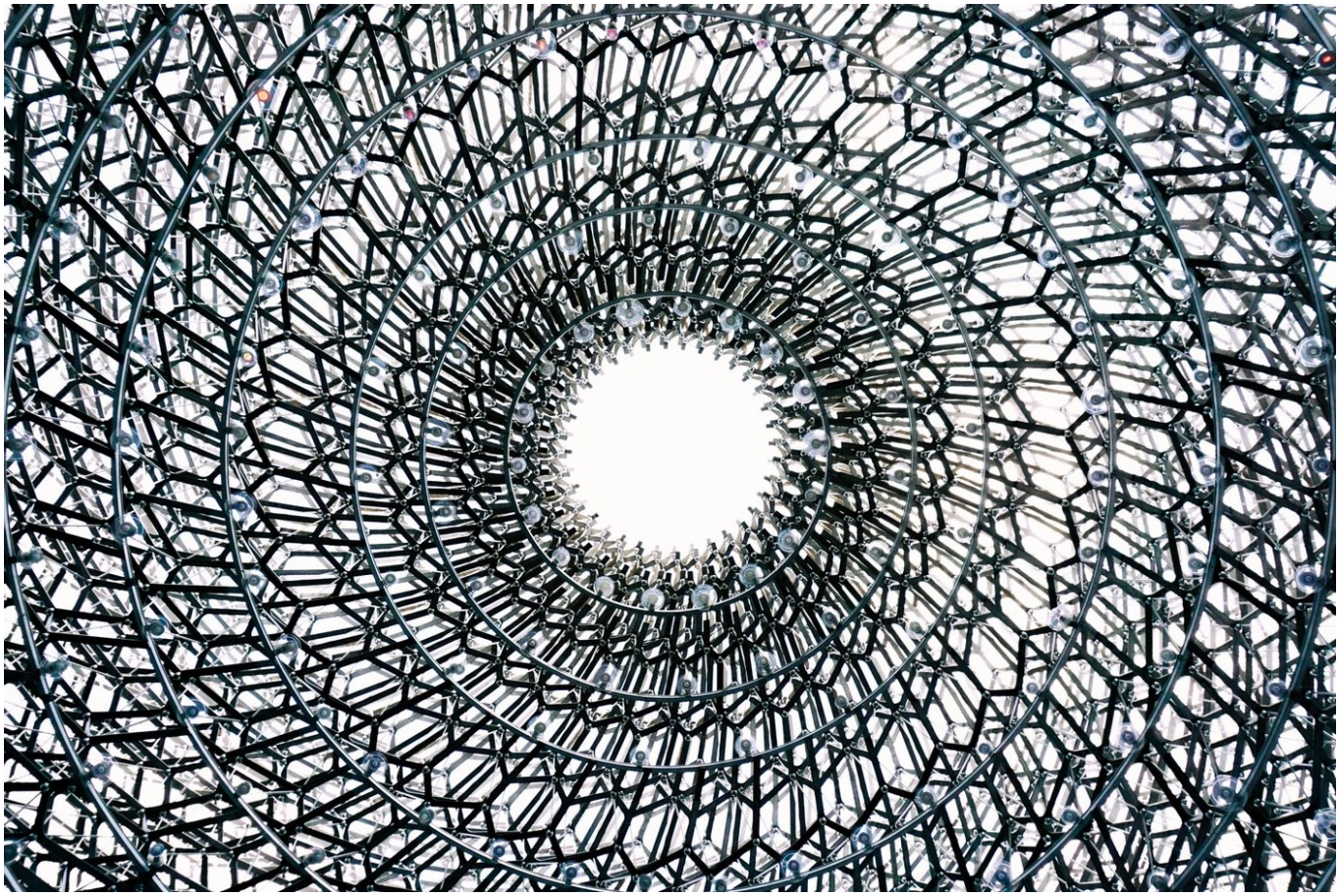
Docker swarm cluster on AWS with Ansible

Svyatoslav Pavlov • February 06, 2024



Introduction

In this tutorial we'll be using Ansible to deploy a **swarm cluster** of 3 nodes on AWS.

Ansible

From the official documentation: *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.*

Ansible scripts are meant to reach a desired `state` and not to do a list of `commands`. If you wish to install `curl` on a server, Ansible will try to install curl ONLY IF it isn't installed yet.

This is an ansible `task` which download a file at a specific `dest` with specific `user permissions`. And in case the file already exists at the desired `dest` it will only replace the file if the content has changed.

```
- name: Fetching super_saiyan.py
  get_url:
    url: https://vegeta.world/super_saiyan.py
    dest: /my/path/super_saiyan.py
    owner: goku
    group: dragonballz
    mode: u=rwx,g=rwx,o=rx
    force: yes
```

When running an ansible script you'll see **statuses** such as `ok` or `changed`.

- `ok`: nothing has been done, either the state is already met or we just set a `fact`.
- `changed`: A command has been run to meet the desired state.

Docker swarm

Docker swarm is an orchestration service. Easy to use, it allows us to run our webapps at scale in production. You can see a summary list of its features here.

Some ressources from Docker official documentation on how to setup and use Docker swarm:

- https://docs.docker.com/engine/swarm/swarm-tutorial/
- https://docs.docker.com/get-started/part4/

Setup and deploy ec2

For this project I'm using `Python 3.7.4` and `pip 19.0.3` and I'm running under `Fedora 30`. In this tutorial I assume:

- You have an AWS account
- You are familiar with `ec2` and `security groups` from AWS

- You have an IAM user with **Programmatic access** (AWS documention)

Dependencies

For this project we will need:

- ansible
- boto3
- boto
- botocore

```
# install dependencies
pip install ansible boto boto3 botocore
```

Here are the versions I have while I'm writing this.

```
[goku@vegeta tutorial]$ pip freeze | grep -E
"boto3|boto|botocore|ansible"
ansible==2.8.5
boto==2.49.0
boto3==1.9.233
botocore==1.12.233
```

Export in your shell
the `AWS_SECRET_ACCESS_KEY`, `AWS_ACCESS_KEY_ID` and `AWS_REGION` variables.

```
export AWS_ACCESS_KEY_ID='YOUR_AWS_API_KEY'
export AWS_SECRET_ACCESS_KEY='YOUR_AWS_API_SECRET_KEY'
# I'm currently on eu-west-1
export AWS_REGION='YOUR_AWS_REGION'
```

Let's test if everything is good. Create a file `test_ec2_playbook.yml`.

```
# test_ec2_playbook.yml
---
  # we don't connect to any servers, we use the AWS api
- hosts: localhost
  # Do not use ssh
  connection: local
```

```
  # Do not fetch variables from the server (there is none)
  gather_facts: False
  tasks:

    - name: "Gathering facts about all subnets and set fact:
'vpc_subnets_facts'"
      # AWS module to gather facts (variables) about subnets
      #
https://docs.ansible.com/ansible/latest/modules/ec2_vpc_subnet_facts_m
odule.html
      ec2_vpc_subnet_facts:
      # Create a fact "vpc_subnets_facts" from the result
      register: vpc_subnets_facts

    - name: "Printing 'vpc_subnets_facts' fact"
      #
https://docs.ansible.com/ansible/latest/modules/debug_module.html
      debug:
        var: vpc_subnets_facts
```

Now run it. It will print all your `subnets` in the region.

```
ansible-playbook test_ec2_playbook.yml
```

If you have something like this, you're all good ! (if you see warnings, ignore them).

```
[...]
                "id": "subnet-e6bd14bc",
                "ipv6_cidr_block_association_set": [],
                "map_public_ip_on_launch": true,
                "owner_id": "859761867273",
                "state": "available",
                "subnet_arn": "arn:aws:ec2:eu-west-
1:859761867273:subnet/subnet-e6bd14bc",
                "subnet_id": "subnet-e6bd14bc",
                "tags": {},
                "vpc_id": "vpc-453c0e23"
            },
            {
                "assign_ipv6_address_on_creation": false,
```

```
                "availability_zone": "eu-west-1c",
                "availability_zone_id": "euw1-az1",
                "available_ip_address_count": 4091,
                "cidr_block": "172.31.0.0/20",
                "default_for_az": true,
                "id": "subnet-94c0acf2",
                "ipv6_cidr_block_association_set": [],
                "map_public_ip_on_launch": true,
                "owner_id": "859761867273",
                "state": "available",
                "subnet_arn": "arn:aws:ec2:eu-west-
1:859761867273:subnet/subnet-94c0acf2",
                "subnet_id": "subnet-94c0acf2",
                "tags": {},
                "vpc_id": "vpc-453c0e23"
            }
        ]
    }
}

PLAY RECAP
*********************************************************************
*********************************
localhost                  : ok=2    changed=0    unreachable=0
failed=0    skipped=0    rescued=0    ignored=0
```

Choose one `subnet`, write down your `vpc_id` and your `subnet_id`. Here they are `"vpc_id": "vpc-453c0e23"` and `"subnet_id": "subnet-94c0acf2"`. If you have multiples, just pick one. We will need them later.

Setup architecture

Now we need to deploy our ec2 instances. In our script we will have 4 steps:

- Create `security_groups`
- Create a `key pair` (or use pre-existing one)
- Find an `AMI`, in this tutorial we'll use **Debian 9 (Stretch)**
- Create 3 `ec2`

Because we have 4 clear steps, we will write a file for each with their belonging **tasks**.

Create security groups

In AWS the security groups allow us to manage our ports. This is like a firewall around the server. It gives us features which make easier to manage our ports.

Create a folder `tasks`. Within it create a file named `create_security_groups.yml`.

Do not worry about `"{{ VPC_ID }}"` this is a **variable** and we will see later how to set a **value** for it.

```
# tasks/create_security_groups.yml
---
- name: create Swarm security groups
  #
https://docs.ansible.com/ansible/latest/modules/ec2_group_module.html
  ec2_group:
    # The id of the security group
    name: swarm
    description: >-
      Allow swarm members to communicate between themselves,
      allow ssh and open ports for the example voting app.
    # The shown name
    tags:
      Name: swarm
    # Left as it is, we will set the variable later
    vpc_id: "{{ VPC_ID }}"
    rules:
      - proto: tcp
        from_port: 22
        to_port: 22
        rule_desc: "Allow ssh from everywhere."
        cidr_ip: 0.0.0.0/0
        cidr_ipv6: ::/0
      - proto: tcp
        from_port: 2377
        to_port: 2377
        group_name: swarm
        rule_desc: "Cluster management communications."
      - proto: tcp
```

```
        from_port: 7946
        to_port: 7946
        group_name: swarm
        rule_desc: "Communication among nodes."
      - proto: udp
        from_port: 7946
        to_port: 7946
        group_name: swarm
        rule_desc: "Communication among nodes."
      - proto: udp
        from_port: 4789
        to_port: 4789
        group_name: swarm
        rule_desc: "Overlay network traffic."
      - proto: tcp
        from_port: 8080
        to_port: 8080
        rule_desc: "Open ports for visualizer."
        cidr_ip: 0.0.0.0/0
        cidr_ipv6: ::/0
      - proto: tcp
        from_port: 5000
        to_port: 5001
        rule_desc: "Open ports for the vote and result apps."
        cidr_ip: 0.0.0.0/0
        cidr_ipv6: ::/0
```

Here we create our `security_group` in order to allow `ssh`, the ports for the `service` we will deploy and the ports needed by `docker swarm` but only limited to the members of the `swarm`.

Create key pair (or use pre-existing one)

In order to connect by ssh to our servers will we need a **key pair**.

```
# tasks/create_key_pair.yml
---
- name: "Trying to create a key pair"
  #
https://docs.ansible.com/ansible/latest/modules/ec2_key_module.html
```

```
  ec2_key:
    name: "{{ KEY_PAIR_NAME }}"
    # If it exists do not overwrite
    force: false
  register: ansible_tutorial_private_key
  # Do not stop the playbook if it fails
  ignore_errors: yes

#################################################################
# ALL the code below will only run if the key pair is created #
# look at the "when" keys which controls when to run a task   #
#################################################################

- name: "Saving private key to {{ KEY_PAIR_LOCAL_PATH }}"
  # https://docs.ansible.com/ansible/latest/modules/copy_module.html
  copy:
    dest: "{{ KEY_PAIR_LOCAL_PATH }}"
    content: "{{ ansible_tutorial_private_key['key']['private_key']
}}"
    # Permissions for private key
    mode: '0600'
    # We do not want to overwrite a pre existing key
    force: no
  # We wish to run another task in case of error
  register: save_ansible_tutorial_private_key
  # ONLY run if "ansible_tutorial_private_key" was a success
  when: ansible_tutorial_private_key.changed == true
  # Do not stop the playbook if it fails
  ignore_errors: yes

- name: "Deleting {{ KEY_PAIR_NAME }} since it couldn't be saved"
  #
https://docs.ansible.com/ansible/latest/modules/ec2_key_module.html
  ec2_key:
    name: "{{ KEY_PAIR_NAME }}"
    # Delete it
    state: absent
  # ONLY run if the previous tasks didn't write to the file
  when: ansible_tutorial_private_key.changed == true and
save_ansible_tutorial_private_key.changed == false
```

```
  register: delete_key_pair

  # Stop the playbook and mark it as failed
  # https://docs.ansible.com/ansible/latest/modules/fail_module.html
- fail:
    msg: >
      The file {{ KEY_PAIR_LOCAL_PATH }} already exists but the
      key pair {{ KEY_PAIR_NAME }} doesn't exists on AWS.
      Change either the KEY_PAIR_NAME or the KEY_PAIR_LOCAL_PATH
  # ONLY run if the previous tasks didn't write to the file
  when: delete_key_pair.changed == true
```

When you create a **key pair**, AWS gives you the private key.

:warning: This is the only time you have access to it ! if you don't download it or loose it, the **key pair** becomes useless.

There is 4 ways those `tasks` can be triggered:

- Neither the key pair nor the file exist. The script will create the key pair and create a file with its private key.
- The key pair exists and the file too. Nothing is done since this is what we already want.
- The key pair doesn't exist but the file does. We won't erase the file as we do not know what is contains, we will delete the key pair as we couldn't save the private key and raise an error asking the user to check his variables.
- The key pair exists but the file doesn't. We will delete the key pair as we can't save the private key and raise an error asking the user to check his variables.

Find an AMI Debian 9 (Stretch)

```
# tasks/set_fact_ec2_ami.yml
---
- name: "set fact: Debian 9 (Stretch) AMI"
  #
https://docs.ansible.com/ansible/latest/modules/ec2_ami_facts_module.h
tml
```

```
  ec2_ami_facts:
    # see the full list of filtered elements:
    # https://eu-west-1.console.aws.amazon.com/ec2/home?region=eu-
west-1#Images:visibility=public-images;search=379101102735,debian-
stretch-hvm-x86_64-gp2-*;sort=desc:name
    owners: 379101102735
    filters:
      name: "debian-stretch-hvm-x86_64-gp2-*"
  register: debian_ami

- name: "set fact: get id from Debian 9 (Stretch) AMI"
  #
https://docs.ansible.com/ansible/latest/modules/set_fact_module.html
  set_fact:
    # There is many debian AMI, let's take the first
    debian_ami_id: "{{ debian_ami.images[0].image_id }}"
```

Those tasks are straightforward. We look for a **Debian 9 (Stretch)** AMI made by Debian (379101102735), take the first and save the id in a `fact`.

Create ec2

We wish to create 3 ec2. 1 **manager** and 2 **workers**. We will iterate over a **variable** with the key `loop`.

```
# tasks/create_ec2.yml
---
- name: Creating ec2 instances
  # https://docs.ansible.com/ansible/latest/modules/ec2_module.html
  ec2:
    key_name: "{{ KEY_PAIR_NAME }}"
    # The security group
    group: "{{ item.group }}"
    instance_type: t2.nano
    image: "{{ debian_ami_id }}"
    # Wait until the ec2 is created (not running yet)
    wait: yes
    # Garanty only 1 ec2 with those count_tags
    exact_count: 1
    # Tags to identify uniquely an ec2
```

```
    count_tag:
      Name: "{{ item.tags.Name }}"
    # Other tags to attach to the ec2
    instance_tags: "{{ item.tags }}"
    # The subnet where to create the ec2
    vpc_subnet_id: "{{ SUBNET_ID }}"
    assign_public_ip: yes
  # Run this tasks for all elements in EC2_INSTANCES
  loop: "{{ EC2_INSTANCES }}"
  register: create_ec2
```

Notice the `loop: "{{ EC2_INSTANCES }}"` at the end of the task. It iterates the same module over the content of `EC2_INSTANCES` var.

Now let's create our `EC2_INSTANCES` variable in a file. Create a folder `vars` and within it a file named `ec2.yml`.

```
# vars/ec2.yml
---
- EC2_INSTANCES:
  - tags:
      Name: "swarm-manager"
      SwarmType: manager
      Swarm: yes
    group:
      - swarm
  - tags:
      Name: "swarm-worker-a"
      SwarmType: worker
      Swarm: yes
    group:
      - swarm
  - tags:
      Name: "swarm-worker-b"
      SwarmType: worker
      Swarm: yes
    group:
      - swarm
```

We store the `tags` and the `security_group` properties.

Create the playbook

In order to use our new tasks we will create a **playbook** which includes all 4 of them. Just like the **playbook** we wrote for the test. Create a `create_ec2_playbook.yml` file at the root of your project.

```
# create_ec2_playbook.yml
---
- hosts: localhost
  connection: local
  gather_facts: False
  vars:
    # The vpc_id and subnet_id choosen earlier
    VPC_ID: "YOUR_VPC_ID"
    SUBNET_ID: "YOUR_SUBNET_ID"
    # The key pair name on AWS
    KEY_PAIR_NAME: ansible_tutorial
    # The path to the private key on your machine
    KEY_PAIR_LOCAL_PATH: "~/.ssh/ansible_tutorial.pem"
  # Load variables from file
  vars_files:
    - vars/ec2.yml
  tasks:
      # Include all tasks within file
    - include_tasks: tasks/create_security_groups.yml
    - include_tasks: tasks/create_key_pair.yml
    - include_tasks: tasks/set_fact_ec2_ami.yml
    - include_tasks: tasks/create_ec2.yml
```

Please fill the `VPC_ID` and `SUBNET_ID` vars with the `vpc_id` and the `subnet_id` we have chosen after running `ansible-playbook test_ec2_playbook.yml`.

(optional) If you wish to use a pre-existing `key_pair` please change `KEY_PAIR_NAME` to its name on AWS and `KEY_PAIR_LOCAL_PATH` to the local path to the private key on your machine.

Your current project structure should look like this now.

```
[goku@vegeta tutorial]$ tree
.
├── create_ec2_playbook.yml
├── tasks
│   ├── create_ec2.yml
│   ├── create_key_pair.yml
│   ├── create_security_groups.yml
│   └── set_fact_ec2_ami.yml
├── test_ec2_playbook.yml
└── vars
    └── ec2.yml

2 directories, 7 files
```

Now run it !

```
ansible-playbook create_ec2_playbook.yml
```

Setup Docker swarm cluster

Now that we have our architecture running we wish to deploy a docker **swarm cluster**.

Inventory

In order to run our scripts against the desired servers Ansible uses an `inventory`. In short, an `inventory` is where you store your **servers' ip addresses**. You name them, create groups, setup variables by hosts / groups, it has a lot of features. It can be a folder or file.

But in our case, we create our servers **dynamically**, which means each time we deploy new servers, their **ip address** changes. So we will use the **AWS ec2 plugin** which creates dynamic `hosts`.

If you wish to know more about **inventories** you check the documentation here.

Ansible and Dynamic Amazon EC2 Inventory Management

Create an `inventory` folder at the project root. Within, we will create two files:

- `ec2.py` the AWS core plugin, you can download it here

- `ec2.ini` the configuration file, here

Make `ec2.py` executable to avoid any issues with ansible.

```
chmod +x inventory/ec2.py
```

Test if the plugin is working as desired. It may take a few moment to get the result as the plugin is doing multiples requests on AWS

```
inventory/ec2.py --list
```

You should see something like this:

```
[...]
       "ec2_tag_Name": "swarm-worker-b",
        "ec2_tag_Swarm": "True",
        "ec2_tag_SwarmType": "worker",
        "ec2_virtualization_type": "hvm",
        "ec2_vpc_id": "vpc-453c0e23"
      }
    }
  },
  "ec2": [
    "34.254.184.38",
    "34.244.247.140",
    "34.255.136.188"
  ],
  "tag_Name_swarm_manager": [
    "34.254.184.38"
  ],
  "tag_Name_swarm_worker_a": [
    "34.244.247.140"
  ],
  "tag_Name_swarm_worker_b": [
    "34.255.136.188"
  ],
  "tag_SwarmType_manager": [
    "34.254.184.38"
  ],
  "tag_SwarmType_worker": [
```

```
    "34.244.247.140",
    "34.255.136.188"
  ],
  "tag_Swarm_True": [
    "34.254.184.38",
    "34.244.247.140",
    "34.255.136.188"
  ]
}
```

Here you can see all the different **servers**, **groups** and **variables** available.

if you have any troubles or wish to know more about it, here is the documentation.

Project setup

Now create an `ansible.cfg` file at the root of our project to specify a few default options.

```
# ansible.cfg
[defaults]
# (optional) the host check keys from ssh
# is enabled by default and may causes
# issues while following the tutorial
host_key_checking=false

# Tells ansible where the inventory files are
inventory=inventory/
```

Here we disable `host_key_checking` to remove constraints from this tutorial as it may causes issues (it is security related). We also define where the `inventory files` are.

We have set the `KEY_PAIR_NAME` and `KEY_PAIR_LOCAL_PATH` variables in the `create_ec2_playbook.yml` playbook. But we will need them in our next playbooks. So we'll move them in a variable file.

Create a folder `group_vars` and create a file `all.yml` in it. Move your `KEY_PAIR_NAME` and `KEY_PAIR_LOCAL_PATH` variables in it.

```
# group_vars/all.yml
---
KEY_PAIR_NAME: ansible_tutorial
KEY_PAIR_LOCAL_PATH: "~/.ssh/ansible_tutorial.pem"
```

All files within `groups_vars` are automatically read by Ansible depending on the current `group` or `host` currently running. Here `all.yml` will be used for **all** `hosts` or `groups`. This way we ensure those variables are accessible from everywhere.

By default our `ec2` can be accessed with the user `admin` (this is related to the AMI we're using). To specify Ansible which user and which private key to use we will create an other file `ec2.yml` in `group_vars`. All **dynamic hosts** from the AWS ec2 plugin are part of the **ec2 group**.

```
# group_vars/ec2.yml
---
ansible_user: admin
ansible_ssh_private_key_file: "{{ KEY_PAIR_LOCAL_PATH }}"
```

Those variables will be used ONLY by the `hosts` that belonged to the **ec2 group**.

To check everything is setup correctly we'll write a little playbook. **Note** this test will work only if your servers are **online** and **running**.

```
# test_dynamic_aws_host_playbook.yml
---
  # Run the playbook for all ec2 instances sharing the tag Swarm=True
- hosts: "tag_Swarm_True"
  tasks:
    - name: "Ping the ec2 with Swarm=True tag"
      #
https://docs.ansible.com/ansible/latest/modules/ping_module.html
      ping:
```

Our current project structure should look like this:

```
[goku@vegeta tutorial]$ tree
```

```
.
├── ansible.cfg
├── create_ec2_playbook.yml
├── group_vars
│   ├── all.yml
│   └── ec2.yml
├── inventory
│   ├── ec2.ini
│   └── ec2.py
├── tasks
│   ├── create_ec2.yml
│   ├── create_key_pair.yml
│   ├── create_security_groups.yml
│   └── set_fact_ec2_ami.yml
├── test_dynamic_aws_host_playbook.yml
├── test_ec2_playbook.yml
└── vars
    └── ec2.yml

4 directories, 13 files
```

Run it (may take a few moments because of the AWS dynamic host)

```
ansible-playbook test_dynamic_aws_host_playbook.yml
```

If you see something like this everything is working as expected :ok_hand:

```
PLAY [tag_Swarm_True]
*********************************************************************
*********************************************

TASK [Gathering Facts]
*********************************************************************
*********************************************
ok: [34.240.55.248]
ok: [34.242.96.200]
ok: [18.203.135.17]

TASK [Ping the ec2 with Swarm=True tag]
*********************************************************************
****************************
```

```
ok: [18.203.135.17]
ok: [34.240.55.248]
ok: [34.242.96.200]

PLAY RECAP
*********************************************************************
*****************************************************
18.203.135.17              : ok=2    changed=0    unreachable=0
failed=0    skipped=0    rescued=0    ignored=0
34.240.55.248              : ok=2    changed=0    unreachable=0
failed=0    skipped=0    rescued=0    ignored=0
34.242.96.200              : ok=2    changed=0    unreachable=0
failed=0    skipped=0    rescued=0    ignored=0
```

Something is wrong if you see something like this :collision:

```
PLAY [tag_Swarm_true]
*********************************************************************
*********************************************
skipping: no hosts matched
```

It means either your `ec2` aren't online and running or they do not have the tag `Swarm=True` (case sensitive).

Install Docker

We will install **Docker** on a **Debian 9 (Stretch)**. I've followed the instruction from the official Docker documentation and wrote the tasks.

```
# tasks/install_docker.yml
---
- name: Ensure dependencies are installed
  # https://docs.ansible.com/ansible/latest/modules/apt_module.html
  apt:
    name:
      - apt-transport-https
      - ca-certificates
      - curl
      - gnupg-agent
      - software-properties-common
    state: present
```

```
- name: Add Docker apt key
  #
https://docs.ansible.com/ansible/latest/modules/apt_key_module.html
  apt_key:
    url: https://download.docker.com/linux/debian/gpg
    id: 9DC858229FC7DD38854AE2D88D81803C0EBFCD88
    state: present
  register: add_repository_key

- name: Get debian version
  #
https://docs.ansible.com/ansible/latest/modules/command_module.html
  command: lsb_release -cs
  register: debian_version_name

- name: Add Docker repository
  #
https://docs.ansible.com/ansible/latest/modules/apt_repository_module.
html
  apt_repository:
    repo: "deb [arch=amd64] https://download.docker.com/linux/debian
{{ debian_version_name.stdout }} stable"
    state: present
    update_cache: true

- name: Ensure docker is installed
  # https://docs.ansible.com/ansible/latest/modules/apt_module.html
  apt:
    name:
      - docker-ce
      - docker-ce-cli
      - containerd.io
    state: present
    update_cache: true

- name: Ensure Docker is started and enabled at boot
  #
https://docs.ansible.com/ansible/latest/modules/service_module.html
  service:
```

```
    name: docker
    state: started
    enabled: true

- name: Ensure handlers are notified now to avoid firewall conflicts
  # https://docs.ansible.com/ansible/latest/modules/meta_module.html
  meta: flush_handlers

- name: "Ensure the user {{ ansible_user }} is part of the docker
group"
  # https://docs.ansible.com/ansible/latest/modules/user_module.html
  user:
    name: "{{ ansible_user }}"
    groups: docker
    append: yes

########################################################################
####################
#               We install the docker python module as it is
recommended               #
#                   for the docker_swarm module that we will use later
#
#
https://docs.ansible.com/ansible/latest/modules/docker_swarm_module.ht
ml#requirements #
########################################################################
####################

- name: Ensure docker python module and jsondiff are installed
  # https://docs.ansible.com/ansible/latest/modules/pip_module.html
  pip:
    name:
      - docker
      # jsondiff and pyyaml are needed by the docker_stack module
      - jsondiff
      - pyyaml
  register: pip_install_docker
  ignore_errors: yes

- name: Fetching pip
```

```
  #
https://docs.ansible.com/ansible/latest/modules/get_url_module.html
  get_url:
    url: https://bootstrap.pypa.io/get-pip.py
    dest: "/home/{{ ansible_user }}/get-pip.py"
    mode: u=rwx,g=rwx,o=rx
  when: pip_install_docker is failed

- name: Installing pip
  #
https://docs.ansible.com/ansible/latest/modules/command_module.html
  command: "python /home/{{ ansible_user }}/get-pip.py"
  when: pip_install_docker is failed

- name: Installing docker python module and jsondiff
  # https://docs.ansible.com/ansible/latest/modules/pip_module.html
  pip:
    name:
      - docker
      # jsondiff and pyyaml are needed by the docker_stack module
      - jsondiff
      - pyyaml
  when: pip_install_docker is failed
```

Init Docker Swarm

Docker is installed, we can now init the **Swarm** on a **manager node**. Docker Swarm is easy to setup:

- Init the Swarm on a server and save the `join token worker` and `join token manager`
- On the other `managers` make them join the Swarm thanks to the `join token manager`
- On the `workers` use the `join token worker`

We will ask **Ansible** to use the first **Swarm manager node** to init the swarm. We could specify a specific server but I prefer to just take the first one. In a large **swarm**

cluster you have multiple `manager` and `worker` nodes and by taking the first, our script will still be working in case the very server we specified is `down` for example.

```
# tasks/init_swarm.yml
---
- name: Setting up Swarm
  # Similar to a Try/Except
  #
https://docs.ansible.com/ansible/latest/user_guide/playbooks_blocks.ht
ml#blocks-error-handling
  block:
    - name: Initiating the swarm with default parameters
      #
https://docs.ansible.com/ansible/latest/modules/docker_swarm_module.ht
ml
      docker_swarm:
        state: present
      register: init_swarm

    - name: "set fact: join token worker"
      set_fact:
        join_token_worker: "{{
init_swarm.swarm_facts.JoinTokens.Worker }}"

    - name: "set fact: join token manager"
      set_fact:
        join_token_manager: "{{
init_swarm.swarm_facts.JoinTokens.Manager }}"

  rescue:
    - name: Getting join manager token from existing Swarm
      #
https://docs.ansible.com/ansible/latest/modules/command_module.html
      command: docker swarm join-token manager -q
      register: join_token_manager_command

    - name: Getting join worker token from existing Swarm
      #
https://docs.ansible.com/ansible/latest/modules/command_module.html
      command: docker swarm join-token worker -q
```

```
      register: join_token_worker_command

    - name: "set fact: join_token_manager"
      set_fact:
        join_token_manager: "{{ join_token_manager_command['stdout']
}}"

    - name: "set fact: join_token_worker"
      set_fact:
        join_token_worker: "{{ join_token_worker_command['stdout'] }}"
```

Join the worker nodes

Our Swarm is initiated, from the **worker nodes**, use the `join token worker` and specify a **manager node's address** to make them part of the Swarm.

```
# tasks/join_worker_node.yml
---
- name: "set fact: first Swarm manager host"
  set_fact:
    first_swarm_manager_host: "{{ groups['tag_SwarmType_manager'][0]
}}"

- name: 'set fact: list remote_addrs'
  set_fact:
    # Create a list of all managers' private ip addresse
    list_remote_addrs: >-
      {{ list_remote_addrs | default([]) }} + [ '{{ item }}:2377' ]
  loop: >-
    {{ groups['tag_SwarmType_manager'] | map('extract', hostvars,
'ec2_private_ip_address') | list }}

- name: Joining worker to the swarm
  #
https://docs.ansible.com/ansible/latest/modules/docker_swarm_module.ht
ml
  docker_swarm:
    state: join
    timeout: 60
```

```
    # Using PRIVATE IP ADDRESS as they are in the same VPC
    advertise_addr: >-
      {{ ec2_private_ip_address }}:2377
    # hostvars contains all variables related to a host
    #
https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables
.html#accessing-information-about-other-hosts-with-magic-variables
    join_token: >-
      {{ hostvars[first_swarm_manager_host].join_token_worker }}
    # Using PRIVATE IP ADDRESS as they are in the same VPC
    remote_addrs: "{{ list_remote_addrs }}"
```

Here we use the `hostvars`. The `hostvars` allows us to access variables from other hosts, here we take two variables:

- `join_token_worker` the token given by the first **swarm manager** who allows us to join the Swarm
- `ec2_private_ip_address` the **private ip address**. Since our servers share the same `subnet` they can communicate by **private ip address**. And our security groups **ONLY** opens the **Docker ports** to the servers member of the **swarm** security group and this implies communicating by **private ip address**.

```
# deploy_swarm_playbook.yml
---
  # Take the first ec2 whose tag SwarmType is equal to "manager"
- hosts: tag_SwarmType_manager[0]
  # Use sudo for all tasks
  become: yes
  tasks:
    - include_tasks: tasks/install_docker.yml
    - include_tasks: tasks/init_swarm.yml

- hosts: tag_SwarmType_worker
  # Use sudo for all tasks
  become: yes
  tasks:
    - include_tasks: tasks/install_docker.yml
    - include_tasks: tasks/join_worker_node.yml
```

Deploy the cluster :rocket: (if it is `unreachable` wait a bit until the `ec2` get to the `running` state).

```
ansible-playbook deploy_swarm_playbook.yml
```

Check cluster

Our `Swarm cluser` should be ready. To check it let's connect to it by ssh.

(Optional) If you're too lazy to check your `swarm manager node's ip` upper or from the **aws console** you can run this command:

```
[goku@vegeta tutorial]$ inventory/ec2.py --list | grep
tag_SwarmType_manager -A 2

# output
"tag_SwarmType_manager": [
    "34.242.96.200" # <- here
  ],
# ajust ~/.ssh/ansible_tutorial.pem if needed
[goku@vegeta tutorial]$ ssh -i ~/.ssh/ansible_tutorial.pem
admin@34.242.96.200

admin@ip-172-31-8-57:~$ docker node ls

# Tada ! Your swarm is running
ID                            HOSTNAME            STATUS
AVAILABILITY        MANAGER STATUS      ENGINE VERSION
scqe6rb9h0016g8wmeohfkv32     ip-172-31-5-233     Ready
Active                                    19.03.2
yhokk5ae4l22ssmpkslqbz4p7 *   ip-172-31-8-57      Ready
Active              Leader                19.03.2
td9vjq0myikqhr7dzwky8enut     ip-172-31-15-91     Ready
Active                                    19.03.2
```

Deploy service

We have our **swarm cluster** running. It is time to deploy our first service.

I'm using a stack file from the repo https://github.com/dockersamples. Their projects are complex enough for development and testing. We will be using the docker-stack.yml from the `example-voting-app`. It uses `redis`, `postgres`, an app in `python`, an app in `nodejs`, a worker in `.NET` and even a visualizer !

I've created a `example-voting-app` folder and a `docker-compose.yml` file in it.

```
# example-voting-app/docker-compose.yml
version: "3"
services:

  redis:
    image: redis:alpine
    networks:
      - frontend
    deploy:
      replicas: 1
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure
  db:
    image: postgres:9.4
    volumes:
      - db-data:/var/lib/postgresql/data
    networks:
      - backend
    deploy:
      placement:
        constraints: [node.role == manager]
  vote:
    image: dockersamples/examplevotingapp_vote:before
    ports:
      - 5000:80
    networks:
      - frontend
    depends_on:
      - redis
    deploy:
```

```
      replicas: 2
      update_config:
        parallelism: 2
      restart_policy:
        condition: on-failure
  result:
    image: dockersamples/examplevotingapp_result:before
    ports:
      - 5001:80
    networks:
      - backend
    depends_on:
      - db
    deploy:
      replicas: 1
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure

  worker:
    image: dockersamples/examplevotingapp_worker
    networks:
      - frontend
      - backend
    depends_on:
      - db
      - redis
    deploy:
      mode: replicated
      replicas: 1
      labels: [APP=VOTING]
      restart_policy:
        condition: on-failure
        delay: 10s
        max_attempts: 3
        window: 120s
      placement:
        constraints: [node.role == manager]
```

```
  visualizer:
    image: dockersamples/visualizer:stable
    ports:
      - "8080:8080"
    stop_grace_period: 1m30s
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints: [node.role == manager]

networks:
  frontend:
  backend:

volumes:
  db-data:
```

Create a new task file `deploy_example_voting_app_stack.yml` under `tasks`:

```
# tasks/deploy_example_voting_app_stack.yml
---
- name: Copying example voting app's docker-compose.yml file on the
server
  # https://docs.ansible.com/ansible/latest/modules/copy_module.html
  copy:
    src: "example-voting-app/docker-compose.yml"
    dest: "/home/{{ ansible_user }}/docker-compose.yml"
    owner: "{{ ansible_user }}"
    group: "{{ ansible_user }}"
    mode: u=rw,g=rw,o=r
    force: yes

- name: Deploying example voting app's stack
  #
https://docs.ansible.com/ansible/latest/modules/docker_stack_module.ht
ml
  docker_stack:
    state: present
```

```
    name: "{{ STACK_NAME | default('example_voting_app') }}"
    compose:
      - "/home/{{ ansible_user }}/docker-compose.yml"
```

The playbook `deploy_stack_playbook.yml`

```
# deploy_stack_playbook.yml
---
- hosts: tag_SwarmType_manager[0]
  tasks:
    - include_tasks: tasks/deploy_example_voting_app_stack.yml
```

Deploy ! This is the very last step. :rocket:

```
ansible-playbook deploy_stack_playbook.yml
```

Once it's done, you may need to wait 10~20 seconds while waiting for all services to get deployed.

Now you can access your `apps` from any `nodes` :tada: !

(Optional) If you wish the list of all your servers' **public IP address** you can run this.

```
inventory/ec2.py --list | grep tag_Swarm_True -A 3
```

Pick any one of them, doesn't matter if it is a `manager` or a `worker` the mode routing mesh does it all for you. We have those apps running:

- **vote** - port: 5000
- **result** - port: 5001
- **visualizer** - port: 8080

On any servers you can access **vote**, **result** and **visualizer** !

Clean the tutorial

Now that we have accomplished what we wanted, let's clean up our `servers` and `security_groups`. Here are the `tasks` and `playbook` to delete our `ec2` and `security_group`.

Remove the `ec2`.

```
---
# tasks/remove_ec2.yml
- name: "set fact: Swarm ec2 instances"
  #
https://docs.ansible.com/ansible/latest/modules/ec2_instance_facts_mod
ule.html
  ec2_instance_facts:
    filters:
      "tag:Swarm": "True"
  register: swarm_ec2

- name: "Removing ec2 instances"
  # https://docs.ansible.com/ansible/latest/modules/ec2_module.html
  ec2:
    instance_ids: "{{ item.instance_id }}"
    wait: yes
    state: absent
  loop: "{{ swarm_ec2.instances }}"
  when: swarm_ec2.instances != []
```

Remove the `security_group`.

```
---
# tasks/remove_security_groups.yml
- name: "set fact: security_groups"
  #
https://docs.ansible.com/ansible/latest/modules/ec2_group_facts_module
.html
  ec2_group_facts:
    filters:
      group-name:
        - swarm
  register: security_groups
```

```
- name: "Removing security groups"
  #
https://docs.ansible.com/ansible/latest/modules/ec2_group_module.html
  ec2_group:
    group_id: "{{ item.group_id }}"
    state: absent
  loop: "{{ security_groups.security_groups }}"
```

The playbook.

```
# remove_ec2_playbook.yml
---
- hosts: localhost
  connection: local
  gather_facts: False
  tasks:
    - include_tasks: tasks/remove_ec2.yml
    - include_tasks: tasks/remove_security_groups.yml
```

Clean it. You wouldn't want to forget you have 3 servers running somewhere on AWS.

```
ansible-playbook remove_ec2_playbook.yaml
```