What are Ansible Playbooks?

Ansible playbooks are a way to send commands to remote computers in a scripted way. Instead of using Ansible commands individually to remotely configure computers from the command line, you can configure entire complex environments by passing a script to one or more systems.

Ansible playbooks are written in the YAML data serialization format. If you don't know what a data serialization format is, think of it as a way to translate a programmatic data structure (lists, arrays, dictionaries, etc) into a format that can be easily stored to disk. The file can then be used to recreate the structure at a later point. JSON is another popular data serialization format, but YAML is much easier to read.

Each playbook contains one or more plays, which map hosts to a certain function. Ansible does this through something called tasks, which are basically module calls.

## Exploring a Basic Playbook

Let's look at a basic playbook:

---

- hosts: droplets

tasks:

- name: Installs nginx web server

apt: pkg=nginx state=installed update\_cache=true

notify:

- start nginx

handlers:

- name: start nginx

service: name=nginx state=started

Let's break this down in sections so we can understand how these files are built and what each piece means.

The file starts with:

---

This is a requirement for YAML to interpret the file as a proper document. YAML allows multiple "documents" to exist in one file, each separated by ---, but Ansible only wants one per file, so this should only be present at the top of the file.

YAML is very sensitive to white-space, and uses that to group different pieces of information together. You should use only spaces and not tabs and you must use consistent spacing for your file to be read correctly. Items at the same level of indentation are considered sibling elements.

Items that begin with a - are considered list items. Items that have the format of key: value operate as hashes or dictionaries. That's pretty much all there is to basic YAML.

YAML documents basically define a hierarchical tree structure with the containing elements further to the left.

On the second line, we have this:

---

- hosts: droplets

This is a list item in YAML as we learned above, but since it is at the left-most level, it is also an Ansible "play". Plays are basically groups of tasks that are performed on a certain set of hosts to allow them to fulfill the function you want to assign to them. Each play must specify a host or group of hosts, as we do here.

Next, we have a set of tasks:

---

- hosts: droplets

tasks:

- name: Installs nginx web server

apt: pkg=nginx state=installed update\_cache=true

notify:

- start nginx

At the top level, we have "tasks:" at the same level as "hosts:". This contains a list (because it starts with a "-") which contains key-value pairs.

The first one, "name", is more of a description than a name. You can call this whatever you would like.

The next key is "apt". This is a reference to an Ansible module, just like when we use the ansible command and type something like:

ansible -m apt -a 'whatever' all

This module allows us to specify a package and the state that it should be in, which is "installed" in our case. The update-cache=true part tells our remote machine to update its package cache (apt-get update) prior to installing the software.

The "notify" item contains a list with one item, which is called "start nginx". This is not an internal Ansible command, it is a reference to a handler, which can perform certain functions when it is called from within a task. We will define the "start nginx" handler below.

---

- hosts: droplets

tasks:

- name: Installs nginx web server

apt: pkg=nginx state=installed update\_cache=true

notify:

- start nginx

handlers:

- name: start nginx

service: name=nginx state=started

The "handlers" section exists at the same level as the "hosts" and "tasks". Handlers are just like tasks, but they only run when they have been told by a task that changes have occurred on the client system.

For instance, we have a handler here that starts the Nginx service after the package is installed. The handler is not called unless the "Installs nginx web server" task results in changes to the system, meaning that the package had to be installed and wasn't already there.

We can save this playbook into a file called something like "nginx.yml".

Just for some context, if you were to write this same file in JSON, it might look something like this:

[

{

"hosts": "droplets",

"tasks": [

{

"name": "Installs nginx web server",

"apt": "pkg=nginx state=installed update\_cache=true",

"notify": [

"start nginx"

]

}

],

"handlers": [

{

"name": "start nginx",

"service": "name=nginx state=started"

}

]

}

]

As you can see, YAML is much more compact and most people would say more readable.

## Running an Ansible Playbook

Once you have a playbook built, you can call it easily using this format:

ansible-playbook playbook.yml

For instance, if we wanted to install and start up Nginx on all of our droplets, we could issue this command:

ansible-playbook nginx.yml

Since the playbook itself specifies the hosts that it should run against (namely, the "droplets" group we created in the last tutorial), we do not have to specify a host to run against.

However, if we would like to filter the host list to only apply to one of those hosts, we can add a flag to specify a subset of the hosts in the file:

ansible-playbook -l host\_subset playbook.yml

So if we only wanted to install and run Nginx on our "host3", we could type this:

ansible-playbook -l host3 nginx.yml

## Adding Features to the Playbook

Right now our playbook looks like this:

---

- hosts: droplets

tasks:

- name: Installs nginx web server

apt: pkg=nginx state=installed update\_cache=true

notify:

- start nginx

handlers:

- name: start nginx

service: name=nginx state=started

It is simple and it works, but all it is doing is installing a piece of software and starting it. That's not very beneficial by itself.

We can start to expand the functionality by adding tasks to our playbook.

### Add a Default Index File

We can tell it to transfer a file from our Ansible server onto the host by adding some lines like this:

---

- hosts: droplets

tasks:

- name: Installs nginx web server

apt: pkg=nginx state=installed update\_cache=true

notify:

- start nginx

- name: Upload default index.html for host

copy: src=static\_files/index.html dest=/usr/share/nginx/www/ mode=0644

handlers:

- name: start nginx

service: name=nginx state=started

We can then make a directory called static\_files in our current directory and place an index.html file inside.

mkdir static\_files

nano static\_files/index.html

Inside of this file, let's just create a basic html structure:

<html>

<head>

<title>This is a sample page</title>

</head>

<body>

<h1>Here is a heading!</h1>

<p>Here is a regular paragraph. Wow!</p>

</body>

</html>

Save and close the file.

Now, when we re-run the playbook, Ansible will check each task. It will see that Nginx is already installed on the host, so it will leave it be. It will see the new task section and replace the default index.html file with the one from our server.

### Registering Results

When you are installing and configuring services manually, it is almost always necessary to know whether your actions were successful or not. We can cook this functionality into our playbooks by using "register".

For each task, we can optionally register its result (failure or success) in a variable that we can check later on.

When using this functionality, we also have to tell Ansible to ignore errors for that task, since normally it aborts the playbook execution for that host if any trouble happens.

So, if we want to check whether a task has failed or not to decide on subsequent steps, we can use the register functionality.

For instance, we could tell our playbook to upload an index.php file if it exists. If that task fails, we could instead try to upload an index.html file. We will check for the failure condition in the other task because we only want to upload the HTML file if the PHP file fails:

---

- hosts: droplets

tasks:

- name: Installs nginx web server

apt: pkg=nginx state=installed update\_cache=true

notify:

- start nginx

- name: Upload default index.php for host

copy: src=static\_files/index.php dest=/usr/share/nginx/www/ mode=0644

register: php

ignore\_errors: True

- name: Remove index.html for host

command: rm /usr/share/nginx/www/index.html

when: php|success

- name: Upload default index.html for host

copy: src=static\_files/index.html dest=/usr/share/nginx/www/ mode=0644

when: php|failed

handlers:

- name: start nginx

service: name=nginx state=started

**Note**: We have not configured our host to handle PHP files at this time, so even if you did upload a PHP file, it would not be processed correctly.

This new version tries to upload a PHP index file to the host. It registers the success of the operation into a variable called "php".

If this operation was successful, the task to remove the index.html file is run next.

If the operation failed, the index.html file is uploaded instead.