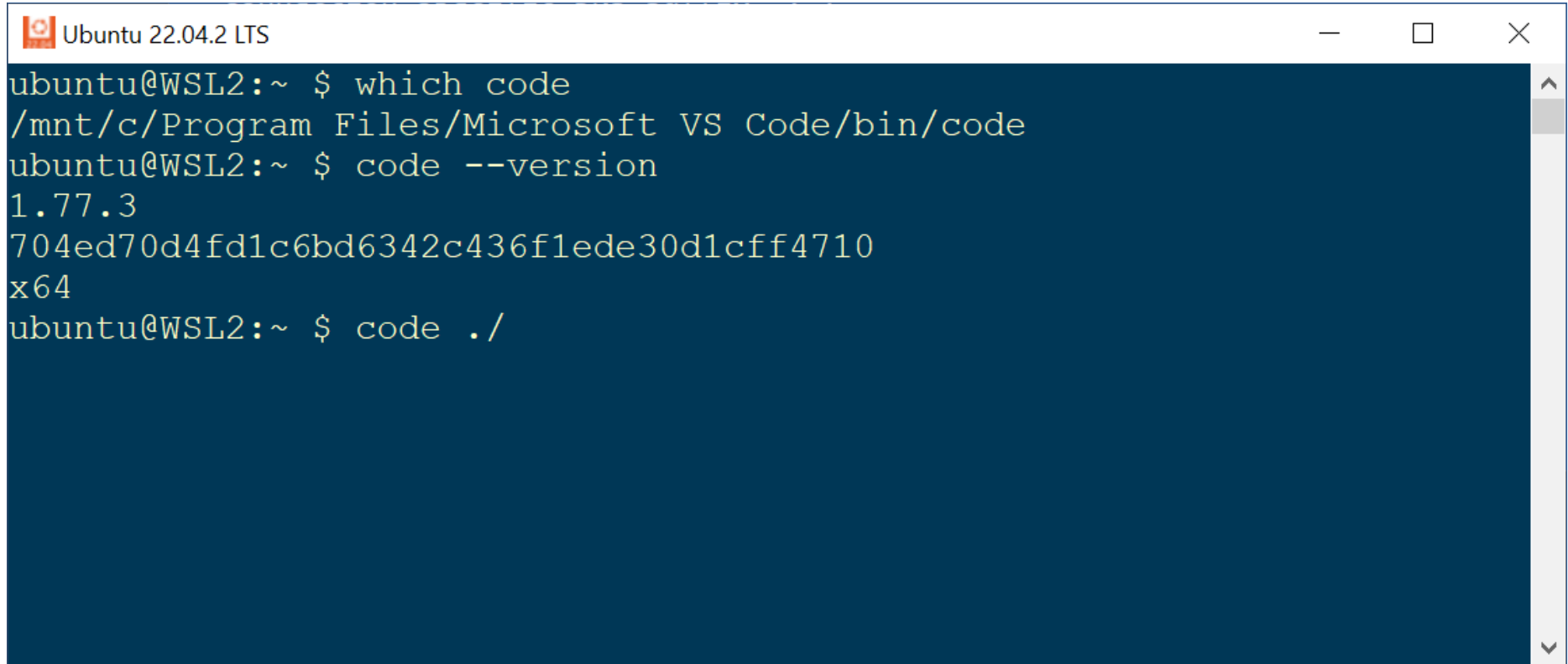
The screenshot shows the Visual Studio Code interface with a remote terminal connected to an Ubuntu VM. The terminal window displays the following commands and output:

```
ubuntu@ubuntu-desktop-vm:~/Coding$ gcc -Wall -g main.c -o main
ubuntu@ubuntu-desktop-vm:~/Coding$ ./main
Hello on Raspberry Pi!
Current date and time: Fri May 12 08:23:24 2023
ubuntu@ubuntu-desktop-vm:~/Coding$ file ./main
./main: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=5f97fdcf60d8846842987a2755d4510b82d46311, for GNU/Linux 3.2.0, with debug_info, not stripped
ubuntu@ubuntu-desktop-vm:~/Coding$
```

The code editor shows the following C program:

```
1 #include <stdio.h>
2 #include <time.h>
3
4 int main(int argc, char *argv[]) {
5     time_t current_time;
6     struct tm *local_time;
7 }
```

## Using the VS Code IDE in WSL2-Ubuntu VM

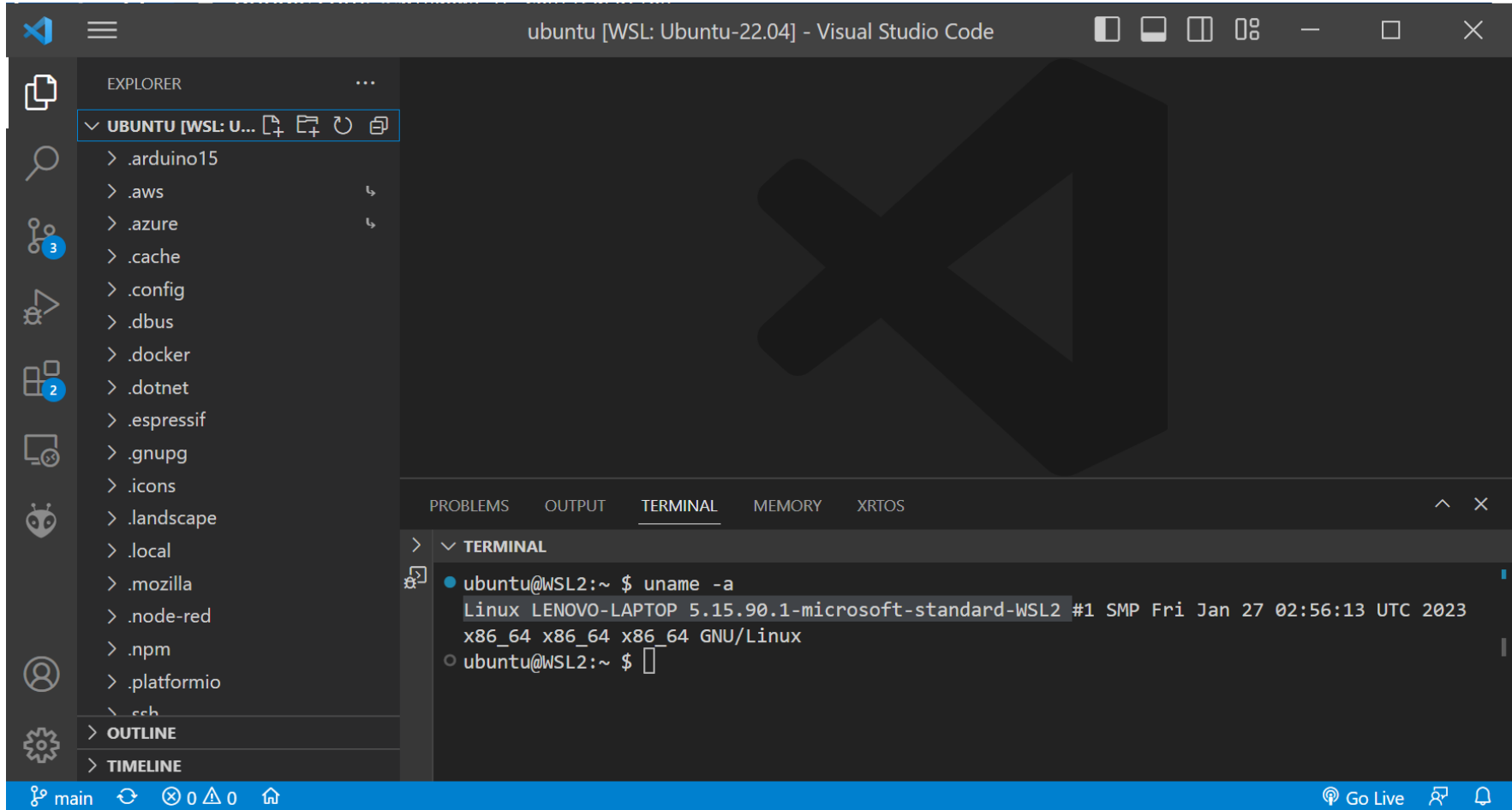
A terminal window titled 'Ubuntu 22.04.2 LTS' with standard window controls. The terminal shows the following commands and output:

```
ubuntu@WSL2:~ $ which code
/mnt/c/Program Files/Microsoft VS Code/bin/code
ubuntu@WSL2:~ $ code --version
1.77.3
704ed70d4fd1c6bd6342c436f1ede30d1cff4710
x64
ubuntu@WSL2:~ $ code ./
```

- 1) Start **WSL2-Ubuntu**.
- 2) Run the following command in the terminal to start **VS Code**.

```
$ code ./
```

# Using the VS Code IDE in WSL2-Ubuntu



# Conclusions

- We have learned how to use **SSH (Secure Shell)** to access a remote Ubuntu server, using password-based and public-key authentication.
- We have learned how to use **VS Code IDE with Remote Development Extensions Pack** for remote software development.