**How to get your code running on AWS virtual machine using Nexus Cloud and Ansible**



**Intro**

So, you have your amazing code runs on your local machine. That is nice. But wouldn’t it be much better to run it on the cloud?

If you agree this post is for you.

It sounds difficult to get your code running on the precious cloud, but in this post, I will show you step by step how to reach there.

We are going to use AWS as a cloud, and we will write some Ansible playbooks to handle all this magic.

Even if you are not familiar with AWS nor Ansible, stay with me.

Hello AWS

There are several cloud providers as you probably know. I chose to work with AWS.

We are going to use EC2 virtual machine. We will create an instance, upload our code into it and eventually will run it.

Ansible- welcome aboard

Ansible is an IT automation engine that automates many IT needs such as configuration management, cloud provisioning, application deployment and much more.

Ansible script which is called ‘playbook’ is written in YAML. It is a simple language that allows you to describe your automation job in a way that approaches plain English..

And finally, Nexus Repository

We are going to use Nexus Repository for pulling our latest repository.

Why Nexus? Because it supports all popular tools: Jenkins, Docker, Eclipse, IntelliJ, Helm, npm and more.

Pin, Office, Notice, Clerical, Tack, Violet Before we begin please verifying you have AWS and Nexus accounts.

**Body:**

Let’s do it! Let’s run our code on the cloud!

We are going to pull our version from our repository (GitLab in my case). Then we will push it to Nexus repo since we want our repo to be hosted on the cloud as well.

After that we will create a virtual machine on AWS called EC2. **(**Elastic Compute Cloud)

**We will copy repo from Nexus into EC2 instance and finally will deploy the code and run it.**

Steps of getting your code running on AWS virtual machine:

1. Pull your code from repository and push it to Nexus repository
2. Set up EC2 instance
3. Copy image from Nexus cloud to instance
4. Run your playbook
5. Run your code on EC2 instance

We will write a playbook for each step. 

1. Clone your repository and store it on Nexus Repository

--

# tasks file for preparing docker images in Nexus

- name: GitLab - Login

  shell: "docker login -u {{ user\_name }} -p {{ password }} https://gitlab.yourRepo.com:4567"

- name: Pull docker images

  shell: 'docker pull gitlab.yourRepo.com:4567/project:"{{ subversion }}"'

- name: GitLab - Logout

  shell: "docker logout https://gitlab.yourRepo.com:4567"

- name: Nexus-AWS - Login

  shell: "docker login -u {{ nexus\_user }} -p {{ nexus\_pass }} https://nexus.yourRepo.cloud:8085"

- name:  Nexus-AWS TAG

  shell: "docker tag gitlab.yourRepo.com:4567/project:{{ subversion }} nexus.yourRepo.cloud:8085/repository/docker-internal:{{ subversion }}"

- name:  Push to Nexus-AWS

  shell: "docker push nexus.yourRepo.cloud:8085/repository/docker-internal:{{ subversion }}"

As you can see in the playbook above, each stage has a name which describe it and a shell command.

Lines 3-4: login to our repository on GitLab. Need to replace {{user\_name}} and {{password}} with your real username and password. You should not write it hard coded on the playbook of course.

Lines 6-7: pull images from repository. Supply subversion you want to pull. (latest probably)

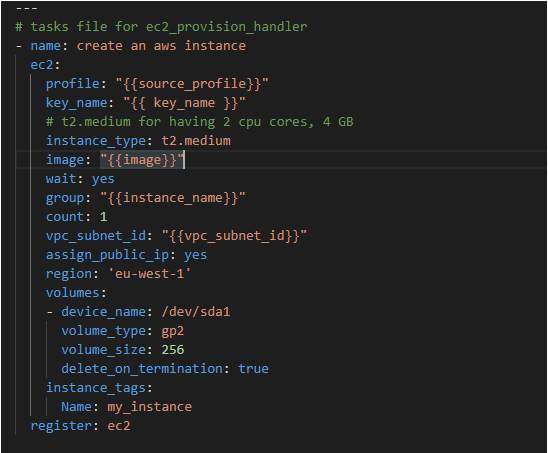
Lines 9-10: logout from repo. (after every login we have to logout)

Lines 12-13: login to Nexus. Need to replace {{nexus\_user}} and {{nexus\_pass}} with your real

username and password for Nexus repo.

We will call this playbook: prepare\_nexus\_images.yml. Now we will run it:

ansible-playbook prepare\_nexus\_images.yml --extra-vars {"user\_name":"my\_name", "password":"W3a$!GHY6re$%^H","nexus\_user":"my\_username", "nexus\_pass":"my\_nexus\_pass"}

1. Set up EC2 instance 

What do we have in this playbook?

Create an aws instance stage:

We build our ec2 instance. To do so we have some properties that we need to define:

Profile: Uses a boto profile

Key\_name: key pair to use on the instance. The SSH key must exist on AWS. You should see it on AWS GUI under Key Pairs on EC2 Dashboard:



instance\_type: which EC2 type you want to use

image: ami-id to use for this instance ([what is ami](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html)?)

wait: (yes or no) Wait for the instance to reach its desired state before returning.

group: security group or a list if security groups. ([what is security group for EC2 instance](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-security-groups.html)?)

count: how many instances do you want to set

vpc\_subnet\_id:  subnet ID in which to launch the instance. VPC stands for Virtual Private Cloud. It is a virtual network dedicated to each AWS account. Subnet is a range of IP addresses in your VPC.

assign\_public\_ip: (yes or no) if you want your instance to have a public IP address. If you need to reach it. In our case we need to reach it in the deploy stage.

region: your AWS account region

volumes: a list of hash/dictionaries of volumes to add to the new instance; ‘[{“key”:”value”, “key”:”value”}]’

volume\_type: gp2 = General Purpose SSD

delete\_on\_termination: (true or false) if you want to delete additional volumes that you attached to your instance, after your instance terminates. (by default, these volumes are not terminated)

instance\_tags: a hash/dictionary of tags to attach to your instance

register: ec2 registers the output in ‘ec2’ variable. You will be able to run query to find out different properties of your instance

Another block we want to add for later use is:

- add\_host:

    name: "{{ ec2.instances[0].public\_ip }}"

    groups: ec2\_host

We add EC2 public IP to group named ec2\_host to be able to run a playbook on this instance later.

Put it together with the above blocks and we get the full playbook for stage #2:

---

# tasks file for ec2\_provision\_handler

- name: create an aws instance

  ec2:

    profile: "{{source\_profile}}"

    key\_name: "{{ key\_name }}"

    # t2.medium for having 2 cpu cores, 4 GB

    instance\_type: t2.medium

    image: "{{image}}"

    wait: yes

    group: "{{instance\_name}}"

    count: 1

    vpc\_subnet\_id: "{{vpc\_subnet\_id}}"

    assign\_public\_ip: yes

    region: 'eu-west-1'

    volumes:

    - device\_name: /dev/sda1

      volume\_type: gp2

      volume\_size: 256

      delete\_on\_termination: true

    instance\_tags:

      Name: my\_instance

  register: ec2

- add\_host:

    name: "{{ ec2.instances[0].public\_ip }}"

    groups: ec2\_host

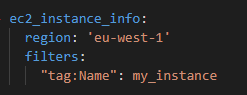
Let’s save it as ec2\_provision\_handler.yml

Some useful blocks:

* We can add debug messages by using debug section:

Here I print the public\_ip of my new EC2 instance.

* We can gather information about all instances in region: ‘eu-west-1’ with name = ‘my\_instance’ by using this block:



1. Pull image from Nexus into EC2 instance

- name: copy nexus cloud key to ec2 instance

  shell: scp -i "{{public\_key}}" -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null -r "{{nexus\_key}}" ubuntu@{{ec2.instances[0].public\_ip}}:/home/ubuntu/

- name: pull repo from nexus

  shell: docker pull nexus.yourRepo.cloud:8086/repository/docker-internal

First, we copy Nexus key into our EC2 instance. We want EC2 instance to communicate with Nexus machine. Therefore, it must have its key (pem file). To reach it we use the public IP address we assigned it. Notice that we use ‘scp’ command to ubuntu@public\_ip since ubuntu is the default user of EC2 machine. We need Nexus key to have permission to pull our repo from there.

Then we pull our uploaded repo from Nexus.

Let’s save this playbook as: ec2\_prepare\_handler.yml

1. Run your playbooks

By now you have 2 playbooks:

ec2\_provision\_handler.yml – runs on your local machine, pull repo, and set up EC2 instance.

ec2\_prepare\_handler.yml – runs on EC2 instance and pull repo from Nexus.

We will write a playbook which will run both playbooks on the relevant host.

It will look like this:

---

- hosts: localhost

  gather\_facts: False

  roles:

    - ec2\_provision\_handler

- hosts: ec2\_host

  become: true

  remote\_user: ubuntu

  roles:

    - ec2\_prepare\_handler

We will call it ec2\_handler.yml.

Now we will run it by using the following command:

ansible-playbook ec2\_handler.yml --extra-vars {"aws\_access\_key":"AKIA3G4WER3451", "aws\_secret\_key":"W3a$!fsdffsdfsdf354$%^H",

  "nexus\_user":"my\_username", "nexus\_pass":"my\_nexus\_pass", "profile":"my\_source\_profile", "image":"my\_image", "group":"my\_group", "vpc\_subnet\_id"="my\_subnet\_id"}

1. Now you have the deploy directory on your EC2 instance and all you need to do is to run your code the same way you run it on your local machine.

For those who survive until here:

We set up an EC2 instance, pushed our repo to Nexus and then connected to our instance and pulled the repo from Nexus.

We learnt to write Ansible playbooks and how to run it.

In future post I will write about how to automate the process by creating a pipeline in Jenkins.

Meanwhile you can create a vars file containing all variables that we pass when we run ec2\_handler.yml

We will use it in next post.

