**Universidad ICESI**

**Christian López Morcillo**

**Juan Diego García**

**Luisa Vivas Mayor**

**Rodrigo Esteban Rivera**

**Curso:** Sistemas Operativos

**Docente:** Daniel Barragán C.

**Tema:** Despliegue automático de un espejo

**Correo:** daniel.barragan at correo.icesi.edu.co

**Objetivos**

* Emplear herramientas de aprovisionamiento automático para la realización de tareas en infraestructura
* Instalar y configurar espejos de sistemas operativos en forma automática para el soporte al aprovisionamiento de máquinas virtuales en clústers de cómputo
* Especificar y desplegar ambientes conformados por contenedores virtuales para la realización de tareas específicas

**Solución proyecto:**

Para el aprovisionamiento de los servicios solicitados fue necesario instalar Aptly y primero se debe crear las llaves RSA que se emplean para transferir los archivos de forma segura. para esto se debe utilizar los siguientes comandos:

$ gpg --gen-key

abrimos una nueva sesión con el fin de generar la entropía:

$ cat /dev/urandom

Después de generar las llaves, pasamos a importar las llaves que contiene la máquina a la base trustedkeys.

$ gpg --no-default-keyring --keyring /usr/share/keyrings/ubuntu-archive-keyring.gpg --export | gpg --no-default-keyring --keyring trustedkeys.gpg --import

Ahora se agrega el repositorio de Aptly el archivo sources list de la máquina para la instalación de aptly y se realiza mediante el siguiente comando:

$ deb http://repo.aptly.info/ squeeze main > /etc/apt/sources.list

importamos la llave del servidor Aptly mediante el siguiente comando:

# sudo apt-key adv --keyserver keys.gnupg.net --recv-keys 9E3E53F19C7DE460

Después de realizar lo anterior podemos instalar apropiadamente como cualquier otro paquete de software:

# apt-get update  
# apt-get install aptly

**Creación mirror :**

Después de instalar aptly se crea el miror, mediante el siguiente comando:

$ aptly mirror create -architectures=amd64 -filter='Priority (required) | Priority (important) | Priority (standard) postgresql' -filter-with-deps mirror-xenial http://mirror.upb.edu.co/ubuntu/ xenial main

$ aptly mirror update mirror-xenial

Este comando permite definir cuáles paquetes se instalarán y el segundo comando permite actualizar el mirror con estas configuraciones.

Seguido de esto realizamos un snapshot del mirror :

$ aptly snapshot create mirror-snap-xenial from mirror mirror-xenial

Se mezcla los snapshot:

$ aptly publish snapshot mirror-snap-xenial  
$ aptly serve

Ahora bien para el cliente, se debe agregar la llave pública generada por el mirror:

$ apt-key add my\_key.pub

Luego, se configura para apuntar al mirror y de esta manera poder descargar los paquetes, desde el repositorio:

$ echo "deb http://direccionIpMirror:8080/ xenial main" > etc/apt/sources.list

Se actualiza el Apt para guardar los cambios.

$ apt-get update -y

Se crea un archivo denominado docker-compose.yml en el directorio del proyecto y se pegó lo siguiente:

|  |
| --- |
| Aqui docker-compose.yml |

Este archivo Compose define dos máquinas, un cliente y un mirror. En cada uno de ellos se definen los nombres y las características:

* El mirror se construye a través del Dockerfile que se encuentra en la carpeta mirror. Se pasa por variable se le inserta un arreglo de variables de los elementos que se van a inyectar en los elementos del mirror, en este caso vim-gnome,python3. También se exponen hacia las máquinas contenedoras el puerto 8080.
* El cliente se construye a través del Dockerfile que se encuentra en la carpeta cliente. Se definen variables del sistema del contenedor cliente con el parámetro links. En caso del puerto lo que se hace es un mapeo del puerto 8080 a través 8090.

**Cliente**

Empezando por el lado del cliente, el primer archivo que es ejecutado el Dockerfile en la carpeta del cliente. Este archivo se encuentra la configuración necesaria :

|  |
| --- |
| Aqui Dockerfile del cliente |

Se define el contenedor cliente, se hace una definición de como se va a crear el contenedor. Cojemos la imagen del contenedor que ya tiene instalado, en este caso apache httpd:2.2

Agregamos una llave pública para hacer una conexión segura encriptada que luego será usada con una llave privada. La llave privada es importada sobre el mirror y la pública es dada a todos los cliente. Es decir, el cliente la comunicación la encripta con una llave pública y el mirror con una llave privada.

Entonces se copia el archivo public.asc contenido en la carpeta keys en el directorio /tmp del contenedor, seguidamente se agrega a la llave con el comando apt-key y finalmente se elimina el archivo temporal.

Despues se copia de la carpeta /conf/ el archivo Entrypoint.sh al directorio /scripts/ del contenedor.

Un Entrypoint consiste en mostrar lo que se va a mostrar en consola el estado de la conexión con el mirror.

|  |
| --- |
| Aqui archivo Entrypoint.sh del cliente |

Finalmente, se prueba que el mirro está conectando. Se agrega la dirección del mirror en el directorio */etc/apt/sources.list* que corresponde a la lista de los repositorios.

Luego, se instalan los paquetes se limpia y se actualiza para finalmente realizar *httpd-foreground*.

**Mirror**

Por parte del mirror, el Dockerfile se encarga de levantar los servicios para ejecutarse de la forma correcta.

|  |
| --- |
| Aquí Dockerfile del mirror |

En este punto se debe tener muy en cuenta que se debe agregar una llave privada. Llave fue creada por aparte, pues dentro de un contenedor es imposible generar entropía. Esta llave privada es copiada de la carpeta /keys/ directorio /keys/ en el contenedor, así como el archivo aptly.conf al directorio del contenedor /etc/.

Aclaramos que estos pasos fueron realizados en base a la guía de instalación de aplty.

Se importan las llaves que previamente fueron agregadas para luego eliminar el archivo de la llave por seguridad. Se realizan los pasos de instalación de Aptly y sus dependencias. Entre estas dependencias son xz-utils y bzip2.

Se instala Except para crear respuestas automáticas cuando en una función o instancia se pide el ingreso de datos. Se implementa Except agregando a la lista del repositorio del contenedor, se actualiza y se actualiza.

Se continua con la guía de instalación de Aptly para la configuración del mirror.

Al momento de publicar el snapshot se agrega en el contenedor el archivo publish\_snapshot.sh y se le dan permisos de ejecución

|  |
| --- |
| Aqui archivo publish\_snapshot.sh del mirror |

En este archivo se automatizan las respuesta a las preguntas que pide el servicio. Luego este archivo es ejecutado.

Seguidamente se copia el archivo Entrypoint al contenedor del mirror y se conceden permisos de ejecución.

|  |
| --- |
| aqui entrypoint.sh del mirror |

Mientras este archivo es ejecutado se leen por parámetros el array de las variables que fueron creadas en el archivo del docker-compose, donde se instalan cada una de estas dependencias sobre el snapshot del mirror.

Finalmente se ejecuta el Entrypoint.sh desde el Dockerfile del contenedor

**Dificultades**

A la hora de crear una llave pública para el mirror y una llave privada para cada cliente, no se lograba generar una conexión. Ante el problema anterior, se usó un modo promiscuo, es decir, se usaron llaves públicas y de este modo los clientes se lograron autenticar ante el mirror. Se genera en un computador a parte, se importa la pública en los clientes del mirror y la privada en el mirror.

Un problema que surgió en el desarrollo del proyecto fué la dificultad de la identificación de la dirección ip de cada contenedor. Lo anterior se soluciona identificando los FQDN (id) que proporciona Docker para cada contenedor.

Otro problema fue garantizar que primero se iniciará el mirror antes que el cliente, esto se mitigó, utilizando depends\_on, lo cual es una utilidad que permite definir el orden de ejecución de los contenedores.

Para los programas en los cuales se requería de interacción humana, por ejemplo, crear el mirror, se utilizó la herramienta expect que, se encarga de emular la interacción humana.

Una dificultad encontrada en el desarrollo fue, como inyectar un arreglo de variables dentro de los comandos de Docker. Se solucionó creando una variable de sistema separada por comas y luego se hizo un split para crear un array, se usó el array para realizar los comandos de Docker.