Reading Time: 5 minutes

**Introduction**



Ansible is an IT automation tool intended to facilitate the management of remote servers. Ansible requires Python (version 2.7 or 3.5 and higher) to run. Ansible is run from a centralized control node and can manage any server accessible over SSH. Remote servers that are managed by Ansible are called managed nodes.

By default, Ansible communicates with managed nodes using OpenSSH. SSH is not the only communication mechanism Ansible supports. You can run tasks on the control server locally, in a docker container or even a Windows server. The control node will require Linux to run.

**What is IT Automation?**

Today, System Administrators and DevOps Engineers must manage complex IT infrastructure in the cloud and across multiple sites. Modern web applications typically consist of an API service, a front-end service, and a database service. Manually managing such complex systems would require a lot of time and would be prone to mistakes.

IT automation uses software to provision IT infrastructure, deploy applications and, to manage services configuration changes. The software you use to fill this role needs to provide a reliable and repeatable way to manage your IT tasks. Ansible’s approach to IT automation centers around the ‘playbook’. You can think of an Ansible playbook as a recipe that describes the steps needed to set up your IT infrastructure, deploy your applications and then configure those services.

**Install Ansible**

**Step 1: Update your Control Node**

Any time you are installing new software, it is a good idea to ensure your *existing* operating system software is up to date. Let us start with that task first.

yum update

**Step 2: Install the EPEL Repository**

Installing Ansible is rather straightforward. First, we will need to install the CentOS 7 EPEL repository.

yum install epel-release

**Step 3: Install Ansible**

Next, we install the Ansible package from the EPEL repository.

yum install ansible

**Step 4a: Create a User for Ansible**

As security best practice denotes, it is a good idea to avoid logging into your Linux servers as root. We will create a non-root user on our *control* node and our *managed* nodes that will run our Ansible playbooks. This user defines the admin Ansible will utilize to log into our managed nodes. Here we have used “*ansible*”’ for the user, but any username can be substituted. To follow along with our examples, you will want to use the same username on both the *Control* node and our managed *Nodes*. Log onto the *control* node to add a user and set a password.

useradd ansible  
passwd ansible

Log onto the managed *node*, add the ansible user, and set the password.

useradd anisble  
passwd ansible

**Step 4b: Configure the Control Node User for Passwordless Super User Access**

On the managed *node*, we need to ensure our Ansible user can utilize the *sudo* command without a password. Run the following command to open the sudoers file for editing.

visudo

Type ‘i’ to enter input mode and add the following to the end of the file. Type ‘[ESC]:wq’ to save your changes.

ansible ALL=(ALL) NOPASSWD: ALL

**Step 5: Configure our Ansbile User for SSH Access**

We need to ensure our ansible user can access the managed *node* over SSH without a password. We will set up an SSH key pair to allow this. Log onto the *control* node as the admin user and run the following command to generate an SSH key pair. Note: Just hit enter at the prompts to accept the defaults.

ssh-keygen

Now we can copy the public key to our managed *node* with the following command.

ssh-copy-id [IP or DNS name of your managed node]

**Step 6: Create an Ansible Inventory**

Ansible requires an [inventory list](https://www.liquidweb.com/kb/how-to-install-and-configure-ansible/) so it can identify your managed nodes. To add our managed node to the inventory, we need to login to our Control node as the ansible user. Next, we will add a new inventory file. Make sure you are logged onto the *Control* node as the admin user.

vim /home/ansible/inventory

Type ‘i’ to enter insert mode and add the managed *node* hostname to the inventory file.

192.168.1.10

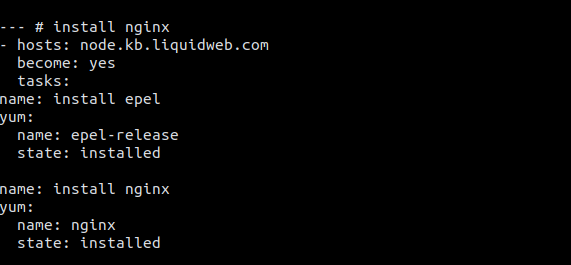
Next, type ‘[ESC]+:wq’ to save the file.

**Step 7: Create an Ansible Playbook**

To test our configuration, we will create a simple playbook to install Nginx on our managed node. First, we will create and open a new file. File names are not particularly important as far as Ansible is concerned. You should, of course, use descriptive file names. Make sure you are logged onto the control node as the admin user.

vim /home/ansible/install-nginx.yml

Ansible’[s playbooks](https://www.liquidweb.com/kb/how-to-install-and-configure-ansible/) are written in a language called YAML. YAML is intended to be human-readable. Looking at the text below, you should be able to tell what the expected results are. I’ll break the syntax down in just a moment, but for the time being, type ‘i’ to enter insert mode and add the following text to your playbook. Then type ‘[ESC]+:wq’ to save and exit.



Ansible’s playbooks execute ‘plays’. A play is a list of tasks that will be performed on the nodes. In the example above, we used the ‘hosts’ keyword to specify a list of just a single node. You can, however, specify a list of hosts, using comma-separated values. To install software with Ansible, we require root access to utilize yum. We use the keyword “become” in the play to instruct Ansible that the root user is required to execute the task.

The ‘tasks’ keyword initiates the list of tasks to be completed. Each task is given a unique name using the ‘name’ keyword. We then use the yum module provided by Ansible to install the first EPEL repository, and then we use the second entry to install Nginx.

**Step 8: Run the Playbook**

Running a playbook is rather easy. We use the “ansible-playbook” command and then specify the inventory file with the “-i” option followed by the path to the playbook. Make sure you are logged onto the control node as the admin user.

ansible-playbook -i /home/ansible/inventory /home/ansible/install-nginx.yml

**Conclusion**

We have only scratched the surface with Ansible so far. You can group the servers in your inventory together by using group names. This grouping allows you to execute playbooks on only your webservers or, only on your database servers. You can also run ad-hoc commands. Ad-hoc commands are tasks that you need to run only once. For instance, if you needed to reboot all of your web servers. All in all, Ansible is an excellent tool you can use to save time, money and effort to automate tasks across a single or multiple remote servers.