How TO work with ansible

Use this document to learn and use Ansible

Homework Tips Checklist for Parents

# Ansible Basics

# What is ansible doing

# How Is it working

# how to configure a server

# architecture diagram

Chapter 01. Ansible Basics

Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates.

Ansible’s main goals are simplicity and ease-of-use. It also has a strong focus on security and reliability, featuring a minimum of moving parts, usage of OpenSSH for transport (with other transports and pull modes as alternatives), and a language that is designed around auditability by humans–even those not familiar with the program.

Due to its simplicity Ansible is appropriate for managing all environments, from small setups with a handful of instances to enterprise environments with many thousands of instances.

Ansible manages machines in an agent-less manner. There is never a question of how to upgrade remote daemons or the problem of not being able to manage systems because daemons are uninstalled. Because OpenSSH is one of the most peer-reviewed open source components, security exposure is greatly reduced. Ansible is decentralized–it relies on your existing OS credentials to control access to remote machines. If needed, Ansible can easily connect with Kerberos, LDAP, and other centralized authentication management systems.

Chapter 02. What is Ansible Doing

Ansible is one of the most popular configuration management tools.

Configuration management is a process of controlling configuration changes in a way that the system maintains integrity over time. Even though the term did not originate in the IT industry, currently it is broadly used to refer to the software and the hardware. In this context, it concerns the following aspects:

Application configuration: This involves software properties that decide how the system works, which are usually expressed in the form of flags or properties files passed to the application, for example, the database address, the maximum chunk size for file processing, or the logging level. They can be applied during different development phases: build, package, deploy, or run.

Infrastructure configuration: This involves server infrastructure and environment configuration, which takes care of the deployment process. It defines what dependencies should be installed on each server and specifies the way applications are orchestrated (which application is run on which server and in how many instances).

***The configuration management tool reads the configuration file and prepares the environment respectively (installs dependent tools and libraries, deploys the applications to multiple instances).***

Chapter 03. How Is It Working

Ansible, one of the most popular configuration management tool is an open source product with free basic versions and paid enterprise editions.

Ansible uses YAML as Configuration Language and is agentless, using the standard SSH protocol for communication.

The agentless feature is a significant advantage because it implies no need to install anything on servers.

Ansible is quickly trending upwards and is often considered as the best tool to use for the Continous Delivery process.

So How does exactly Ansible work?

Ansible doesn’t break its process into steps or procedures. To understand how the software works however, we need to look at the different stages one at a time. This will show us how Ansible delivers a great performance with very little overhead.

1. Setting up the Architecture

Ansible connects nodes and pushes out small programs called ‘Ansible Modules’. These modules act as the resource for the desired state of the system. Ansible executes these modules and removes them once the execution is complete. As users, we just need to work on our terminal, text editor and in some cases a version controller to keep track of the changes.

2. SSH keys and their role

SSH keys are promoted as the best way to access Ansible. However; you can choose other options like Kerberos as well. Authorization keys are supported and they can control the access of different modules to different users, which will come in handy in many IT systems. Furthermore, root logins are not necessarily required to access Ansible.

Eg:

|  |  |
| --- | --- |
| 1  2 | ssh-agent bash  ssh-add ~/.ssh/id\_rsa |

**3. Managing Inventory**  
Ansible puts all the machines that it is managing on a system in a simple INI file. The machines are grouped as per the user’s choice inside the INI file. New machines can be added without any SSL signings, which also takes out the pain of NTP or DNS issues. As well as that, Ansible can easily plugin to sources such as Rackspace, EC2, Openstack and others. Here’s what the inventory looks like:

[webservers]  
www1.example.com  
www2.example.com

[dbservers]  
db0.example.com  
db1.example.com

**4. Using Ansible**  
Once your instance is available, you can just start using Ansible without any extra overheard. All the requirements in the form of resource modules and running commands have already been taken care of. Ansible itself comes with its entire arsenal full of modules. It couldn’t be more simple!  
Eg:

|  |  |
| --- | --- |
| 1  2  3 | ansible all -m ping  ansible foo.example.com -m yum -a "name=httpd state=installed"  ansible foo.example.com -a "/usr/sbin/reboot" |

**Key Features of Ansible:**  
1. Models the IT infrastructure around the systems interrelating with each other, thus ensuring faster end results.  
2. Module library can reside on any system, without the requirement of any server, daemons or databases.  
3. No additional setup required, so once you have the instance ready you can work on it straight away.  
4. Easier and faster to deploy as it doesn’t rely on agents or additional custom security infrastructure.  
5. Uses a very simple language structure called playbooks. Playbooks are almost similar to the plain English language for describing automation jobs.  
6. Ansible has the flexibility to allow user-made modules that can be written in any programming language such as Ruby, Python. It also allows adding new server side behaviors extending Ansible’s connection types through Python APIs.

Ansible works by configuring client machines from a computer that has the Ansible components installed and configured.

It communicates over normal SSH channels to retrieve information from remote machines, issue commands, and copy files. Because of this, an Ansible system does not require any additional software to be installed on the client computers.

This is one way that Ansible simplifies the administration of servers. Any server that has an SSH port exposed can be brought under Ansible's configuration umbrella, regardless of what stage it is at in its life cycle. This means that any computer that you can administer through SSH, you can also administer through Ansible.

Ansible takes on a modular approach, making it easy to extend to use the functionalities of the main system to deal with specific scenarios. Modules can be written in any language and communicate in standard JSON.

Configuration files are mainly written in the YAML data serialization format due to its expressive nature and its similarity to popular markup languages. Ansible can interact with hosts either through command line tools or its configuration scripts, which are known as Playbooks

Chapter 04. How To Configure A Server

Prerequisites

To follow this tutorial, you will need:

Two or more Ubuntu 18.04 servers. One of these will be used as your Ansible server, while the remainder will be used as your Ansible hosts. Each should have a non-root user with sudo privileges and a basic firewall configured.

The examples throughout this guide specify three Ansible hosts, but the commands and configurations shown can be adjusted for any number of clients.

*Step 1 — Installing Ansible*

To begin using Ansible as a means of managing your various servers, you need to install the Ansible software on at least one machine.

To get the latest version of Ansible for Ubuntu, you can add the project's PPA (personal package archive) to your system. Before doing this, though, you should first update your package index and install the software-properties-common package. This software will make it easier to manage this and other independent software repositories:

sudo apt update

sudo apt install software-properties-common

Then add the Ansible PPA by typing the following command:

sudo apt-add-repository ppa:ansible/ansible

Press ENTER to accept the PPA addition.

Next, refresh your system's package index once again so that it is aware of the packages available in the PPA:

sudo apt update

Following this update, you can install the Ansible software:

sudo apt install ansible

Your Ansible server now has all of the software required to administer your hosts.

*Step 2 — Configuring SSH Access to the Ansible Hosts*

As mentioned previously, Ansible primarily communicates with client computers through SSH. While it certainly has the ability to handle password-based SSH authentication, using SSH keys can help to keep things simple.

On your Ansible server, use the cat command to print the contents of your non-root user’s SSH public key file to the terminal’s output:

cat ~/.ssh/id\_rsa.pub

Copy the resulting output to your clipboard, then open a new terminal and connect to one of your Ansible hosts using SSH:

ssh sammy@ansible\_host\_ip

Switch to the client machine’s root user:

su -

As the root user, open the authorized\_keys within the ~/.ssh directory:

nano ~/.ssh/authorized\_keys

In the file, paste your Ansible server user’s SSH key, then save the file and close the editor (press CTRL + X, Y, then ENTER). Then run the exit command to return to the host’s non-root user:

exit

Lastly, because Ansible uses a python interpreter located at /usr/bin/python to run its modules, you’ll need to install Python 2 on the host in order for Ansible to communicate with it. Run the following commands to update the host’s package index and install the python package:

sudo apt update

sudo apt install python

Following this, you can run the exit command once again to close the connection to the client:

exit

Repeat this process for each server you intend to control with your Ansible server. Next, we’ll configure the Ansible server to connect to these hosts using Ansible’s hosts file.

*Step 3 — Setting Up Ansible Hosts*

Ansible keeps track of all of the servers that it knows about through a hosts file. We need to set up this file first before we can begin to communicate with our other computers.

Open the file with sudo privileges, like this:

sudo nano /etc/ansible/hosts

Inside the file, you will see a number of example configurations that have been commented out (with a # preceding each line). These examples won’t actually work for us since the hosts listed in each one are made up. We will, however, keep these examples in the file to help us with configuration if we want to implement more complex scenarios in the future.

The hosts file is fairly flexible and can be configured in a few different ways. The syntax we are going to use, though, looks like this:

[group\_name]

alias ansible\_ssh\_host=your\_server\_ip

In this example, group\_name is an organizational tag that lets you refer to any servers listed under it with one word, while alias is just a name to refer to one specific server.

So, in our scenario, we are imagining that we have three servers we are going to control with Ansible. At this point, these servers are accessible from the Ansible server by typing:

ssh root@ansible\_host\_ip

You should not be prompted for a password if you have set this up correctly. For the purpose of demonstration, we will assume that our hosts' IP addresses are 203.0.113.1, 203.0.113.2, and 203.0.113.3. We will set this up so that we can refer to these individually as host1, host2, and host3, or as a group with the name servers.

This is the block that we should add to our hosts file to accomplish this:

/etc/ansible/hosts

[servers]

host1 ansible\_ssh\_host=203.0.113.1

host2 ansible\_ssh\_host=203.0.113.2

host3 ansible\_ssh\_host=203.0.113.3

Hosts can be in multiple groups and groups can configure parameters for all of their members. Let's try this out now.

With our current settings, if we tried to connect to any of these hosts with Ansible, the command would fail (assuming you are not operating as the root user). This is because your SSH key is embedded for the root user on the remote systems and Ansible will by default try to connect as your current user. A connection attempt will get this error:

Output

host1 | UNREACHABLE! => {

"changed": false,

"msg": "Failed to connect to the host via ssh.",

"unreachable": true

}

On the Ansible server, we're using a user called sammy. Ansible will try to connect to each host with ssh sammy@server. This will not work if the sammy user is not on the remote system as well.

We can create a file that tells all of the servers in the "servers" group to connect as the root user.

To do this, we will create a directory in the Ansible configuration structure called group\_vars. Within this folder, we can create YAML-formatted files for each group we want to configure:

sudo mkdir /etc/ansible/group\_vars

sudo nano /etc/ansible/group\_vars/servers

We can put our configuration in here. YAML files start with "---", so make sure you don't forget that part.

/etc/ansible/group\_vars/servers

---

ansible\_ssh\_user: root

Save and close this file when you are finished.

If you want to specify configuration details for every server, regardless of group association, you can put those details in a file at /etc/ansible/group\_vars/all. Individual hosts can be configured by creating files named after their alias under a directory at /etc/ansible/host\_vars.

*Step 4 — Using Simple Ansible Commands*

Now that we have our hosts set up and enough configuration details to allow us to successfully connect to our hosts, we can try out our very first command.

Ping all of the servers you configured by typing:

ansible -m ping all

Ping output

host1 | SUCCESS => {

"changed": false,

"ping": "pong"

}

host3 | SUCCESS => {

"changed": false,

"ping": "pong"

}

host2 | SUCCESS => {

"changed": false,

"ping": "pong"

}

This is a basic test to make sure that Ansible has a connection to all of its hosts.

The all means all hosts. We could just as easily specify a group:

ansible -m ping servers

We could also specify an individual host:

ansible -m ping host1

We can specify multiple hosts by separating them with colons:

ansible -m ping host1:host2

The -m ping portion of the command is an instruction to Ansible to use the "ping" module. These are basically commands that you can run on your remote hosts. The ping module operates in many ways like the normal ping utility in Linux, but instead it checks for Ansible connectivity.

The ping module doesn't really take any arguments, but we can try another command to see how that works. We pass arguments into a script by typing -a.

The "shell" module lets us send a terminal command to the remote host and retrieve the results. For instance, to find out the memory usage on our host1 machine, we could use:

ansible -m shell -a 'free -m' host1

Shell output

*host1 | SUCCESS | rc=0 >>*

*total used free shared buffers cached*

*Mem: 3954 227 3726 0 14 93*

*-/+ buffers/cache: 119 3834*

*Swap: 0 0 0*

With that, your Ansible server configured and you can successfully communicate and control your hosts.

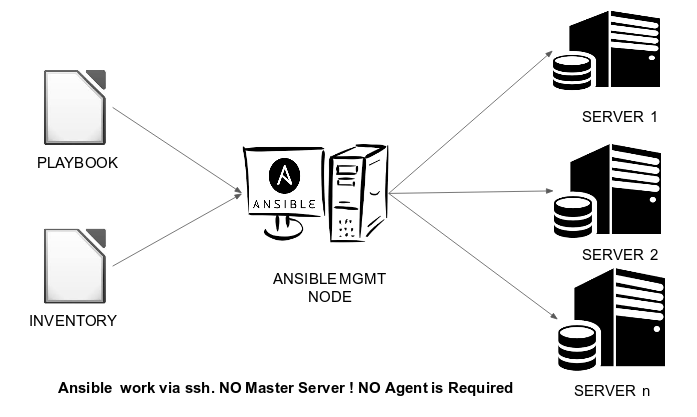
*Conclusion*

We have configured Ansible and verified that it can communicate with each host. We have also used the ansible command to execute simple tasks remotely.

Although this is useful, we have not covered the most powerful feature of Ansible in this document: Playbooks. Ansible Playbooks are a powerful, simple way to manage server configurations and multi-machine deployments. For an introduction to Playbooks, see this guide. Additionally, we encourage you to check out the official Ansible documentation to learn more about the tool.

Chapter 05. Architecture Diagram

Ansible mgmt node is the Ansible server which is responsible for configuring playbook over the server (Ubuntu/trusty64 or centos). A user needs to write scripts in a playbook and mention the IP Address in the inventory host. Just by running the ansible command, the user will be able to install scripts which are written in the playbook in each server and ensure that all the servers have passwordless ssh.



Concepts and Terminology:

Ansible mgmt node:- Ansible management node is the machine where the Ansible is installed & which is responsible for configuration of all server machines.

Inventory:- Inventory is a file which consists of IP Address of all servers on which configuration is to be done.

Playbook: Playbook is used to manage configuration which can be deployed on the remote machines. Playbook is designed in simple human-readable format, so that it becomes easy to understand the code.

Task:- Task is the file where a user defines the steps that he needs to perform on the Playbook.

Module:- Ansible has in-built modules that can be executed directly on remote hosts or through Playbooks. Users can also write their own modules. These modules can control system resources, like services, packages, or files or handle execution of system commands.

Role: Role is used for organizing Playbooks and other files in order to facilitate sharing and reusing portions of a provisioning.

Play: A provisioning executed from start to finish is called a Play.

Facts:Used to get the system information while running the playbook, like network interfaces or operating system.

Handlers: Handlers which are similar to a task, can be used to restart or to stop a service.