**SSH, the Secure Shell :**  
  
The SSH system, is a protocol for remote logins and for securing network services on an insecure network. SSH’s capabilities include remote command execution, shell access, file transfer, port forwarding, network proxy services, and even VPN tunnelling.  
  
SSH is a client/server protocol that uses cryptography for authentication, confidentiality, and integrity of communications between two hosts.  
  
**OpenSSH essentials :**  
  
**• ssh, the client  
• sshd, the server daemon  
• ssh-keygen, for generating public/private key pairs  
• ssh-add and ssh-agent, tools for managing authentication keys  
• ssh-keyscan, for retrieving public keys from servers  
• sftp-server, the server process for file transfer over SFTP  
• sftp and scp, file transfer client utilities**  
  
In the most common and basic usage, a client establishes a connection to the server, authenticates itself, and subsequently opens a shell to execute commands. Many users can log in simultaneously. A pseudo-terminal is allocated for each, connecting their input and output to the remote system.   
  
To initiate this process, a user invokes ssh with the remote host as the first argument:  
  
**$ ssh web01.k8slab.in**  
  
ssh attempts a TCP connection on port 22 When the connection is established, the server sends its public key for verification. If the server isn’t already known then ssh prompts the user to confirm  
Once the user accepts the key, the fingerprint is added to **~/.ssh/known\_hosts** for future use.  
  
**$ cat known\_hosts**

10.10.10.210 ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBMSL2CWUtLhBfUZtv4u7Ynk4RUvFq7tQ8kL7ztO0XB0aZVuokfsCBt+i6gI3wbtUOXJ5KOx0ifxXjyUWVPHP8kA=  
  
Once the host key has been accepted, the server lists the authentication methods it supports. simple UNIX password authentication, trusted hosts, public keys, GSSAPI for integration with Kerberos.

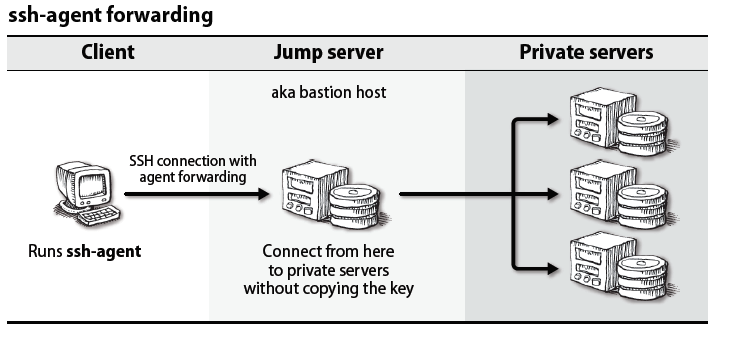
Of these, public key authentication is the most commonly used and is the method we recommend for most sites. Configuration is found in the **/etc/ssh** directory.  
  
**ssh\_config = Site-wide client configuration  
sshd\_config = Server configuration  
\*\_key 0600 = Private keys for every algorithm supported by the server  
\*\_key.pub = A public key to match each private key**  
  
In addition OpenSSH uses **~/.ssh** for storing public and private keys, for per-user client.  
  
The ssh client :  
  
For example, to check the disk space of /var/log:

**$ ssh web01.k8slab.in "df -h /var/log"**  
If you specify a command, ssh authenticates itself to the host, runs the command, and exits without opening an interactive shell. If you do not specify a username, ssh uses your local username on the remote host.  
  
# Specify username to call SSH on remote machine  
  
**$ ssh vipin@web01.k8slab.in**  
# For example, you could disable host checks for a server:  
  
**$ ssh -o StrictHostKeyChecking=no web01.k8slab.in**  
Useful SSH client configuration options :  
  
**AddKeysToAgent** = Automatically add keys to ssh-agent   
**ConnectTimeout** = Connection timeout in seconds   
**ControlMaster** = Allow connection multiplexing  
**DynamicForward** = Set up a SOCKS4 or SOCKS5 proxy  
**ForwardAgent** = Enable ssh-agent forwarding   
**Host** = Marker for a new host alias  
**IdentityFile** = Path to an authentication private key ~/.ssh/id\_rsa  
**Port** = Port to connect on 22  
**RequestTTY** = Specify whether a TTY is needed  
  
**# Public key authentication :**  
  
OpenSSH (and the SSH protocol generally) can use public key to authenticate users to remote systems. As a user, you start by creating a public/private key pair. You give the public key to the server administrator, who adds it to the server in the file **~/.ssh/authorized\_keys**. You can then log in to the remote server by running ssh with the remote username and matching private key.  
  
1) First create/generate Public/Private key :  
  
[vagrant@docker01 ~]$ ssh-keygen  
Generating public/private rsa key pair.  
Enter file in which to save the key (/home/vagrant/.ssh/id\_rsa):  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:  
Your identification has been saved in /home/vagrant/.ssh/id\_rsa.  
Your public key has been saved in /home/vagrant/.ssh/id\_rsa.pub.  
The key fingerprint is:  
SHA256:+PnUoI6g6huuY92KzaJqIlY70sBVkY/XNQYgulT4rXE vagrant@docker01.k8slabs.in  
The key's randomart image is:

+---[RSA 2048]----+  
| .+o.... |  
| .oo. + |  
| oo + . o . |  
| ...+ E . |  
|. .. \* S . |  
| o . . . o o |  
| .= + + . . |  
|=\*=\* o o o |  
|/B=+o . . . |  
+----[SHA256]-----+  
  
The public key (~/.ssh/id\_rsa.pub) and private key (~/.ssh/id\_rsa) files are created, Never share the private key! ssh-keygen sets the permissions on the public and private key correctly as 0644 and 0600, respectively.  
  
2) Provide public key 'id\_rsa.pub' from ~/.ssh/. Admin will add public key for user  
  
follow these steps:  
1. Ensure that the user has an active account with a valid shell.  
2. Get a copy of the user’s public key from the user.  
3. Create the user’s .ssh directory with permissions 0700.  
4. Add the public key to ~user/.ssh/authorized\_keys and set the permissions of that file to 0600.  
  
for example :  
  
[vagrant@docker01 .ssh]$ grep vagrant /etc/passwd  
vagrant:x:1000:1000::/home/vagrant:/bin/bash  
mkdir -p ~vagrant/.ssh && chmod 0700 ~vagrant/.ssh  
$ cp /tmp/id\_rsa.pub >> ~vagrant/.ssh/authorized\_keys  
$ chmod 0600 ~vagrant/.ssh/authorized\_keys

**#ssh-agent :**  
ssh-agent is even more useful when you leverage its key forwarding feature, which makes the loaded keys available to remote hosts while you are logged in to them through ssh.

You can use this feature to jump from one server to another without copying your private key to remote systems.



To enable agent forwarding, either add ForwardAgent yes to your **~/.ssh.config** file or use **ssh -A**.

**sshd: the OpenSSH server :**

The OpenSSH server daemon, **sshd**, listens on port 22 (by default) for connections from clients.

Its configuration file, **/etc/ssh/sshd\_config**, boasts myriad options, some of which may need to be tuned for your site.

In Linux, you can also run **sudo systemctl reload sshd**. The changes take effect for new connections.

**File transfers :**

You can use **scp** to copy files from your system to a remote host, from a remote host to your system, or between remote hosts. The syntax mirrors that of **cp** with some extra decorations to designate hosts and usernames.

$ **scp ./file web01.k8slab.in:**

$ **scp web01.k8slab.in:file ./file**

$ **scp web01.k8slab.in:file web02.k8slab.in:file**