C-001 Regarding Setup of a Development Environment

Vincent W. Finley[*]

July 2025
Rev A

Setting up a modern developer environment is a non-obvious undertaking. General instructions regarding how to setup a working software development environment are discussed herein.

Instructions pertain to an environment where:

- A local IDE is connected to a remote VM, and
- The VM is configured to run a developer container.

Instructions are specific to the author's containerized environment. However, these instructions can be adapted to other project environments.

1 Introduction

Note: Acronyms and Abbreviations at end of article.

Your software development environment can either speed or hinder progress.

Recent years have seen an explosion in software developer tooling. More power is within the reach of more developers, at the cost of more choices and increased complexity.

Aggregating a collection of tooling into an integrated whole is the key to attaining a successful developer environment. Seldom is found comprehensive documentation describing how independent tools can be assembled into a working development environment.

This article shows how to integrate several tools into a working development environment.

Prior to 20 years ago, software development centered around the desktop.

- Software was written, compiled, tested and run on a desktop PC.
- The end product targeted users who installed the released software on their own desktop PCs.
- This system had the advantage of simplicity.
- Setting up the development environment was relatively straightforward.

[*]©2025: Bear, DE, USA: CC BY 4.0 license
https://creativecommons.org/licenses/by/4.0/legalcode

Typically an IDE, runtime engine and supporting libraries were installed on the developer's networked desktop PC.

- Everything needed to develop code was contained locally on the desktop PC.
- The environment was truly an integrated whole.
- Only an external code repository was needed to share code amongst the development team.

Software development became more complex with the rise of Web applications. No longer was the software product intended to be run on the desktop. The user interface now ran in a browser, and the low-level application logic ran on a server somewhere on a network.

Around 10+ years ago, when containerized microservice applications appeared on scene, development became more complex. Developers found themselves writing code remotely. Their working code and compiler resided on a remote machine, whereas their physical laptop PC acted as just a terminal.

The old way:

- Monolith desktop development model.
- Everything on your local machine.
- Remote source control.

The new way:

- Microservice/cloud/container model.
- Local machine is just a terminal.
- Development on remote (virtual) machine.
- Developer containers.
- Remote staging/deployment.
- Remote source control.

2 Solution

As with most developers, you are probably working on either a laptop or desktop PC. You are likely running a version of Microsoft Windows. Perhaps you are reluctant to uninstall the Windows distribution that came with your machine? Or perhaps your employer's IT policy prevents you?

Good news, there's a solution to develop in a more friendly open-source Linux environment. Docker/Podman containers accommodate hosting of guest environments on your available hardware. Your compiler and development libraries could live in a VSCode developer Docker container, while your code could reside on the hosting system.

If the hosting system were some remote Linux machine with a Docker/Podman installation, then you would have a pretty good development environment. Sure, you could install Docker/Podman directly onto your Windows laptop and run your containers there. But wouldn't you rather develop in a true Linux environment? Especially when you are writing cloud deployable microservices and web-apps?

The advantage of remoting into the development hosting environment is that you may be working with multiple teams on multiple projects. Each project or team could have their own development hosting environment. In the morning, you could point your IDE on your laptop to project A. After lunch, you could point it to project B. You could also work on different applications within the same project environment, by simply running different containers. All this is possible without reconfiguring or disturbing your laptop.

2.1 Strategy

Ok, so what do we want to accomplish? Let's assume we want to write Golang code for some project.

- We will need a container setup to build/run our Golang program.
- We will need Docker/Podman installed on a remote Linux machine, to run the container.
- We will also need a client to manage code in our source control repository.
- We will need an IDE installed on our Windows laptop, talking to the remote Linux machine.

Our development environment will roughly look like Figure 1.

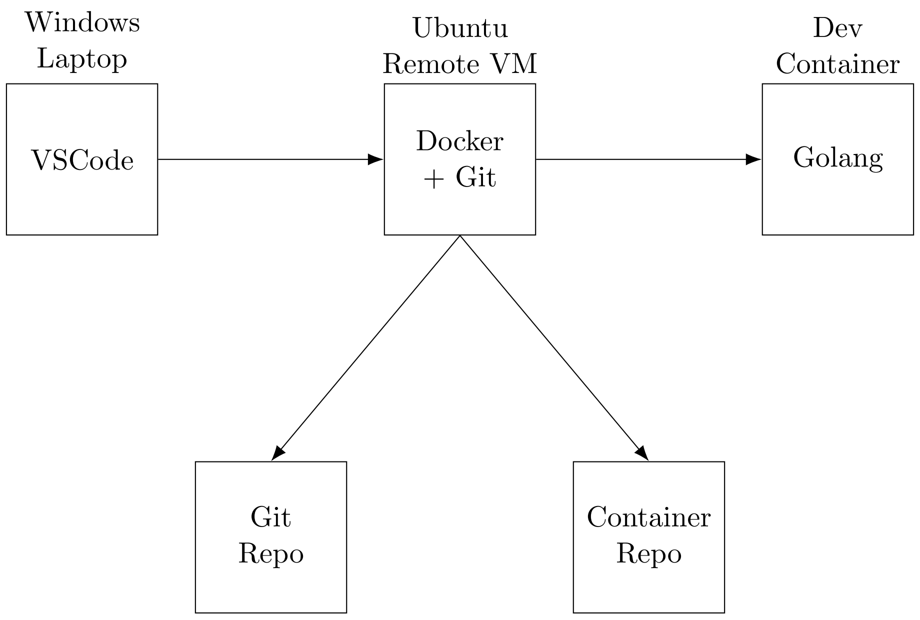

Figure 1: The Development Environment

2.2 Remote Dev Machine

For this exercise, many of the technology choices will have already been made for us. For example, our Windows 11 laptop is a given.

Other technology choices are open-ended. The biggest choice we need to make is related to the remote development machine.

- Which Linux distribution will we run?

- Will the machine be physical or virtual?
- What hardware are we running on?

If you have a remote physical machine that you can dedicate to running a Linux distribution, you are indeed lucky. In most cases you will be interested in running a Linux distribution as a Virtual Machine inside a hypervisor.

Fortunately, there are many hypervisors available. WSL, HyperV, vSphere, ESXi, ProxMox, VirtualBox: are some of the hypervisors you could use to run the Linux instance in a VM. Many can be installed either on a networked server or locally on your laptop. Which you choose depends upon budget, licensing and compute hardware.

Another popular solution is to run the Linux instance in a Cloud VM service. For example AWS EC2 or Google Compute Engine are popular choices.

For my environment I will be installing Oracle VirtualBox hypervisor on my laptop. Yes, the Linux instance can run in a VirtualBox hosted VM on the same Windows laptop where my IDE runs. From the IDE's point of view, the Linux instance will appear to be on a remote network machine. The IDE doesn't care if the remote machine lives in an EC2 instance or in my laptop's hypervisor

2.3 Linux Distribution

The best place to start searching for a Linux distribution is DistroWatch[1]. There you will find hundreds of Linux (and BSD) operating system distributions.

Linux distributions fall into two main families that differ by package manager and available packages. Each has its pros and cons. The RHEL/Fedora family uses the YUM package manager. The Debian/Ubuntu family uses the APT package manager.

RHEL/Fedora has packages to install Podman and Buildah. Podman increases security by enforcing rootless containers.

Debian/Ubuntu has packages to install Docker. Docker is easier to work with since much existing tooling had been developed for it. Sharing project files with developer containers is also easier in Docker.

For my development environment I will choose an Ubuntu[2] Desktop distribution since I want the ease of working with Docker.

2.4 Source Control

Git is the de facto standard for modern source control. Choosing Git is a no-brainer since I will be using a repository in GitHub to keep my source code.

2.5 IDE

Installing VSCode on my laptop is a matter of convenience. Better IDEs probably exist, but I am familiar with VSCode.

[1]https://distrowatch.com
[2]https://ubuntu.com/

VSCode has the advantage of developer container support. It also has extensions that integrate remote SSH connections.

During routine usage VSCode acts almost like a thin-client terminal. Once VSCode is setup, developing code in a dev container on a remote machine appears similar to developing code on your local laptop. This makes for a nice developer experience, to the point where one forgets where the code and compiler actually lives.

2.6 Password Manager

As you develop complex systems, you will accumulate (and forget) multiple passwords. It is helpful to secure your account logins and passwords in a single place.

Several systems are available to keep your passwords handy and safe. I recommended you install one of the following password managers: Bitwarden, 1Password and KeePassXC; on your PC.

2.7 Laptop

VirtualBox hypervisor will be running on the same laptop where VSCode will be installed. For performance reasons it is worth noting the laptop configuration. Knowing the laptop configuration helps when setting expectations and making comparisons to other hardware environments.

```
Device Specifications
---------------------
Manufacturer: Lenovo
Model: IdeaPad Slim 5 16IRU9
Processor: Intel(R) Core(TM) 7 150U   1.80 GHz
Installed RAM: 16.0 GB (15.7 GB usable)
Product ID: 00342-22290-21080-AAOEM
System type: 64-bit operating system, x64-based processor
Pen and touch: No pen or touch input is available for this display


Windows Specifications
----------------------
Edition: Windows 11 Home
Version: 24H2
Installed on: 2/14/2025
OS build: 26100.3775
Experience: Windows Feature Experience Pack 1000.26100.66.0
```

2.8 Environment Summary

After much consideration, the development below was selected. By no means is this an optimal solution. You can swap in/other components of this environment to suit your development scenario.

- Host OS: Windows 11 Home Edition
- Password Manager: KeePassXC 2.7.9
- Hypervisor: Oracle VirtualBox Version 7.1.6 r167084 (Qt6.5.3)

- IDE: Visual Studio Code 1.100.2 (user setup)
- Guest OS: Ubuntu 22.04.5 LTS Desktop
- Container runtime: Docker 28.8.1, build 4eba377
- Source code repository: GitHub
- Container image repository: DockerHub, Microsoft Artifact Registry
- Golang container: mcr.microsoft.com/devcontainers/go:1-1.23-bookworm
- Golang version: go1.23.8 linux/amd64

3 Hypervisor

Let's begin setting up the development environment. We will install Oracle VirtualBox onto the laptop.

Do the following:

1. Browse to https://www.virtualbox.org/wiki/Downloads
2. Select the link for "Windows hosts".
3. Once the download has completed, open the download location and run the *.exe file.
4. When prompted if you want the app to make changes to your Device, click Yes.
5. On the welcome panel, click Next.
6. Accept the terms of the license agreement and click Next.
7. On the Custom Setup panel, accept all the defaults, and click Next.
8. On the Warning: Network Interfaces panel, click Yes.
9. After installation has completed, reboot your laptop.
10. Once you've logged back into your laptop, launch Oracle VirtualBox from the Windows start menu.

4 Virtual Machine

Setting up a Virtual Machine is not difficult, but there are many steps involved.

4.1 Download Linux image

To create an Ubuntu Desktop virtual machine on the VirtualBox hypervisor, we will need a distribution image.

1. Browse to https://ubuntu.com/download/desktop
2. Search the page for "check out our alternative downloads", and click it.
3. Search the page under Past Releases for Ubuntu 22.04 LTS (Jammy Jellyfish), and click it.
4. Click the link for "64-bit PC (AMD64) desktop image".

5. Download of the file named ubuntu-22.04.5-desktop-amd64.iso will complete in a few minutes.

4.2 Create New Virtual Machine

Let's create a new virtual machine in VirtualBox with the image we downloaded.

1. From the Windows start menu, launch Oracle VirtualBox.
2. In Oracle VirtualBox Manager application goto: Machine > New...
3. In the Create Virtual Machine popup dialog, we'll set Name="alpha".
4. Click the ISO Image field and search for the ubuntu-22.04.5-desktop-amd64.iso you downloaded.
5. Set Type=Linux
6. Set Subtype=Ubuntu
7. Set Version=Ubuntu (64-bit)
8. Check "Skip Unattended Installation".
9. Expand the Hardware section.
10. Increase Processors to 2 CPUs.
11. Click the Finish button.

4.3 Start the Virtual Machine

Okay, let's launch Ubuntu inside VirtualBox.

1. In the Oracle VirtualBox Manager application, search the left panel for the machine "alpha" you just created.
2. Right-click on "alpha" and select Start > Normal Start.
3. Once the boot screen appears in the preview pane, click "Show" on the top menu.
4. Click inside the boot window that appears, select "Try or Install Ubuntu", and hit enter.

4.4 Install Ubuntu

Time to install Ubuntu onto the VM!

1. Once the Ubuntu desktop appears, you should see a window with the title "Install".
2. The page has two options, choose your language then select the option "Install Ubuntu".
3. On the Keyboard Layout page, choose your keyboard layout and click "Continue".
4. On the Updates and Other Software page, choose Minimal Installation, uncheck "Download Updates while installing Ubuntu", check "Install third-party software..."
5. Click "Continue".
6. On the Installation Type page, select "Erase disk and install Ubuntu".
7. Click "Install Now".

8. On the pop-up box that says “Write the changes to disk?”, click “Continue”.
9. Select your timezone and click “Continue”.
10. On the “Who are you?” page, populate your name, set the computer name to “alpha”, set your username and password.
11. Choose “Require my password to log in”.
12. Click “Continue”.
13. Ubuntu will begin installing onto the VM, be patient it will take 10-15 minutes to complete.
14. Once the installation has completed, click the “Restart Now” button.
15. Wait a few moments, an Ubuntu screen will appear telling you to remove the installation media. DO NOT Press Enter Yet!
16. Go into the VirtualBox Manager application, select the “alpha” machine, and click Settings on the top menu.
17. Select Storage from the Left panel.
18. Under Controller IDE, click the disk icon.
19. Uncheck Live CD/DVD. Click OK.
20. Now go back to the remove installation media screen, and press Enter.
21. Wait for the VM to shutdown.

5 Initial Login

We will restart the VM named alpha, and login for the first-time.

5.1 First time login pages

1. In VirtualBox right-click the VM named alpha.
2. Select Start > Normal Start
3. When the Ubuntu login window appears, login.
4. When the desktop appears, a setup window named “Online Accounts” will appear.
5. Click Skip.
6. On the page Enable Ubuntu Pro, select Skip for Now and then click Next.
7. On the Help Improve Ubuntu, make a choice and click Next.
8. On the Privacy page, enable Location Services, and click Next.
9. On the Your Ready To Go Page, click Done.

5.2 Screen Blank

Until you get everything setup, it would be good to turn off the annoying screen blanking.

1. In the Ubuntu desktop, goto Activities > Settings > Privacy > Screen
2. Change Blank screen to: Never
3. Change Automatic Screen Lock to: Disabled
4. Change Lock Screen on Suspend to: Disabled

5.3 Power Auto Suspend

Since the alpha VM is running on the laptop, it will lose connections when the laptop goes to sleep to save power. Let's prevent this from happening.

1. In the Ubuntu desktop, got Activities > Settings > Power
2. Change Screen Blank to: Never
3. Change Automatic Power Saver to: Disabled
4. Change Automatic Suspend to: Off

5.4 Screen Resolution

Finally it would be good for the alpha guest VM to make full use of the display on the laptop host.

1. In the Ubuntu desktop, got Activities > Settings > Displays
2. You will need to try different settings for Resolution.
3. Click the green Apply button.
4. If you like the resolution setting click Keep Changes, otherwise click Revert Settings.
5. For my laptop, Resolution=1680x1050 seems to work best.

5.5 Shutdown

Shutdown the alpha VM to prepare for the next section.

6 Network Setup

The Ubuntu desktop distribution has been installed into a VM that runs in VirtualBox on the laptop. As far as the laptop is concerned, the VM is a wholly independent machine, even though the VM lives on the laptop.

We need a way to talk to the VM from the laptop. To achieve this we will be making a change to the network, and then gathering some information.

Initially, VirtualBox created the VM with a single default network adapter that supports Network Address Translations (NAT). This default adapter is fine for connecting to the outside world: browsing to an internet website, or pinging google.com. In other words, outbound traffic is supported.

However, inbound traffic is unsupported. If we were to ping the "alpha" VM from the commandline on the laptop, we would get no reply.

6.1 Add Bridge Adapter

To support inbound traffic, we need to add another network adapter.

1. In VirtualBox: Select machine alpha (powered off).
2. Goto Settings > Network > Adapter 2 tab
3. Check Enable Network Adapter
4. Change Attached to: = Bridged Adapter
5. Click OK

6.2 Gather IP Addresses

Let's verify the network is configured correctly, and can send/receive traffic. While we are at it, let's find the alpha VM IP addresses.

1. You should still be logged into the alpha VM from the previous section.
2. Click Activities > Settings > Network
3. For each of the two Ethernet connections:
   - Click on the Gear button
   - Click on the Details tab
   - Write down the IPv4 addresses for later.
   - For example: 10.0.2.15 and 10.0.0.246
4. Close the Settings application.

6.3 Outbound connection

We have two network adapters each with a IPv4 address associated to it. It is unclear which address is the mapped to the NAT adapter and which is mapped to the Bridged adapter. In other words we don't know which is being used for outbound and inbound traffic. However we can test to verify the outbound connection is working properly.

1. On the Ubuntu desktop, click Activities.
2. Search for and select "terminal".
3. Once the terminal application launches, at the commandline type:

   ```
   ping google.com
   ```

4. Hit Enter
5. Verify you get a response back.
6. Hit Ctrl+C to stop pinging.

6.4 Inbound connection

Okay, let's verify that the inbound (Bridged) connection is working. It is also an opportunity to determine which address is the inbound address, and which is the outbound address.

1. On the laptop, open a Windows CMD terminal.
2. In at the commandline in the terminal you opened above, run ping with one of the addresses you wrote down above.
3. For example:

   ```
   ping 10.0.2.15
   ```

4. If you get a response, it is the inbound (Bridged) traffic address, otherwise it is the outbound (NAT) address.
5. Write either "inbound" or "outbound" next to this address.
6. At the commandline try pinging the other address.
7. For example:

   ```
   ping 10.0.0.246
   ```

8. If you get a response, it is the inbound (Bridged) traffic address, otherwise it is the outbound (NAT) address.
9. Write either "inbound" or "outbound" next to this address.

7 Software Updates

You have verified that the network is configured and functional. While your are still logged into the desktop, it is a great opportunity to update all the software on the system.

There are a couple ways to update.

7.1 Method 1: Update Graphically

Ubuntu desktop has an application named Software Updater to make system updates convenient.

You may notice a message box that periodically appears. It will say "Updated software is available for this computer. Do you want to update it now?".

You can also manually run the Software Updater application by going to: Activities > Software Updater

When prompted, click Install Now. Be sure to reboot after the update completes.

7.2 Method 2: Update Commandline

A better way to update is from the commandline. This method better supports automation and will work with headless (server) Ubuntu distributions.

First launch a terminal: Activities > Terminal

Then enter the following at the commandline.

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

You should reboot after the upgrade completes.

8 Hosts File

At this point we have an Ubuntu Linux desktop distribution running in a virtual machine. Our virtual machine is named “alpha”.

A couple sections ago, we had established communication between our laptop and the “alpha” VM.

Alpha responded to the Ping at address 10.0.0.246. This address is the inbound traffic address associated with the Bridged network adapter.

This address will be used to communicate with alpha hereafter. However, the address is difficult to remember and tedious to type frequently.

It would be good to alias the address to a friendly name like “alpha”. Let’s update the laptop’s hosts file to associate the address to the friendly name.

To edit the hosts file on a Windows 11 system, run Notepad as admin.

1. Click Windows start icon.
2. Type “notepad”.
3. Expand menu under Open.
4. Select as “Run as administrator”.
5. Open file

   ```
   C:\Windows\System32\drivers\etc\hosts
   ```

6. Add following entry to the file:

   ```
   10.0.0.246  alpha # Ubuntu VM
   ```

7. Save the file.
8. Close Notepad.

Verify that the friendly name is associated with the address. At a Windows cmd shell prompt, do the following and verify alpha replies to the ping.

```
C:\Users\vfinl>ping alpha

Pinging alpha [10.0.0.246] with 32 bytes of data:
Reply from 10.0.0.246: bytes=32 time=1ms TTL=64
Reply from 10.0.0.246: bytes=32 time<1ms TTL=64
Reply from 10.0.0.246: bytes=32 time<1ms TTL=64
Reply from 10.0.0.246: bytes=32 time<1ms TTL=64

Ping statistics for 10.0.0.246:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms
```

9 SSH Service

Most Linux **server** distributions are already setup for inbound SSH connectivity. However, Linux **desktop** distributions are setup to support outbound SSH connections only. The Ubuntu desktop distribution is no exception. It is setup to be an SSH client, but not a server. Inbound SSH support must be setup manually.

9.1 Install SSH Server

To install the SSH service on an Ubuntu system[3], open a terminal on alpha and enter the following:

```
$ sudo apt-get install openssh-server
$ systemctl status ssh

# If ssh.service is not running, do the following command:
$ sudo systemctl restart ssh
```

10 OpenSSH Client

Keep in mind, SSH (Secure SHell) is a protocol. The implementation of the protocol has a client and server associated with it.

OpenSSH client should already be installed on Windows 11 by default. Earlier Windows versions may have it disabled.

If OpenSSH client is unavailable on your laptop, search the internet on how to enable or install it. You will find the project page here: https://www.openssh.com/

The best way to verify if OpenSSH client is available is to enter the following at the Windows CMD prompt:

```
$ ssh
$ where ssh
```

[3]https://devconnected.com/how-to-install-and-enable-ssh-server-on-ubuntu-20-04/

As part of OpenSSH support for Windows, ssh-keygen commandline utility is probably bundled in.

10.1 Initial SSH remote login

In the previous section, you enabled the SSH service on the remote VM named alpha. Let's test the connection!

1. On your laptop, open a Windows command shell and do the commands below.
2. When prompted "Are you sure you want to continue connecting" enter "yes".
3. Then enter your password for alpha.

```
C:\Users\vfinl>ssh vfinley@alpha

The authenticity of host 'alpha (10.0.0.246)' can't be established.
ED25519 key fingerprint is SHA256:YTJSgWrfm9oppwq+vtpskrQjk0hPw80dJ8xzBAfacOI.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes

Warning: Permanently added 'alpha' (ED25519) to the list of known hosts.

vfinley@alpha's password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-57-generic x86_64)

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.
```

10.2 Check BASH shell functionality

Congratulations, you are SSH'd into alpha! To verify everything is working, enter "ls" at the BASH shell prompt. Inspect contents of your user directory, it should contain Desktop, Documents, etc. as shown below.

```
vfinley@alpha:~$ ls
Desktop  Documents  Downloads  Music  Pictures
  Public  snap  ssh  Templates  Videos
```

Now exit the shell by entering exit.

```
vfinley@alpha:~$ exit
```

10.3 Generate Key

SSH secures authentication and communication by using a public+private key pair. Using the key pair makes it convenient since you won't need to login every time you connect to the remote alpha VM.

You wil need to:

1. Create the key pair.
2. Save the pair to your laptop.
3. Then give the public key to the remote VM named alpha (Ubuntu Desktop).

When asked for a file in which to save the key, hit Enter to save it to the default location. When creating the key pair you can set an optional password (or leave it blank) when asked for a passphrase. If you set a passphrase you will prompted for it once a session when making connections.

Here's how do generate a keypair and copy it to the remote VM named alpha. On the laptop, at the Windows CMD prompt do the following:

```
C:\Users\vfinl> ssh-keygen
Generating public/private ed25519 key pair.
Enter file in which to save the key (C:\Users\vfinl/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\vfinl/.ssh/id_ed25519
Your public key has been saved in C:\Users\vfinl/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:rp5oKbld4wp4biuWaVsnoALpThqM2x16ud5Poty4xnY vfinl@victor
The key's randomart image is:
+--[ED25519 256]--+
|                 |
|                 |
|                 |
| .               |
|o   .   S        |
|=  o . .         |
|+=o==o+ =        |
|+*XXAO.B .       |
|o*XOYBB..        |
+----[SHA256]-----+
```

10.4 Copy Public Key

Okay, now we need to copy the public key over to the alpha VM. There are two possible ways to do this. If you have ssh-copy-id already installed we can use it to copy the key. Otherwise don't worry, we can copy the key manually.

To copy the key with ssh-copy-id, goto where the keys were generated in the previous subsection. And do the following:

```
C:\Users\vfinl> cd C:\Users\vfinl\.ssh

C:\Users\vfinl\.ssh> dir

C:\Users\vfinl\.ssh> ssh-copy-id -i id_ed25519.pub vfinley@alpha
```

To manually copy the public key:

1. In the Windows CMD shell, goto where the public key was generated in the previous section.
2. Use the Windows type command to display the line from the .pub file.
3. Copy the line from the .pub file.
4. ssh into the alpha remote VM.
5. Goto the .ssh hidden directory.
6. Edit or create the file named authorized˙keys
7. Paste the line from the .pub file into the authorized˙keys file.
8. Save the file and quit the editor.

Here's an example of how to manually copy the public key.

```
# Goto where the ssh keys live.
C:\Users\vfinl>cd .ssh

C:\Users\vfinl\.ssh>dir
 Volume in drive C is Windows-SSD
 Volume Serial Number is 9484-3D6D

 Directory of C:\Users\vfinl\.ssh

05/06/2025  07:07 AM    <DIR>          .
05/05/2025  06:49 AM    <DIR>          ..
05/06/2025  07:23 AM                62 config
05/05/2025  06:51 AM               399 id_ed25519
05/05/2025  06:51 AM                95 id_ed25519.pub
05/05/2025  06:49 AM               816 known_hosts
05/05/2025  06:49 AM                88 known_hosts.old
               5 File(s)          1,460 bytes
               2 Dir(s)  332,340,908,032 bytes free

# Display the line in the public key file.
# We will copy and paste this.
C:\Users\vfinl\.ssh>type id_ed25519.pub
ssh-ed25519 \
  AAABC3NzaC1lZEI1NTE5AAAAILbyJgcZLOc2E8cEe4m5myuCEcpsrvtkWmGhHCb+ok50 \
    vfinl@victor
```

Okay, now let's login to the alpha VM and paste the public key into the authorized_keys file.

```
# SSH into the alpha VM.
C:\Users\vfinl\.ssh>ssh vfinley@alpha
vfinley@alpha's password:

Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-59-generic x86_64)

# Goto the hidden .ssh directory on alpha, and edit the authorized_keys
vfinley@alpha:~$ cd .ssh
vfinley@alpha:~/.ssh$ vi authorized_keys

# Now, copy/paste the line from above into authorized_keys file
ssh-ed25519 \
  AAABC3NzaC1lZEI1NTE5AAAAILbyJgcZLOc2E8cEe4m5myuCEcpsrvtkWmGhHCb+ok50 \
    vfinl@victor

# Save the line you just pasted, by hitting Escape then colon wq, then Enter.
<ESC>
:wq
<ENTER>

# Now exit the SSH session
vfinley@alpha:~/.ssh$ exit
logout
Connection to alpha closed.
```

10.5 Testing SSH connection with keyed login

Let's verify that the public SSH key is being used. At the Windows CMD prompt, you should be able to ssh into the alpha VM. If you set a passphrase when generating the keypair, you will be prompted for the passphrase. Otherwise if you left passphrase empty, you will be automatically connected.

```
# No passphrase was set.
C:\Users\vfinl>ssh vfinley@alpha
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-59-generic x86_64)
....
....
....
Last login: Tue May 20 10:36:47 2025 from 10.0.0.137

# Type exit to leave ssh and return to Windows CMD prompt.
vfinley@alpha:~$ exit
logout
Connection to alpha closed.

C:\Users\vfinl>
```

11 Microsoft Visual Studio Code (VSCode)

Now that the SSH keys and connection have been established, it is a convenient time to setup VSCode. Sure there are other IDEs you could install, but we'll be installing VSCode as the baseline example. VSCode has a couple advantages: it is widely used by developers, and it is freely available.

11.1 Setup VSCode

To setup VSCode:

1. On your your laptop, browse to the Microsoft App Store at https://apps.microsoft.com/
2. Search for VSCode or go here:

   ```
   https://apps.microsoft.com/detail/xp9khm4bk9fz7q?hl=en-US&gl=US
   ```

3. Download and Install VSCode on your laptop.
4. After VSCode has installed, launch it.
5. The current VSCode version is Version 1.100.2, you version may be newer.

11.2 Install Remote Development extension pack

Once VSCode is up and running, we need to install the Remote Development extension pack. The pack contains 4 extensions that will help us connect VSCode to the alpha VM.

1. In VSCode goto View > Extensions or type (Ctrl+Shift+X)
2. In the "Search Extensions in Marketplace" search bar, type "Remote Development".
3. The Remote Development pack icon (has number 4) should appear at the top of the search results.
4. Click the blue install button under Remote Development.

5. After the Remote Development pack installs you may be asked to restart VSCode.

11.3 Connect VSCode to the alpha VM

Let's point VSCode to our alpha VM. Doing so will let us use VSCode to develop code on alpha.

1. First we need to verify our alpha VM is running in VirtualBox.
2. Next, in VSCode click Remote Explorer icon on the left vertical panel.
3. Select Remotes(Tunnels/SSH) from drop down at top of Remote Explorer panel.
4. Hover over the SSH section on the panel.
5. Click the Plus (New Remote) to the right of SSH.
6. In the "Enter SSH Connection Command" box, enter "ssh your_username@remotemachine -A", for example: ssh vfinley@alpha -A
7. In the "Select SSH configuration file to update" box enter the first file in the list, for example C:/Users/vfinl/.ssh/config

11.4 Initial Connection

Let's use VSCode to connect to the alpha VM.

1. In Remoter Explorer SSH list, find the host (alpha) you just added.
2. Hover over alpha
3. Click the right arrow (Connect in current window) button.
4. Wait about 30 sec for VSCode server to install.
5. Click Explorer icon in left bar (Ctrl+Shift+E), click "Open Folder" button.
6. Select default path, click OK
7. You will see dialog box "Do you trust the authors of the files in this folder?"
8. Click "Yes, I trust the authors"

11.5 Testing the terminal

Once we've got a connection to our alpha VM, we need to check the terminal is available.

1. In VSCode goto View > Terminal
2. Click in the terminal panel that appears at bottom and enter: ping google

12 Git

Every development project, big and small, needs source code control. Git is the de facto standard for source control in modern development projects. We will set up our project to use a Git client with GitHub as our online repository. In practice you may substitute online and onprem Git servers, including GitLab, Azure DevOps, etc.

12.1 Git Install on Remote VM

Connect to the alpha remote VM with VSCode. Open the terminal in VSCode, it should be connected to the alpha VM. Do the following:

```
# Install Git client on the alpha VM.
$ sudo apt install git
```

12.2 Initalize user information

Setup your user information. It will be used to identify you when reviewing Git change history.

```
# Register your name and email with Git.
# Be sure to change user name and email to YOUR name and email address
$ git config --global user.name "John Doe"
$ git config --global user.email "john.doe@youremail.com"
```

12.3 Personal Access Token (PAT)

Personal Access Tokens (PATs) provide more security and flexibility than user passwords. Many Git servers are configured to use PATs exclusively. You will need to generate a PAT at least once per year. Don't worry if you forget or lose your PAT, you can always generate a new one.

To create a PAT do the following:

1. Login to GitHub or other online/onprem Git service.
2. Click on the user icon
3. Select "Settings" from the menu.
4. Left menu, scroll to bottom, click "Developer Settings".
5. Left menu, under Access Tokens, select Fine-Grained Tokens
6. Click green "Generate new token" button.
7. Fill in the "Token name" field with a name you will remember.
8. Set the Expiration date to a convenient date.
9. Select "All Repositories"
10. Under "Permissions / Repository permissions" at a minimum give yourself "Read and write" access to Contents and Pull requests.
11. Click the green "Generate token" button at the bottom.
12. In the popup dialog click the green "Generate token" again.
13. In the page that appears, copy the PAT (including the github_pat_ prefix), name and expiration date.
14. Save it somewhere for later use, like in a password manager (KeePassXC, Bitwarden, etc.).

12.4 Identify Git Repository

The first thing you'll need to do is to identify a Git repository that you will be working out of.

1. In browser find a Git project where you have write priviliges. In my case I have prviliges to write: https://github.com/vwfinley/regarding
2. Click on green "Code" button.
3. Click on the "Local" tab.
4. Select "Https"
5. Copy the URL in the "Clone using the web URL" field. For example: https://github.com/vwfinley/regarding.

12.5 Clone Project

Next you'll need to clone the project. In the VSCode BASH shell terminal that is connected to our alpha VM do the following.

```
# Create a directory for your project, then cd to go there.
$ mkdir ~/Documents/projects
$ cd ~/Documents/projects/

# Clone the repo with the URL you copied above.
~/Documents/projects$ git clone https://github.com/vwfinley/regarding.git
Cloning into 'regarding'...
remote: Enumerating objects: 2047, done.
remote: Counting objects: 100% (271/271), done.
remote: Compressing objects: 100% (179/179), done.
remote: Total 2047 (delta 188), reused 158 (delta 90), pack-reused 1776 (from 2)
Receiving objects: 100% (2047/2047), 3.91 MiB | 4.60 MiB/s, done.
Resolving deltas: 100% (1243/1243), done.
```

12.6 Git Password Saver Setup

Typing your password/PAT over-and-over for Git(Hub/Lab) is annoying. Fortunately there is a better way.

At the terminal Bash shell prompt enter:

```
$ git config --global credential.helper store
```

Next time you enter password/PAT info for Git, you password/PAT will be remembered thereafter. Some Git Repositories are setup to use a PAT in place of a password, when prompted for a password use the PAT instead.

12.7 Initial Checkin

Before changes appear in a remote Git repository, a change must be made to your working copy. Let's make a change to the project that was previously cloned, and check-in the change.

```
# Make change to the project (for example).
~/Documents/projects$ cd regarding/
~/Documents/projects/regarding$ ls
A  B  LICENSE.md  README.md
~/Documents/projects/regarding$ mkdir C
~/Documents/projects/regarding$ ls
A  B  C  LICENSE.md  README.md
~/Documents/projects/regarding$ mkdir C/C-001
~/Documents/projects/regarding$ cd C/C-001

# Create new file
~/Documents/projects/regarding/C/C-001$ echo "Hello World" > README.md

# Initial check-in
~/Documents/projects/regarding/C/C-001$ git add README.md
~/Documents/projects/regarding/C/C-001$ git commit -m "New File"
[main 5ed57a6] New File
 1 file changed, 1 insertion(+)
 create mode 100644 C/C-001/README.md
```

12.8 Initial Push

Now, let's push the change to the remote repository so we can see it.

```
 ~/Documents/projects/regarding/C/C-001$ git push

# When prompted for password paste your PAT token.
~/Documents/projects/regarding/C/C-001$ git push
Username for 'https://github.com': vwfinley
Password for 'https://vwfinley@github.com': <paste_PAT_token_here>

Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (5/5), 365 bytes | 365.00 KiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/vwfinley/regarding.git
   7ff2319..5ed57a6  main -> main
```

12.9 VSCode Git Integration

Support for source code control is built-in to VSCode. You will notice as you make changes to a cloned project, that the Source Control icon on the VSCode left vertical panel will update with the number of pending code changes.

To check-in and push the changes, do the following:

1. Click the icon or type Ctrl+Shift+G.
2. Click the plus icon next to each file you want to commit.
3. Type a commit message above the blue Commit button.
4. Click the blue commit button.
5. Click the blue Sync Changes button, or...
6. ... click the triple dotted More Actions button > Pull, Push > Sync

13 Docker

We eventually want to do our development in a developer container. Before we can think about containers we need to install a container runtime on our alpha VM.

The easiest way to install a container runtime is to install either Docker or Podman. Since much tooling was developed with Docker in-mind, Docker is the simplest to work with. Podman is more secure, however mapping directories from the remote host to a rootless container can tricky.

When your remote host is Fedora/RHEL, you will probably use Podman. Otherwise you will probably use Docker.

Since our alpha VM is running Ubuntu, we will install Docker. Instructions to install Docker given by:

```
https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository
```

They are repeated here in case that webpage disappears. To install Docker, at the VSCode Bash terminal connected to the alpha VM, do the following:

1. Set up Docker's apt repository.

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
      -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) \
  signed-by=/etc/apt/keyrings/docker.asc] \
  https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && \
  echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

2. Install the Docker packages.

```
    sudo apt-get install \
         docker-ce docker-ce-cli containerd.io docker-buildx-plugin \
         docker-compose-plugin
```

3. Verify that the installation is successful by running the hello-world image:

```
    sudo docker run hello-world
```

4. Add yourself or another user to the docker group

```
    # Add yourself to the docker group
    # https://docs.docker.com/engine/install/linux-postinstall/

    # Adding to docker group
    sudo groupadd docker
    sudo usermod -aG docker $USER
    newgrp docker
    docker run hello-world
```

14 Developer Container

Finally we are at the point where we can create a developer container for our development environment. Having a dev container will insure uniformity and repeatability, and generally make life easier. We show how to create a dev container for Golang, however you could select/create dev containers for other languages too. Although we won't get into customization, dev containers can be customized by modifying the devcontainer.json file.

14.1 Create a new dev container

Let's create a basic dev container for our Golang development.

1. In VSCode: Ctrl+Shift+P
2. Search for and select: "Dev Containers: New Dev Container..." from the Select Dev Container Configuration prompt.
3. Search for an appropriate Golang container, for example "Go devcontainers".
   (a) This item has an official checkmark certificate icon.
   (b) Description starts with "Develop Go based applications..."
   (c) You could select other developer containers depending on the languages you want to develop.
4. Select "Create Dev Container go". You could also select "Additional Options...", and pick your desired options.
5. Wait a few mins while container image downloads and installs

6. A new instance of VSCode wil launch.
7. while you are waiting, you can click the box that says "Connecting to dev container (show log)" and watch the progress.

14.2 Container shell prompt

Did it work? It would be good to test out our dev container before writing any code.

1. Once the container has downloaded and installed, open a new terminal in VSCode by clicking View > Terminal.
2. You should see the prompt change to "vscode -> /workspaces/go $" to indicate you are now running inside the developer container.
3. At the prompt, enter "go version"

You should see something that looks like this:

```
vscode -> /workspaces/go $ go version
go version go1.23.8 linux/amd64
```

14.3 Devcontainer file

When the dev container was created, a file was also created in a hidden directory. Search for and open the file at .devcontainer/devcontainer.json

The contents of the will resemble the JSON object below.

```
{
  "name": "Go",
  // Or use a Dockerfile or Docker Compose file.
  // More info: https://containers.dev/guide/dockerfile
  "image": "mcr.microsoft.com/devcontainers/go:1-1.23-bookworm"

  // Features to add to the dev container.
  // More info: https://containers.dev/features.
  // "features": {},

  // Use 'forwardPorts' to make a list of ports inside the
  // container available locally.
  // "forwardPorts": [],

  // Use 'postCreateCommand' to run commands after the container is created.
  // "postCreateCommand": "go version",

  // Configure tool-specific properties.
  // "customizations": {},

  // Uncomment to connect as root instead.
  // More info: https://aka.ms/dev-containers-non-root.
  // "remoteUser": "root"
}
```

The important part of the JSON is the value for the “image” field. We see the value is “mcr.microsoft.com/devcontainers/go:1-1.23-bookworm”. This tells us that you will find the developer container in the Microsoft Artifact Registry located at “https://mcr.microsoft.com”. You can browse Microsoft’s repository for other developer containers. You can also browse other repositories for containers.

14.4 Development and Customization

As you add files to your project, they should be available in your dev container and appear in VSCode. Developing in a dev container is a broad subject, beyond the scope of this article. You should know that you can customize and fine-tune your dev container by modifying devcontainer.json file.

You are encouraged to read more here: https://code.visualstudio.com/docs/devcontainers/containers

14.5 Ready to Develop!

Pin your VSCode session on the Windows toolbar so you can quickly launch it next time.

1. Right-click the VSCode icon on the Windows toolbar at the bottom of the screen.
2. Under “Recent Folders” hover over “go [Dev Container]”
3. Click the pin icon.
4. Next time you want to launch your Golang developer container, you will find it in the Pinned.

5. Just Right-click the VSCode icon.

At this point your development environment is ready to use. You are open for business!

15 Other Considerations

As you go, you'll no doubt want to add things to your environment to make life easier. Read the following subsections regarding helpful additions and fine-tuning.

15.1 VSCode Extensions

VSCode has a nice system for adding extensions. Earlier we used it to add the Remote Development extension pack so that we connect VSCode in to our remote alpha VM.

There are thousands of VSCode extensions. Some useful, others not so.

Through experience and habit I have found the following extensions to help me in my work.

- Code Spell Checker: by Street Side Software
- vscode-pdf: by tomoki1207
- Markdown PDF: by yzane
- Docker: by Microsoft
- Go: by Go Team at Google

To install them and other extensions goto Ctrl+Shift+X in VSCode.

15.2 Docker repositories

The main public repositories are not the only container repositories that exist. Other semi-public and government sponsored repositories also exist. Private corporate repositories may also exist, for example hosted with Artifactory.

- DockerHub (https://hub.docker.com/)
- RedHat Quay (https://access.redhat.com/products/red-hat-quay)
- Microsoft Artifact Registry (https://mcr.microsoft.com/)
- IronBank (https://p1.dso.mil/ironbank)

You will want to setup Docker or Podman to recognize all the container repositories you plan to use for your work. You may also want to setup Docker or Podman to automatically login to those repositories when you begin a development session.

16 Conclusion

This article described the setup of a basic remote containerized Golang development environment. While you may not be writing Golang code, you can follow the general steps to setup an environment for whatever language your project needs.

Every project is different. Your software development project will be different than mine. I guarantee it!

- Instead of writing Golang code, you may be writing Java code.
- Instead of hosting a remote VM in VirtualBox, you remote VM maybe running in an AWS EC2 instance.
- Instead of running Ubuntu in your VM, you may be running RHEL.
- Instead of containers in Docker you may have containers in Podman.
- Instead of VSCode on a Windows Laptop, you may have Emacs on a Linux desktop box.

Yes, each of these scenarios are different. However, despite the differences from project to project, the general approach to setting up a development environment is similar. You can take the steps that are laid out in this article and substitute whichever technologies you are using for your project.

What is important is the process. Recognize the patterns!

Best wishes on your software development project.

17 Acronyms, Abbreviations and Terminology

- 1Password : A password manager (https://1password.com/)
- APT : Advanced Package Tool (https://en.wikipedia.org/wiki/APT_(software))
- AWS : Amazon Web Service (https://aws.amazon.com/)
- Azure : Microsoft cloud platform (https://azure.microsoft.com/)
- BASH : Bourne Again Shell (https://en.wikipedia.org/wiki/Bash_(Unix_shell))
- Bitwarden : A password manager (https://bitwarden.com/)
- BSD : Berkeley Software Distribution (https://en.wikipedia.org/wiki/Berkeley_Software_Distribution)
- Buildah : Container image tool (https://buildah.io/)
- CD : Compact Disk
- CMD : Windows command shell
- Debian : Linux Distribution (https://www.debian.org/)
- DevOps : Developer Operations
- Distro : Distribution
- Docker : Container Tool (https://www.docker.com/)
- DockerHub : Container library (https://hub.docker.com/)
- DVD : Digital Video Disc (https://en.wikipedia.org/wiki/DVD)
- EC2 : Amazon Elastic Compute Cloud (https://en.wikipedia.org/wiki/Amazon_Elastic_Compute_Cloud)
- Emacs : Family of text editors (https://en.wikipedia.org/wiki/Emacs)

- ESXi : Hypervisor by VMWare (https://en.wikipedia.org/wiki/VMware_ESXi)
- Fedora : A Linux Distribution (https://fedoraproject.org/)
- Git : Source version control system (https://git-scm.com/)
- GitHub : (https://github.com/)
- GitLab : (https://gitlab.com/)
- Golang : Go Programming Language (https://go.dev/)
- HTTPS : Hypertext Transfer Protocol Secure (https://en.wikipedia.org/wiki/HTTPS)
- HyperV : Hypervisor by Microsoft (https://en.wikipedia.org/wiki/Hyper-V)
- IDE : Integrated Development Environment
- IP : Internet Protocol
- IPv4 : Internet Protocol version 4 address
- ISO : International Organization for Standardization virtual machine image
- IT : Information Technology
- Java : Java programming Language (https://www.oracle.com/java/)
- JSON : JavaScript Object Notation (https://en.wikipedia.org/wiki/JSON)
- KeePassXC : Password manager (https://keepassxc.org/)
- Linux : Operating system kernel (https://en.wikipedia.org/wiki/Linux)
- NAT : Network Address Translation (https://en.wikipedia.org/wiki/Network_address_translation)
- onprem : On Premises installation
- OpenSSH : An implementation of SSH (https://www.openssh.com/)
- PAT : Personal Access Token
- PC : Personal Computer
- Podman : Podman container tool (https://podman.io/)
- ProxMox : Hypervisor and virtual environment (https://www.proxmox.com/)
- Repo : Repository
- RHEL : Red Hat Enterprise Linux (https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux)
- SSH : Secure SHell protocol (https://en.wikipedia.org/wiki/Secure_Shell)
- Ubuntu : Ubuntu Linux Distribution https://ubuntu.com/
- URL : Uniform Resource Locator (https://en.wikipedia.org/wiki/URL)
- VirtualBox : Oracle VirtualBox Hypervisor (https://www.virtualbox.org/)
- VM : Virtual Machine (Virtual Personal Computer)

- VMWare : Technology company acquired by Broadcom (https://www.vmware.com/)
- VSCode : Visual Studio Code (https://code.visualstudio.com/)
- vSphere : A hypervisor by VMWare (https://en.wikipedia.org/wiki/VMware_vSphere)
- WSL : Windows Subsystem for Linux (https://en.wikipedia.org/wiki/Windows_Subsystem_for_Linux)
- YUM : Yellowdog Updater Modified (https://en.wikipedia.org/wiki/Yum_(software))