Ansible - Orchestration of EC2 Instance

Report by: KUMAR MAYANK

Abstract :

Many organizations are using Containers as an integral part of the DevOps workflows because of their flexibility and portability. Ansible is the way to automate Docker container in our environment. Ansible enables to operationalize our Docker container build and deployment process in ways that one likely doing manually today. Playbooks are Ansible's configuration, deployment, and orchestration language. They can describe a policy we want our remote systems to enforce, or a set of steps in a general IT process. Ansible also contains a number of modules for controlling Amazon Web Services (AWS). Ansible automation can help to manage AWS environment like a fleet of services instead of a collection of servers.

Summary :

The main components of Ansible are: playbooks, configuration management, deployment, and orchestration. The first step to start with ansible is to setup ***Inventory.*** Ansible works against multiple systems in infrastructure at the same time. It does this by selecting portions of systems listed in Ansible's inventory, which defaults to being saved in the location **/etc/ansible/hosts**, providing nodes *IP* or *Domain Name* (as well ansible_ssh_user=″user_name" and ansible_ssh_pass="user_password"). Ansible can be operated over all the nodes provided in the Inventory or it also supports groups of nodes in Inventory, by using group names in square brackets, example: "[group_name1]". These groups can be used in classifying systems and deciding what systems are being controlled at what times and for what purpose.

After configuring Inventory file, all the nodes can be orchestrated by different ways such as using Ad-hoc commands or using Ansible-playbook. The simpler way or to test the node, Ad-hoc commands comes handy. For example, to test if all the nodes are reachable, rather than pinging each node we can ping all nodes with a single command - $ ***ansible all -m ping***

Playbooks are completely different way to use ansible than in ad-hoc task execution mode, and are particularly more powerful. Playbooks are expressed in YAML format. Each playbook is composed of one or more '*play'* in a list. The goal of a play is to map a group of hosts to well defined roles. Basic components in Playbooks are *hosts, remote_user, tasks* and *handlers. hosts:* lists one or more group, *remote_user:* is just a name of user, example(*root*). *tasks:* Each play contains a list of tasks. Tasks are executed in order, one at a time. The goal of each task is to execute a module, with very specific

arguments. *handlers:* Handlers are lists of tasks, not really any different from regular tasks, that are referenced by a globally unique name, and are notified by notifiers used in tasks.

Playbook can work more efficiently using *Roles,* making Playbook more organised and flexible. Roles are ways of automatically loading certain vars_files, tasks, and handlers based on a known file structure. Grouping content by roles also allows easy sharing of roles with other users. *Roles* uses a particular Directory Structure, each having a particular use case. ***tasks*** - contains the main list of tasks to be executed by the role. ***handler -*** contains handlers, which may be used by this role or even anywhere outside this role. ***vars -*** variables for the role. ***files -*** contains files which can be deployed via this role. ***templates -*** contains templates which can be deployed via this role. ***meta -*** defines some meta data for this role. Each directory must contain a ***main.yml*** file.

Major work done over research period was dedicated and getting hands on experience over the automation of AWS application with Ansible. *The power of AWS meets Ansible Simplicity.*

Ansible can be used to define, deploy, and manage a wide variety of AWS services. Even the most complicated of AWS environments can be easily described in Ansible playbooks. Ansible has nearly 100 modules supporting AWS capabilities.

First, a driver system is needed to be configured. One of AWS service ec2 instance is configured as the driver system. The driver ec2 instance (rhel 7 free tier) can be configure as, installing ***ansible, boto***, Ansible's EC2 module uses python-boto library to call AWS API, and boto needs AWS credentials in order to function. After setting up, update credentials in boto which is saved in location ***~/.boto*** .

For getting hands on experience during the research, I had used ansible to create a new security group, to launch a new ec2 instance, dynamically updating the Inventory file to set up the newly launched instance, installing apache server(httpd service), configuring httpd service to deploy webpages to document root other than default, and starting up the httpd service. Using ansible playbook, all the setup from creating new security group to launching an ec2 instance and configuring it, all can be done over a single command.

Elaboration:

After configuration of the driver ec2 instance and setting up the ansible playbook with roles and defined Directory structure, ansible-playbook is executed, which performs tasks one by one as: creating a new security group, create an EC2 key, saving EC2 key in driver instance, create an EC2 instance, dynamically updating Inventory by adding IP of new EC2 instance to hosts, wait for SSH to EC2 instance to come up, creating repository, installing httpd service, creating document root, adding webpages to document root, adding httpd conf file and starting up the *httpd* service.

For further explanation through illustrated code please follow link below:

https://github.com/max6746/ansible_aws