Show Submission Credentials

P0. Secure Shell (SSH) Secure Shell (SSH)

✔ Secure Shell (SSH)

✔ Remote Screen Management

Secure Shell (SSH)

Secure Shell (SSH)

Secure Shell (SSH) is a network protocol used for secure data communication, remote login and remote command execution. You should be quite familiar with SSH already, but there are some quirks to using SSH with EC2 instances, which are covered below.

In order to get SSH working with an instance on EC2, you must ensure that the SSH clients and servers are installed correctly, that the instance's security group allows incoming connections on port 22, and that your authentication mechanism is working correctly.

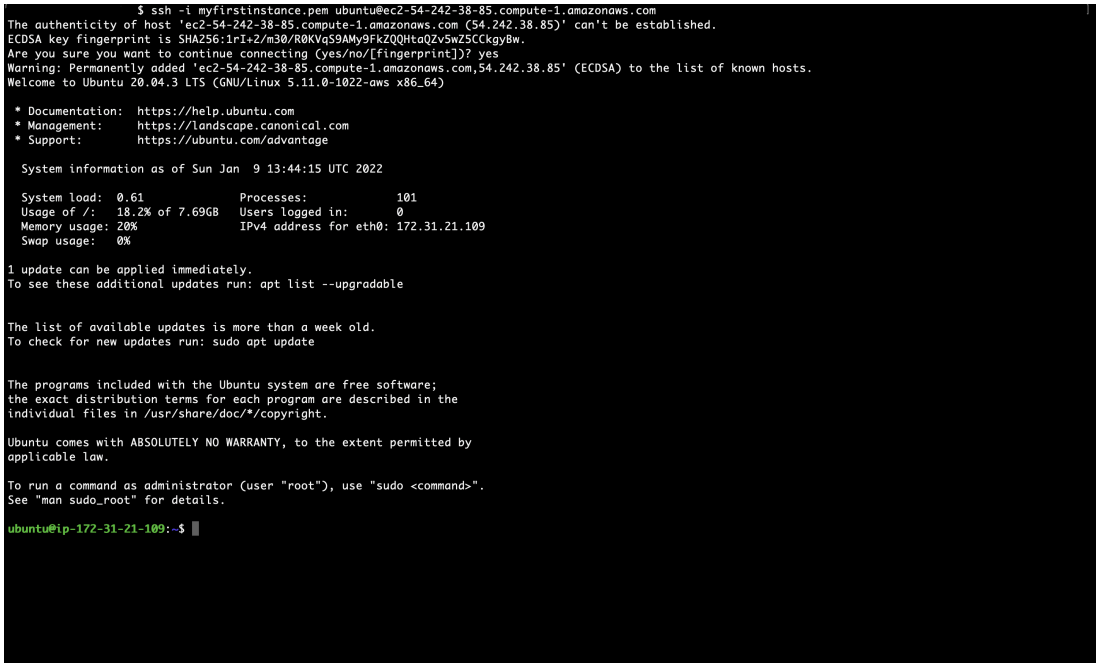

Figure 1: Connecting to AWS EC2 Instances through SSH

Setting up SSHD (If not installed already)

All EC2 images come with an ssh daemon (`openssh`) pre-installed. In case you have a fresh linux install on your own machine, please follow the steps below to install and configure `openssh` :

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

Configuration settings for ssh are on `/etc/ssh/sshd.config`

Depending on your version of Ubuntu, sshd can be managed either using `init.d` :

```
sudo /etc/init.d/ssh
```

or by using service:

```
sudo service sshd [start|stop|restart]
```

Authentication Mechanisms

SSH supports a number of authentication mechanisms. You might be most familiar with password authentication (where you enter a username/password when connecting to a remote machine). Password authentication security is based on the length and complexity of the password, and is prone to brute-force or dictionary-based attacks. This is especially dangerous on a publicly-accessible EC2 instance.

To plug the vulnerability and to increase security, all EC2 instances are configured to use “key-based” authentication instead. This is based on public-key cryptography. A key pair consists of two keys: one private and one public. The public and private keys are mathematically

linked; they are generated using algorithms such as RSA or DSA. However, from the knowledge of only one of the keys (only private or only public), it is either impossible or very expensive to calculate the other key.

You can also generate your own key pair. In Linux, ssh key pairs are typically stored in the `~/.ssh` directory. We recommend using this directory as it has the correct permissions (read-write for user and no permissions for anyone else - `700`). The following command can be used to generate a key-pair in Linux using the ssh-keygen tool in standard OpenSSH installation (You are free to create an RSA key using any other tool):

```
# Generate a new key with your email id as a label
ssh-keygen -t rsa -b 4096 -C "email_id@domain.com"

# Enter the file where you want to save the key: (recommended - choose default)

# You will be asked enter a pass-phrase for your key twice.
# (Use a strong pass-phrase. Longer pass-phrases are more secure than shorter on
es.)

# After you provide the pass-phrase. The console will print the location of the
key and the key fingerprint.
```

You can now import your key on Amazon EC2 and use this to create an SSH connection to a remote instance. You can see how to do this here (http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html#how-to-generate-your-own-key-and-import-it-to-aws).

As an example, the following `ssh` command will execute the command `uname -a` on the remote instance `some-instance.ec2.amazonaws.com`. This is especially useful to run commands on many remote instances in a script.

```
ssh -i private_key_file.pem some-instance.ec2.amazonaws.com 'uname –a'
```

Information

Troubleshooting SSH Problems

If you receive a permission-denied error when trying to connect to an EC2 instance, verify that you are using the private key to connect. In addition, verify that your private key files have unix permissions `400` and the directory containing the key is `700`.

Running `ssh -v` will provide more verbose output, which may be useful for debugging connection issues.

Information

SSH Timeout

Sometimes you need to execute long-running scripts on your EC2 instance, however SSH terminal sessions will typically timeout after being inactive for a certain amount of time. If the sessions timeout, all your running scripts will be terminated . There are several

ways you could avoid SSH timeout:

1. Disable SSH timeout on your SSH Client (https://docs.oseems.com/general/application/ssh/disable-timeout).
2. Use `nohup` to run your scripts.
3. Preserve your terminal sessions by using remote screen management (as explained in the next section).

Remote Screen Management

Remote Screen Management

When working with remote machines over SSH, the bash (or similar linux shells) are used. These shells work in interactive mode, allowing the user to run commands and launch processes. However, due to network connectivity issues, remote SSH shells can be volatile and bash by default will terminate all launched processes if the session is disconnected. This can be especially frustrating when running long jobs or scripts that get terminated due to a network issue. The `nohup` command is useful to run a program in the background and continue to run the command even if the session is disconnected.

Another useful tool to manage a remote SSH session is `byobu`. `byobu` is a multi-terminal manager application for Linux, similar to `screen` or `tmux`. We highly recommend using `byobu` to keep the state of your remote sessions alive, even if you disconnect from the SSH session. You can install `byobu` on your instance if it's not already present.

Launch `byobu` by running the command `byobu`. This will launch a new shell session in your instance. You should see a screen similar to the image below. You can create multiple shell sessions by pressing F2. To toggle between shell sessions, press F3/F4. To disconnect from `byobu` (and keep the shells alive), press F6. To terminate a shell session, simply exit the shell.

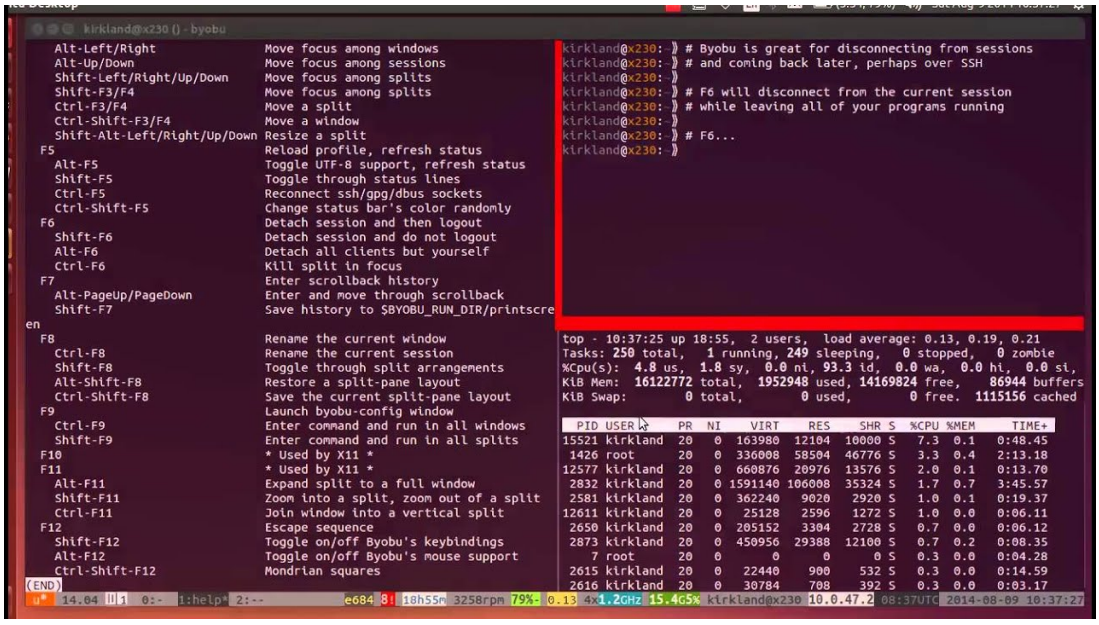

Figure 2: Byobu Screenshot. source (https://www.byobu.org)