

Best Practices for SSH on a Production Cloud Server

Learn essential SSH security practices for production cloud servers, including key-based authentication, port changes, and access restrictions to protect your infrastructure.

Contents

01	Introduction	3
02	Prerequisites	3
03	Configure the Local Host SSH Client	4
04	Restrict the root Login and Disable Password Authentication	10
05	Configure an Alternate Port Number to Reduce Unauthorized Login Attempts	14
06	Configure Fail2ban to Reduce Brute-force Attacks	16
07	Configure SSH Agent Forwarding for Non-Root Users	18
08	How to Use SSH Tunneling (Port Forwarding)	21
09	Use Scp to Copy Files Between Hosts	24
010	Use Rsync to Backup Files and Folders	25
011	More Information	26

Introduction

SSH (Secure Shell) connects to a remote cloud server via a command-line or GUI client. SSH is a network protocol that is also used by other applications like scp and rsync. Scp copies files between hosts on a network. Rsync can use ssh to transfer files and folders between a local and remote host. Rsync transfers files and folders that have changed since the last transfer. Entire projects can be efficiently backed up via rsync.

This guide explains how to configure SSH on a local host and on a remote Vultr host, including how to use the Vultr Web Console to prevent accidental login lockouts. It explains how to:

- Configure the local host SSH client to use key-based authentication
- Restrict the root login and disable password authentication
- Configure an alternate port number to reduce unauthorized login attempts
- Configure fail2ban to reduce brute-force attacks
- Configure SSH Agent Forwarding for Non-Root Users
- How to use SSH Tunneling
- Use **scp** to copy files between hosts
- Use **rsync** to backup files and folders

This guide uses the OpenSSH ssh command-line client that is available on most Unix-like hosts, including macOS and Linux. Use the [Windows Subsystem for Linux](#) on a Windows 10/11 PC to access the OpenSSH ssh client.

For the examples in this guide, the Vultr server FQDN hostname is `ap1.example.com`. The local macOS host is `imac1`. The local Debian host is `db1`.

Note: The SSH commands work the same for a macOS or Debian Linux terminal session except as noted.

Prerequisites

You need a [Vultr Account](#). This guide uses macOS and the [Debian Linux OS](#) to show how to use SSH. But most other Linux environments that use the

OpenSSH ssh client should work. You first create a private/public SSH key on your local host and then install the public key in your Vultr Account. Then you can automatically install that SSH key when you deploy a Vultr server. Or you can install the SSH key later if you have already created the server.

You should be familiar with:

- Installing command-line applications on macOS via the [Homebrew](#) brew command or the apt command on Linux.
- Working with a firewall, such as UFW, to allow and disable ports.
- Working with the command line.

Note: Command lines starting with `$` use a non-privileged account, and commands starting with `#` use the root account.

Configure the Local Host SSH Client

1. Generate an SSH Key Pair

[Generate a modern SSH key pair](#) with the following command:

```
george@imac1:~  
$ ssh-keygen -t ed25519 -C "hostname"
```

Often the `-c` key comment is an email address. However, the key comment can be any text. You can set it to your hostname to manage which SSH keys belong to which host. Be sure to use a unique filename to save the key pair for each host and enter a passphrase when prompted. Use the `pwgen` command to generate a long and secure passphrase:

```
george@imac1:~  
$ pwgen -s 64 1  
H7zhGF06DeMAf1gvMCF0aWcbmLJrcE9VXI10j81zLBGN2GFpFhhxw8hT4qyyU4j l
```

Note: Use the brew or apt command to install pwgen.

Here is a complete example:

```
george@imac1:~
$ ssh-keygen -t ed25519 -C "ap1"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/Users/george/.ssh/id_ed25519): /Users/
george/.ssh/ap1_ed25519
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/george/.ssh/ap1_ed25519.
Your public key has been saved in /Users/george/.ssh/ap1_ed25519.pub.
The key fingerprint is:
SHA256:RHQ9bSUV9TwKahLLr0crwYZvMVIcxJiFbSG0+PDKzr8 ap1
The key's randomart image is:
+--[ED25519 256]--+
|      .0=+ .. ..+*|
|  . o+.* .  o oo.|
| o . .o.o . o .o|
| +  .+o . . . .|
|  o ++So .   |
|. . o *+.    |
| o  + =..    |
|o    +.o     |
| o.E. ..o    |
+-----[SHA256]-----+
```

Copy and paste your passphrase when prompted. The passphrase is not visible when you enter it. Entering the filename is optional. `~/.ssh/id_ed25519` is the default filename. Create a unique key pair for each host.

2. Install the Public Key

Log into your Vultr account. Under your account name, click SSH Keys. Click Add SSH Key. Create a name for your SSH key. Copy and paste your public key, `~/.ssh/ap1_ed25519.pub`, into the key field. Click Add SSH Key.

Deploy a new Debian 11 server. Select your SSH key in the `SSH Keys` section. The SSH key is automatically installed for the `root` user in `~/.ssh/authorized_keys`

Note: You may need to wait a few minutes after the server is running for Cloud-init to complete. Open the View Console window and wait for the Cloud-init finished message.

If you have already created your server, you can select `Reinstall SSH Keys` under the Settings tab and (re)install your SSH key.

Note: This deletes all your data and reinstalls the server OS.

You can use `ssh-copy-id` to install your key on an existing server. This command installs the public SSH key in `~/.ssh/authorized_keys` on your Vultr host. Before executing this command, be sure that `ap1.example.com` exists in your DNS data or in `/etc/hosts` on your local host. Enter your root password when prompted:

```
george@imac1:~
$ ssh-copy-id -i ~/.ssh/ap1_ed25519 root@ap1.example.com
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/Users/george/.ssh/ap1_ed25519.pub"
The authenticity of host 'ap1.example.com (66.42.117.31)' can't be established.
ECDSA key fingerprint is SHA256:fqAViH4GleZ07oMYkr+Qu92kewU7xzZeeUSmfPDME0A.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/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
root@ap1.example.com's password:

Number of key(s) added:          1

Now try logging into the machine, with:  "ssh 'root@ap1.example.com'"
and check to make sure that only the key(s) you wanted were added.
```

Note: The authenticity of host message appears the first time when connecting to the server. Answer `yes` to continue connecting. The example ssh command to log into the server does not work unless you use the default identity filename, `~/.ssh/id_ed25519`. Please refer to the next section, Test the Public Key. If you previously used SSH to log into your server and then you reinstalled your server, you may see this error message:

```
$ ssh-copy-id -i ~/.ssh/ap1_ed25519 root@ap1.example.com
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/Users/george/.ssh/
ap1_ed25519.pub"
/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: ERROR:
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
ERROR: @    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!    @
ERROR: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
ERROR: IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
  o o o
```

Just delete the previous ap1 entry in `~/.ssh/known_hosts` and run the command again.

3. Test the Public Key

Execute the following command to verify that you can log into your server using your SSH key:

```
george@imac1:~
$ ssh -i ~/.ssh/ap1_ed25519 root@ap1.example.com
Enter passphrase for key '/Users/george/.ssh/ap1_ed25519':
Linux ap1 5.10.0-11-amd64 #1 SMP Debian 5.10.92-1 (2022-01-18) x86_64
  o o o
root@ap1:~#
```

The `-i` option is not required if you use the default identity name, `id_ed25519`. You have to enter the passphrase every time unless you use the SSH authentication agent, `ssh-agent`, to hold the private keys. Use the `ssh-add` command to store the private keys in the ssh-agent:

```
george@imac1:~
$ ssh-add -K ~/.ssh/ap1_ed25519
Enter passphrase for /Users/george/.ssh/ap1_ed25519:
Identity added: /Users/george/.ssh/ap1_ed25519 (ap1)
```

Note: The `-K` option stores the passphrase in the macOS keychain to save it over macOS restarts. It does not work with the Linux version of `ssh-add`. Refer to the next section, [Create the Local SSH Configuration File](#), to see how to automatically add the passphrase to the `ssh-agent` after starting the first `ssh` session.

Both macOS and Debian Linux use the `ssh-agent` to store private keys. The `ssh-agent` starts automatically and works with the macOS keychain to store the keys. For the Debian Linux server terminal session, the `ssh-agent` starts for each terminal session after making the following environment changes:

```
george@db1:~
$ echo 'eval `ssh-agent` > /dev/null' >> ~/.profile

george@db1:~
$ echo 'eval `ssh-agent -k` > /dev/null' >> ~/.bash_logout
```

Restart your local Linux terminal session after making the above changes. Use `ssh-add` to store the private key:

```
george@db1:~
$ ssh-add ~/.ssh/ap1_ed25519
Enter passphrase for /home/george/.ssh/ap1_ed25519:
Identity added: /home/george/.ssh/ap1_ed25519 (ap1)
```

You can now connect to the server without using a passphrase during the Linux terminal session.

4. Create the Local SSH Configuration File

You can create an [SSH configuration file](#) that stores SSH options for hosts and users. This simplifies the SSH command line used to connect to the server. You can also define a host alias instead of entering the host FQDN. The SSH config file contains global and per-host options. Here is the `~/.ssh/config` file for the example server:

```
Host *
  AddKeysToAgent yes
```

```
UseKeychain yes
IdentitiesOnly yes
AddressFamily inet

Host ap1.example.com ap1
  Hostname ap1.example.com
  Port 22
  User george
  IdentityFile ~/.ssh/ap1_ed25519
```

Note: For Linux, delete the UseKeychain option.

The `Host *` line defines the global options. The `AddKeysToAgent` option adds the private keys to the `ssh-agent`. The **macOS only** `UseKeychain` option adds the passphrase to the macOS keychain. The `IdentitiesOnly` option limits the authentication entity to the one defined by the `IdentityFile`. The `AddressFamily` option specifies using IPv4 when connecting. The `Host ap1.example.com ap1` line defines two host aliases for the ssh connection. It includes the FQDN and the short hostnames. The next lines define the DNS FQDN host, port number, user, and the authentication identify file. The `Port` defaults to the standard port 22 if not specified, but the option can define a non-standard port number. The `User` option defines a default user if the user is not specified on the ssh command line.

With this `~/.ssh/config` file in place, connect to the server with the short alias name:

```
george@imac1:~
$ ssh root@ap1
Enter passphrase for key '/Users/george/.ssh/ap1_ed25519':
Linux ap1 5.10.0-11-amd64 #1 SMP Debian 5.10.92-1 (2022-01-18) x86_64
 o o o
root@ap1:~#
```

Enter the passphrase when making the first ssh connection to the server. The identity is automatically added to the ssh-agent:

```
george@imac1:~
$ ssh-add -l
256 SHA256:XoidSmFlhy1gx5wiYDMMPYFvNY5xiRdwEi/i61rNVmA ap1 (ED25519)
```

The macOS keychain contains the passphrase. To verify, open the macOS KeyChain App, click the default login chain, and search for ap1. For Linux, the passphrase remains available during the life of the terminal session.

When connecting to the default user on the server, just do:

```
george@imac1:~  
$ ssh ap1  
Linux apnet1 5.10.0-13-amd64 #1 SMP Debian 5.10.106-1 (2022-03-17) x86_64  
  o o o  
george@ap1:~  
$
```

Note: This example assumes that the default user account is available on the server.

Restrict the root Login and Disable Password Authentication

It's not good practice to allow the root user to log in to the server via SSH. You should use the Vultr Web Console or log in as a regular user and then switch to the root account. This maintains a log of regular users accessing the root account. Before restricting root SSH access and disabling password authentication, you need to create a regular user with sudo privileges.

1. Create the Default Regular User

Log into your root account and create the regular user and give them sudo permissions:

```
root@ap1:~# adduser george
```

Give the user a secure password when prompted. Fill in the additional user information as desired. Add the user to the sudo group:

```
root@ap1:~# usermod -aG sudo george
```

Verify that you can log into the new user account:

```
$ ssh ap1
george@ap1.example.com's password:
Linux ap1 5.10.0-11-amd64 #1 SMP Debian 5.10.92-1 (2022-01-18) x86_64
 o o o
george@ap1:~$
```

Installed the SSH public key for the new user:

```
george@imac1:~
$ ssh-copy-id -i ~/.ssh/ap1_ed25519 ap1
```

Enter the user password when prompted.

Verify that you can log into the user account without a password:

```
george@imac1:~
$ ssh ap1
Linux ap1 5.10.0-11-amd64 #1 SMP Debian 5.10.92-1 (2022-01-18) x86_64
 o o o
george@ap1:~$
```

Verify that you use sudo to change to the root account:

```
george@ap1:~$ sudo -i
```

```
We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:
```

- #1) Respect the privacy of others.
- #2) Think before you type.
- #3) With great power comes great responsibility.

```
[sudo] password for george:
root@ap1:~#
```

Enter your user password when prompted. The sudo command remembers your password for 15 minutes.

Note: For the Debian OS, the trust message prints the first time you use sudo.

You can see the root account accessed by the regular user in `/var/log/auth.log`:

```
Apr  5 21:09:34 ap1 sudo:    george : TTY=pts/1 ; PWD=/root ; USER=root ; COMMAND=/bin/
bash
Apr  5 21:09:34 ap1 sudo: pam_unix(sudo:session): session opened for user root(uid=0)
by george(uid=1000)
Apr  5 21:09:45 ap1 sudo: pam_unix(sudo:session): session closed for user root
```

You can also use su to access the root account:

```
george@ap1:~$ su -
Password:
root@ap1:~#
```

Note: Enter the **root** password.

Again, you can monitor this access in the `/var/log/auth.log`:

```
Apr  5 21:12:38 ap1 su: (to root) george on pts/1
Apr  5 21:12:38 ap1 su: pam_unix(su-l:session): session opened for user root(uid=0) by
george(uid=1000)
Apr  5 21:12:49 ap1 su: pam_unix(su-l:session): session closed for user root
```

Normally, you use sudo to execute individual commands that require elevated privileges.

For example:

```
george@ap1:~$ ls -l /root
ls: cannot open directory '/root': Permission denied

george@ap1:~$ sudo ls -l /root
total 4
drwxr-xr-x 2 root root 4096 Apr  5 20:35 orig
```

Looking again at `/var/log/auth.log`:

```
Apr  5 21:21:13 ap1 sudo:    george : TTY=pts/1 ; PWD=/home/george ; USER=root ;  
COMMAND=/usr/bin/ls -l /root  
Apr  5 21:21:13 ap1 sudo: pam_unix(sudo:session): session opened for user root(uid=0)  
by george(uid=1000)  
Apr  5 21:21:13 ap1 sudo: pam_unix(sudo:session): session closed for user root
```

2. Restrict root Access and Disable Password Authentication

As root, make the following changes to `/etc/ssh/sshd_config`:

- Change `PermitRootLogin yes` to `PermitRootLogin no`
- Change `#PasswordAuthentication yes` to `PasswordAuthentication no`. Make sure to remove the `#` comment at the beginning of the line if it exists.

Restart the `sshd` daemon:

```
george@ap1:~$ sudo systemctl restart sshd
```

Try to log into the root and user account from a host that does not have SSH authentication (or disable it):

```
george@db1:~  
$ ssh root@ap1  
root@ap1.example.com: Permission denied (publickey).  
  
george@db1:~  
$ ssh george@ap1  
george@ap1.example.com: Permission denied (publickey).
```

Even with SSH authentication enabled, the root authentication fails:

```
george@imacl:~  
$ ssh root@ap1  
root@ap1.example.com: Permission denied (publickey).
```

But the user account access works:

```
george@imac1:~  
$ ssh george@ap1  
Linux ap1 5.10.0-11-amd64 #1 SMP Debian 5.10.92-1 (2022-01-18) x86_64  
o o o  
george@ap1:~$
```

Configure an Alternate Port Number to Reduce Unauthorized Login Attempts

Changing the SSH port number reduces the number of unauthorized login attempts. You can see these login attempts in the `/var/log/auth.log` file:

```
george@ap1:~$ sudo grep sshd /var/log/auth.log | grep root  
Apr 6 16:28:42 ap1 sshd[30840]: Disconnected from authenticating user root  
43.133.165.86 port 48314 [preauth]  
Apr 6 16:31:49 ap1 sshd[30887]: Disconnected from authenticating user root  
43.133.165.86 port 48382 [preauth]  
Apr 6 16:34:58 ap1 sshd[30895]: Disconnected from authenticating user root  
43.133.165.86 port 48448 [preauth]  
o o o
```

SSH uses a privileged port (less than 1024) by design. It can use any port number that is not assigned to another service. Look at `/etc/services` to verify unassigned port numbers. For example, to change the SSH port number to 522, do:

1. If the firewall is active, allow port 522:

```
george@ap1:~$ sudo ufw allow 522/tcp
```

Note: This example uses the UFW firewall, which is active on a new Vultr Debian/Ubuntu instance.

2. Change the port number to 522 in `/etc/ssh/sshd_config` and restart the sshd daemon:

```
/etc/ssh/sshd_config:
< #Port 22
---
> Port 522

george@ap1:~$ sudo systemctl restart sshd
```

Your ssh session should remain active. However, use the Vultr Web Console if disconnected and review your changes.

3. Verify that you cannot connect using port 22:

```
george@imac1:~
$ ssh ap1
ssh: connect to host ap1.example.com port 22: Connection refused
```

4. Update your local host `~/.ssh/config` file to use port 522 for the ap1 host and verify that you can connect to the server:

```
Host ap1.example.com ap1
  Hostname ap1.example.com
  Port 522
  User george
  IdentityFile ~/.ssh/ap1_ed25519

george@imac1:~
$ ssh ap1
Linux ap1 5.10.0-11-amd64 #1 SMP Debian 5.10.92-1 (2022-01-18) x86_64
o o o
george@ap1:~$
```

5. If the firewall is active, disable port 22:

```
george@ap1:~$ sudo ufw delete allow 22
```

If you are using the [Vultr Firewall](#), you should follow the same procedure.

Configure Fail2ban to Reduce Brute-force Attacks

Fail2ban works with the **server installed** firewall, for example, UFW, to ban abusive hosts that make multiple login attempts during a short interval.

Install fail2ban:

```
george@ap1:~$ sudo apt install fail2ban
```

By default, SSH protection is active:

```
george@ap1:~$ sudo fail2ban-client status
Status
|- Number of jail:  1
`- Jail list:      sshd

george@ap1:~$ sudo fail2ban-client status sshd
Status for the jail: sshd
|- Filter
|  |- Currently failed:  0
|  |- Total failed:  0
|  `-- File list:  /var/log/auth.log
`- Actions
   |- Currently banned:  0
   |- Total banned:  0
   `-- Banned IP list:
```

However, if you have changed the SSH port number, you need to update the fail2ban SSH configuration by creating the `/etc/fail2ban/jail.local` configuration file:

```
[DEFAULT]
ignoreip = 127.0.0.1/8 ::1 72.xx.yy.zz

[sshd]
port = 522
findtime = 15m
```

```
bantime = 2h
maxretry = 3
```

This example also shows how you can update the find time interval, the ban time, and the maximum number of attempts per findtime before banning the abusive host. It also includes a DEFAULT section where you can set global options like IP addresses to ignore. You can set ignoreip to your SSH source IP to prevent accidental lockouts. Restart fail2ban after making these changes:

```
george@ap1:~$ sudo systemctl restart fail2ban
```

Note: If you forget the ignoreip DEFAULT section like I did, log into your Vultr View Console window and execute the unban command noted below.

Any banned IPs are still banned after the restart. The status commands lists the banned IPs:

```
Status for the jail: sshd
|- Filter
| |- Currently failed:      2
| |- Total failed: 167
| `-- File list: /var/log/auth.log
`- Actions
  |- Currently banned:      3
  |- Total banned: 29
  `-- Banned IP list:      36.110.228.254 208.115.245.214 67.52.195.108
```

You can unban an IP address before the ban time has expired with the following command:

```
sudo fail2ban-client set sshd unbanip 67.52.195.108
1
```

Use the status command to verify the unban status. You can also view the status history in the fail2ban log file:

```
root@ap1:~# egrep "Ban|Unban" /var/log/fail2ban.log | grep 67.52.195.108
2022-04-11 14:55:15,351 fail2ban.actions [35842]: NOTICE [sshd] Ban
67.52.195.108
2022-04-11 15:34:53,087 fail2ban.actions [53811]: NOTICE [sshd] Restore Ban
```

```
67.52.195.108
2022-04-11 15:47:48,050 fail2ban.actions [53811]: NOTICE [sshd] Unban
67.52.195.108
```

Fail2ban restores the banned status after restarting the fail2ban service. The log file notes the unbanned status after executing the command to unban the IP address. This fail2ban failed/banned login example used SSH port 22. After changing the SSH port number, the number of failed login attempts are typically lower.

Configure SSH Agent Forwarding for Non-Root Users

SSH agent forwarding enables you to access server B from server A with your server-B private SSH key stored on your local host. An example use would be logging into server A, the ap1 server, and making a pull request from GitHub.com. You do not have to store your GitHub SSH keys on the remote server to access GitHub. The remote server uses SSH agent forwarding to retrieve the GitHub SSH keys from the SSH agent on the local host. SSH forwarding must be explicitly enabled for the remote server. Here is an example using GitHub and the ap1 server that shows how to configure SSH agent forwarding.

1. Create and Install GitHub SSH Keys

If you have not already created and installed SSH keys for GitHub, do:

```
george@imac1:~
$ ssh-keygen -t ed25519 -C "GitHub"
```

Save your GitHub keys at `~/.ssh/GitHub` and use a secure passphrase.

Add your GitHub SSH private key to the SSH agent on the local host.

```
george@imac1:~
$ ssh-add -K ~/.ssh/GitHub_ed25519
```

```
Enter passphrase for /Users/george/.ssh/GitHub_ed25519:  
Identity added: /Users/george/.ssh/GitHub_ed25519 (GitHub)
```

You can list the SSH keys stored in the SSH agent with this command:

```
george@imac1:~  
$ ssh-add -L  
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIMjM9PSuyKe3p4gJokzGHzy5mQXq52UDajrY9A2dRfFG  
GitHub
```

Install your GitHub public SSH key using the [GitHub Setting](#) page.

2. Create the GetHub SSH Agent Forwarding Configuration

Add the GitHub SSH configuration in your `~/.ssh/config` file:

```
Host github.com  
  Hostname github.com  
  Port 22  
  IdentityFile ~/.ssh/GitHub_ed25519
```

Verify that your GitHub SSH keys are working from your local host:

```
george@imac1:~  
$ ssh git@github.com  
Warning: Permanently added the ECDSA host key for IP address '140.82.113.4' to the list  
of known hosts.  
PTY allocation request failed on channel 0  
Hi georgew509! You've successfully authenticated, but GitHub does not provide shell  
access.  
Connection to github.com closed.
```

Note: You cannot log into GitHub via SSH, but you can test that your SSH authentication is working.

Enable SSH agent forwarding in your SSH `~/.ssh/config` file for the `ap1` host:

```
Host ap1.example.com ap1  
  Hostname ap1.example.com  
  Port 522
```

```
User george
IdentityFile ~/.ssh/ap1_ed25519
ForwardAgent yes
```

Log into your remote `ap1` server and verify that SSH agent forwarding is working:

```
george@imac1:~
$ ssh ap1
Linux ap1 5.10.0-11-amd64 #1 SMP Debian 5.10.92-1 (2022-01-18) x86_64
o o o
george@ap1:~$ ssh git@github.com
o o o
PTY allocation request failed on channel 0
Hi georgew509! You've successfully authenticated, but GitHub does not provide shell
access.
Connection to github.com closed.
```

The remote server receives the GitHub authentication request and forwards it to the SSH agent on the local host. The authentication request is successful. If you try to connect to GitHub from a remote host that does not have SSH agent forwarding enabled, you see this error:

```
git@github.com: Permission denied (publickey).
```

3. SSH Agent Forwarding Does Not Work for the root Account

SSH agent forwarding does not work for the root account because SSH access is off for the root account:

```
george@ap1:~$ sudo -i
[sudo] password for george:
root@ap1:~# ssh git@github.com
o o o
git@github.com: Permission denied (publickey).
```

By design, the root account cannot forward the SSH request to the local host.

How to Use SSH Tunneling (Port Forwarding)

You can use SSH tunneling or port forwarding to create an encrypted SSH connection to access a remote service from a local host. Or you can create a reverse SSH tunnel where the remote host can access a service running on your local host even if your local host is on a local network behind a Network Address Translation (NAT) router.

1. Create an SSH Tunnel

The following example shows how to create an SSH tunnel to use a local MariaDB client to access a remote MariaDB service. The local client connects on port 3307 to avoid conflicts with the local MariaDB server. The remote MariaDB server listens on port 3306. A remote dbadmin user with a password exists on the remote MariaDB server.

This SSH command creates the tunnel for the MariaDB service between the local host and the ap1 server:

```
george@imac1:~  
$ ssh -L 3307:127.0.0.1:3306 -N ap1
```

This command creates a non-interactive tunnel from the local host on port 3307 to the remote host at 127.0.0.1:3306. This SSH command runs in the foreground and displays any errors that occur. You can add the `-f` option to run the tunnel in the background.

In a new local host shell window, install and run the local MariaDB client:

```
george@imac1:~  
$ brew install mariadb  
  
george@imac1:~  
$ mariadb --host=127.0.0.1 --port=3307 -u dbadmin -p  
Enter password:  
Welcome to the MariaDB monitor. Commands end with ; or \g.  
Your MariaDB connection id is 562
```

```
Server version: 10.5.15-MariaDB-0+deb11u1 Debian 11

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

As noted above, the local macOS MariaDB client connects to the remote Debian MariaDB server.

2. Create a Reverse SSH Tunnel

The following example shows how to create a reverse SSH tunnel to access a service on the local host from the remote host. The local host is running a [Hugo](#) web server on port 1313. The remote host is using the Lynx text web browser on port 8080.

Note: You can follow the [Hugo Quick Start](#) guide to create a Hello, World Hugo test site.

This SSH command creates the reverse tunnel between the local host and the ap1 server:

```
george@imac1:~
$ ssh -R 8080:127.0.0.1:1313 -N ap1
```

This command creates the non-interactive reverse tunnel from the remote host on port 8080 to the local host, 127.0.0.1, on port 1313. This SSH command runs in the foreground and displays any errors that occur. You can use the netstat command on the remote host to verify that the sshd daemon is listening on port 8080:

```
george@ap1:~$ sudo netstat -lptn | grep 8080
tcp        0      0 127.0.0.1:8080      0.0.0.0:*           LISTEN    59739/
sshd: george
tcp6      0      0 :::1:8080          :::*                LISTEN    59739/
sshd: george
```

Start the Hugo web server, or other local service, on the local host:

```
george@imac1:~/Hugo/apnet
$ hugo server -D
Start building sites ...
hugo v0.96.0+extended darwin/amd64 BuildDate=unknown
  o o o
Web Server is available at http://localhost:1313/ (bind address 127.0.0.1)
Press Ctrl+C to stop
```

The Hugo web server is listening on 127.0.0.1:1313.

On the ap1 remote host install the lynx web browser and access the local host Hugo web server:

```
george@ap1:~$ sudo apt install lynx

george@ap1:~$ lynx http://127.0.0.1:8080
```

This opens a text window in the remote shell session displaying the Hugo web server content.

If the Hugo web server is not running, the local shell session running the SSH tunnel displays an error message:

```
connect_to 127.0.0.1 port 1313: failed.
```

3. Disabling Tunneling for the root User

Again, disabling SSH access for the root user, disables SSH tunneling for the root user:

```
george@imac1:~
$ ssh -L 3307:127.0.0.1:3306 -N root@ap1
root@ap1.example.com: Permission denied (publickey).
```

Use Scp to Copy Files Between Hosts

Scp securely copies files between hosts using ssh for the data transfer. With the remote host alias defined in `~/.ssh/config`, remote file references start with alias:. This is the example ap1 local SSH configuration in `~/.ssh/config`:

```
Host ap1.example.com ap1
  Hostname ap1.example.com
  Port 522
  User george
  IdentityFile ~/.ssh/ap1_ed25519
  ForwardAgent yes
```

This config file defines two host aliases--a long and short form.

This scp command copies a file from the local host to the remote host, relative to the user george home directory, to the `~/backup` directory:

```
$ scp logo.png ap1:backup
logo.png

george@imac1:~/tmp
$ ssh ap1 "ls backup"
logo.png
```

This scp command copies and renames the file:

```
$ scp logo.png ap1:myfile
logo.png

george@imac1:~/tmp
$ ssh ap1 "ls -lR ~/"
/home/george/:
total 12
drwxr-xr-x 2 george george 4096 Apr 13 20:50 backup
-rw-r--r-- 1 george george 8134 Apr 13 20:58 myfile

/home/george/backup:
```

```
total 8
-rw-r--r-- 1 george george 8134 Apr 13 20:55 logo.png
```

Use scp to copy remote files to the local host:

```
george@imac1:~
$ scp ap1:backup/logo.png .
logo.png
```

Use Rsync to Backup Files and Folders

Rsync can use the SSH protocol to transfer files and folders between hosts. It uses the same host alias specification defined in the SSH config file. It uses SSH when the source or destination host specification contains a single colon (:) separator. You can use rsync to backup a user's home file system:

```
george@imac1:~/tmp
$ rsync -av --delete ap1:/home/george .
receiving file list ... done
george/
george/.bash_history
george/.bash_logout
george/.bashrc
george/.profile
george/myfile
george/.ssh/
george/.ssh/authorized_keys
george/.ssh/known_hosts
george/backup/
george/backup/logo.png

sent 210 bytes  received 23286 bytes  15664.00 bytes/sec
total size is 22576  speedup is 0.96
```

The options are:

- `-a`: archive mode
- `-v`: increase verbosity
- `--delete`: delete extraneous files from dest dirs

Another useful option is `--exclude=PATTERN`. This option excludes copying files and folders that match a PATTERN.

Rsync copies files and folders that have changed:

```
george@imac1:~/tmp
$ rsync -av --delete ap1:/home/george .
receiving file list ... done

sent 16 bytes  received 320 bytes  672.00 bytes/sec
total size is 22576  speedup is 67.19
```

For this example, the files and folders on the remote and local hosts are the same. It did not need to copy any files. Refer to the rsync manual page for additional backup examples and usage. One detail to note is the last folder name in the path to copy. The example shows the george folder and its content copied. If the last folder name ends with a trailing slash (/), it only copies the folder content.

More Information

Vultr has other featured SSH articles if you'd like to learn more.

- [How to Use Two-Factor Authentication with Sudo and SSH on Linux with Google Authenticator](#)
- [How to Use SSH with Vultr Servers](#)
- [How to Add and Delete SSH Keys](#)
- [How To Connect to your Vultr Cloud Server with SSH, RDP, or SFTP](#)
- [How To Setup Fail2Ban On CentOS](#)
- [Changing Your SSH Port For Extra Security on CentOS 6 or 7](#)

Here are other helpful SSH resources:

- A collection of [OpenSSH Manual Pages](#)
- [Connecting to MySQL Remotely from Windows with SSH](#)
- [SSH Command - Usage, Options, Configuration](#)



VULTR

