ANSIBLE CONTAINER

Álvaro Sola Olivero

I.E.S Gonzalo Nazareno

Administración Sistema Informáticos en Red

ÍNDICE

1. Introducción

Ansible Container es un proyecto de código abierto que tiene como objetivo permitir la automatización de todo el proceso de creación, despliegue y administración de contenedores a través de Ansible playbooks.

A continuación se describe los conocimientos básicos de las tecnología que depende Ansible Container.

1.1. ¿Qué es Ansible?

Ansible es una plataforma de automatización TI que facilita el despliegue de aplicaciones, la orquestación multi-nivel y la gestión de configuración de infraestructuras, acelerando la producción, alineando las conversaciones con los procesos de construcción de software, de aseguramiento y de suministro TI.

Se caracteriza por su gran simplicidad y facilidad de uso, además de ser seguro y bastante fiable, utilizando un lenguaje fácil de comprender para todos los públicos.

1.2. ¿Qué es DOCKER?

Docker es una plataforma encargada de crear contenedores ligeros y portables para las aplicaciones de software, facilitando los despliegues sin tener en cuenta el sistema operativo empleado, pudiendo llevar de un lado a otro un contenido.

Nos permite añadir en un contenedor los servicios necesarios que una aplicación necesita para ser ejecutada, además de la propia aplicación.

Docker permite una independencia entre las aplicaciones, la infraestructura, los desarrolladores y las operaciones de TI para crear un modelo para una mejor colaboración e innovación.

1.3. ¿Qué es Red Hat?

Red Hat es el proveedor mundial conocido de soluciones Open Source empresarial, el cual ha lanzado Ansible Container en Junio de 2016 bajo el proyecto Ansible, que proporciona un marco de automatización de TI Open Source sencillo, potente y sin agentes.

Es conocida en gran medida por su sistema operativo empresarial Red Hat Enterprise Linux y por la adquisición del proveedor de middleware empresarial de código abierto JBoss. Proporciona almacenamiento, plataformas de sistemas operativos, middleware, aplicaciones, productos de administración y servicios de soporte, capacitación y consultoría.

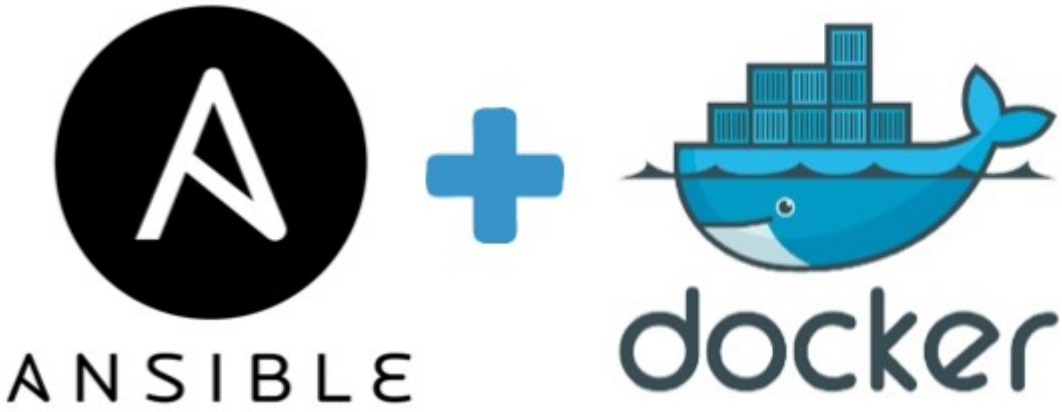
1.4. ¿Qué es Ansible Container?

Ansible Container es una tecnología creada para proporcionar un flujo de trabajo centrado en Ansible para construir, ejecutar, probar y desplegar contenedores.

Ansible Container permite la creación completa de contenedores Linux con formato Docker y poder orquestar a partir de Ansible Playbooks. Su principal función es crear una aplicación a partir de un fichero YAML, en lugar de usar un fichero Dockerfile, pudiendo enumerar las funciones que queremos que tenga el contenedor, además de mejorar la forma de crear y configurar contenedores a partir de plantillas, datos encriptados, etc.

Ansible permite a los desarrolladores y operadores implementar entornos y aplicaciones de forma más rápida y sencilla, pudiendo automatizar las actividades rutinarias como la configuración de red, implementaciones en la nube y la creación de entornos de desarrollo.
Con las capacidades de Ansible Container, los usuarios serán capaces de automatizar su infraestructura y sus redes con Ansible Playbooks, usando Playbooks adicionales para incluir sus aplicaciones y desplegarlas en una plataforma de contenedor de aplicaciones.

Junto a Ansible Container, el proyecto Ansible, ha lanzado nuevos módulos Kubernetes que permiten la producción de plantillas Kubernetes directamente desde el Ansible Playbook. Esta función permite construir contenedores Linux e implementar a una plataforma de aplicación de contenedores basada en Kubernetes como Red Hat Open Shift, de manera mucho más ágil y eficiente.



La creación y despliegue de forma automatizada de contenedores ofrecidos por Ansible se suman a la infraestructura de contenedores de Red Hat, incluyendo:

- Un sistema operativo estable y basado en contenedores, Red Hat Enterprise Linux Atomic Host.

- Una plataforma empresarial de contenedor de aplicaciones Kubernetes y Docker, a través de Red Hat OpenShift y el nuevo servicio de nube pública OpenShift Online.

- Gestión de infraestructuras, automatización y control de entornos híbridos con Red Hat CloudForms, Red Hat Insights, Red Hat Satellite y Ansible Tower by Red Hat.

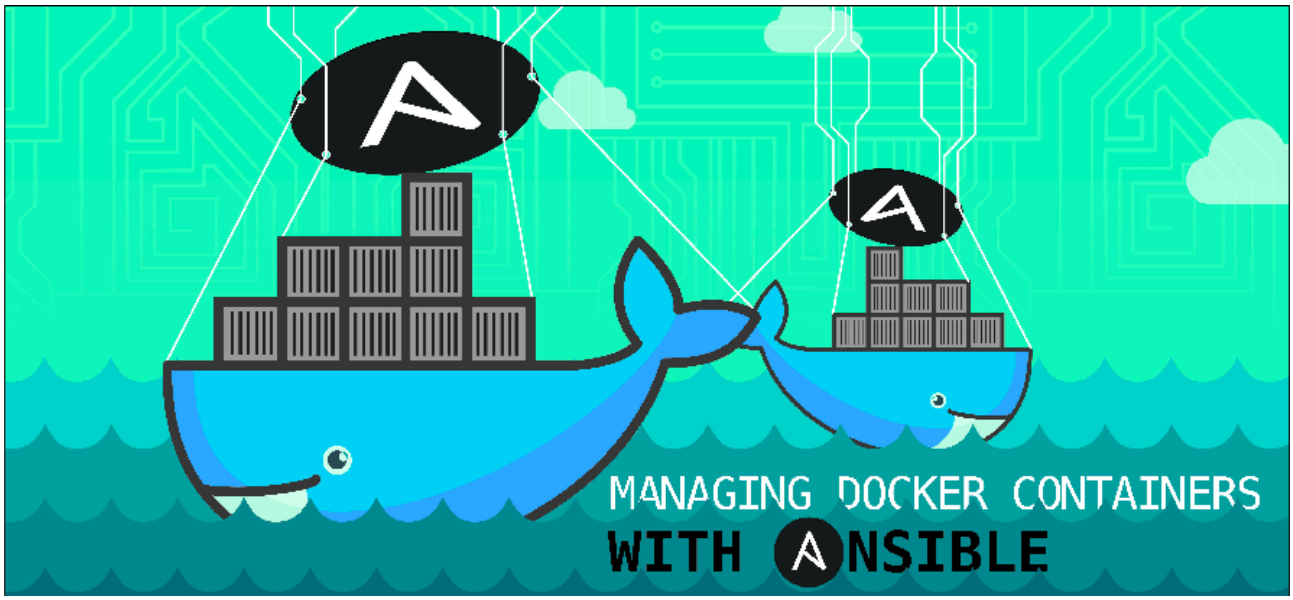
- Arquitectura de nube híbrida y privada masivamente escalable para el despliegue de contenedores a gran escala a través de Red Hat OpenStack Platform y Red Hat Cloud Suite, que también incluye Red Hat OpenShift.

Como dice el director de Ansible Community, Greg DeKoenigsberg, "muchos usuarios de nuestra comunidad ya se han acostumbrado a utilizar Ansible para desplegar y gestionar sus contenedores, y el proyecto Ansible Container es una encapsulación de muchas de las mejores ideas de la comunidad, reunidas en una sola herramienta. La cantidad de usuario que hemos recibido en un período relativamente corto de tiempo nos confirma que estamos apuntando a una necesidad crítica en el espacio de la mecanización de contenedores”.

2. Objetivos

El objetivo de este proyecto es crear y administrar contenedores a través de una receta Ansible sin tener que utilizar ficheros Dockerfile y desplegar una aplicación de prueba para mostrar el funcionamiento.

Debido al enfoque actual para crear y administrar contenedores es manual y anticuado, realizo este proyecto para automatizar todo el proceso de creación, despliegue y administración de una aplicación Flask que interactúe con un servidor web sobre un contenedor



(1) Crear y administrar contenedores.

(2) Desplegar una aplicación Flask interactuando con un servidor web Nginx.

3. Instalación

El proceso de instalación de Ansible Container se realizará en un sistema Linux, en una distribución Debian Stretch. Existen dos formas de realizar la instalación, a través de un entorno de Python o directamente instalarlo en el sistema, a demás hay que tener en cuenta los requisitos previos para poder instalarlo.

3.1. Instalación de requisitos

Para realizar la instalación de Ansible Container se necesitan instalar previamente las siguientes dependencias.

3.1.1. Python 2.7 o Python 3.5

Por defecto en el sistema operativo Debian Stretch tiene instalado el paquete Python con la versión 2.7.

```
root@debian:/home/usuario# python -V
Python 2.7.13
```

3.1.2. Pip

Instalación del módulo “python-pip” de Python.

```
root@debian:/home/usuario# apt install python-pip
```

Por defecto instala la versión 2.7. Esta versión soporta la instalación de Ansible Container.

```
root@debian:/home/usuario# pip -V
pip 9.0.1 from /usr/lib/python2.7/dist-packages (python 2.7)
```

3.1.3. Setuptools 20.0.0+

Actualizar Setuptools a la última versión.

```
root@debian:/home/usuario# pip install --upgrade setuptools
```

3.2. Instalación Ansible Container

Ansible Container depende del motor del contenedor para construir, ejecutar y desplegar su proyecto. Cuando se instala Ansible Container, se debe especificar qué motores se desea que admita su instalación. Los motores soportados actualmente son:

- Docker - El motor Docker.
- K8s - Kubernetes , en un servicio remoto o en una instalación local utilizando MiniKube.
- Openshift - Red Hat OpenShift , en un servicio remoto o en una instalación local usando MiniShift.

Para especificar que motores instalar, se indica el nombre separado por comas, entre corchetes como parte del comando de pip install . Por ejemplo, si tiene la intención de utilizar Docker para el desarrollo de contenedores pero implementar su proyecto en Kubernetes, se debe especificar los motores Docker y k8s.

La instalación de Ansible Container se puede realizar en un entorno virtual de Python o en el propio sistema operativo.

La ventaja de instalarlo en un entorno virtual de Python es que cualquier dependencias que instale, no afecta al sistema operativo, el inconveniente es que al realizar la instalación faltan módulos, que si están por defecto en el sistema.

3.2.1. Entorno Virtual de Python

Instalación de virtualenv (entorno virtual de Python).

```
root@debian:/home/usuario# apt-get install virtualenv
```

Creación y configuración de un entorno virtual de Python en Debian.

```
usuario@debian:~$ virtualenv ansible-container
```

Acabamos de crear un entorno virtual de Python en el que se han instalado todos los paquetes de Python necesarios para poder utilizar Python pip dentro de ese entorno y que dichos paquetes se instalen por un usuario sin privilegios sin afectar a los paquetes de Python del sistema.

Cada vez que vayamos a utilizar los paquetes de Python de este entorno virtual o queramos instalar algún paquete nuevo, debemos activarlo mediante la instrucción.

```
usuario@debian:~$ source ansible-container/bin/activate
```

Ahora estamos dentro del entorno virtual. Ya podemos realizar en nuestro entorno virtual las instalaciones que queramos. Cuando queramos salir del entorno virtual de python ejecutamos la siguiente instrucción.

```
(ansible-container) usuario@debian:~$ deactivate
```

Por último se procede a instalar el paquete, es necesario especificar que motores se desean instalar en Ansible Container, en este caso para desplegar mi aplicación necesito solamente el motor Docker.

```
(ansible-container) usuario@debian:~$ pip install ansible-container[docker]
```

3.2.2. Sistema

La instalación de Ansible Container en el sistema operativo es un proceso más sencillo debido a que se instala directamente en el sistema. Se especifica igualmente en este proceso de instalación que motores se desea instalar en Ansible Container.

```
root@debian:/home/usuario# pip install ansible-container[docker]
```

4. INICIACIÓN

Los proyectos realizado con Ansible Container se identifican por su estructura de sistema de archivos. La ruta del proyecto contiene toda la fuente y archivos necesarios para construir y orquestar contenedores.

4.1. Conductor Container

El despliegue de Ansible Container a través del proceso de construcción "build", ocurre dentro de un contenedor especial llamado "Conductor Container".

Contiene todo lo necesario para construir imágenes de contenedores. Durante la construcción de contenedores se instala todas las dependencias necesarias a partir del "Conductor Container".

Debido a esto se recomienda que la distribución del "Conductor Container" sea la misma distribución de los contenedores creados. Por ejemplo, si está creando imágenes de contenedores basadas en Centos 7, es buena idea usar la distribución Centos 7 en el "Conductor Container". Dicha imagen de distribución se especifica en el fichero de configuración "container.yml".

Ansible Container ofrece imágenes preparadas para las distribuciones de Linux más utilizadas, desde las que puede construir la imagen del "Conductor Container" de su proyecto. Esto hace que la configuración sea de una dificultad menor. La lista de imágenes bases son las siguientes.

- CentOS 7
- Fedora 24, 25 y 26
- Debian Jessie, Stretch y Wheezy
- Ubuntu Preciso, Confiable, Xenial y Zesty
- Alpine 3.4 y 3.5

4.2. Comenzando desde cero

Ansible Container proporciona una forma de iniciar un proyecto simplemente ejecutando "ansible-container init" desde el directorio de su proyecto, el cual crearía los siguientes ficheros.

- ansible.cfg
- ansible-requirements.txt
- container.yml
- meta.yml
- requirements.yml
- .dockerignore

Para iniciar el proceso de compilación se ejecuta "ansible-container build". Construye y lanza el "Conductor Container". Él ejecuta instancias a partir de las imágenes del contenedor base que se especifica en el fichero "container.yml".

Para iniciar el contenedor se ejecuta "ansible-container run". Inicia los contenedores de las imágenes compiladas descritas en el fichero "container.yml".

Para cargar las imágenes creadas en un registro de contenedores se ejecuta "ansible-container deploy".

Entonces, ¿qué se especifica en los ficheros que hacen que esto funcione?

4.2.1. Container.yml

Es un fichero de sintaxis YAML donde se describe los servicios, cómo construirlos y ejecutarlos, los repositorios, etc. Es similar a otros formatos de contenedores como Docker Compose o OpenCompose.

Ansible Container utiliza este archivo para determinar qué imágenes crear, qué contenedores ejecutar y conectar, y qué imágenes enviar a su repositorio. Se ejecuta automáticamente.

Un ejemplo del fichero “container.yml”.

```
version: "2"
services:
  web:
    from: "centos:7"
    roles:
      - common
      - apache
    ports:
      - "80:80"
   command:["/usr/bin/dumb-init","/usr/sbin/apache2ctl","-D", "FOREGROUND"]
    dev_overrides:
      environment:
        - "DEBUG=1"
```

Estos parámetros indican lo siguiente.

1) Establece la versión 2.

2) Cada uno de los contenedores que desea establecer debe estar bajo la palabra clave "services".

3) La imagen que se especifica, es la imagen base del contenedor.

4) Cada rol debe estar en el directorio "roles/" del proyecto creado.

5) Los puertos que abre el contenedor por donde escucha la aplicación.

6) Se necesita levantar todos los servicios necesarios para desplegar la aplicación, debido a que en los contenedores por defecto hay que activar los demonios.

7) Opcionalmente se puede especificar la opción "dev_overrides". Durante la compilación esta opción anula la configuración de su entorno en producción.

4.2.2. Meta.yml

Es un fichero donde se puede compartir el proyecto en "Ansible Galaxy" para que otros puedan utilizarlo como plantilla para crear sus propios proyectos.

4.2.3. Ansible-requirements.txt

Es un fichero donde se especifica las dependencias de la biblioteca Python que podría necesitar nuestra aplicación. Este archivo sigue el formato pip para las dependencias de Python. Al ejecutar Ansible cuando se crea el "Conductor Container", estas dependencias se instalan.

4.3.4. Requirements.yml

Es un fichero donde se especifica los roles del fichero “container.yml”, si están en "Ansible Galaxy" o en un repositorio remoto.

4.3.5. Ansible.cfg

Es un fichero donde se establece la configuración de Ansible dentro del contenedor durante la compilación.

4.3.6. .Dockerignore

Es un fichero donde se establece los archivos que se desea ignorar durante la construcción del contenedor.

5. CREACIÓN DE LA INFRAESTRUCTURA

El sistema operativo donde se crea la receta Ansible Container será Debian Stretch, el sistema deberá tener los siguientes requisitos para poder crear el proyecto.

- Docker Engine 2.7.0
- Ansible 2.3+
- Ansible Container

Se crea un directorio donde estarán los ficheros.

```
usuario@debian:~$ mkdir proyecto
```

Ansible Container proporciona una forma conveniente de iniciar su aplicación simplemente ejecutando "ansible-container init".

```
usuario@debian:~$ cd proyecto/
usuario@debian:~/proyecto$ ansible-container init
Ansible Container initialized.
usuario@debian:~/proyecto$ ls
ansible.cfg
ansible-requirements.txt
container.yml
meta.yml
requirements.yml
```

Ansible Container ofrece una ventaja sobre Dockerfiles y Docker Compose, que es la capacidad de crear contenedores fácilmente utilizando la sintaxis de Ansible.

Se crea dos roles en la raíz de nuestro proyecto.

1) Flask
2) Nginx

Dentro de cada directorio se crea subdirectorios asociados para los ficheros de configuración.

```
usuario@debian:~/proyecto$ mkdir -p roles/nginx/tasks roles/nginx/templates
roles/flask/tasks roles/flask/files
```

Quedando así la estructura de nuestro proyecto.

```
+ - - ansible.cfg
+ - - container.yml
+ - - ansible-requirements.txt
+ - - meta.yml
+ - - requirements.yml
+ - - roles
    + - - flask
    ¦   + - - tasks
    ¦   + - - files
    + - - nginx
        + - - tasks
        + - - templates
```

Se procede a continuación a configurar la aplicación Flask, para ello se crea un fichero de configuración dentro de “roles/flask/files”.

```
usuario@debian:~/proyecto$ nano roles/flask/files/app.py

from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello World! Realizado por Alvaro Sola'

if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)
```

A continuación se procede a definir los parámetros para instalar y configurar la aplicación. Para ello se crea un fichero dentro “roles/flask/tasks”.

```
usuario@debian:~/proyecto$ nano roles/flask/tasks/main.yml

---
- name: Actualizar repositorios
  apt:
    update_cache: yes
    cache_valid_time: 600

- name: Instalar python-pip
  apt:
    name: "{{ item }}"
    state: present
  with_items:
    - "python-pip"

- name: Instalar requisitos
  pip:
    name: "{{ item }}"
    state: present
  with_items:
    - "flask"
    - "gunicorn"

- name: Crear aplicacion
  file:
   path: "/app"
   state: directory

- name: Configurar aplicacion
  copy:
    src: app.py
    dest: /app/app.py
```

A continuación se procede a configurar el servidor web Nginx, para ello se crea un fichero de configuración con los parámetros deseados, dentro de “roles/nginx/templates”.

```
usuario@debian:~/proyecto$ nano roles/nginx/templates/default

# NGINX Configuracion
# Main Section
# Parametros para definir numero de procesos trabajando.
worker_processes  1;

# events Block
# Parametros para definir la conexion
events {
    worker_connections  1024;
}

# http Block
# Parametros para definir el trafico HTTP que puede manejar NGINX
http {
    include       mime.types;
    default_type  application/octet-stream;

    sendfile        on;
    keepalive_timeout  65;

# server Block
# Parametros para definir un especifico virtual host

server {
   # Define the directory where the contents being requested are stored
   # root /usr/src/app/project/;

   # Define the default page that will be served If no page was requested
   # (ie. if www.kennedyfamilyrecipes.com is requested)
   # index index.html;

   # Define the server name, IP address, and/or port of the server
```

```
   listen 8080;
   # server_name xxx.yyy.zzz.aaa

    # Define the specified charset to the “Content-Type” response header
field
   charset utf-8;

   # Configure NGINX to deliver static content from the specified folder
   #location /static {
   #    alias /usr/src/app/project/static;
   #}

   # Configure NGINX to reverse proxy HTTP requests to the upstream server
(Gunicorn (WSGI server))
   location / {
       # Define the location of the proxy server to send the request to
       proxy_pass http://flask:5000;

       # Redefine the header fields that NGINX sends to the upstream server
       proxy_set_header Host $host;
       proxy_set_header X-Real-IP $remote_addr;
       proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

       # Define the maximum file size on file uploads
       client_max_body_size 5M;
   }
  }
}
```

A continuación se procede a definir los parámetros para instalar y configurar el servidor web. Para ello se crea un fichero dentro de “roles/nginx/tasks”.

```
usuario@debian:~/proyecto$ nano roles/nginx/tasks/main.yml

---
- name: Actualizar repositorio
  apt:
    update_cache: yes
    cache_valid_time: 600

- name: Instalar nginx
  apt:
    name: "{{ item }}"
    state: present
  with_items:
    - "nginx"

- name: Configurar nginx
  template:
    src: default
    dest: /etc/nginx/nginx_ansible.conf
```

Por último se necesita configurar el fichero “container.yml”. Este fichero describe que imágenes usar y que contenedores crear, es muy similar al fichero Dockerfile o Docker Compose.

```
usuario@debian:~/proyecto$ nano container.yml

version: "2"
settings:

  conductor:
      # The Conductor container does the heavy lifting, and provides a
portable
     # Python runtime for building your target containers. It should be
derived
```

```
    # from the same distribution as you're building your target containers
with.
    base: "ubuntu:xenial"
    # roles_path:   # Specify a local path containing Ansible roles
    # volumes:      # Provide a list of volumes to mount
    # environment:  # List or mapping of environment variables

   # Set the name of the project. Defaults to basename of the project
directory.
   # For built services, concatenated with service name to form the built
image name.
  project_name: "proyecto"

   # The deployment_output_path is mounted to the Conductor container, and
the
   # `run` and `deployment` commands then write generated Ansible playbooks
to it.
  # deployment_output_path: ./ansible-deployment

  # When using the k8s or openshift engines, use the following to authorize
with the API.
   # Values set here will be passed to the Ansible modules. Any file paths
will be mounted
   # to the conductor container, allowing the `run` command to access the
API.
  #k8s_auth:
    # path to a K8s config file
    #config_file:
    # name of a context found within the config file
    #context:
    # URL for accessing the K8s API
    #host:
    # An API authentication token
    #api_key:
    # Path to a ca cert file
    #ssl_ca_cert:
    # Path to a cert file
```

```
    #cert_file:
    # Path to a key file
    #key_file:
    # boolean, indicating if SSL certs should be validated
    #verify_ssl:

  # When using the k8s or openshift engines, use the following to set the
namespace.
  # If not set, the project name will be used. For openshift, the namespace
maps to a project,
  # and description and display_name are supported.
  #k8s_namespace:
  #  name:
  #  description:
  #  display_name:

services:

  # Aplicacion
  flask:
    from: "ubuntu:xenial"
    roles:
      # Applying a role to a container
      - role: flask
    ports:
      - "5000"
     command: ["gunicorn", "--bind", "0.0.0.0:5000", "app:app", "--chdir",
"/app"]

  # Servidor web
  nginx:
    from: "ubuntu:xenial"
    roles:
      # Syntax for expressing extra role variables
      - role: "nginx"
        # These varibles are passed to the nginx config template
    ports:
```

```
      - "8080:8080"
    command: ["nginx", "-c", "/etc/nginx/nginx_ansible.conf", "-g", "daemon
off;"]

  # Add your containers here, specifying the base image you want to build
from.
  # To use this example, uncomment it and delete the curly braces after
services key.
  # You may need to run `docker pull ubuntu:trusty` for this to work.

  # web:
  #   from: "centos:7"
  #   ports:
  #     - "80:80"
    #      command:  ["/usr/bin/dumb-init",  "/usr/sbin/apache2ctl",  "-D",
"FOREGROUND"]
  #   dev_overrides:
  #     environment:
  #       - "DEBUG=1"
registries: {}
  # Add optional registries used for deployment. For example:
  #  google:
  #    url: https://gcr.io
  #    namespace: my-cool-project-xxxxxx
```

Ansible Container proporciona una forma para crear nuestro contenedor ejecutando la instrucción "ansible-container build" en el directorio raíz del proyecto.

```
root@debian:/home/usuario/ansible-container# ansible-container build
Building Docker Engine context...
Starting Docker build of Ansible Container Conductor image (please be
patient)...
Parsing conductor CLI args.
Docker™ daemon integration engine loaded. Build starting.
project=proyecto
Building service...     project=proyecto service=flask
```

```
PLAY [flask] ***************************************************************

TASK [Gathering Facts] *****************************************************
ok: [flask]

TASK [flask : Actualizar repositorios] *************************************
changed: [flask]

TASK [flask : Instalar python-pip] *****************************************
changed: [flask] => (item=[u'python-pip'])

TASK [flask : Instalar requisitos] *****************************************
changed: [flask] => (item=flask)
changed: [flask] => (item=gunicorn)

TASK [flask : Crear aplicacion] ********************************************
changed: [flask]

TASK [flask : Configurar aplicacion] ***************************************
changed: [flask]

PLAY RECAP *****************************************************************
flask                      : ok=6    changed=5    unreachable=0    failed=0

Applied role to service role=ordereddict([('role', 'flask')]) service=flask
Committed                  layer                  as                  image
image=sha256:baa4d71b15bfdc220457f824b24e43388d877e21abf06a0fe25545e617d1cb
66 service=flask
Build complete. service=flask
Building service...     project=proyecto service=nginx

PLAY [nginx] ***************************************************************

TASK [Gathering Facts] *****************************************************
ok: [nginx]

TASK [nginx : Actualizar repositorio] **************************************
```

```
changed: [nginx]

TASK [nginx : Instalar nginx] ********************************************
changed: [nginx] => (item=[u'nginx'])

TASK [nginx : Configurar nginx] ******************************************
changed: [nginx]

PLAY RECAP ***************************************************************
nginx                      : ok=4    changed=3    unreachable=0    failed=0

Applied role to service role=ordereddict([('role', 'nginx')]) service=nginx
Committed                  layer                  as                  image
image=sha256:ee24e8b88efee02dab5cc730bc8da8bde0f4f45b509db9345b6662c757220e
8e service=nginx
Build complete. service=nginx
All images successfully built.
Conductor    terminated.    Cleaning    up.                    command_rc=0
conductor_id=d6d7a34ee336d8a583482c3e92954af07c2e784549326d980d98bc8713bdbf
a4 save_container=False
```

Ansible Container proporciona una forma para iniciar nuestro contenedor ejecutando la instrucción “ansible-container run”.

```
root@debian:/home/usuario/ansible-container# ansible-container run
Parsing conductor CLI args.
Engine integration loaded. Preparing run.       engine=Docker™ daemon
Verifying service image service=flask
Verifying service image service=nginx

PLAY [Deploy proyecto] ***************************************************

TASK [docker_service] ****************************************************
changed: [localhost]

PLAY RECAP ***************************************************************
localhost                  : ok=1    changed=1    unreachable=0    failed=0
```

```
All services running.   playbook_rc=0
Conductor   terminated.   Cleaning   up.                    command_rc=0
conductor_id=4ebc96b6275db032188f4e73cb365edb851e11bd3a96b62d204261a9353606
32 save_container=False
```

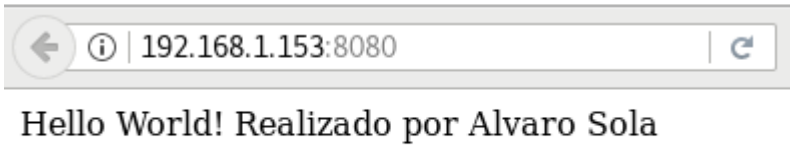
Se puede verificar el estado del contenedor y del servicio ejecutando la siguiente instrucción de Docker. (Por motivos de legibilidad disminuyo el tamaño de la letra).

```
root@debian:/home/usuario/proyecto# docker ps
WARNING: Error loading config file: /root/.docker/config.json: read /root/.docker/config.json: is a directory
CONTAINER ID         IMAGE                              COMMAND                  CREATED              STATUS               PORTS
NAMES
b9d53c24cef9         proyecto-nginx:20180521111126   "nginx -c /etc/nginx…"   15 seconds ago      Up 13 seconds       0.0.0.0:8080-
>8080/tcp    proyecto_nginx_1
70044aaf2abc         proyecto-flask:20180521110941   "gunicorn --bind 0.0…"   15 seconds ago      Up 12 seconds       0.0.0.0:32770-
>5000/tcp   proyecto_flask_1
root@debian:/home/usuario/proyecto# curl http://127.0.0.1:8080/
```

Se procede a comprobar que funciona correctamente la aplicación.

```
root@debian:/home/usuario/proyecto# curl http://127.0.0.1:8080/
Hello World! Realizado por Alvaro Sola
```

Desde el navegador se puede comprobar que está desplegada la aplicación correctamente.



Por último para poder acceder al contenedor se ejecuta la instrucción de Docker.

```
root@debian:/home/usuario/proyecto#   docker   exec   -i   -t   ID_CONTAINER
/bin/bash
```

6. PROBLEMAS Y SOLUCIONES

6.1. Instalación

Durante la instalación de Ansible Container en la distribución Centos 7, me he encontrado problemas de falta de dependencias y módulos. Por ejemplo.

```
from pip.req import parse_requirements
ImportError: No module named req
Command "python setup.py egg_info" failed with error code 1 in /tmp/pip-install-zv39PO/ansible-container/
```

El error es debido a que no esta instalado el módulo especificado. La solución sería instalar el siguiente módulo.

```
python2-requests.noarch
```

6.2. Creación

Durante la ejecución de la receta, puede aparecer un error de dependencias de Docker, esto es debido a que normalmente al instalar Docker instala la última versión y Ansible Container puede pedir la versión de Docker 2.7. La solución es instalar Docker con la versión especificada.

```
root@debian:/home/usuario/proyecto# pip install docker==2.7.0
```

6.3. Contenedores

Ejecutar el proyecto Ansible Container estando levantado puede dar fallos al volverlo ejecutar debido a que los contenedores están creados y se intenta crear un contenedor con el mismo registro.

Una solución es limpiar el registro de contenedores.

```
docker rm $(docker ps -aq)
docker rmi $(docker images -q)
```

7. RESULTADOS Y CONCLUSIÓN

7.1. Resultados

En este proyecto se puede ver como crear y administrar contenedores Docker a través de una receta Ansible sin tener que tener conocimientos de Docker, debido a que Ansible Container no necesita utilizar fichero Dockerfile.

Como prueba de funcionamiento se puede ver como automatizar el despliegue de una aplicación Flask que interactúe con un servidor web Nginx en un contenedor Ubuntu.

Se puede desplegar este proyecto a través de mi repositorio Git Hub, ejecutando las instrucciones especificadas en el repositorio.

```
https://github.com/alvarosola/ansible-container
```

7.2. Conclusión

La herramienta aún se encuentra en desarrollo activo, no siempre es fácil solucionar problemas, debido a que no hay suficiente documentación, pero aún así el proyecto Ansible-container definitivamente es muy prometedor.

En mi opinión incluso en su estado actual es una tecnología muy útil, debido a su poder de crear y administrar contenedores utilizando toda la potencia de Docker con la ayuda de Ansible Playbooks.

8. BIBLIOGRAFÍA

Las siguientes url son los enlaces que he seguido para poder realizar este proyecto.


```
https://docs.ansible.com/ansible-container/

https://www.ansible.com/integrations/containers/ansible-container

https://docs.ansible.com/ansible-container/installation.html

https://docs.ansible.com/ansible-container/getting_started.html

https://www.revistacloudcomputing.com/2016/06/red-hat-presenta-ansible-
container/
```