

PROJECT FSP

Automatización de Infraestructuras y Despliegue de aplicaciones web con Ansible

Por Franco Sergio Pereyra

VISIÓN GENERAL

Mi proyecto muestra un ejemplo práctico de cómo se puede utilizar Ansible para automatizar la creación de infraestructura y despliegue de aplicaciones web en la nube pública de AWS.

El código fuente de la aplicación web desplegada en este proyecto ha sido desarrollado por mí. Esta aplicación te permite escribir o recibir mensajes con otros clientes mediante un sistema de login basado en cookies, donde la información de cada cookie se almacena en una base de datos MySQL. La infraestructura donde se ejecuta la aplicación web está formada por un balanceador de carga, dos servidores web y un servidor de bases de datos. Por último, tendríamos una instancia que estaría monitoreando las demás instancias. Todo creado y automatizado por Ansible.

OBJETIVOS

1. Hacer que Ansible se conecte a la API de AWS y que cree una infraestructura “simple” formada un balanceador de carga con Nginx, dos instancias con el servidor web Apache, un servidor con MySQL y una instancia para monitorear las demás instancias.
2. Instalación y configuración del software para cada una de las instancias, haciendo uso de roles de Ansible. Algunas de estas tareas consisten en instalar el sistema gestor de bases de datos MySQL, crear una base de datos, crear un usuario que pueda conectarse a esa base de datos, etc.
3. Todo el proceso será automático para garantizar que el proceso de despliegue sea repetible y fácilmente escalable.

Recomendación: [Para ver mejor este archivo, por favor ampliarlo un 150%](#)

ÍNDICE

1. [Instalación de ANSIBLE y conexión hacia la nube de AWS](#)
2. [Automatización de infraestructura AWS EC2](#)
3. [Automatización de software para cada Nodo](#)
4. [Bibliografía](#)

1. Instalación de Ansible y Conexión hacia la nube de AWS

1.1. Creación de la máquina Ansible

Primero entramos a la consola de amazon y creamos la instancia donde instalaremos ansible, esta instancia tiene que tener lo siguiente:

- A. AMI: `ami-0472eef47f816e45d` (imagen ubuntu 20.04)
- B. Puerto: `22` abierto
- C. Instance type: `t2.small`
- D. Keypair: `IAW2`

Una vez creado, abrimos Visual Studio Code e instalamos la extensión “Remote SSH” que nos permitirá conectarnos por ssh a nuestra máquina.

Luego tenemos que crear un fichero config donde le indicaremos el usuario, la ip pública de Ansible y la ruta de la clave que vamos a utilizar para poder así conectarnos a Ansible

```
Host Ash-Twin

HostName 52.200.2.147

User ubuntu

IdentityFile C:\Users\fsp28\Documents\IAW2.pem
```

1.2. Instalación de ANSIBLE y CLI AWS

Conectados, podemos empezar instalarla ansible y CLI aws .

Para poder instalar ansible, primero usamos el comando `apt-add-repository -y ppa:ansible/ansible` para añadir los paquetes de ansible al sistema y lo instalamos con `apt-get install -y ansible`

Una vez instalamos ansible, instalamos las dependencias necesarias para poder usar ansible con el módulo AWS o `ec2_instance`

```
# Instalamos python
apt install python3-pip -y

# Instalamos Boto Framework - AWS SDK para que ansible pueda llegar a AWS
usando boto SDK

pip install boto boto3

apt-get install python3-boto -y
```

Con ansible preparado, empezamos la instalación de CLI AWS, simplemente instalamos el zip que nos da amazon

```
# Intalamos AWSCLI

apt install zip -y

cd /tmp/

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"

unzip awscliv2.zip

./aws/install
```

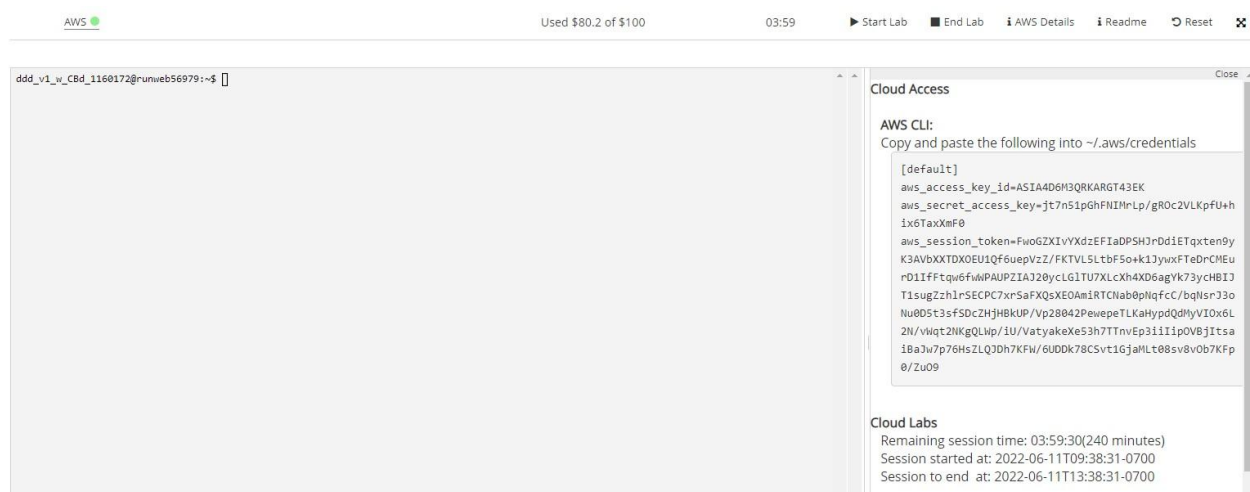
Quitamos el tiempo que trae al crear la máquina y ponemos el tiempo del país en el que estamos. (Si no no podríamos conectarnos al servidor del AWS)

```
rm /etc/localtime

ln -s /usr/share/zoneinfo/Europe/Madrid /etc/localtime
```

Al instalarlo se nos creará una carpeta llamada `.aws/` donde tendremos que crear ahí dos archivos: El archivo `credentials` y `config`.

El archivo `credentials` tendremos que añadir el AWS details del servidor de Amazon, así estaríamos conectados directamente desde la consola en nuestra máquina Ansible



Por último, generamos una clave pública y privada que permitirá a ansible conectarse a las instancias de amazon, esto lo hacemos con el comando `ssh-keygen`

Copiamos la clave pública en la carpeta `.ssh` y se las enviamos a las máquinas que queramos modificar

```
scp -i aws-ansible.pem ubuntu@54.173.51.24:/home/ubuntu/.ssh/id_rsa.pub .
cat id_rsa.pub | ssh -i "IAW2.pem" ubuntu @ ec2 -52 -200 -2 -147. compute -1.
amazonaws.com "cat - >> /home/ubuntu/.ssh/authorized_keys2"
```

2. Automatización de infraestructura AWS EC2

2.1. Creación de una playbook de Ansible para crear instancias de AWS EC2

Creamos la carpeta CreateEC2 y le añadimos las siguientes subcarpetas:

- **group_vars**: Aquí guardaremos las variables globales.
- **inventory**: Aquí ponemos y asignamos a nodos las ip de nuestras instancias.
- **roles**: Aquí crearemos las instrucciones que queremos que haga Ansible.

Y un fichero con extensión yml, por ejemplo **“create.yml”**. Este fichero indica a qué máquinas y roles tiene que seguir Ansible.

Entramos dentro de la carpeta roles y añadimos una carpeta con el nombre que más se asemeje la instrucción que queramos añadir, por ejemplo: **“Create-security-group”**, **“Create-ip-elastic”** y **“Create-instance”**. Dentro de esas carpetas, creamos las carpetas **task** y **template** y en cada Task ponemos el fichero **main.yml**. **Ahora indico lo que hace cada carpeta/archivo**

EJEMPLO DEL CONTENIDO DE CADA FICHERO/CARPETA

Fichero de la Carpeta **Group_vars** los ficheros de estas carpetas se llaman **all.yml**, nota: Todos los ficheros yml tiene que tener - - - al principio de su contenido (Esto lo indico una vez para que vean como sería el contenido de cada carpeta)

```
---  
  
keypair: IAW2  
  
instance_type: t2.small  
  
image: ami-0472eef47f816e45d  
  
count: 1  
  
region: us-east-1  
  
region2: us-east-1a  
  
SECURITY_GROUP: Brothers  
  
SECURITY_GROUP3: Balancer-sg
```

```
SECURITY_GROUP4: nfs-sg
SECURITY_GROUP5: nagios
SECURITY_GROUP6: ansible
SECURITY_GROUP7: mysql
INSTANCE_NAME_WWW1: Ash-Twin
INSTANCE_NAME_WWW2: Ember_Twin
INSTANCE_NAME_BALANCER: Balancer
INSTANCE_NAME_SERVER: nfs_server
INSTANCE_NAME_NAGIOS: Nagios_server
INSTANCE_NAME_ANSIBLE: Ansible
INSTANCE_NAME_MYSQL: Mysql
```

Fichero de la Carpeta **Inventory**, los ficheros se llaman **host**(Esto lo indico una vez para que vean como sería el contenido de cada carpeta)

```
[localhost]
44.201.108.234 ansible_user=ubuntu ansible_connection=ssh

[Brothers]
3.233.31.203 ansible_user=ubuntu ansible_connection=ssh
52.20.167.206 ansible_user=ubuntu ansible_connection=ssh

[NFS]
3.86.201.241 ansible_user=ubuntu ansible_connection=ssh

[Balancer]
184.73.75.144 ansible_user=ubuntu ansible_connection=ssh

[Mysql-Server]
34.235.80.234 ansible_user=ubuntu ansible_connection=ssh

[Nagios]
3.213.191.141 ansible_user=ubuntu ansible_connection=ssh
```

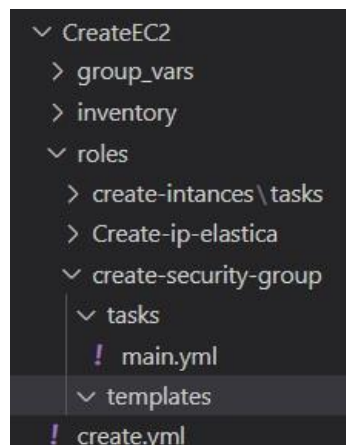
Fichero **create.yml** ,en este fichero indico el nodo que quiero y en qué orden se apliquen las tasks del role y que lo aplique como superusuario con el comando become: true (Esto lo indico una vez para que vean como sería el contenido de cada carpeta)

```
---
- hosts: localhost
  become: true
  roles:
    - create-security-group
    - create-instances
    - Create-ip-elastica
```

Carpeta **roles** y su contenido.

Dentro del role tendremos carpetas donde asignamos instrucciones específicas y esas instrucciones las guardamos en las carpetas task. Crearemos carpetas templates para guardar archivos de configuración que queramos modificar

(Esto lo indico una vez para que vean como sería el contenido de cada carpeta)



Fichero **main.yml** ,en este fichero indico lo que quiero hacer, instalar SSH,que muestre los procesos de cada nodo, copiar un fichero de configuración en la instancia seleccionada ... etc. (Esto lo indico una vez para que vean como sería el contenido de cada carpeta)Ej: Crear un security group

```
---
- name: Security Group para los brothers
  local_action:
    module: ec2_group
    name: "{{ SECURIRY_GROUP }}"
    description: Security Group para los brothers
    region: "{{ region }}"
    rules:
      - proto: tcp
        from_port: 22
        to_port: 22
        cidr_ip: 0.0.0.0/0
      - proto: tcp
        from_port: 80
        to_port: 80
        cidr_ip: 0.0.0.0/0
      - proto: tcp
        from_port: 2049
        to_port: 2049
```



```

        cidr_ip: 0.0.0.0/0
    - proto: tcp
      from_port: 5666
      to_port: 5666
      cidr_ip: 0.0.0.0/0
    - proto: tcp
      from_port: 19999
      to_port: 19999
      cidr_ip: 0.0.0.0/0
  rules_egress:
    - proto: all
      cidr_ip: 0.0.0.0/0
  register: basic_firewall

```

Carpeta **Template**, acá guardaremos los ficheros de configuración personalizadas donde podemos aplicarle variables. para implementarlo hay que indicarlo en una task y tiene que tener como extensión **.j2** EJ:

```

1 upstream fronted {
2     server {{ IP_SERVER }};
3     server {{ IP_SERVER2 }};
4 }
5
6 server {
7     server_name fspcorp.ddns.net;
8     location / {
9         proxy_pass http://fronted;
10    }
11 }
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26

```

```

9   pkg: "{{ item }}"
10  update-cache: yes
11  state: latest
12  with_items:
13    - nginx
14
15  - name: Copiamos el fichero de Configuración de Nginx
16    template:
17      src: templates/default_balancer.j2
18      dest: /etc/nginx/sites-available/default
19      owner: root
20      group: root
21      mode: '0644'
22
23  - name: Iniciamos el servicio Docker
24    shell: 'systemctl restart nginx'
25    args:
26      chdir: /home/ubuntu

```

FIN DE EJEMPLO

Esto sería los task que haría al ejecutar la playbook Create E2C, primero creó el grupo de seguridad, luego la instancia con el módulo ec2 instances y le asigno una ip elástica para que no cambie de ip.

```

- name: Security Group para el NFS
  local_action:
    module: ec2_group
    name: "{{ SECURITY_GROUP4 }}"
    description: Security Group para el NFS
    region: "{{ region }}"
    rules:
      - proto: tcp
        from_port: 22
        to_port: 22
        cidr_ip: 0.0.0.0/0
      - proto: tcp
        from_port: 2049
        to_port: 2049
        cidr_ip: 0.0.0.0/0
    rules_egress:
      - proto: all
        cidr_ip: 0.0.0.0/0
  register: basic_firewall2

```

```

---
- name: Lanzamos las instancias de Brother Ember
  ec2_instance:
    key_name: "{{ keypair }}"
    security_group: "{{ SECURITY_GROUP }}"
    instance_type: "{{ instance_type }}"
    image_id: "{{ image }}"
    wait: yes
    region: "{{ region }}"
    count: "{{ count }}"
    tags:
      Name: "{{ INSTANCE_NAME_WWW1 }}"
  register: ec2

```

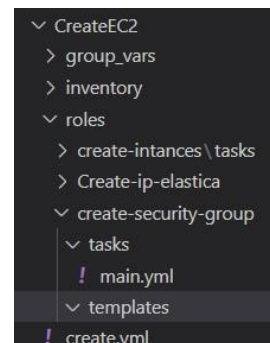
```

---
- name: "Creamos las Elastic IP"
  shell: ../06_Create_ip_elastica.sh
  args:
    chdir: ../Create-ip-elastica

```

Para ejecutar una playbook nos dirigimos a CreateEC2 y en una terminal ponemos: `ansible-playbook`, el nombre del fichero `.yaml` y la indicación a la carpeta `inventory` Ej:

`ansible-playbook create.yml /inventory -vvv`



Y al ejecutarlo me va a crear todos los grupos de seguridad necesarios y personalizados para cada instancia, luego creará las instancias **Brother-Ember**, **Brother-Ash**, **Mysql-Server**, **Balancer** y **Nagios** donde les asignará una ip estática a cada una de ellas.

Con esto podemos empezar la automatización de software para cada una de las anteriores máquinas.

Aquí dejo el repositorio de GitHub con todo mi código pasado a Ansible para que lo puedan ver mejor:

https://github.com/FSP-1/Practica_FSP/tree/main/infraestructura/CreateEC2

3. Automatización de software para cada Nodo

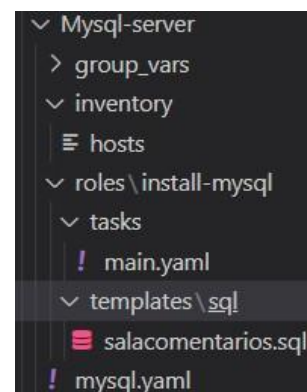
3.1. MySQL Server (Sistema Gestor de Bases de datos)

Ahora que sabemos la estructura y las funciones de cada parte de la estructura necesaria para que una playbook de ansible funcione, podemos empezar a explicar la automatización del software más rápido.

Empezamos con `mysql-server`, creamos la estructura de Ansible (`group_vars`, `inventory`, `roles`, `task...` etc). En `inventory` solo ponemos el nodo y la ip del `mysql` →

```
[Mysql-Server]
34.235.80.234 ansible_user=ubuntu ansible_connection=ssh
```

En roles creamos la carpeta **“install-mysql”** y dentro de ella una task y un template para guardar nuestra base de datos personalizada. Base de Datos que hice → [BD](#)



Una vez dentro del task, empezamos a automatizar el software necesario para tener un mysql-server. Primero instalamos mysql y unas dependencias que nos ayudará a ejecutar los módulos de ansible :

```
- name: Instalamos mysql y algunas dependencias para que podamos aplicar el módulo
community.mysql
apt:
  pkg: "{{ item }}"
  update-cache: yes
  state: latest
with_items:
  - mysql-server
  - libmysqlclient-dev
  - python3-mysqldb
  - python3-pip
```

Luego copiaremos la BD que pusimos en templates en mysql

```
- name: Copiamos la base de datos a la máquina Mysql
template:
  src: templates/sql/salacomentarios.sql
  dest: /tmp/salacomentarios.sql
  owner: root
  group: root
- name: Añadimos a la base de datos
shell: 'mysql -u root < /tmp/salacomentarios.sql'
args:
  chdir: /home/ubuntu
```

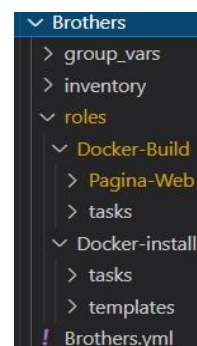
Por último, usamos el módulo mysql.mysql_user para crearnos un usuario en mysql, donde la usaremos más tarde para conectarnos a la BD. Modificamos el fichero de configuración de Mysql para que acepte conexión de cualquier red y reiniciamos el servicio para que los cambios se apliquen.

```
- name: Creamos el usuario Franco y le damos todos los permisos de la base de datos a
el
community.mysql.mysql_user:
  name: Franco
  password: 11Ww222Ee
  priv: 'Franco_db.*:ALL'
  state: present
- name: Configuramos MySQL para aceptar conexiones desde cualquier interfaz de red
shell: 'sed -i "s/127.0.0.1/0.0.0.0/" /etc/mysql/mysql.conf.d/mysqld.cnf'
args:
  chdir: /home/ubuntu
```

Ejecutamos el playbook con el archivo mysql.yml (Recuerdo que simplemente este archivo indica a ansible a que nodo y que task debe de ejecutar en la instancia)

3.2. TWIN-BROTHERS (Página web hecha con docker-compose)

Creamos la carpeta **Brother** con la estructura de ansible y creamos dos roles. Uno para instalar **docker-compose** y otro para ejecutar el archivo docker-compose.yml (Este crearía un contenedor **apache** y **netdata**)



Estos roles se ejecutará en las instancias Ember y Ash ya que en el inventario están en el mismo nodo

Inventory:

```
[Brothers]
3.233.31.203 ansible_user=ubuntu ansible_connection=ssh
52.20.167.206 ansible_user=ubuntu ansible_connection=ssh
```

Brothers.yml

```
---
- hosts: Brothers
  become: true
  roles:
    - Docker-install
    - Docker-Build
```

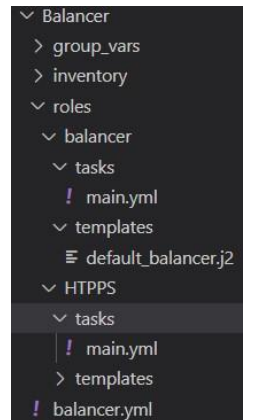
Como dicho antes, el primer rol instala docker compose y el segundo ejecuta el docker-compose.yml (Es un archivo que crea los contenedores docker con volúmenes, así los datos de cada contenedor se guardarán en vez de eliminarse al parar dicho contenedor).

Al ejecutarlo creará en las instancias dos contenedores, una que será apache server con la página web (Carpeta **html** y carpeta **Página-Web**) y otro llamado netdata (Un programa que monitorea) que monetizara siempre el estado de las instancias y contenedores docker. Dejo un enlace directo para que puedan ver el archivo y el contenido de la página web

[Docker-compose de mi Página Web](#)

3.3. BALANCER (Nginx con un DNS y con certificado de Certbot)

Hacemos la carpeta **Balancer** y una estructura Ansible en ella crearemos dos roles(**balancer** y **HTTPS**), una para instalar y configurar el servidor nginx que estara balanceando entre las ip de los Brothers con un DNS y otro rol para ejecutar la creación del certificado certbot para tener una página con https



El inventario con ip y nombre del nodo para la instancia balancer y un group_vars con las ip de los Brothers y mi cuenta de gmail con mi ddns que cree para la creación del certificado Certbot

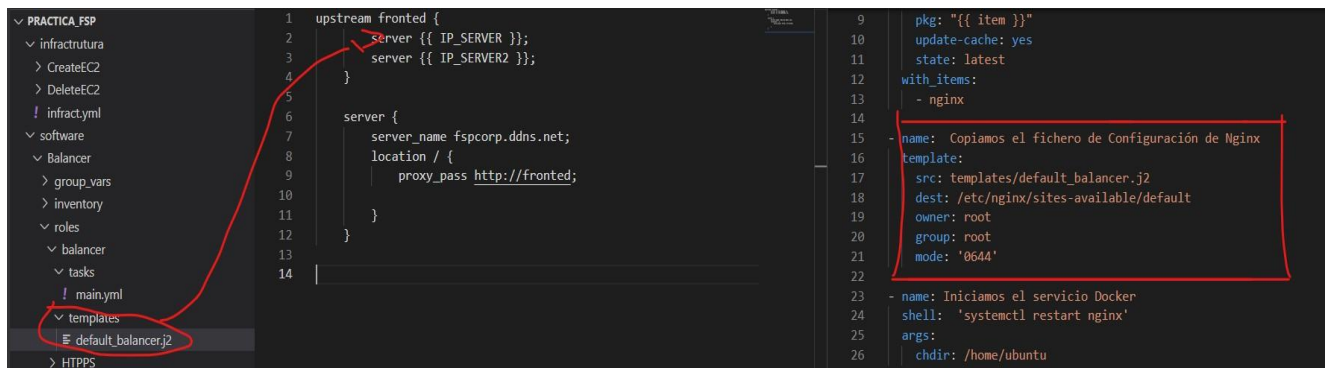
Group_vars

```

IP_SERVER: 52.20.167.206
IP_SERVER2: 3.233.31.203

EMAIL_HTTPS: f.s.p282002@gamil.com
DOMAIN: fspcorp.ddns.net
  
```

En el rol balancer, instalo el servidor web nginx y le añado mi archivo de configuración



En el rol HTTPS crear y asigna un certificado a mi dns

```
- name: Instalamos certbot con snap
  shell: 'snap install --classic certbot'
  args:
    chdir: /tmp/

- name: Solicitamos el certificado HTTPS
  shell: certbot --nginx -m {{ EMAIL_HTTPS }} --agree-tos --no-eff-email -d {{ DOMAIN }}
  args:
    chdir: /home/ubuntu

- name: Reload nginx to activate specified site
  service: name=nginx state=restarted
```

3.4. Nagios y Netdata (Nagios 4 con apache y certificado y los netdata de los Brothers en la instancia Nagios)

En este crearemos la carpeta **Nagios** que tendrá cinco roles

A). Uno sería **netdata** donde se conectaría al puerto 81 (Esto en un servidor nginx) al los puertos 19999 de los brothers y con un proxy inverso como configuración del nginx.

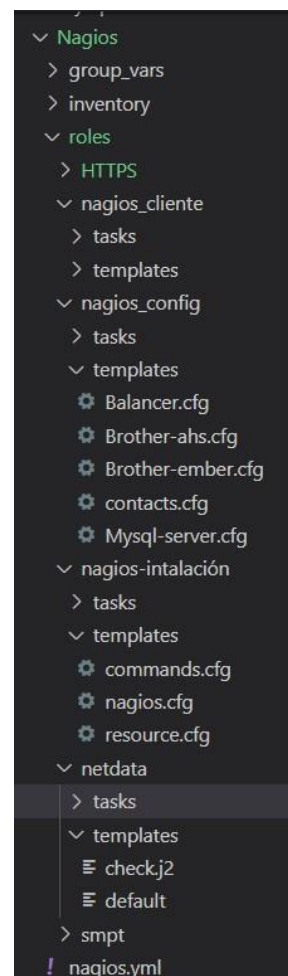
Entramos por <http://checkfsp.ddns.net/netdata/server1> para ver el netdata del **Brother-Ember**

Entramos por <http://checkfsp.ddns.net/netdata/server2> para ver el netdata del **Brother-Ash**

Dejo un enlace directo al fichero de configuración → [Netdata de los Brothers](#)

B). Los otros 3 Roles se encargan de instalar/Configurar Nagios, tanto en el servidor como en los clientes.

Primero iría el rol **nagios-instalación** donde descargarla e ejecutaría el demon de nagios 4.4.6. El paquete instalaría nagios, grupo y user nagios conjunto la página web que tiene. Antes instalará todas las dependencias que necesita para funcionar y finalmente cambiaría algunos ficheros de configuración para que las demás instancias puedan contactar con nagios-server. Como el fichero **commands.cfg** este son los comandos/plugins que usa Nagios para monitorear sus instancias.



Luego seguiría el rol **nagios-cliente**, en este instalamos nrpe y los plugins de nagios en todas las instancias clientes (Brothers, Balancer y Mysql). Cambiaríamos el archivo de configuración de nrpe para que apunte a la ip de la instancia Nagios-server.

Y en acabado, el rol **nagios-config**, acá creamos la carpeta server donde guardaremos los hosts de cada instancia. Cada fichero host lo podremos configurar para monitorear lo que queramos de ese host indicando nombre e ip del host y los servicios a monitorear

Ej → [Mysql-server host](#)

C). El último rol **HTTPS** es igual que el del Balancer, crear un certificado con certbot a mi dominio **checkfsp.ddns.ne**

Inventory:

```
[Nagios]
3.213.191.141  ansible_user=ubuntu ansible_connection=ssh

[Nrpe]
3.233.31.203  ansible_user=ubuntu ansible_connection=ssh
52.20.167.206  ansible_user=ubuntu ansible_connection=ssh
184.73.75.144  ansible_user=ubuntu ansible_connection=ssh
34.235.80.234  ansible_user=ubuntu ansible_connection=ssh
```

Nagios.yml

```
---
- hosts: Nagios
  become: true
  roles:
    - netdata
    - HTTPS

- hosts: Nagios
  become: true
  roles:
    - nagios-intalación

- hosts: Nrpe
  become: true
  roles:
    - nagios_cliente

- hosts: Nagios
  become: true
  roles:
    - nagios_config
```


Group_vars

```
IP_SERVER: 52.20.167.206
IP_SERVER2: 3.233.31.203

EMAIL_HTTPS: f.s.p282002@gamil.com
DOMAIN: checkfsp.ddns.net
```

Nagios https:(Ampliar 200% para ver el enlace del nagios https)

Nagios
 Last Updated: Mon Jun 13 15:55:49 UTC 2022
 Updated every 90 seconds
 Nagios® Core™ 4.4.6 - www.nagios.org
 Logged in as nagiosadmin

Current Network Status

Host Status Totals

Up	Down	Unreachable	Pending
5	0	0	0

Service Status Totals

Ok	Warning	Unknown	Critical	Pending
25	2	0	2	0

Service Status Details For All Hosts

Host	Service	Status	Last Check	Duration	Attempt	Status Information
Balancer	Current Load	CRITICAL	06-13-2022 15:55:24	1d 23h 15m 32s	2/4	CRITICAL - load average: 0.62, 0.30, 0.11
	Current Users	OK	06-11-2022 16:16:10	8d 4h 6m 33s	1/4	USERS OK - 0 users currently logged in
	Root / Partition	OK	06-11-2022 16:17:10	8d 3h 35m 47s	1/4	DISK OK - free space: /dev 980 MB (100% inode=99%);
	Total Processes	OK	06-11-2022 16:18:11	8d 3h 35m 1s	1/4	PROCS OK: 111 processes
	nginx port	OK	06-11-2022 16:15:48	3d 6h 29m 18s	1/3	TCP OK - 0.001 second response time on 184.73.75.144 port 443
Brother-ahs	Current Load	WARNING	06-11-2022 16:17:37	1d 23h 43m 12s	4/4	WARNING - load average: 0.02, 0.03, 0.11
	Current Users	OK	06-13-2022 15:55:36	1d 23h 57m 47s	1/4	USERS OK - 0 users currently logged in
	Root / Partition	OK	06-11-2022 16:16:48	1d 23h 54m 1s	1/4	DISK OK - free space: /dev 986 MB (100% inode=99%);
	Total Processes	OK	06-11-2022 16:17:14	1d 23h 53m 35s	1/4	PROCS OK: 118 processes
	apache2 port	OK	06-11-2022 16:17:44	1d 23h 58m 5s	1/3	TCP OK - 0.001 second response time on 3.233.31.203 port 80
Brother-ember	Current Load	WARNING	06-11-2022 16:17:49	1d 23h 43m 0s	4/4	WARNING - load average: 0.00, 0.04, 0.12
	Current Users	OK	06-11-2022 16:17:49	1d 23h 58m 0s	1/4	USERS OK - 1 users currently logged in
	Root / Partition	OK	06-11-2022 16:40:41	1d 23h 57m 24s	1/4	DISK OK - free space: /dev 986 MB (100% inode=99%);
	Total Processes	OK	06-11-2022 16:17:58	1d 23h 57m 51s	1/4	PROCS OK: 122 processes
	apache2 port	OK	06-11-2022 16:12:35	1d 23h 53m 14s	1/3	TCP OK - 0.001 second response time on 52.20.167.206 port 80
Mysql-server	Current Load	OK	06-11-2022 16:17:10	1d 23h 38m 39s	1/4	OK - load average: 0.05, 0.05, 0.00
	Current Users	OK	06-11-2022 16:15:58	1d 23h 39m 51s	1/4	USERS OK - 0 users currently logged in
	Root / Partition	OK	06-11-2022 16:15:46	1d 23h 40m 3s	1/4	DISK OK - free space: /dev 980 MB (100% inode=99%);
	Total Processes	OK	06-11-2022 16:16:46	1d 23h 39m 3s	1/4	PROCS OK: 106 processes
	netdata port	OK	06-11-2022 16:18:05	1d 23h 57m 44s	1/3	TCP OK - 0.001 second response time on 52.20.167.206 port 19999

Nagios-Netdata: (Ampliar 200% para ver el enlace del proxy)

System Overview
 Overview of the key system metrics.

cpu

Disk Read: 0.0 MB/s

Disk Write: 0.0 MB/s

CPU: 4.0 %

Net Inbound: 54.9 kilobits/s

Net Outbound: 0.75 megabits/s

Used RAM: 49.6 %

System Overview

- System Overview
- CPUs
- Memory
- Disks
- Networking Stack
- IPv4 Networking
- IPv6 Networking
- Network Interfaces
- Firewall (netfilter)
- systemd Services
- Applications
- User Groups

4. Bibliografía

- 1) <https://ualmtorres.github.io/CursoAnsible/tutorial/>
- 2) <https://www.youtube.com/watch?v=HOqA1zVEWSk>
- 3) https://docs.ansible.com/ansible/latest/collections/amazon/aws/ec2_instance_module.html#ansible-collections-amazon-aws-ec2-instance-module
- 4) <https://manuelfrancoblog.wordpress.com/2017/11/29/instalacion-de-nagios4-en-clients-y-servidores/>
- 5) <https://learn.netdata.cloud/docs/get-started>
- 6) <https://www.coachdevops.com/2021/07/ansible-playbook-for-provisioning-new.html>
- 7) <https://www.php.net/manual/en/function.setcookie.php>
- 8) https://support.nagios.com/kb/article/nrpe-check_nrpe-socket-timeout-after-n-seconds-617.html
- 9) <https://www.noip.com/es-MX>
- 10) https://runebook.dev/es/docs/ansible/collections/community/mysql/mysql_user_module
- 11) <https://josejuansanchez.org/iaw/practica-aws-cli/index.html#configuraci%C3%B3n-de-aws-cli>
- 12) <https://aws.amazon.com/es/premiumsupport/knowledge-center/ec2-linux-fix-permission-denied-errors/>

Mi GitHub → https://github.com/FSP-1/Practica_FSP

Gracias por leer mi proyecto

