Open the Amazon EC2 console and here you will see a navigation pane on the left side. In the navigation pane, under NETWORK & SECURITY, choose Key Pairs.

First create a key pair to login to Putty. When you download a keypair, the file extension is .pem (Privacy Enhanced Mail). Then convert this .pem file to .ppk (Putty Private Key) file using PuttyGen.  
Open PuttyGen and select load to load the .pem file and then select “Save Private Key” and save it with .ppk extension.

Then open the Putty and expand Connection->SSH and select Auth and browse for .ppk file.

Enter the username as Ubuntu and it won’t ask for the password as you have used a private key to login.  
**Note:** For Amazon instance, the username is ec2-user.

**Sudo--  
Note:** In Linux (and UNIX in general), there is a Super User named **root**. **By default, the root account password is locked in Ubuntu.** This means that you cannot login as root directly or use the su command to become the root user. However, since the root account physically exists it is still possible to run programs with root-level privileges. This is where sudo comes in - it allows authorized users (normally "Administrative" users) to run certain programs as root without having to know the root password.

This means that in the [terminal](https://help.ubuntu.com/community/UsingTheTerminal) you should use sudo for commands that require root privileges; simply prepend sudo to all the commands you need to run as root.

Sudo performs the following command with super-user (root) capabilities. Many actions that require modifying system files or installing applications require extra permissions to go through.

**Apt-get--** The apt-get utility is a powerful and free package management command line program that is used to work with Ubuntu’s **APT** (Advanced Packaging Tool) library to perform installation of new software packages, removing existing software packages, upgrading of existing software packages and even used to upgrading the entire operating system.

Running “**sudo apt-get update**” simply makes sure your list of packages from all **repositories and PPA's** is up to date. If you do not run this command, you could be getting older versions of various packages you are installing, or worse, dependency issues. If you have just added a PPA and have not updated, nothing from the PPA will work at all as you do not have a package list from that PPA or repository.

Sudo apt-get update (You should have internet connection to update the repositories)  
sudo apt-get install <package1> <package2>

Running the above commands one after the other ensures that the latest packages installed.  
**Note:** If you have an internet connection you can install software from any Ubuntu repository.  
**Note:** You will get the some of the repositories with Ubuntu CD itself   
**Personal Package Archive (PPA):** Using a Personal Package Archive (**PPA**), you can distribute software and updates directly to Ubuntu users. Create your source package, upload it and Launchpad will build binaries and then host them in your own apt repository.

That means Ubuntu users can install your packages in just the same way they install standard Ubuntu packages and they'll automatically receive updates as and when you make them. Every individual and team in Launchpad can have one or more PPAs, each with its own URL.  
Packages you publish in your PPA will remain there until you remove them, they're superseded by another package that you upload or the version of Ubuntu against which they're built becomes obsolete.  
**Repository:** There are literally thousands of Ubuntu programs available to meet the needs of Ubuntu users. Many of these programs are stored in software archives commonly referred to as *repositories*. Repositories make it easy to install new software, while also providing a high level of security, since the software is thoroughly tested and built specifically for each version of Ubuntu.

**Apt-cache**-- The apt-cache command line tool is used for searching apt software package cache. In simple words, this tool is used to search software packages, collects information of packages and also used to search for what available packages are ready for installation on **Debian** or **Ubuntu** based systems.

5 useful apt-cache commands:  
1) sudo apt-cache pkgnames : lists all the available packages  
2) sudo apt-cache search vsftpd: Using “search” with apt-cache will display a list of matched packages with short description. Here we are trying to find out the description of vsftpd.  
To find and list down all the packages starting with ‘vsftpd‘, you could use the following command  
apt-cache pkgnames vsftpd  
3) sudo apt-cache show netcat: Is used to check the package version and other info. Here netcat is a package.  
4) Sudo apt-cache showpkg vsftpd: Use the ‘showpkg’ sub command to check the dependencies for particular software packages.  
5) Sudo apt-cache stats: The ‘stats’ sub command will display overall statistics about the cache.

20 useful apt-get commands:   
1) sudo apt-get update: The ‘update’ command is used to resynchronize the package index files from their sources specified in/etc/apt/sources.list file.  
2) Sudo apt-get upgrade: The ‘upgrade’ command is used to upgrade all the currently installed software packages on the system. Currently installed packages will not be removed and also the packages which are not already installed will neither be retrieved nor installed to satisfy upgrade dependencies.  
3) Sudo apt-get install netcat: To install a single package  
4) sudo apt-get install nethogs goaccess: To install multiple packages  
5) sudo apt-get install '\*name\*': You can use regular expressions also to install multiple packages.  
6) Sudo apt-get install packageName --no-upgrade: Using sub ‘–no-upgrade’ command will prevent already installed packages from upgrading.  
7) Sudo apt-get install packageName --only-upgrade: The ‘–only-upgrade’ command do not install new packages but it only upgrade the already installed packages and disables new installation of packages.  
8) Sudo apt-get install vsftpd=2.3.5-3ubuntu1: To install only specific version of packages, simply use the ‘=‘ with the package-name and append desired version.  
9) Sudo apt-get remove vsftpd: To un-install software packages without removing their configuration files (for later re-use the same configuration).  
10) Sudo apt-get purge vsftpd: To remove software packages including their configuration files, use the ‘purge’ sub command.  
  
**Note:** I think we can use both the above commands like below.  
Sudo apt-get remove --purge awscli  
11) sudo apt-get clean: The ‘clean‘ command is used to free up the disk space by cleaning retrieved (downloaded) .deb files (packages) from the local repository.  
12) sudo apt-get --download-only source vsftpd: To download only source code of particular package, use the option ‘–download-only source‘ with ‘package-name’ as shown.  
13) Sudo apt-get source vsftpd: Use this command to download and unpack source code of a package to a specific directory.  
14) Sudo apt-get --compile source goaccess: You can also download, unpack and compile the source code at the same time, using option ‘–compile‘.  
15) Sudo apt-get download nethogs: Using ‘download’ option, you can download any given package without installing it.  
16) Sudo apt-get changelog vsftpd: The ‘changelog’ flag downloads a package change-log and shows the package version that is installed.  
17) Sudo apt-get check: The ‘check’ command is a diagnostic tool. It used to update package cache and checks for broken dependencies.  
18) Sudo apt-get build-dep netcat: This ‘build-dep’ command searches the local repositories in the system and install the build dependencies for package. If the package does not exist in the local repository it will return an error code.  
19) sudo apt-get autoclean: The ‘autoclean‘ command deletes all .deb files from /var/cache/apt/archives to free-up significant volume of disk space.  
20) Sudo apt-get autoremove vsftpd: The ‘autoremove’ sub command is used to auto remove packages that were certainly installed to satisfy dependencies for other packages and but they were now no longer required. For example, the following command will remove an installed package with its dependencies.

**YUM (Yellowdog Updater Modified)**: It is an open source command-line as well as graphical based package management tool for RPM (**Red Hat Package Manager**) based **Linux** systems. It allows users and system administrator to easily install, update, remove or search software packages on a system. It was developed and released by Seth Vidal under GPL (General Public License) as an open source, means anyone can allow to download and access the code to fix bugs and develop customized packages. YUM uses numerous third party repositories to install packages automatically by resolving their dependencies issues.  
**Ex:** yum install firefox: To install a firefox package.  
**Ex:** yum info firefox: Gives you information of a package.  
In the same way we can use other commands also as with apt-get.

PIP is a [package management system](https://en.wikipedia.org/wiki/Package_manager) used to install and manage [software packages](https://en.wikipedia.org/wiki/Package_(package_management_system)) written in [Python](https://en.wikipedia.org/wiki/Python_(programming_language)). Many packages can be found in the [Python Package Index](https://en.wikipedia.org/wiki/Python_Package_Index) (PyPI)

To Install python based applications we need **python-pip**. First install python-pip using below command. This command first installs python and then pip will be installed.  
--sudo apt-get install **python-pip**

Now to install **awscli** (aws command line interface) which is a python based program we issue below command.  
--sudo pip install awscli (You can install awscli using apt-get also but you may not get the latest version).   
after installing awscli, we need to configure it. Below command is used to configure aws.  
--aws configure  
This prompts for the AK (Access Key) and SK (Secret Key) which are like username and password to connect to aws from the command prompt.  
**Note:** You will get AK and SK from your account in aws by clicking on security credentials.

Now install **ansible** using PIP. Using **ansible** we will make everything automate like creating **security groups**, Keys, AMI-ID, Region, launch instance etc.

--sudo pip install ansible  
--/usr/local/bin/ansible --version  
--ansible --version  
--which ansible

**Playbooks:**Simply put, playbooks are the basis for a really simple configuration management and multi-machine deployment system, unlike any that already exist, and one that is very well suited to deploying complex applications.  
Playbooks are expressed in **YAML format** and have a minimum of syntax, which intentionally tries to not be a programming language or script, but rather a model of a configuration or a process.  
We use YAML because it is easier for humans to read and write than other common data formats like XML or JSON. Further, there are libraries available in most programming languages for working with YAML.  
All YAML files (regardless of their association with Ansible or not) can optionally begin with ---and end with... . This is part of the YAML format and indicates the start and end of a document.  
**Note:** Ansible uses “{{ var }}” for variables  
  
foo: "{{ variable }}"

While you might run the main **/usr/bin/ansible** program for ad-hoc tasks, playbooks are more likely to be kept in source control and used to push out your configuration or assure the configurations of your remote systems are in spec.

### Hosts and Users:

For each play in a playbook, you get to choose which machines in your infrastructure to target and what remote user to complete the steps (called tasks) as.

The hosts line is a list of one or more groups or host patterns, separated by colons. The remote\_user is just the name of the user account:

---

- hosts: webservers

remote\_user: root

Remote users can also be defined per task:

---

- hosts: webservers

remote\_user: root

tasks:

- name: test connection

ping:

remote\_user: yourname

**Tasks list:**  
Each play contains a list of tasks. Tasks are executed in order, one at a time, against all machines matched by the host pattern, before moving on to the next task. It is important to understand that, within a play, all hosts are going to get the same task directives. It is the purpose of a play to map a selection of hosts to tasks.  
The goal of each task is to execute a module, with very specific arguments. Variables, as mentioned above, can be used in arguments to modules.  
Modules are ‘**idempotent**’, meaning if you run them again, they will make only the changes they must in order to bring the system to the desired state. This makes it very safe to rerun the same playbook multiple times. They won’t change things unless they have to change things.  
The **command** and **shell** modules will typically rerun the same command again, which is totally ok if the command is something like chmod or setsebool, etc. Though there is a creates flag available which can be used to make these modules also idempotent.  
**Note:** Every task should have a name, which is included in the output from running the playbook.

### Here is what a basic task looks like. As with most modules, the service module takes key=value arguments:

tasks:

- name: make sure apache is running

service: name=httpd state=started

The **command** and **shell** modules are the only modules that just take a list of arguments and don’t use the key=value form. This makes them work as simply as you would expect:

tasks:

- name: disable selinux

command: /sbin/setenforce 0

# [Playbook Roles and Include Statements](http://docs.ansible.com/ansible/playbooks_roles.html#id3)

# [Common Return Values](http://docs.ansible.com/ansible/common_return_values.html#id2)

# [Best Practices](http://docs.ansible.com/ansible/playbooks_best_practices.html#id8)

**Modules in ansible:**   
All the below modules are core modules and wherever you are executing this module, that host should have python>=2.6 and **boto**.  
The ones below the ec2 are called options or attributes.  
**ec2:** create, terminate, start or stop an instance in ec2.  
Mandatory options or attributes: Apart from the below ones, we have lot other options.  
image: *ami* ID to use for the instance.  
instance\_type: instance type to use for the instance  
count—No.of instances to launch.  
  
*# Basic provisioning example*

- ec2:

key\_name: mykey

instance\_type: t2.micro

image: ami-123456

wait: yes

group: webserver

count: 3

vpc\_subnet\_id: subnet-29e63245

assign\_public\_ip: yes

*# Advanced example with tagging and CloudWatch*

- ec2:

key\_name: mykey

group: databases

instance\_type: t2.micro

image: ami-123456

wait: yes

wait\_timeout: 500

count: 5

instance\_tags:

db: postgres

monitoring: yes

vpc\_subnet\_id: subnet-29e63245

assign\_public\_ip: yes

# ec2\_group: maintain an ec2 VPC security group. Mandatory options or attributes: Apart from the below ones, we have lot other options. description: Description of the security group. name: Name of the security group. rules: List of firewall inbound rules to enforce in this group (see example). If none are supplied, a default all-out rule is assumed. If an empty list is supplied, no inbound rules will be enabled. rules\_egress: List of firewall outbound rules to enforce in this group (see example). If none are supplied, a default all-out rule is assumed. If an empty list is supplied, no outbound rules will be enabled.

- name: example ec2 group

ec2\_group:

name: example

description: an example EC2 group

vpc\_id: 12345

region: eu-west-1a

aws\_secret\_key: SECRET

aws\_access\_key: ACCESS

rules:

- proto: tcp

from\_port: 80

to\_port: 80

cidr\_ip: 0.0.0.0/0

- proto: tcp

from\_port: 22

to\_port: 22

cidr\_ip: 10.0.0.0/8

- proto: tcp

from\_port: 443

to\_port: 443

group\_id: amazon-elb/sg-87654321/amazon-elb-sg

- proto: tcp

from\_port: 3306

to\_port: 3306

group\_id: 123412341234/sg-87654321/exact-name-of-sg

- proto: udp

from\_port: 10050

to\_port: 10050

cidr\_ip: 10.0.0.0/8

- proto: udp

from\_port: 10051

to\_port: 10051

group\_id: sg-12345678

- proto: icmp

from\_port: 8 *# icmp type, -1 = any type*

to\_port: -1 *# icmp subtype, -1 = any subtype*

cidr\_ip: 10.0.0.0/8

- proto: all

*# the containing group name may be specified here*

group\_name: example

rules\_egress:

- proto: tcp

from\_port: 80

to\_port: 80

cidr\_ip: 0.0.0.0/0

group\_name: example-other

*# description to use if example-other needs to be created*

group\_desc: other example EC2 group

ec2\_key: **maintain an ec2 key pair.**Mandatory options or attributes: Apart from the below ones, we have lot other options.  
name: Name of the key pair  
*# Note: None of these examples set aws\_access\_key, aws\_secret\_key, or region.*

*# It is assumed that their matching environment variables are set.*

*# Creates a new ec2 key pair named `example` if not present, returns generated*

*# private key*

- name: example ec2 key

ec2\_key:

name: example

*# Creates a new ec2 key pair named `example` if not present using provided key*

*# material. This could use the 'file' lookup plugin to pull this off disk.*

- name: example2 ec2 key

ec2\_key:

name: example2

key\_material: 'ssh-rsa AAAAxyz...== me@example.com'

state: present

*# Creates a new ec2 key pair named `example` if not present using provided key*

*# material*

- name: example3 ec2 key

ec2\_key:

name: example3

key\_material: "{{ item }}"

with\_file: /path/to/public\_key.id\_rsa.pub

*# Removes ec2 key pair by name*

- name: remove example key

ec2\_key:

name: example

state: absent

**service:** Controls services on remote hosts.  
Mandatory options or attributes:Apart from the below ones, we have lot other options.  
name: Name of the service.  
*# Example action to start service httpd, if not running*

- service: name=httpd state=started

*# Example action to restart network service for interface eth0*

- service: name=network state=restarted args=eth0

*# Example action to start service foo, based on running process /usr/bin/foo*

- service: name=foo pattern=/usr/bin/foo state=started