Universidad de Granada

Cloud Computing

Máster Profesional en Ingeniería Informática

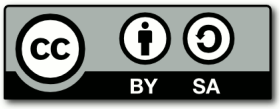
Servicio de almacenamiento en cloud de alta disponibilidad

Autor
Ernesto Serrano Collado

Profesor
Manuel Parra

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Granada, 5 de mayo de 2018



Servicio de almacenamiento en cloud de alta disponibilidad

Ernesto Serrano Collado

Resumen

Palabras clave: *cloud computing, openstack, docker, owncloud, nginx, postgresql, ldap*

Los objetivos de esta práctica son:

1. Gestionar y desplegar Máquinas Virtuales (MVs) de forma automatizada.
2. Desplegar servicios y aplicaciones en infraestructuras distribuidas, utilizando herramientas de automatización de gestión y configuración de software.
3. Diseñar un servicio de aprovisionamiento dinámico de recursos.
4. Aprender a orquestar el conjunto de servicios y aplicaciones.

Yo, **Ernesto Serrano Collado**, alumno de la titulación **Máster Profesional en Ingeniería Informática** de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, autorizo la ubicación de la siguiente copia de mi Trabajo (*Servicio de almacenamiento en cloud de alta disponibilidad*) en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Ernesto Serrano Collado

Granada, a 5 de mayo de 2018

Índice general

1. Introducción **1**

2. Implementación **3**

3. Conclusiones **9**

4. Bibliografía **11**

Capítulo 1

Introducción

Para el diseño del sistema de almacenamiento de alta disponibilidad en Cloud, hay que considerar los componentes siguientes que deben ser desplegados en la arquitectura que se ha provisto dentro de OpenStack:

1. Servicio de alta disponibilidad / balanceador de carga. Este servicio será el encargado de enrutar y balancear las peticiones de los clientes a los servicios de almacenamiento redundantes que se desplegarán. Se usará HAproxy o NGINX. Se recomienda utilizar NGINX, debido a su facilidad de uso.

2. Servicio de almacenamiento. Es el servicio de almacenamiento y gestión de los datos en cloud. Este servicio se encontrará replicado en al menos dos nodos de la infraestructura y dará soporte de almacenamiento en cloud. Se usará la plataforma de almacenamiento de ficheros owncloud.

3. Servicio de auto-escalado dinámico. Este servicio permite que se creen o destruyan nuevos contenedores para el servicio de almacenamiento. El servicio debe ser programado en el lenguaje que prefieras.

4. Servicios de bases de datos. El servicio de almacenamiento requiere de un sistema de base de datos para poder almacenar toda la información de usuarios, ficheros, metadatos, etc. Para ello como mínimo se usará un nodo, con MariaBD/MySQL/PostgreSQL.

5. Servicio de autenticación. Este servicio se encargará de habilitar el acceso a los usuarios de owncloud en el sistema de almacenamiento en cloud. El servicio de directorio de usuarios será LDAP y podrá estar compartido dentro del nodo de Base de Datos.

Capítulo 2

Implementación

Para la realización de esta práctica me propuse orquestar utilizando el paradigma PaC (Platform as Code), intenté definir la creación de máquinas con Vagrant y provisionar los diferentes servicios completamente con ansible.

Tras mucha prueba y error con la API de OpenStack y con la particularidad de que no tener disponible lo que en OpenStack denominan *Magic ips* (ips publicas para cada maquina desplegada) conseguí tanto la creación de máquinas como el provisionamiento con ansible en las mismas. Todo esto cuando la máquina lo permitía ya que hubieron diversos problemas que obligaban al reinicio de la máquina OpenStack.

Para poder conectarme a las distintas máquinas tuve que hacer uso de un proxypass ssh que se conectaba al servidor atcstack y de ahí saltaba a las distintas máquinas. Para facilitar la tarea agregué la configuración necesaria al archivo `/.ssh/config`

Fragmento de código 2.1: fichero /.ssh/config

```
Host openstack
  Hostname atcstack.ugr.es
  User CC_XXXXXXX

Host machine_158
  User          ubuntu
  HostName      192.168.0.158
  ProxyCommand ssh CC_74003802@atcstack.ugr.es -W %h:%p

Host machine_159
  User          ubuntu
  HostName      192.168.0.159
  ProxyCommand ssh CC_74003802@atcstack.ugr.es -W %h:%p

Host machine_160
  User          ubuntu
  HostName      192.168.0.160
  ProxyCommand ssh CC_74003802@atcstack.ugr.es -W %h:%p

```

```
Host machine_161
  User          ubuntu
  HostName      192.168.0.161
  ProxyCommand ssh CC_74003802@atcstack.ugr.es -W %h:%p
```

Aunque el código Vagrant funcionaba bien, las continuas caídas del servicio `OpenStack` hizo que el resto de la configuración la hiciera en instancias en mi propia cuenta de AWS.

Para poder iniciar sesión entre las distintas máquinas se ha generado un par de claves publica-privada que podemos encontrar en el directorio `app_keys`.

La configuración del servidor de base datos fue bastante trivial, agregando usuarios y asignando los roles necesarios. Se ha optado por `PostgreSQL` por diferenciar el entorno del típico `LAMP` (Linux + Apache + MySQL + PHP).

La configuración del servidor `LDAP` si ha sido mas complejo pero al final se ha resuelto instalando el *wrapper* `ldapscripts` y configurando `debconf` para asignarle la contraseña de superusuario que queramos al paquete `slapd`.

`OwnCloud` necesita un directorio para persistencia, por lo que si queremos balancear de forma correcta entre máquina la solución mas sencilla es compartir un directorio por NFS, así que instalamos un servidor NFS y configuramos lo necesario para que los demás servidores puedan montar dicha unidad. El código de OwnCloud realiza una serie de bloqueos en el fichero de configuración y dichos bloqueos (`flock`) no son soportados por NFS por lo que al final se ha optado por inyectar un fichero `Config.php` dentro del contenedor que evite dichos bloqueos.

Una vez configurado el servidor de BBDDD pasamos a configurar los dos servidores que correrán los contenedores Docker de OwnCloud, en los mismos se han instalado tanto Docker como docker-compose que nos facilitará levantar y escalar contenedores de forma dinámica.

Se ha utilizado el contenedor 'owncloud/server' que podemos encontrar en Docker Hub y que está publicado por los propios desarrolladores de OwnCloud.

Aunque dicho contenedor incorpora el plugin user_ldap, dicho plugin no está activado de forma automática por lo que hemos tenido que definir la variable de entorno `OWNCLOUD_APPS_ENABLE=user_ldap` para que active el plugin.

Aparte de activar el plugin debemos configurarlo cosa que por defecto no han contemplado los desarrolladores. Al menos permiten ejecutar scripts al

finalizar un despliegue. Para ello basta con colocar los scripts que queramos en el directorio /etc/post_server.d/ del contenedor. Mirando en la documentación de OwnCloud (y tras mucha mas prueba y error) he desarrollado un sencillo script que haciendo uso del cliente `occ` establece los valores de configuración necesarios para permitir la conexión al servidor LDAP

Fragmento de código 2.2: bash version

```
#!/usr/bin/bash
LDAP_SERVER={{database}}
# Enable ldap plugin (better do it in env vars)
# su www-data -c "php occ app:enable user_ldap -n -q --no-ansi
# Enable and configure LDAP and external storage plugin
occ ldap:create-empty-config -n -q --no-ansi
occ ldap:set-config 's01' ldapBase dc=example,dc=com -n -q --no-ansi
occ ldap:set-config 's01' ldapBaseGroups dc=example,dc=com -n -q --no-ansi
occ ldap:set-config 's01' ldapBaseUsers dc=example,dc=com -n -q --no-ansi
occ ldap:set-config 's01' ldapCacheTTL 600 -n -q --no-ansi
occ ldap:set-config 's01' ldapConfigurationActive 1 -n -q --no-ansi
occ ldap:set-config 's01' ldapEmailAttribute mail -n -q --no-ansi
occ ldap:set-config 's01' ldapExperiencedAdmin 0 -n -q --no-ansi
occ ldap:set-config 's01' ldapExpertUsernameAttr uid -n -q --no-ansi
occ ldap:set-config 's01' ldapGroupDisplayName cn -n -q --no-ansi
occ ldap:set-config 's01' ldapGroupFilter objectClass=posixGroup -n -q --no-ansi
occ ldap:set-config 's01' ldapGroupFilterMode 0 -n -q --no-ansi
occ ldap:set-config 's01' ldapGroupMemberAssocAttr uniqueMember -n -q --no-ansi
occ ldap:set-config 's01' ldapHost ${LDAP_SERVER} -n -q --no-ansi
occ ldap:set-config 's01' ldapLoginFilter '(&(|(objectclass=posixAccount))(uid=%uid))' -n -q --no-ansi
occ ldap:set-config 's01' ldapLoginFilterEmail 0 -n -q --no-ansi
occ ldap:set-config 's01' ldapLoginFilterMode 0 -n -q --no-ansi
occ ldap:set-config 's01' ldapLoginFilterUsername 1 -n -q --no-ansi
occ ldap:set-config 's01' ldapNestedGroups 0 -n -q --no-ansi
occ ldap:set-config 's01' ldapPagingSize 500 -n -q --no-ansi
occ ldap:set-config 's01' ldapPort 389 -n -q --no-ansi
occ ldap:set-config 's01' ldapQuotaAttribute mailQuota -n -q --no-ansi
occ ldap:set-config 's01' ldapTLS 0 -n -q --no-ansi
occ ldap:set-config 's01' ldapUserDisplayName cn -n -q --no-ansi
occ ldap:set-config 's01' ldapUserFilter objectClass=posixAccount -n -q --no-ansi
occ ldap:set-config 's01' ldapUserFilterMode 0 -n -q --no-ansi
occ ldap:set-config 's01' ldapUuidGroupAttribute auto -n -q --no-ansi
occ ldap:set-config 's01' ldapUuidUserAttribute auto -n -q --no-ansi
```

Una vez configurado LDAP de forma automática en nuestro contenedor procedimos a levantar mas de un contenedor. OwnCloud por defecto no contempla esta configuración por lo que tuvimos que hacer un bypass del script `/usr/local/bin/owncloud-installed`, que es uno de los scripts de comprobación que ejecuta a la hora de inicializar el contenedor y que crea la base de datos. En nuestro caso lo cambiamos para que en caso de existir la base de datos retornara un valor para que pensara que estamos ante una actualización en lugar de en una instalación.

Fragmento de código 2.3: comprobacion de que no está previamente configurado OwnCloud

```
#!/usr/bin/env bash
set -e
if [[ $(PGPASSWORD=$OWNCLOUD_DB_PASSWORD psql -U $OWNCLOUD_DB_USERNAME
     -h {{database}} $OWNCLOUD_DB_NAME -tAc "SELECT to_regclass('
    oc_users');") != "oc_users" ]]
then
  exit 1
else
  exit 0
fi
```

Una vez resuelto este problema ya hemos podido levantar múltiples contenedores con la sencilla instrucción `docker-compose up --scale owncloud=N` siendo N el numero de contenedores que queremos tener. El propio docker-compose se encarga de reescalar si el numero es distinto del numero de contenedores en ejecución.

Hay que tener en cuenta que al permitir levantar contenedores de forma dinámica no podemos fijar los puertos, así que en el docker-compose le indicamos que asigne un puerto aleatorio a cada contenedor que redirija al puerto 80 que es el que está sirviendo OwnCloud.

Una vez configuradas las máquinas Docker hemos pasado a configurar el balanceador, que consiste en un nginx configurado para incluir un fichero de configuración donde sacar la lista de servidores que llamará a través de `proxypass`.

Dicha lista de contenedores la hemos extraído de la salida del comando `docker ps --format ".Ports"` que nos devuelve los puertos asignados a los contenedores. De esta forma nuestro sistema permite levantar tantos contenedores como queramos sin tener que fijar un numero de puertos.

Una vez configurado el script de balanceo, hemos procedido a probar que tal se comportaba viendo que como sospechábamos `OwnCloud` no está preparado de forma nativa para ser balanceado al guardar las sesiones de los usuarios en el sistema interno de `PHP` que guarda en disco las mismas siendo lo recomendado en estos casos es guardar la configuración de las sesiones en la base de datos.

Para el balanceo de carga se ha utilizado un sencillo script en bash que incorpora las funciones necesarias para consultar el estado de `NGINX` y conectarse a los dos nodos `Docker` para mediante docker-compose hacer un reescalado automático. El script detecta automáticamente los puertos en los que se ha levantado Docker y los incorpora a la lista del *upstream* de `NGINX` haciendo un reinicio controlado (*reload*) que permite que el servicio no esté

caído en ningún momento como si pasaría con un reinicio normal (*restart*). Dicho scripts guarda un log de las tareas realizadas en el archivo `saed.log`

Para dicho balanceo nos hemos limitado a realizar la operación matemática proporcionada por el profesor que en nuestras pruebas de carga con `siege` no nos ha dado valores diferentes de 1 o 2, quizá por filtrado de la propia AWS, lo que hemos hecho es utilizar ese numero como valor del numero de contenedores a levantar en cada nodo.

Capítulo 3

Conclusiones

Esta práctica ha supuesto un gran reto ya que en mi opinión el sistema `OwnCloud` no está preparado para ser balanceado de forma sencilla, al menos utilizando los contenedores `Docker` proporcionados por los desarrolladores. De igual forma dichos contenedores no están preparados para integrar LDAP sin realizar diversas tareas de programación, de igual forma el código de OwnCloud no está utilizando correctamente las sesiones PHP para funcionar en un entorno balanceado.

Aun así me ha permitido trabajar con `OpenStack` así como jugar con una configuración que requería de ingenio para poder saltarse las restricciones de la plataforma.

Capítulo 4

Bibliografía

Se ha utilizado la documentación oficial de los proyectos, así como el código fuente de los proyectos utilizados.

- Documentación de Docker: https://docs.docker.com/compose/environment-variables/
- Documentación de Nginx: https://nginx.org/en/docs/
- Documentación de OwnCloud: https://doc.owncloud.org/
- Documentación de Ansible: http://docs.ansible.com/
- StackOverflow: https://stackoverflow.com
- Creative Commons Share Alike 4.0: https://creativecommons.org/licenses/by-sa/4.0/