How to Set up an SSH Server on a Home Computer

 Zach Duey Aug 27 '17

#beginners #networking #sysadmin

Around 18 months ago, I built a desktop computer so I would have a little more firepower at my fingertips when I needed it. My hope was to avoid having to pay for cloud computing services during Kaggle competitions and other side projects. Early on, I realized that it would be nice to be able to use this machine remotely, so I found a few resources for setting up your home PC as an SSH server. However, I was not able to find a single resource that provided enough background information that I was not just copy/pasting the commands.

Although no advanced knowledge of any particular topic is required to set up your own SSH server, there are many concepts to get your head around. I will explain more detail below, but the short version is that an SSH server is a process running on your computer that waits for outside computers to request access via a specific port, authenticates that user, and then allows access to the computer. If half of those terms do not make sense, do not worry. I was in the same position when I first tried getting set up. Here is a quick list of the concepts/terms that we will cover:

1. Server
2. SSH

4. Public/Private IP Addresses
5. Ports
6. Port Forwarding
7. Public/Private Key Authentication

I use Ubuntu 14.04 LTS as my operating system, however, the steps will be similar on any unix-like platform. To follow along, be sure to have access to the following:

1. The unix-based computer that will host the SSH server
2. A different computer to test the remote connection to the server
3. Access to a different network than the one to which your host machine is connected (optional, but recommended)

For the purposes of this guide, 'host' will indicate the computer running the SSH server, while 'client' will refer to any computer requesting access to the host. At one point in this guide, your host computer will also be a client.

Initial Setup

SSH stands for "secure socket shell" and is what will allow us to establish a secure connection between two computers. Our end goal is to be able to issue commands from a client machine that are executed by the host machine. For a more thorough coverage of SSH, take a look at this great guide by Digital Ocean.

term is a process running on a computer that is tasked with managing access to a computer's resources over a network. We will be installing the openssh-server application, which will allow us to run an SSH server on our machine that will handle requests for access to the host computer from other devices.

Start by updating your system packages:

```
sudo apt-get upgrade
```

Install the openssh-server application and client. You should also install the openssh-client on machines that will be used as clients.

```
sudo apt-get install openssh-client
sudo apt-get install openssh-server
```

You should now have an SSH server process running on your machine. Check with the following:


```
ps -A | grep sshd
```

You should see something like:

```
[number] ?  00:00:00 sshd
```

For those of you who copy/pasted the above command (something I have certainly been guilty of), you will learn a lot more by taking the time to look up any commands you do not recognize. If you are new (or just want a refresher on) to shell commands, check out the Learn the Command Line course from Code Academy.

Check that you can 'login' to the host machine from the host machine, i.e. your host machine is also the client in this case.

```
ssh localhost
```

Move around a little in the shell (cd, pwd, etc.) to make sure that everything works and then type "exit" to end the session.

Next, we will connect using a true 'client' machine. For this to work, make sure that your client machine is on the same network as the server. If you are following along from your home or work wifi network, you should be all set.

Start by determining the IP address of your host machine by listing information about the network interfaces on your machine.

```
ifconfig
```

if you have installed a wireless card on your desktop. If your host machine is connected to the network via an ethernet cable, use the address associated with the "eth[number]" entry. If you are connected via wifi, use the "wlan[number]" entry. If you want to see just the list of IP addresses for the devices, you can use:

```
ifconfig | grep "inet addr"
```

Next, try logging in from the client machine using that IP address:

```
ssh username@X.X.X.X
```

You should be prompted for the password of the account you used in your ssh command. For security reasons, the characters will not be displayed in the shell. If successful, you should see a "Welcome to Ubuntu X.X.X" message or something similar depending on your OS.

This is a bit more useful than our previous test, but it still does not solve our initial problem of being able to access the machine from outside the same network. Go ahead and try it out if you have the ability to connect your client machine to a different network than your host machine.

A quick option if you have a little data to spare on your cellular plan is to turn on tethering and connect your client machine to that network,

on this separate network, try that ssh command again:

```
ssh username@X.X.X.X
```

Most likely, you will notice that the command hangs and then eventually times out. Go ahead and disconnect from this other network and we will finish with the last step of the setup. But first, we will take a second to cover a few more background concepts that will make this next step clearer.

NAT and Public/Private IP Addresses

It may be surprising, but the internet is only able to handle a finite number of users being on line at a single time. In a world where almost every electronic device has access to the internet, it is possible to get close to the theoretical maximum. In order to circumvent this problem, Network Address Translation (NAT) was created to allow a set of devices on a private network to share a single IP address. On the private network, each device receives its own private IP address. For example, the IP address you found above with "ifconfig" is the private IP address of your computer on your local network.

Any time you acess a web page from one of the devices on your home network, the request is routed (hence the term router) through an NAT device, which translates the private IP address into a request using the public IP address assigned to the router by your ISP. This is the IP address you will ultimately need to use when connecting from outside

going to www.whatsmyip.org. You should see something in the form of XX.XX.XXX.XXX. You may be tempted to try that ssh command again with this address, but in all likelihood it still will not work.

Although NAT solves the device-limit problem, it adds a layer of complexity to setting up a home computer to accept SSH connections. When a client machine sends a request to connect to the public IP address, your router does not know which of the devices on your private network the request is meant for. Luckily, the solution is usually straightforward and involves setting up 'port fowarding' on your router. Bear with me as we continue down the terminology rabbit hole and go over ports. Feel free to skip ahead if you are already familiar with the concept.

Ports and Port Fowarding

Although computers have a variety of physical ports that most people are familiar with (USB, HDMI, VGA, etc.), the ports I am referring to here are networking ports and are logical, not physical. When you start a server (i.e. the SSH server you just set up in the previous steps), you bind it to one of these logical ports. The process then 'listens' for messages sent to this port by other programs (either internal or external). Each port has a number and some of these numbers are reserved for use by specific types of services. As an example, servers handling HTTP request use port 80 as the default. When you access a web page via a URL, you are really sending a message to an HTTP server at port 80 for a specific resource (web page or other content) that is managed by that service. In general, port 22 is used for the SSH service.

requests made using a specific port to a particular device on your private network. It is then the responsibility of that device to handle the request. You may be wondering: why don't just send the request directly to the computer via the private IP address and avoid all of this port forwarding nonsense? Like the name implies, these IP addresses are private, so once you are on a different network, the internet-at-large has no knowledge of this private IP address and therefore the request to connect will fail.

Every router is a little bit different when it comes to setting up port forwarding. This may sound like a cop out, but honestly, your best bet is to look up directions for your specific router. For example, here are instructions for an AT&T router and Comcast XFINITY. In general, you will need to take the following actions:

1. Log in to your router's admin page
2. Navgiate to the page for adding a service (SSH is usually one of the default options)
3. Select or enter the port number where requests will be made (22 by default for SSH)
4. Select or input the private IP address you found earlier of your host machine
5. Save the updated settings

Now, we can repeat our ssh command using the public IP address and that request should be redirected by our router to our host machine. Connect back to that outside network and try:

```
ssh username@[public IP address]
```

You should again be prompted for your host machine password and then admitted access.

More Secure SSH Server Configuration

For this final section, we will be making some changes to the configuration settings for our SSH server. On Ubuntu, these settings are located in the /etc/ssh/sshd_config file. A couple of quick notes:

- To edit this file, you will need to open it as a super user
- Any time you update the config settings, you need to restart the SSH server to have them take effect

```
sudo restart ssh
```

When a running SSH server receives a request over port 22 (or whatever port it is bound to), it must check that the requesting user has permission to access the computer. This can be done with a password, however, it is considered much more secure to use public key authentication. Public key cryptography works by having a user generate a public key that can be given to anyone, while keeping the private key hidden. Authentication works by using the public key (made available by the user to the process doing the authentication) to verify that the requesting user (whose request to access the service is encrypted using their private key) matches a user who is allowed access. Once this connection is established, more efficient methods of

other information using this secure connection.

If you have not already done so, you will need to generate a public/private key pair to use when logging in to your machine. Note that you will need to do this on any device you wish to be allowed access to your SSH server. There are a number of good guides out there for doing this, so to avoid re-inventing the wheel, I suggest following the Ubuntu Instructions

Now that you have those keys ready, we will enable public key authentication on the SSH server. Open the sshd_config file as a super user for the next set of steps and start by changing:

```
PasswordAuthentication yes
```

to

```
PasswordAuthentication no
```

This will prevent users from being able to access the server with a password, which will help against a brute force attack that attempts to guess the password. Another recommendation is to change the port that the SSH service uses. Within your config file, comment out the existing port specification and choose a new port.

```
# Port 22
Port [new port number]
```

Remember, based on an earlier step, your router will still attempt to send SSH requests to port 22, so you will need to go back into your router's config settings and update the port number for the SSH service. If you are on a machine with multiple user accounts, you can also limit which users are allowed to log in through SSH. At the bottom of the config file add:

```
AllowUsers [user1] [user 2] ...
```

You can also deny specific users and add/deny groups, however, it is unlikely that you will need to do this for a home computer.

Next, disable root login via SSH by changing:

```
PermitRootLogin without-password
```

to

```
PermitRootLogin No
```

attacks by limiting the number of concurrent connections to the SSH server. For a home computer, keep this number low by changing:

```
#MaxStartups 10:30:60
```

to

```
MaxStartups 3
```

With an SSH connection, you can tunnel graphical windows. If you do not plan to use this ability,then go ahead and shut it off by changing

```
X11Forwarding yes
```

to

```
X11Forwarding no
```

In addition to forwarding windows, you can also forward ports via SSH. For example, if you had a web server running on port 80 on your host machine, you could forward that port to a port on your client machine. This can be handy, but if you do not plan on using this

maliciously.

```
AllowTcpForwarding no
```

There are a bunch of other settings in the ssh_config file, but I do not recommend changing the defaults unless you have a good reason to do so. While the suggested changes will add some security to your server, there are many additional ways to add security. In fact, they probably deserve a post of their own, but I will save that for another time.

Thanks for taking the time to read through this and please feel free to let me know if anything is unclear. This post originally appeared on my personal blog.

External Resources

- Digital Ocean SSH Essentials
- Ubuntu SSH Configuration Guide
- Ubuntu Public/Private Keys Guide
- Secure SSH Configuration

Zach Duey + FOLLOW

Z_Duey zduey zduey.github.io

Add to the discussion

PREVIEW SUBMIT

Praneeth Mendu

Oct 5 '17

a port scanner shows that only ports 21 and 80 are open on my public IP. Does this mean I cannot use true ssh without the help of my IPS ??

1 REPLY

code of conduct - report abuse

Classic DEV Post from Apr 7

How do you deal with burnout and low motivation?

Ryan Norton

The past few months have been a no-code zone for me. I've literally gotten noth...

57 14

READ POST SAVE FOR LATER

What Is Immutable Infrastructure?


Hazel Virdó

This article explains the conceptual and practical differences between mutable and immutable infrastructure.

67 1

READ POST SAVE FOR LATER

Another Post You Might Like

Open Letter to My Younger Self


Lorenzo Pasqualis

I know you won't believe me at first, but please keep reading this letter to the end. You are in your early 20's and you don't believe most things you can't see, but I have to try anyway. I am the older you, writing from 2017, almost 2018. Yes, you are going to make it that far.

67 8

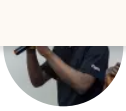
READ POST SAVE FOR LATER


Shapr, the networking secret of successful founders

Laszlo L Mari - Apr 8


Learning Web Development? These Skills Will Make You Stand Out (Part Two)

Colin Morgan - May 10



/[Abejide Femi Jr]\s/ - May 11

Franken-code 'Till You Make It.

Cat Carbonell - May 9