SELF HOST VPN

PROJECT REPORT

18 CSE344T – CLOUD ARCHITECTURE

(2018 Regulation)

II Year/ III Semester

Academic Year: 2022 -2023

By

Kolli loka Prashanth reddy(RA2011028010111)

FACULTY OF ENGINEERING AND TECHNOLOGY

SCHOOL OF COMPUTING

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Kancheepuram , NOVEMBER 2022

BONAFIDE

This is to certify that the project report titled " SELF HOST VPN" is the bonafide work of KOLLI LOKA PRASHANTH REDDY(RA2011028010111) who carried out the task of completing the project within the allotted time under my supervision.

Signature of the Course Faculty

Dr. DEEPA TILAK
Assistant Professor
Department of NWC,
SRM Institute of Science and Technology

Table of Contents

OBJECTIVE:

Build an VPN using linode and openvpn

Allocating VPC server in linode

Installing and configuring Open vpn

Two– factor authentication

Automatic openvpn updates installer

ABSTRACT:

A Hosted VPN service is a virtual private network (VPN) hosted by a thirdparty provider. The provider is responsible for the administration, maintenance, and general support of the hosting service and account. With a hosted VPN, users can connect securely from any location to the network and access company data and applications.

Hosted virtual private networks provide users safe, encrypted web, desktop, and mobile access to an organization's cloud-based apps, data, and files.

As a Layer 3 solution for both LAN-to-LAN and WAN-to-LAN deployments, site-to-site Internet Protocol security (IPsec) is used by cloud VPNs to create an encrypted tunnel to safeguard data in transit between network peers.

To secure communications over IP networks, IPsec cryptographic security keys are used to establish protocols for mutual authentication between clients at the outset of a session and again at any point during the session.

Because of this, Cloud VPNs can create a secure, encrypted connection between two networks, simulating a direct connection through a physical router. Cloud VPN traffic can also be managed with the help of rules and policies, just like traditional network connections configured on the user's end.

While a hosted VPN service does not provide the same level of control, scalability, and security that a self-hosted VPN can, it is a good option for companies who do not have the in-house IT personnel and infrastructure to fully manage their own VPN. A hosted VPN service provides these companies with a way to gain secure access to their cloud-based resources.

The main difference between the services comes down to the administration and maintenance of the services and visibility and control over network connections. Some providers can manage multiple customers on a single network with ease.

Compliance is also something to consider when comparing hosted and self-hosted VPN services. While a self-hosted VPN grants complete control over your network and allows you to meet compliance requirements, a hosted VPN might not be as easily managed.

Existing systems/methods :

Windows has in bulit which provides basic functionalities.

Many third party VPN service providers offering their service

through subscription model or free tier with ad supported model.

Many browsers have in house basic VPN like opera browser ,

Drawbacks in existing Systems

But they can't be trusted always, because they can access the data logs at the other end tunnel. Different countries have different laws on VPN usage and operation.

VPN vendors have comply with regional law in order to operate. Vendors need to share their user data logs to federal governments like India.

VPN is used for data privacy and routing to access content banned by regional internet service provider out of many applications it can offer.

Whereas, Self hosted VPN can be easily created with less complexity, it provides ability to configure settings and security.

Pre-requisite:

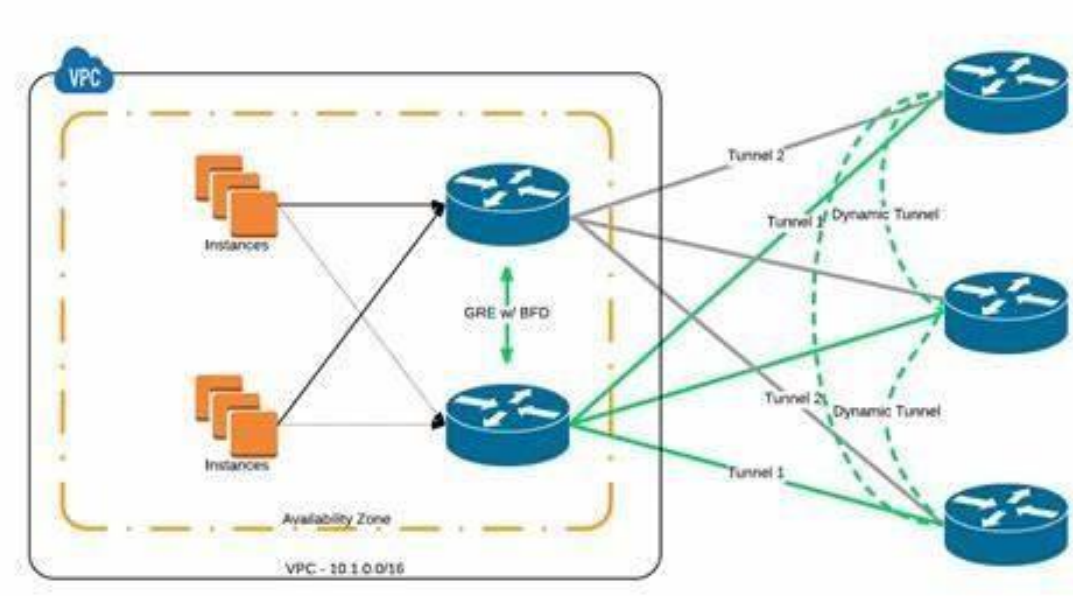
- **Basic understanding of linux kernel**
- **Basic understanding of working of linode**
- **Baic understanding of Open Vpn softaware suite**
- **Linode account**

Technologies:

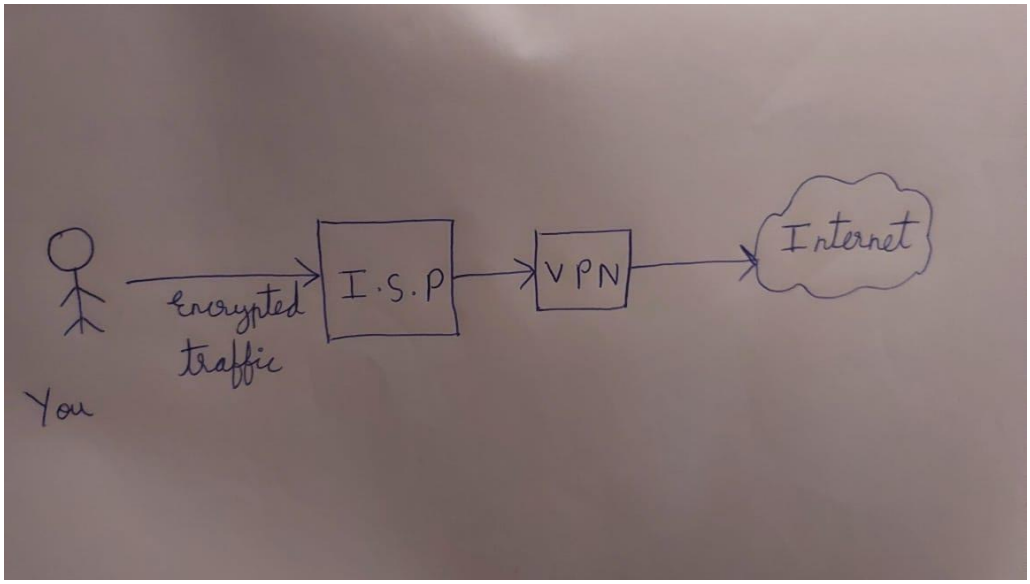
- **Open vpn**
- **linode**

Design

Architecture Diagram



Use case diagram



PROJECT SETUP

Choosing a provider

In order to set up a VPN we need to find where we're going to host it.

There are a lot of VPS providers that offer servers for as little as $2 per month, but there are a few things that you need to consider when choosing a VPS provider:

Creating a VPS

Next thing you need to do is add a server or as Linode calls it, a "linode"..

choose the location, I'm going to choose Mumbai since it's the closest to me physically.

We're going to take the cheapest "Nanode" plan.

After that you'll need to choose a root password and upload an SSH key, which we're not going to do now, i'll explain why later. Lastly tick a box that says "Private IP" and click the create button on the right... and there we go, our server is now created.

Generating SSH keys

Now you should see the control panel of your server, and while the server is starting, let's generate the SSH keys for it. Using a cleartext password to log in to your server is never a good idea since the password is not encrypted in transit and can be exposed on a hostile network. By creating an SSH key we're going to make it so that you can only log in to the server if you have the key file and the password, and at the same time the password is encrypted.

If you're using Linux you probably already know how to open the terminal, if you're on Mac you can find the Terminal app in your Applications folder, and on Windows 10 you'll need to open the PowerShell with administrator's privileges and install SSH using this command:

PS C:\> Add-WindowsCapability -Online -Name OpenSSH.Client*

This is the command that will generate our ssh keys. The RSA algorithm with 4096 key size is what I'd personally recommend, since it's sufficiently secure and widely supported.

ssh-keygen -t rsa -b 4096

Press Enter when asked for the key location to save it to the default one and then enter your password of choice.

Logging in to the server

By now our server has started up and we're ready to log in. Copy the IP address from the server control panel, go back to the terminal and type in **ssh root@ip-address**

Type yes, enter the root password that you specified in the first step and that's it, we're in.

Updating the OS

First and foremost, let's update our operating system and software: apt-get

update && apt-get upgrade

I'll also install my favourite text editor, feel free to use whatever you want though, for example nano. apt install neovim

Creating a user

As much as it's convenient to not have to enter a password every time, we need to create a user account that isn't root. Exposing root login on an SSH server is probably not a good idea even if you have multi factor authentication. Call me paranoid, but I think having to enter root password sometimes is the price I'm willing to pay for some sense of security. Type **useradd -G sudo -m nitesh -s /bin/bash**

That's going to create a user, set bash as default shell for him and permit sudo usage.

Afterwards we'll need to create a password for our user, using passwd

nitesh

Enter your password twice and we're good to go.

Copying SSH key from host to the server

Now that we've created our user it's a good time to copy the public SSH key to the server. Open a second terminal window for your local terminal and enter:

Linux or Mac
ssh-copy-id nitesh@ip_address Windows
type $env:USERPROFILE\.ssh\id_rsa.pub | ssh ip-address "cat >> .ssh/authorized_keys"

You'll be prompted to enter your password and once you do, go back to the terminal window with your server. Don't close the other window yet.

Restricting SSH to key authentication

Now that we've copied the SSH keys to the server we have to restrict authentication to the public key only. Let's edit the sshd configuration file

nvim /etc/ssh/sshd_config

First of all, let's change the default port. This won't do much for security, but it will help with those obnoxious SSH scanners that try to log in with default credentials. Not much, but the security logs will definitely get easier to read. You can use any port that's not taken by other services, but I prefer to use 69. Nice

Port 22 Port
69

Next, we need to disable password authentication so that you're only able to log in using a public key.

PasswordAuthentication no

Last but not least, let's also disable root login

PermitRootLogin no

Now save the file and restart the sshd service using systemctl

restart sshd

Now without closing this window let's go back to our local machine and try to log in with our key:

ssh -i ~/.ssh/id_rsa nitesh@ip_address -p 69

If you see the prompt to enter your key password, that means we're good to go. It's also a good idea to verify that we can't log in with our password anymore:

ssh nitesh@ip_address -p 69

This should give us "Permission denied".

Creating a server alias

But you might have noticed that this command is kind of long and annoying to type, so let's fix that. Create a file in the ".ssh" folder in your home directory called "config" and edit it using your favourite text editor:

nvim ~/.ssh/config

Here we're going to create an alias for our VPS

Host niteshvpn # choose a name for your server
    User nitesh # the username of the user that we created
    Port 69
    IdentityFile ~/.ssh/id_rsa # that's the location of our key file
    HostName ip_address # that's the IP address of our server

Save and close, and now we can login to our server by simply typing ssh niteshvpn

If you also don't want to see this wall of text every time you login, type in touch .hushlogin

Setting up OpenVPN

Log in to your server and install wget if you haven't already. Sometimes it comes with your OS image already, but sometimes it doesn't.

sudo apt install wget

Next, type wget, press Space and paste the link that you copied earlier. Press Enter

Now let's launch the script **sudo bash openvpn-install.sh**

The script will ask you some questions, and in most cases you'll want to pick the default answer. For the port, you can either choose the default port, 1194, but I prefer to choose 443, since 1194 is known as "the OpenVPN port" and in some cases it can be blocked on your network. 443 is the same port that is used for HTTPS, but whereas HTTPS uses TCP, OpenVPN (in this configuration) uses UDP, so they won't conflict with each other.

You're also going to be asked which DNS you want to use. Feel free to choose whatever you like, but I normally choose 1.1.1.1

As for the client name, choose whatever you like.

Now that the configuration is done, press any key and the installation process is going to start. It's fully automated and at the end you'll going to get a configuration file that we'll download to our local machine later on. The problem is that the

script places the file in the root directory by default, and in order to download it later, we need to move it to our user's home directory and give ourselves the correct priviliges:

sudo mv /root/thinkpad.ovpn ~ sudo
chown nitesh thinkpad.ovpn

With this out of the way there's only one thing left to be done on the server's side, and that's to disable the logs. Let's edit the config file:

sudo nvim /etc/openvpn/server/server.conf

And change verb 3 to verb 0 Now restart the

OpenVPN service: systemctl restart openvpn-

server@server.service

Downloading the config file

Now all we need to do is to download the configuration file to our local machine so that we can actually use the VPN. Open a terminal on your local machine and type in sftp servername Next, download the file using the command get configname.ovpn. And finally type exit

Installing mosh

Now, ssh is nice but it does get annoying sometimes, especially when you change your network and your connection drops immediately. Instead, I prefer to use mosh. There's no complicated config file shenenigans, you just install mosh on both your local and your remote machine, and after that you can simply use the mosh command as a drop-in replacement for ssh

Setting up multi-factor authentication

Now, public key authentication is probably secure enough for most, but if you want to be extra fancy, you can also add MFA or multi-factor authentication. The way it works is you install an app on your phone (there are a lot of open source apps on Android like AndOTP) and every time you log in you get a one time password in the app which you need to enter in order to log in. This provides an additional layer of security for your server which can be useful for some of us who are especially paranoid.

The first thing you have to do is to install google-authenticator-libpam Yes, the protocol is made by Google, but it's completely open-source and you don't have to use the Google Authenticator app on your phone, there are many open source options as I've already mentioned **sudo apt install libpam-google-authenticator**

After that, launch the initialization script by typing google-authenticator . There, basically answer yes to all questions except for the one about multiple users and the one about 30 second tokens.

Once you've done that, you might have noticed a big QR code in your command line as well as the recovery codes. Make sure to write those codes down somewhere save, they'll be useful in case you lose the access to the app on your phone. After that what you need to do is launch the authenticator app on your phone, I'll use OTP Auth, add a new account and chooe "Scan a QR code". After you scan the code, the account will be added to the app. And we're done with the phone part for now.

Let's go back to the server terminal and edit the authentication settings file for sshd:

sudo nvim /etc/pam.d/sshd

Here we'll comment out the line that says @include common-auth. Normally the two factor authentication will ask you for your user password and the one time password, but since we're already using a public key with the password, having to enter your password twice is slightly annoying. That way you'll only have to enter the public key password and the one time password.

Next we need to add this line to the end of the file:

auth required pam_google_authenticator.so

Let's save the file and quit. Now we need to edit the SSHD configuration file to make SSH aware of the new authentication method:

sudo nvim /etc/ssh/sshd_config

Here we need to change the following lines:

ChallengeResponseAuthentication yes UsePAM
yes

And add a new line after UsePAM that says:

AuthenticationMethods publickey,password publickey,keyboard-interactive

That's it, let's save the file and exit. And now let's restart the SSH service for the changes to take effect:

sudo systemctl restart sshd

As I mentioned in the beginning, it's always a good idea to try and log in in a separate terminal window without closing the server session. Otherwise if you messed up you'll be locked out of the SSH and obviously you don't want that.

If you try to log in now you'll see that apart from the usual public key password you're also going to be asked for the one time password from your app. Once again, if you're using Gnome, you won't be prompted for the public key until you log out and log back in again, only the one time password from your phone app. Let's enter the password and voila! Now our server is secured by two-factor authentication.

Unattended upgrades

One last thing that I want to show you today is unattended software upgrades. What this means is we're going to have a script that runs apt update and apt upgrade regularly, thus liberating us from the burden of having to log in to the server and do this manually. The server will also be rebooted for kernel updates, but since the reboot takes less than a minute, and since kernel updates are not very frequent, your VPN won't actually have much downtime because of the upgrades. You can also disable the automatic reboots if you prefer to do it yourself. So the first thing we need to do is to install the unattended-upgrades package:

sudo apt install unattended-upgrades apt-listchanges bsd-mailx

Next, enable the stable security updates: sudo dpkg-

reconfigure -plow unattended-upgrades

After that's done, let's edit the config file

sudo nvim /etc/apt/apt.conf.d/50unattended-upgrades

Here we need to set our email address which is going to be used for update notifications:

Unattended-Upgrade::Mail "mail@example.com";

And then also enable automatic reboots, set up cleanup tasks for removing unused kernels and set the automatic reboot time at 5AM.

Unattended-Upgrade::Automatic-Reboot "true"; # this is kind of obvious

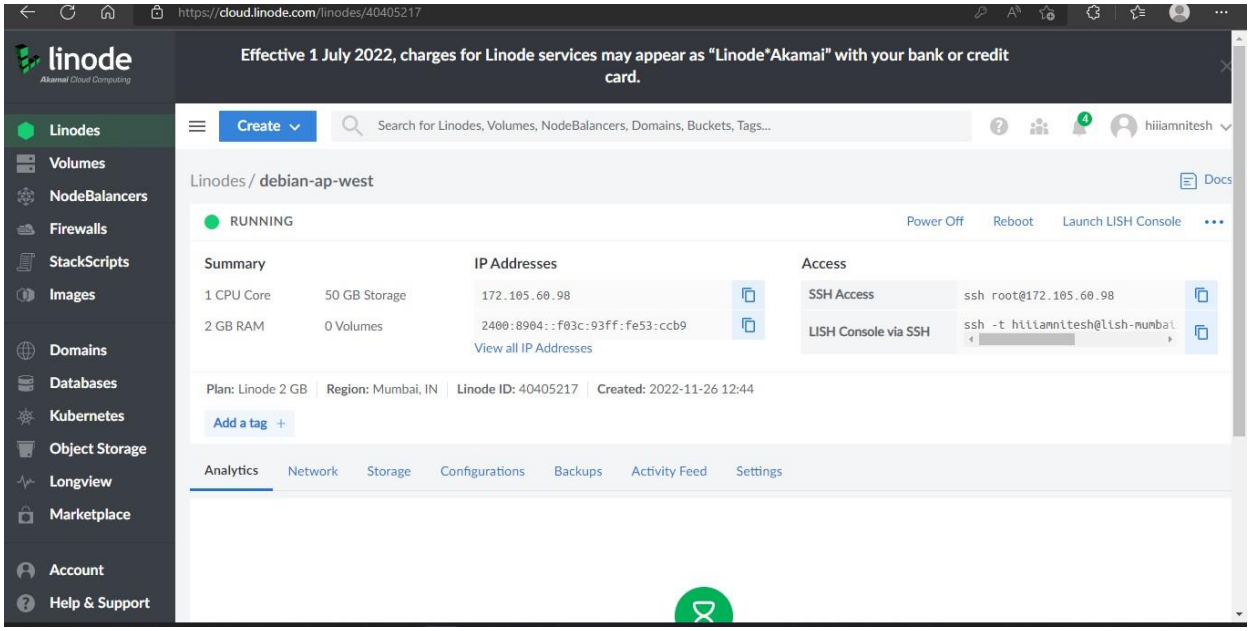
Unattended-Upgrade::Remove-Unused-Kernel-Packages "true";
Unattended-Upgrade::Remove-Unused-Dependencies "true";
Unattended-Upgrade::Automatic-Reboot "true";
Unattended-Upgrade::Automatic-Reboot-Time "05:00"; # here we'll specify when we want our system to reboot sudo unattended-upgrades --dry-run

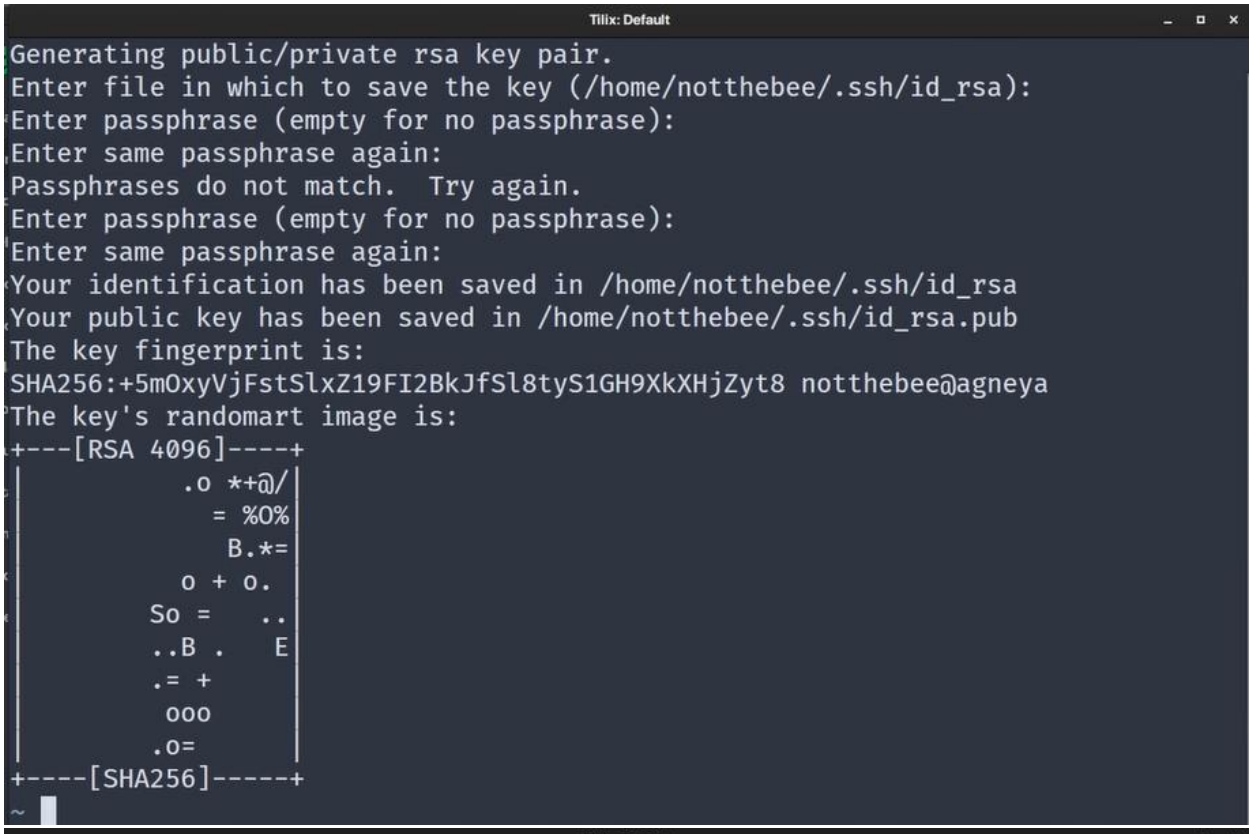
So now your system and all the packages will be updated automatically and you'll get an email every time an upgrade has been performed.

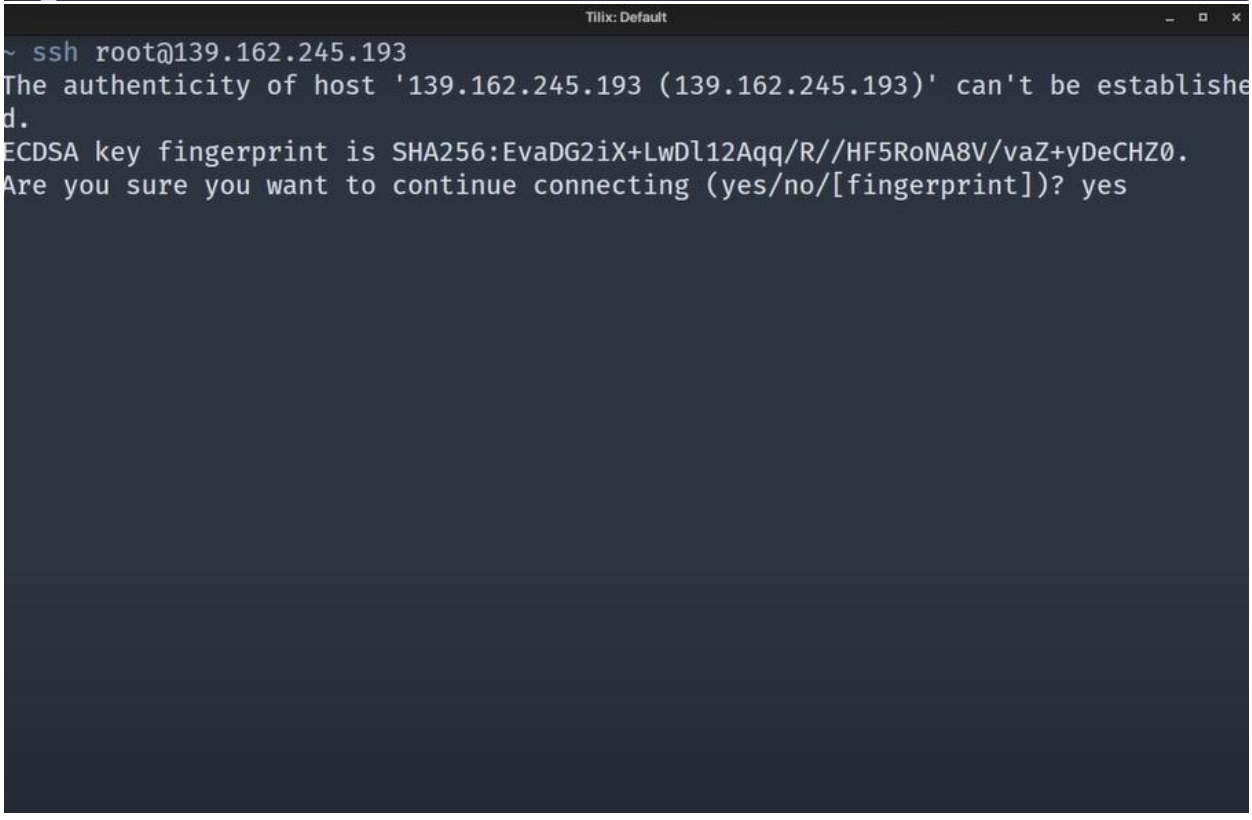
Componenets:

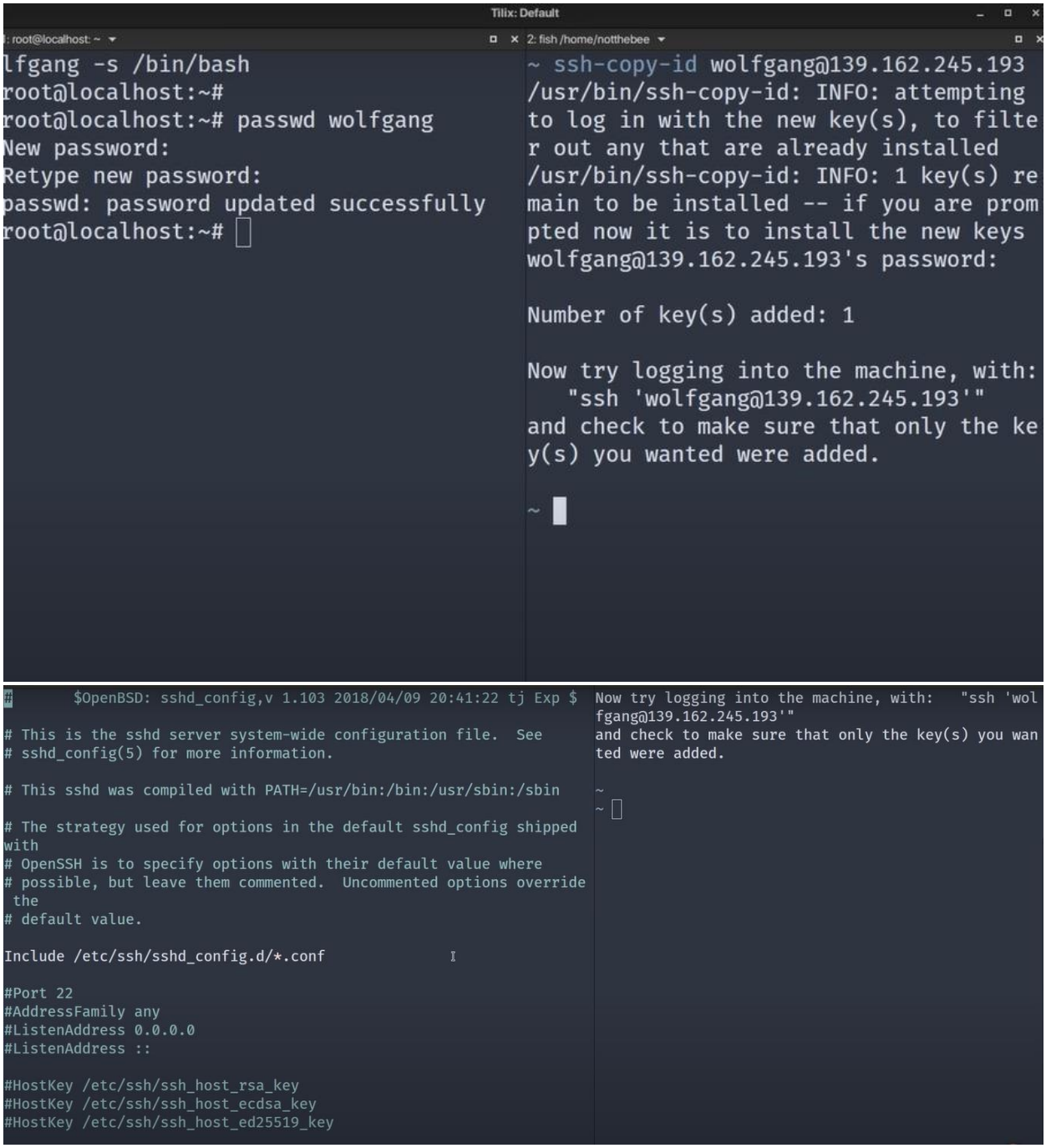
- **SSH KEY**
- **PACKAGES UPADATE**
- **ROOT USER**
- **NON-ROOT USER**
- **MULTI- FACTOR AUTHENTICATION**
- **MOSH**

CODE


https://cloud.linode.com/linodes/40405217
linode
Akamai Cloud Computing
Effective 1 July 2022, charges for Linode services may appear as "Linode*Akamai" with your bank or credit card.
Linodes
Volumes
NodeBalancers
Firewalls
StackScripts
Images
Domains
Databases
Kubernetes
Object Storage
Longview
Marketplace
Account
Help & Support
Create
Search for Linodes, Volumes, NodeBalancers, Domains, Buckets, Tags...
hiiiamnitesh
Linodes / debian-ap-west
Docs
RUNNING
Power Off
Reboot
Launch LISH Console
Summary
1 CPU Core
50 GB Storage
2 GB RAM
0 Volumes
IP Addresses
172.105.60.98
2400:8904::f03c:93ff:fe53:ccb9
View all IP Addresses
Access
SSH Access
ssh root@172.105.60.98
LISH Console via SSH
ssh -t hiiiamnitesh@lish-mumbai
Plan: Linode 2 GB
Region: Mumbai, IN
Linode ID: 40405217
Created: 2022-11-26 12:44
Add a tag
Analytics
Network
Storage
Configurations
Backups
Activity Feed
Settings


Tilix: Default
Generating public/private rsa key pair.
Enter file in which to save the key (/home/notthebee/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Passphrases do not match. Try again.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/notthebee/.ssh/id_rsa
Your public key has been saved in /home/notthebee/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:+5mOxyVjFstSlxZ19FI2BkJfSl8tyS1GH9XkXHjZyt8 notthebee@agneya
The key's randomart image is:
+---[RSA 4096]----+
| .o *+@/|
| = %O%|
| B.*=|
| o + o. |
| So =   .. |
| ..B .   E |
| .= + |
| ooo |
| .o= |
+----[SHA256]-----+
~


Tilix: Default
~ ssh root@139.162.245.193
The authenticity of host '139.162.245.193 (139.162.245.193)' can't be establishe
d.
ECDSA key fingerprint is SHA256:EvaDG2iX+LwDl12Aqq/R//HF5RoNA8V/vaZ+yDeCHZ0.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes


Tilix: Default
1: root@localhost: ~
2: fish /home/notthebee
lfgang -s /bin/bash
root@localhost:~#
root@localhost:~# passwd wolfgang
New password:
Retype new password:
passwd: password updated successfully
root@localhost:~#
~ ssh-copy-id wolfgang@139.162.245.193
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
wolfgang@139.162.245.193's password:
Number of key(s) added: 1
Now try logging into the machine, with:
"ssh 'wolfgang@139.162.245.193'"
and check to make sure that only the key(s) you wanted were added.
~

$OpenBSD: sshd_config,v 1.103 2018/04/09 20:41:22 tj Exp $
This is the sshd server system-wide configuration file. See
sshd_config(5) for more information.
This sshd was compiled with PATH=/usr/bin:/bin:/usr/sbin:/sbin
The strategy used for options in the default sshd_config shipped with
OpenSSH is to specify options with their default value where
possible, but leave them commented. Uncommented options override the
default value.
Include /etc/ssh/sshd_config.d/*.conf
#Port 22
#AddressFamily any
#ListenAddress 0.0.0.0
#ListenAddress ::
#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_ecdsa_key
#HostKey /etc/ssh/ssh_host_ed25519_key
Now try logging into the machine, with: "ssh 'wolfgang@139.162.245.193'"
and check to make sure that only the key(s) you wanted were added.
~
~

```
# This is the sshd server system-wide configuration file.  See
# sshd_config(5) for more information.

# This sshd was compiled with PATH=/usr/bin:/bin:/usr/sbin:/sbin

# The strategy used for options in the default sshd_config shipped with
# OpenSSH is to specify options with their default value where
# possible, but leave them commented.  Uncommented options override the
# default value.

Include /etc/ssh/sshd_config.d/*.conf

Port 22
#AddressFamily any
#ListenAddress 0.0.0.0
#ListenAddress ::

#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_ecdsa_key
#HostKey /etc/ssh/ssh_host_ed25519_key

# Ciphers and keying
#RekeyLimit default none
```

```
#AuthorizedKeysCommand none
#AuthorizedKeysCommandUser nobody

# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to no here!
#PasswordAuthentication yes
#PermitEmptyPasswords no

# Change to yes to enable challenge-response passwords (beware issues with
# some PAM modules and threads)
ChallengeResponseAuthentication no

# Kerberos options
#KerberosAuthentication no
#KerberosOrLocalPasswd yes
#KerberosTicketCleanup yes
#KerberosGetAFSToken no
```

```
#HostKey /etc/ssh/ssh_host_ed25519_key

# Ciphers and keying
#RekeyLimit default none

# Logging
#SyslogFacility AUTH
#LogLevel INFO

# Authentication:

#LoginGraceTime 2m
PermitRootLogin yes
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10

#PubkeyAuthentication yes

# Expect .ssh/authorized_keys2 to be disregarded by default in future.
#AuthorizedKeysFile     .ssh/authorized_keys .ssh/authorized_keys2

#AuthorizedPrincipalsFile none

#AuthorizedKeysCommand none
#AuthorizedKeysCommandUser nobody
```

```
Which IPv4 address should be used?
     1) 139.162.245.193
     2) 192.168.136.2
IPv4 address [1]: 1

Which protocol should OpenVPN use?
   1) UDP (recommended)
   2) TCP
Protocol [1]:
```

```
Which IPv4 address should be used?
     1) 139.162.245.193
     2) 192.168.136.2
IPv4 address [1]: 1

Which protocol should OpenVPN use?
   1) UDP (recommended)
   2) TCP
Protocol [1]:

What port should OpenVPN listen to?
Port [1194]: 443

Select a DNS server for the clients:
   1) Current system resolvers
   2) Google
   3) 1.1.1.1
   4) OpenDNS
   5) Quad9
   6) AdGuard
DNS server [1]: 3

Enter a name for the first client:
Name [client]: thinkpad
```

```
openssl is already the newest version (1.1.1f-1ubuntu2).
The following additional packages will be installed:
  libpkcs11-helper1
Suggested packages:
  resolvconf openvpn-systemd-resolved easy-rsa
The following NEW packages will be installed:
  libpkcs11-helper1 openvpn
0 upgraded, 2 newly installed, 0 to remove and 3 not upgraded.
Need to get 522 kB of archives.
After this operation, 1,343 kB of additional disk space will be used.
Get:1 http://mirrors.linode.com/ubuntu focal/main amd64 libpkcs11-helper1 amd64 1.26-1 [44.3 kB]
Get:2 http://mirrors.linode.com/ubuntu focal/main amd64 openvpn amd64 2.4.7-1ubuntu2 [478 kB]
Fetched 522 kB in 0s (26.7 MB/s)
Preconfiguring packages ...
Selecting previously unselected package libpkcs11-helper1:amd64.
(Reading database ... 72365 files and directories currently installed.)
Preparing to unpack .../libpkcs11-helper1_1.26-1_amd64.deb ...
Unpacking libpkcs11-helper1:amd64 (1.26-1) ...
Selecting previously unselected package openvpn.
Preparing to unpack .../openvpn_2.4.7-1ubuntu2_amd64.deb ...
Unpacking openvpn (2.4.7-1ubuntu2) ...
Setting up libpkcs11-helper1:amd64 (1.26-1) ...
Setting up openvpn (2.4.7-1ubuntu2) ...
 * Restarting virtual private network daemon.
Created symlink /etc/systemd/system/multi-user.target.wants/openvpn.service → /lib/systemd/system/openvpn
Processing triggers for systemd (245.4-4ubuntu3.1) ...
```

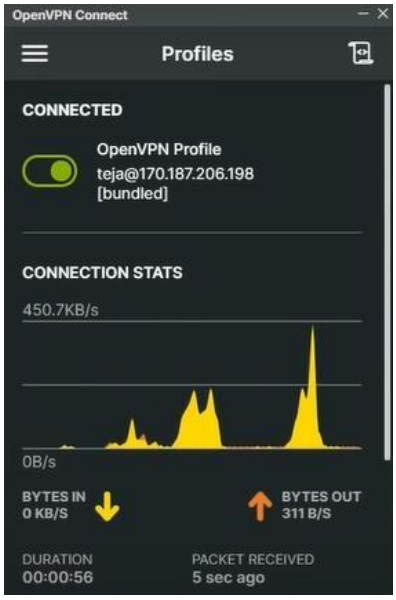
```
port 443
proto udp
dev tun
ca ca.crt
cert server.crt
key server.key
dh dh.pem
auth SHA512
tls-crypt tc.key
topology subnet
server 10.8.0.0 255.255.255.0
server-ipv6 fddd:1194:1194:1194::/64
push "redirect-gateway def1 ipv6 bypass-dhcp"
ifconfig-pool-persist ipp.txt
push "dhcp-option DNS 1.1.1.1"
push "dhcp-option DNS 1.0.0.1"
keepalive 10 120
cipher AES-256-CBC
user nobody
group nogroup
persist-key
persist-tun
status openvpn-status.log
verb
crl-verify crl.pem
explicit-exit-notify
```

```
~ sudo pacman -S mosh
[sudo] password for notthebee:
warning: mosh-1.3.2-11 is up to date -- reinstalling
resolving dependencies...
looking for conflicting packages...

Package (1)     Old Version  New Version  Net Change

community/mosh  1.3.2-11     1.3.2-11       0.00 MiB

Total Installed Size:  0.72 MiB
Net Upgrade Size:      0.00 MiB

:: Proceed with installation? [Y/n] y
(1/1) checking keys in keyring
(1/1) checking package integrity
(1/1) loading package files
(1/1) checking for file conflicts
(1/1) checking available disk space
:: Processing package changes...
(1/1) reinstalling mosh
:: Running post-transaction hooks...
(1/1) Arming ConditionNeedsUpdate...
~
```

```
# Standard Un*x authentication.
#@include common-auth

# Disallow non-root logins when /etc/nologin exists.
account    required     pam_nologin.so

# Uncomment and edit /etc/security/access.conf if you need to set complex
# access limits that are hard to express in sshd_config.
# account  required     pam_access.so

# Standard Un*x authorization.
@include common-account

# SELinux needs to be the first session rule.  This ensures that any
# lingering context has been cleared.  Without this it is possible that a
# module could execute code in the wrong domain.
session [success=ok ignore=ignore module_unknown=ignore default=bad]        pam_selinux.so close

# Set the loginuid process attribute.
session    required     pam_loginuid.so

# Create a new session keyring.
session    optional     pam_keyinit.so force revoke

# Standard Un*x session setup and teardown.
@include common-session

# Print the message of the day upon successful login.
# This includes a dynamically generated part from /run/motd.dynamic
# and a static (admin-editable) part from /etc/motd.
session    optional     pam_motd.so  motd=/run/motd.dynamic
session    optional     pam_motd.so noupdate
```

```
# Create a new session keyring.
session    optional     pam_keyinit.so force revoke

# Standard Un*x session setup and teardown.
@include common-session

# Print the message of the day upon successful login.
# This includes a dynamically generated part from /run/motd.dynamic
# and a static (admin-editable) part from /etc/motd.
session    optional     pam_motd.so  motd=/run/motd.dynamic
session    optional     pam_motd.so noupdate

# Print the status of the user's mailbox upon successful login.
session    optional     pam_mail.so standard noenv # [1]

# Set up user limits from /etc/security/limits.conf.
session    required     pam_limits.so

# Read environment variables from /etc/environment and
# /etc/security/pam_env.conf.
session    required     pam_env.so # [1]
# In Debian 4.0 (etch), locale-related environment variables were moved to
# /etc/default/locale, so read that as well.
session    required     pam_env.so user_readenv=1 envfile=/etc/default/locale

# SELinux needs to intervene at login time to ensure that the process starts
# in the proper default security context.  Only sessions which are intended
# to run in the user's context should be run after this.
session [success=ok ignore=ignore module_unknown=ignore default=bad]        pam_selinux.so open
```

```
#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_ecdsa_key
#HostKey /etc/ssh/ssh_host_ed25519_key

# Ciphers and keying
#RekeyLimit default none

# Logging
#SyslogFacility AUTH
#LogLevel INFO

# Authentication:

#LoginGraceTime 2m
PermitRootLogin no
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10

#PubkeyAuthentication yes

# Expect .ssh/authorized_keys2 to be disregarded by default in future.
#AuthorizedKeysFile     .ssh/authorized_keys .ssh/authorized_keys2

#AuthorizedPrincipalsFile none

#AuthorizedKeysCommand none
#AuthorizedKeysCommandUser nobody
```



References

https://gogetsecure.com/hosted-vpn-services/

https://www.kaspersky.com/resource-center/definitions/what-is-a-vpn

Conclusion

A VPN connection establishes a secure connection between you and the internet. Via the VPN, all your data traffic is routed through an encrypted virtual tunnel. This disguises your IP address when you use the internet, making its location invisible to everyone. A VPN connection is also secure against external attacks.

That's because only you can access the data in the encrypted tunnel – and nobody else can because they don't have the key. A VPN allows you to access regionally restricted content from anywhere in the world. Many streaming platforms are not available in every country. To secure communications over IP networks, IPsec cryptographic security keys are used to establish protocols for mutual authentication between clients at the outset of a session and again at any point during the session.

Because of this, Cloud VPNs can create a secure, encrypted connection between two networks, simulating a direct connection through a physical router. Cloud VPN traffic can also be managed with the help of rules and policies, just like traditional network connections configured on the user's end.