UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Planificación y Gestión de Infraestructuras TIC

<Clúster de alta disponibilidad (Trabajo 1)>

Autores: Miguel de la Cal Bravo y Félix Ángel Martínez Muela

Titulación: Máster en Ingeniería Informática

Fecha: 07/05/2020

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN

En este trabajo práctico utilizaremos las herramientas *Vagrant* y *Ansible* para configurar y aprovisionar varias máquinas virtuales formando con ellas un clúster *Linux* de alta disponibilidad con *Corosync* y *Pacemaker*.

Para ello, partiremos de una máquina con sistema operativo *Ubuntu* nativo en la versión 18.04 *LTS* (en principio la versión no debe ser un problema en el anfitrión, ya que también ha sido probado sobre la 20.04 *LTS*). Sobre ella, se crearán un total de tres máquinas virtuales *Linux* con sistema operativo *Ubuntu* en su versión 14.04 *LTS*, cada una teniendo sus direcciones IP dentro de una red privada, siendo llamadas estas máquinas *nodo1*, *nodo2* y *nodo3*.

Además, este trabajo puede trabajar con un número de nodos N, desde mínimo un nodo hasta un máximo de nueve nodos (podríamos haberlo configurado para más de nueve nodos, pero entendemos que el usuario no empleará en sus ejemplos más de tres o cuatro nodos por temas de recursos en su ordenador, además, esto hubiera complicado en exceso las configuraciones de ficheros para, por ejemplo, la asignación de direcciones IP, etc)

Para la realización de este trabajo utilizaremos *Vagrant* para crear las máquinas virtuales junto con *Ansible* para el aprovisionamiento de estas. Tendremos un único *Vagrantfile* donde se configurará tanto el nodo maestro como los nodos esclavos. En cuanto a la parte de aprovisionamiento, emplearemos diferentes *playbooks* de *Ansible* según sean necesarias las configuraciones, distinguiendo entre un *playbook* para el maestro, otro para los esclavos y un último *playbook* para todos los nodos que forman el clúster.

Nota: en el documento se referencian nodos maestros y nodos esclavos, sin embargo, en realidad el clúster no realiza ningún tipo de distinción entre nodos, por lo que se ha mantenido una arquitectura compatible con maestro/esclavo para poder ser reutilizado el *Vagrantfile* en proyectos que si distingan la configuración de los nodos.

2. PUESTA EN MARCHA

2.1 Instalación de software

Para poner en marcha el proyecto que vamos a realizar, primero es necesario configurar nuestra máquina anfitrión instalando el *software* indicado en la *Tabla 1*:

Software	***Versión***
Ansible[1]	2.9.6
Vagrant[2]	2.2.7
VirtualBox[3]	5.2.34

Tabla 1. Software utilizado en el proyecto

[1] Ansible. https://www.ansible.com/
[2] Vagrant. https://www.vagrantup.com/
[3] VirtualBox. https://www.virtualbox.org/

Este sería todo el *software* necesario para poder levantar las máquinas en nuestro proyecto, sin necesidad de instalar ningún *plugin* adicional de *Vagrant* ni *VirtualBox*.

2.2 Consejos y recomendaciones

Por un lado, se recomienda probar este proyecto con un ordenador con buenos recursos a nivel de *CPU* y memoria *RAM*, ya que virtualizaremos en este ejemplo un total de tres máquinas. También, es recomendable hacer este proceso sobre una máquina nativa *Linux* por temas de rendimiento.

Por otro lado, también es conveniente que el usuario disponga de una buena conexión a Internet, ya que, en primer lugar, se nos descargará el *box* de *VirtualBox* que será utilizado para crear las máquinas, y, posteriormente, será necesario aprovisionar estas máquinas instalando un conjunto de paquetes y demás *software* para obtener finalmente nuestro clúster completamente funcional.

3. SOLUCIÓN PROPUESTA

En este apartado se analiza la solución propuesta para la creación de nuestro clúster de alta disponibilidad con *Corosync* y *Pacemaker*. Cabe mencionar que dicho proyecto ha sido creado y actualizado de forma frecuente en el siguiente repositorio de *GitHub*: https://github.com/FelixAngelMartinez/pgitic_corosync_pacemaker

3.1 Arquitectura del clúster de alta disponibilidad

En la *Figura 1*, se muestra la arquitectura de la infraestructura propuesta en nuestra solución para crear el clúster de alta disponibilidad con *Corosync* y *Pacemaker*.

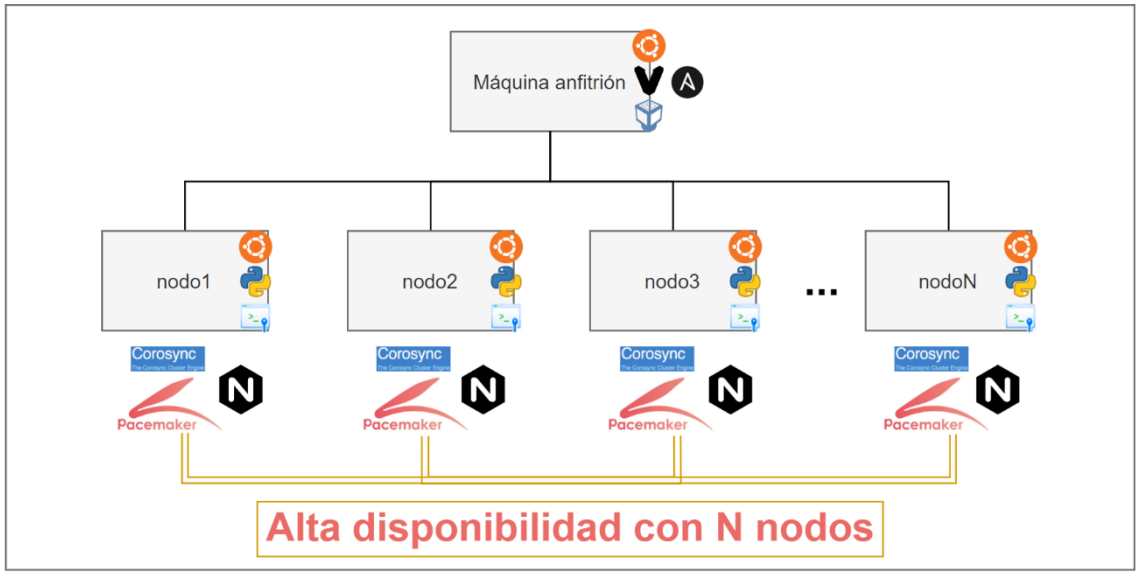

Figura 1. Arquitectura del sistema de alta disponibilidad con Corosync y Pacemaker

En primer lugar, en la parte superior de la arquitectura encontramos nuestra máquina anfitriona, con todo el *software* necesario para crear nuestro clúster, especificado en el apartado anterior.

Por otra parte, en el nivel inferior tenemos los nodos que conforman el clúster, nombrados como *nodo1*, *nodo2*, *nodo3*, …, *nodoN*. Todos ellos tienen sistema operativo *Ubuntu* con *Python* instalado por defecto en su imagen de *VirtualBox* (ya que *Python* es necesario para poder aprovisionar los nodos mediante *Ansible*), y también los configuraremos con una serie de claves privadas que generaremos para que se puedan comunicar entre ellos por *ssh*. Adicionalmente, estos nodos serán aprovisionados con el *software* de *Nginx*, *Corosync* y *Pacemaker*, dando lugar a un clúster de alta disponibilidad para el recurso *Nginx*.

En las arquitecturas clúster, no sólo tenemos la gran ventaja de tener alta disponibilidad[4], sino también los de alta velocidad, balanceo de carga, escalabilidad y resistencia ante ataques *DDoS*.

3.2 Estructura del proyecto desarrollado

Una vez estudiada la arquitectura del clúster que crearemos, comenzaremos con los contenidos de nuestra carpeta del proyecto. En el *Listado 1*, se muestra el árbol de directorios y ficheros que conforman nuestro proyecto (situándonos dentro de la carpeta */src*), distinguiendo los ficheros y carpetas existentes desde el principio en negrita, de aquellos que se generan durante el aprovisionamiento sin negrita.

```
.
├── keys
│   ├── 192.168.128.101-id_rsa.pub
│   ├── 192.168.128.102-id_rsa.pub
│   └── 192.168.128.103-id_rsa.pub
│   └── 192.168.128.10X-id_rsa.pub
├── provision.yml
├── scripts
│   └── crm_configure_fence.sh
├── shared_folders
│   ├── master_shared_folder
│   │   └── master_shared_file.txt
│   └── slave_shared_folder
│       └── slave_shared_file.txt
├── templates
│   ├── authkey
│   ├── corosync.j2
│   ├── crm_configure_fence.j2
│   ├── hosts.j2
│   └── pacemaker.conf
└── Vagrantfile
```

Listado 1. Estructura de directorios y ficheros del proyecto clúster de alta disponibilidad

Como vemos, partimos de un fichero *Vagrantfile* para la creación y configuración de los nodos, distinguiendo entre un nodo "maestro" (que sería nuestro *nodo1*) y los nodos "esclavos" (del nodo *nodo2* al *nodoN*).

También tenemos un único *playbook* de *Ansible*, el cual se estructura en cuatro jugadas de tareas que se ejecutan sobre los nodos que sean necesarios. En estas tareas, los nodos serán aprovisionados con aquellos paquetes, software y configuraciones necesarias para el clúster.

[4] https://blog.infranetworking.com/servidor-en-cluster/#Ventajas_de_los_servidores_en_cluster

En la carpeta */templates*, tendremos aquellas plantillas de configuraciones que utilizaremos para generar los contenidos de diferentes ficheros y un *script* de configuración de *Corosync* y *Pacemaker*. Veremos algunos ficheros de *Jinja2*[5] (extensión *.j2*), muy útiles para configurar los ficheros en función del número de nodos de nuestra infraestructura, empleando sus bucles.

También encontramos una carpeta llamada */shared_folders*, con dos subcarpetas en su interior por si quisiéramos compartir algún fichero en el futuro entre la máquina anfitrión y las máquinas virtuales.

Adicionalmente, sobre la ejecución se crearán las carpetas de */keys* y */scripts*, con las claves *ssh* de los nodos para compartirlas entre los nodos y que tengan comunicación entre todos ellos, así como un script de comandos de *Pacemaker* para configurar el clúster generado a partir de una plantilla *.j2*. Por último, también encontramos el fichero *authkey* dentro de */templates*, ya que este fichero se ha generado en el nodo1 y es la clave de autenticación para *Corosync*.

De esta manera, ya entenderemos todos los ficheros que conforman nuestro proyecto, y podemos seguir con una explicación más detallada de los mismos.

3.3 Funcionamiento y explicación del Vagrantfile

En el *Listado 3* del *ANEXO A. Código del fichero Vagrantfile*, se muestra el código del *Vagrantfile* utilizado para la creación de las máquinas virtuales con *Vagrant* y *VirtualBox*.

El *Vagrantfile* ha sido desarrollado de tal manera, que sea posible ampliar el número de nodos a crear con el mínimo esfuerzo, para ello se ha creado una variable llamada *N_Machines*, mediante la cual cambiando su valor crearemos un número de esclavos *Machines-1*, ya que siempre se crea un nodo máster. Las máquinas son aprovisionadas con la versión de *Ubuntu 14.04 LTS* (con el *box VirtualBox* de *ubuntu/trusty64*), pudiendo cambiar los parámetros de memoria o *CPUs* en función de las características de nuestro equipo.

Se han decidido crear un conjunto de carpetas compartidas, diferenciadas entre el nodo máster y los nodos esclavos, para de esta manera tener un punto común entre nodos del mismo tipo, para futuras instalaciones o lo que fuese necesario en un futuro un medio para compartir archivos rápidamente. Como se indicó previamente, el clúster **NO** posee una arquitectura *maestro/esclavo*, pero se ha desarrollado dicha plataforma para poder ser reutilizada en futuras arquitecturas.

3.4 Funcionamiento y explicación del playbook de Ansible

En cuanto a los aprovisionamientos de estos nodos con la herramienta *Ansible*, tenemos un único *playbook* llamado *provision.yml*, compuesto por un total de cuatro *plays* o jugadas con la finalidad de facilitar y distinguir sobre qué nodos se ejecutan diferentes series de tareas:

- *Play 1: Aprovisionar nodos con paquetes y configuraciones básicas* → Todos los nodos
- *Play 2: Configurar clave de autenticación de Corosync en el nodo 1* → nodo1
- *Play 3: Configurar clúster Corosync y Pacemaker* → Todos los nodos
- *Play 4: Configurar Pacemaker* → nodo1

[5] Jinja2. https://jinja.palletsprojects.com/

Antes de comenzar con la explicación del *playbook*, podemos echar un vistazo previo a su código y el contenido de otros ficheros de plantillas necesarios, en el *ANEXO B. Código del Playbook de aprovisionamiento de Ansible* y *ANEXO C. Código de las plantillas Jinja2 y .conf utilizadas*.

A continuación, se explica jugada a jugada lo que automatiza el *playbook* desarrollado.

3.4.1 Play 1: Aprovisionar nodos con paquetes y configuraciones básicas

La primera jugada, cuyo código se muestra en el *Listado 4*, es lanzada sobre los N nodos del clúster con el usuario remoto *vagrant* y permisos de superusuario.

Además, contiene una serie de tareas previas o tareas base, agrupadas en la sección de *pre-tasks*, cuyos objetivos son: 1) configurar el fichero */etc/hosts* (para que los nodos se conozcan entre ellos como *nodo1*, *nodo2*…) mediante el módulo *template* que obtendrá el contenido a partir de la plantilla de *Jinja2 hosts.j2* dentro de la carpeta *templates/* del proyecto (ver *Listado 8*); 2) permitir el *login* por *ssh* con *root* utilizando el módulo *lineinfile*, para sustituir la línea que comienza por *"PermitRootLogin"* por *"PermitRootLogin without-password"*; y 3) generar y copiar las claves *ssh* entre todos los nodos y reiniciar el servicio *ssh*.

Seguidamente, en la sección *tasks*, tenemos varias tareas que se encargarán de la instalación de paquetes necesarios, así como la configuración de los puertos 5404, 5405 y 5406 del *firewall*.

Con esto, concluiría la primera jugada de nuestro *playbook*, destinada a tareas de instalación y configuraciones básicas del entorno.

3.4.2 Play 2: Configurar clave de autenticación de Corosync en el nodo 1

La segunda jugada del *playbook provision.yml*, cuyo código se muestra en el *Listado 5*, se encarga de generar y configurar una clave de autenticación de *Corosync* en el *nodo1*.

A diferencia de la primera jugada, esta vez las tareas únicamente las lanzaremos sobre el nodo1, manteniendo el usuario *vagrant* y permisos de superusuario. Esta vez tampoco tendremos una sección *pre_tasks*, por lo que partimos directamente de la sección *tasks* donde se instalará un paquete llamado *Haveged* con el que generaremos la clave que, seguidamente, será desinstalado ya que no será necesario más adelante en el proceso.

Por último, nos descargaremos la clave de autenticación de *Corosync* generada en la máquina anfitrión, ya que posteriormente necesitaremos copiarla al resto de nodos de la infraestructura.

3.4.3 Play 3: Configurar clúster Corosync y Pacemaker

En la tercera jugada, mostrada en el *Listado 6*, comenzamos a configurar la arquitectura *cluster* con *Corosync* y *Pacemaker*. Estas configuraciones las aplicaremos sobre todos los nodos del sistema, utilizando el usuario remoto *vagrant* con permisos de administrador.

La jugada comienza nuevamente desde la sección *tasks* copiando la clave de autenticación de *Corosync* (descargada en la jugada anterior) al resto de nodos de la arquitectura. A continuación, se modifica el contenido del fichero de configuración */etc/corosync/corosync.conf* desde una plantilla en *Jinja2* con el módulo *template* de *Ansible* (ver *Listado 9*).

En esta jugada, podemos ver la gran utilidad de *Jinja2* para crear ficheros de configuración "a medida", en función del número de nodos que deseamos configurar empleando bucles en el *.j2*.

Posteriormente, se modifican los ficheros de configuración */etc/corosync/service.d/pcmk* (ver *Listado 10*) y */etc/default/corosync* con los módulos *template* y *lineinfile*. Este último, buscará aquellas líneas que comiencen por *"START="* y las reemplazará poniendo *"START=yes"*. Hecho esto, se reiniciarán los servicios *Corosync* y *Pacemaker* para aplicar los cambios en las configuraciones y, por último, esperaremos con el módulo *wait_for* durante un minuto para que todo se haya reiniciado correctamente, antes de continuar con la última jugada del *playbook*.

3.4.4 Play 4: Configurar Pacemaker

En la cuarta y última jugada del *playbook*, que podemos ver en el *Listado 7*, terminamos el aprovisionamiento del clúster configurando el servicio *Pacemaker* a través de sus comandos.

Como podemos ver, las tareas de este último *play* se agrupan dentro de la sección *tasks*, siendo aplicadas sobre el *nodo1* y haciendo uso como siempre del usuario vagrant con permisos de superusuario.

Nótese el gran uso de los módulos *command* de *Ansible* para aplicar las configuraciones de *Pacemaker*, mediante el comando *crm*. Esto es debido a que, después de buscar e investigar por Internet, no existen módulos de *Ansible* para aplicar estas configuraciones con dicho *software*.

También destacamos el uso de las tareas que hacen uso de los módulos *file*, *template* y *script* para configurar un *script* desde una plantilla en *Jinja2* llamada *crm_configure_fence.j2* (ver *Listado 11*), convertida en *script* en local dentro de una carpeta */script* que crearemos dentro de nuestro proyecto en la máquina anfitrión, que finalmente será ejecutada sobre el *nodo1*.

Finalmente se aplicarán los cambios y levantará la configuración mediante *crm configure commit* y *crm configure up*.

3.5 Creación de las máquinas virtuales del clúster

Explicados todos los *playbook* y el *Vagrantfile*, continuaremos con la ejecución, utilizando el siguiente comando para comenzar con la creación y aprovisionamiento de las máquinas virtuales:

```
$ vagrant up
```

Listado 2. Comando para levantar las máquinas del Vagrantfile

En el comando del *Listado 2*, hacemos uso de la herramienta *Vagrant* para creadas las máquinas especificadas en el fichero *Vagrantfile*. Una vez levantadas, se prosigue de manera automática con el aprovisionamiento de cada una de ellas gracias a los *playbooks* desarrollados en *Ansible*.

Con esto, y si todo ha ido bien, habremos puesto en marcha nuestro entorno de clúster de alta disponibilidad con un total de tres nodos, siendo uno de ellos el nodo maestro y los otros dos nodos esclavos.

4. PRUEBAS DEL CLÚSTER

Una vez hemos levantado nuestros nodos del clúster, es momento de comprobar que todo está creado y configurado perfectamente de acuerdo con nuestras necesidades.

Nota: En los comandos que aparezca *nodoX*, sustituir por un nodo de nuestra máquina, ajustándose a cada caso. Todos los comandos deben ejecutarse dentro de la interfaz *root*. Si el comando va precedido de *$* se ejecutan en la *shell* directamente, si por le contrario van precedidos de *#*, quiere indicar que es un comando el cual se ejecuta dentro de *crm* y sus subcomandos.

Para ello accedemos a un nodo con el comando:

```
$ vagrant ssh nodoX
```

Una vez que nos encontramos dentro de este nodo, lo que deberemos hacer es escribir el siguiente comando:

```
$ sudo corosync-cmapctl | grep members
```

Nos aparecerán los nodos que forman el clúster.

De igual manera, repetiremos estos pasos previos, pero para los nodos, *nodo2* y *nodo3*.

A continuación, realizaremos unas pruebas para comprobar la alta disponibilidad del clúster, pero antes debemos comprobar que está todo en orden, para ello ejecutamos:

```
$ sudo crm cib cibstatus simulate
```

Migrar recursos:

Accedemos a la interfaz *root*:

```
$ sudo su
```

Accedemos al gestor del clúster nos dirigimos a los recursos:

```
$ crm resource
```

Comprobamos dónde está ejecutándose el recurso *IP-nginx*:

```
# status IP-nginx
```

Migramos el recurso *IP-nginx* a otro nodo:

```
# migrate IP-nginx
```

Volvemos a comprobar el estado y vemos como el nodo de ejecución ha cambiado:

```
# status IP-nginx
```

Para ver donde están ejecutándose los recursos debemos de dirigirnos a:

```
$ crm configure show
```

Como podemos ver se ha generado una restricción en los nodos desde los que se ha migrado el recurso, para lo cual deberemos ejecutar el siguiente comando para eliminar dicha restricción (en este caso es *nodo1*, si fuera otro nodo cambiar la terminación del comando):

```
$ crm configure delete cli-ban-IP-nginx-on-nodoX
```

Reiniciar recursos

Para parar un recurso, comprobar que está parado y finalmente volver a levantarlo debemos ejecutar:

```
$ crm resource stop IP-nginx
$ crm resource show
$ crm resource start IP-nginx
```

Simular la caida del servidor Nginx

Para simular la caída de dicho servidor lo primero que debemos hacer es localizar en que nodo se está ejecutando *Nginx*, para ello lo localizaremos con el siguiente comando:

```
$ crm status
```

Una vez que sabemos en qué nodo se está ejecutando procederemos a acceder a dicho nodo en caso de no estar en él, mediante *ssh*:

```
$ ssh vagrant@nodoX
```

Ejecutamos el siguiente comando:

```
$ sudo killall -9 nginx
```

Comprobamos el estado del proceso:

```
$ pgrep -a nginx
```

Si nos esperamos el servicio se vuelve a levantar automáticamente, ejecutando de nuevo el comando anterior y viendo que ya si que nos devuelve resultado.

Si volvemos a tirar el servicio, el clúster decide migrar el recurso a otro nodo y levantarlo en dicho nodo nuevo, por lo cual ya no nos aparecerá nada de comprobamos el estado del proceso.

Para comprobar dicha migración debemos ejecutar el siguiente comando:

```
$ crm_mon -rfn1
$ crm status inactive failcounts bynode
```

Como vemos, hay un parámetro llamado *"migration-threshold=2"*, el cual nos indica que tras 2 fallos el servicio se migra a otro nodo.

Para restaurar el daño realizado ejecutamos la siguiente lista de comandos:

```
$ crm resource
# failcount Nginx-rsc show nodoX
# cleanup Nginx-rsc
# failcount Nginx-rsc show nodoX
```

Simular el fallo de un nodo

Comprobamos el estado de los nodos y donde se están ejecutando antes de simular el fallo:

```
$ crm status
```

Como estamos levantando las máquinas sobre un entorno virtual, nos dirigimos a la máquina anfitriona y ejecutamos el siguiente comando para saber el nombre de las máquinas virtuales:

```
$ VBoxManage list vms
```

Una vez que hemos localizado el nombre de la máquina que debe coincidir con el nombre del *status* previo, procederemos a un apagado de dicha máquina (comando1) o la eliminación por completo (comando2):

```
$ VBoxManage controlvm nodoX poweroff soft --type headless
$ VBoxManage controlvm nodoX pause --type headless
```

Nos conectamos a un nodo que si esté disponible y ejecutamos:

```
$ crm status inactive bynode
```

Procedemos por tanto a iniciar la máquina que teníamos apagada (comando1) o pausada (comando2):

```
$ VBoxManage startvm nodoX --type headless
$ VBoxManage controlvm nodoX resume --type headless
```

Mantenimiento

Si deseamos parar un nodo para mantenimiento, lo que debemos hacer es ejecutar el siguiente comando:

```
$ crm node standby nodoX
# node show
```

Para ponerlo de nuevo en funcionamiento:

```
$ crm node online nodoX
```

Para parar solo un servicio sin que salte ningún fallo se debe ejecutar:

```
$ crm resource unmanage Nginx-rsc
```

Una vez que lo haya solucionamos lo volvemos a monitorizar con:

```
$ crm resource manage Nginx-rsc
```

5. PROBLEMAS ENCONTRADOS

Antes de dar nuestras conclusiones sobre el trabajo realizado, comentaremos por encima algunos de los problemas a los que nos hemos enfrentado durante el proyecto.

Por un lado, tuvimos que decantarnos por una máquina *Ubuntu* "antigua" (14.04), para evitar problemas que experimentamos en versiones más recientes por temas de dependencias, etc.

Aunque finalmente pudimos realizar el proceso de aprovisionamiento al completo, hemos tenido que investigar algunos problemas experimentados al utilizar los comandos de *crm*, ya que la documentación de esta tecnología es muy escasa y apenas hay foros de resolución de problemas.

Por otro lado, uno de los problemas que más dolores de cabeza nos dio fue la adaptación del problema para utilizar un número N de nodos. Para ello, tuvimos que adaptar nuestro fichero *Vagrantfile* para ejecutar el *playbook* mixto pasándole una variable de *Ansible*, posteriormente recogida en una plantilla en *Jinja2* para la correcta configuración de los ficheros *corosync.conf* de cada nodo.

6. CONCLUSIONES

Para concluir este trabajo, daremos nuestra opinión sobre las experiencias que hemos tenido mientras lo hemos realizado, así como algunos puntos fuertes o débiles que queremos destacar.

Puntos fuertes:

Se ha avanzado en el conocimiento de la creación y aprovisionamiento de máquinas virtuales en un entorno de infraestructura como código, lo cual permite a otros compañeros continuar con la infraestructura descrita y seguir aportando valor a la misma.
Hemos tenido que desarrollar sistemas para transmitir información entre los nodos, e incluso entre el Vagrantfile y el propio aprovisionador Ansible, para indicarle el número de nodos a crear y que resulte satisfactoria dicha escalabilidad.

Puntos débiles:

Al crear y aprovisionar un número de máquinas virtuales significativo, si no dispones de un ordenador potente sobre el que crear dichas máquinas virtuales, tardará bastante en dicha tarea, resultando un proyecto pesado en el sentido de ejecución.
El hecho de realizar un trabajo que sea escalable nos ha hecho estar pensando y probando diferentes alternativas para lograr dicha característica.

7. REFERENCIAS

A continuación, se adjuntan las referencias utilizadas para la elaboración del trabajo, así como la presente memoria:

- *How To Create a High Availability Setup with Corosync, Pacemaker, and Floating IPs on Ubuntu 14.04*: https://www.digitalocean.com/community/tutorials/how-to-create-a-high-availability-setup-with-corosync-pacemaker-and-floating-ips-on-ubuntu-14-04
- *High Availability using Corosync + Pacemaker on Ubuntu 16.04*: https://medium.com/@yenthanh/high-availability-using-corosync-pacemaker-on-ubuntu-16-04-bdebc6183fc5
- *Explicación del fichero de configuración de Corosync*: https://linux.die.net/man/5/corosync.conf
- *Documentación oficial de Vagrant, empleada para la creación del Vagrantfile, el cual se encargará de gestionar la creación y ordenes necesarias para la creación de las máquinas virtuales*: https://www.vagrantup.com/docs/
- *Documentación oficial de Ansible, empleada para el aprovisionamiento de las máquinas*: https://docs.ansible.com/

ANEXO A. CÓDIGO DEL FICHERO VAGRANTFILE

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  # Número de nodos que tendrá el cluster
  N_Machines = 3 # N Nodos >= 1 && <=9
  (1..N_Machines).each do |machine_id|
    if machine_id == 1 #Master
      config.vm.define "nodo#{machine_id}", primary: true, autostart: true do |node|
        # Provider
        node.vm.provider "virtualbox" do |vb|
          vb.memory = "1024"
          vb.cpus = 2
          vb.name = "nodo#{machine_id}"
          vb.gui = false
        end
        # Image
        node.vm.box = "ubuntu/trusty64" # Ubuntu 14.04 LTS
        node.vm.boot_timeout = 600
        # Network
        node.vm.network "private_network", ip: "192.168.128.10#{machine_id}"
        # Sync folder
        node.vm.synced_folder "./shared_folders/master_shared_folder",
"/home/vagrant/shared_file" # Master shared folder
        # Provision
        node.vm.provision "ansible" do |ansible|
            ansible.verbose = "v"
            ansible.playbook = "master_Ansible.yml"  # Posibilidad de añadir un
playbook distinto al nodo maestro
            ansible.host_key_checking = "false"
            ansible.limit = "nodo#{machine_id}"
          end
      end
    else  # Slave
      config.vm.define "nodo#{machine_id}",primary: false, autostart: true do |node|
        # Provider
        node.vm.provider "virtualbox" do |vb|
          vb.memory = "1024"
          vb.cpus = 1
          vb.name = "nodo#{machine_id}"
          vb.gui = false
        end
        # Image
        node.vm.box = "ubuntu/trusty64" # Ubuntu 14.04 LTS
        node.vm.boot_timeout = 600
        # Network
        node.vm.network "private_network", ip: "192.168.128.10#{machine_id}"
        # Sync folder
        node.vm.synced_folder "./shared_folders/slave_shared_folder",
"/home/vagrant/shared_file" # Slave shared folder
        # Provision all slaves at the same time
        if machine_id == N_Machines
          node.vm.provision "ansible" do |ansible|
            ansible.verbose = "v"
            ansible.playbook = "slave_Ansible.yml"  # Playbook propio de un esclavo
            ansible.host_key_checking = "false"
            ansible.limit = "all:!nodo1"
          end
          # Aprovisionamiento con dependencias entre nodos máster y esclavos
```

```
            node.vm.provision "ansible" do |ansible|
              ansible.verbose = "v"
              ansible.playbook = "mixed_Ansible.yml"  # Playbook con jugadas mixtas
              # Lanzamos el playbook sobre n nodos del cluster
              ansible.extra_vars = {
                CLUSTER_NODES_NUMBER: "#{N_Machines}"
              }
              ansible.host_key_checking = "false"
              ansible.limit = "all"
            end
          end
        end
      end
    end
  end
```

Listado 3. Código del fichero Vagrantfile

ANEXO B. CÓDIGO DEL PLAYBOOK DE APROVISIONAMIENTO DE ANSIBLE

```
---
### PLAY 1 ###
- name: Aprovisionar nodos con paquetes y configuraciones básicas
  hosts: all
  remote_user: vagrant
  become: yes
  ### Tareas de configuración base ###
  pre_tasks:
  - name: Configurar fichero hosts
    template:
      src: templates/hosts.j2
      dest: /etc/hosts

  - name: Configurar fichero sshd_config
    lineinfile:
      path: /etc/ssh/sshd_config
      regexp: '^PermitRootLogin'
      line: 'PermitRootLogin yes'

  - name: Generar claves
    openssh_keypair:
      path: /home/vagrant/.ssh/id_rsa
      owner: vagrant
      group: vagrant

  - name: Descargar claves públicas de los nodos
    fetch:
      src: /home/vagrant/.ssh/id_rsa.pub
      dest: "keys/{{ ansible_facts['eth1']['ipv4']['address'] }}-id_rsa.pub"
      flat: yes

  - name: Copiar claves SSH a los nodos
    authorized_key:
      user: vagrant
      path: /home/vagrant/.ssh/authorized_keys
      state: present
      key: "{{ lookup('file', item) }}"
    with_fileglob:
      - "keys/*.pub"

  - name: Reiniciar el servicio ssh
    service:
      name: ssh
      state: restarted

  ### Instalación de paquetes y configuración de reglas del firewall ###
  tasks:
  - name: Instalar paquetes necesarios
    apt:
      update_cache: yes
      force_apt_get: yes
      name: "{{ item }}"
      state: latest
    loop:
      - nginx
      - libqb0
      - fence-agents
      - pacemaker
      - ntp

  - name: Configurar reglas de entrada del firewall con iptable
    iptables:
```

```
        chain: INPUT
        in_interface: eth1
        protocol: udp
        destination_port: "{{ item }}"
        ctstate: NEW,ESTABLISHED
        jump: ACCEPT
      loop:
        - "5404"
        - "5405"
        - "5406"

    - name: Configurar reglas de salida del firewall con iptable
      iptables:
        chain: OUTPUT
        out_interface: eth1
        protocol: udp
        source_port: "{{ item }}"
        ctstate: ESTABLISHED
        jump: ACCEPT
      loop:
        - "5404"
        - "5405"
        - "5406"
```

Listado 4. Código de la primera jugada del playbook provision.yml

```
### PLAY 2 ###
- name: Configurar clave de autenticación de Corosync en el nodo 1
  hosts: nodo1
  remote_user: vagrant
  become: yes
  tasks:
  # Paquete haveged necesario para generar la clave de autenticación
  - name: Instalar paquete haveged
    apt:
      force_apt_get: yes
      name: haveged

  - name: Generar la clave de autenticación corosync
    command: corosync-keygen

  # Tras generar la clave, no necesitaremos el paquete haveged
  - name: Desinstalar paquete haveged
    apt:
      force_apt_get: yes
      autoclean: yes
      name: haveged
      state: absent

  - name: Descargar authkey del nodo1 en local
    fetch:
      src: /etc/corosync/authkey
      dest: templates/authkey
      flat: yes
```

Listado 5. Código de la segunda jugada del playbook provision.yml

```
### PLAY 3 ###
- name: Configurar cluster Corosync y Pacemaker
  hosts: all
  remote_user: vagrant
  become: yes
  ### Configurar cluster Corosync ###
  tasks:
  - name: Copiamos authkey de local al servidor secundario en /tmp
    copy:
      src: templates/authkey
      dest: /etc/corosync/
      owner: root
      mode: 400

  - name: Modificar el fichero de configuración de corosync desde plantilla .j2
    template:
      src: templates/corosync.j2
      dest: /etc/corosync/corosync.conf

  - name: Crear el fichero pcmk desde plantilla
    template:
      src: templates/pacemaker.conf
      dest: /etc/corosync/service.d/pcmk

  - name: Cambiar linea de configuración para iniciar corosync
    lineinfile:
      path: /etc/default/corosync
      regexp: '^START='
      line: 'START=yes'

  - name: Reiniciar el servicio corosync
    service:
      name: corosync
      state: restarted

  - name: Ejecutar pacemaker desde el inicio
    command: update-rc.d pacemaker defaults 20 01

  - name: Iniciar el servicio pacemaker
    service:
      name: pacemaker
      state: restarted

  # Esperamos 1 minuto a que se haya iniciado bien el servicio pacemaker
  - name: Sleep 1 minuto
    wait_for:
      timeout: 60
    delegate_to: localhost
    become: no
```

Listado 6. Código de la tercera jugada del playbook provision.yml

```
### PLAY 4 ###
- name: Configurar Pacemaker
  hosts: nodo1
  remote_user: vagrant
  become: yes
  ### Configurar cluster Pacemaker ###
  tasks:
  - name: Configurar crm modo stonith deshabilitado
    command: "crm configure property stonith-enabled=no"

  - name: Configurar crm ignorar política no quorum
    command: "crm configure property no-quorum-policy=ignore"

  - name: Configurar crm stickiness de recursos
    command: "crm configure property default-resource-stickiness=100"

  - name: Configurar recurso IP-nginx
    command: 'crm -F configure primitive IP-nginx ocf:heartbeat:IPaddr2 params ip="192.168.128.110" nic="eth1" cidr_netmask="24" meta migration-threshold=2 op monitor interval=20 timeout=60 on-fail=restart'

  - name: Configurar recurso Nginx-src
    command: "crm -F configure primitive Nginx-rsc ocf:heartbeat:nginx meta migration-threshold=2 op monitor interval=20 timeout=60 on-fail=restart"

  - name: Asegurar que ambos recursos siempre se ejecutan en el mismo nodo
    command: 'crm -F configure colocation lb-loc inf: IP-nginx Nginx-rsc'

  - name: Asegurar que la dirección IP está disponible
    command: "crm -F configure order lb-ord inf: IP-nginx Nginx-rsc"

  ### Fencing agents
  - name: Test fencing agents
    command: "crm configure cib new fencing"

  - name: Habilitar modo Stonith
    command: "crm -F configure property stonith-enabled=yes"

  - name: Crear carpeta local para scripts
    file:
      path: scripts/
      state: directory
      mode: '0755'
    delegate_to: localhost
    become: no

  - name: Generar script crm configurar fences desde plantilla j2
    template:
      src: templates/crm_configure_fence.j2
      dest: scripts/crm_configure_fence.sh
    delegate_to: localhost
    become: no

  - name: Ejecutar script generado
    script: scripts/crm_configure_fence.sh

  - name: Hacemos commit para aplicar los cambios
    command: "crm -F configure commit"

  - name: Levantamos las nuevas configuraciones aplicadas
    command: "crm configure up"
```

Listado 7. Código de la cuarta jugada del playbook provision.yml

ANEXO C. CÓDIGO DE LAS PLANTILLAS JINJA2 Y .CONF UTILIZADAS

```
{# Configuramos tantos nodos como tengamos en la variable CLUSTER_NODES_NUMBER #}
{% for i in range(1,( CLUSTER_NODES_NUMBER | int ) + 1 ) %}
192.168.128.10{{ i }} nodo{{ i }}
{% endfor %}
```

Listado 8. Código del fichero hosts.j2

```
totem {
  version: 2
  cluster_name: lbcluster
  transport: udpu
  interface {
    ringnumber: 0
    bindnetaddr: 192.168.128.0
    broadcast: yes
    mcastport: 5405
  }
}

quorum {
  provider: corosync_votequorum
  two_node: 1
}

nodelist {
{# Configuramos tantos nodos como tengamos en la variable CLUSTER_NODES_NUMBER #}
{% for i in range(1,( CLUSTER_NODES_NUMBER | int ) + 1 ) %}
  node{
    ring0_addr: 192.168.128.10{{ i }}
    name: nodo{{ i }}
    nodeid: {{ i }}
  }

{% endfor %}
}

logging {
  to_logfile: yes
  logfile: /var/log/corosync/corosync.log
  to_syslog: yes
  timestamp: on
}
```

Listado 9. Contenido del fichero corosync.j2

```
service {
  name: pacemaker
  ver: 1
}
```

Listado 10. Contenido del fichero pacemaker.conf

```
1  sudo crm -F configure primitive fence_nodo1 stonith:fence_virsh params
   ipaddr=192.168.128.101 port=22 action=off login=vagrant passwd=vagrant op monitor
   interval=60s
2  {# Configuramos tantos nodos como tengamos en la variable CLUSTER_NODES_NUMBER #}
3  {% for i in range(2,( CLUSTER_NODES_NUMBER | int ) + 1 ) %}
4  sudo crm -F configure primitive fence_nodo{{ i }} stonith:fence_virsh params
   ipaddr=192.168.128.101 port=22 action=off login=vagrant passwd=vagrant delay=15 op
   monitor interval=60s
5  {% endfor %}
6  {% for i in range(1,( CLUSTER_NODES_NUMBER | int ) + 1 ) %}
7  sudo crm -F configure location l_fence_nodo{{ i }} fence_nodo{{ i }} -inf: nodo{{ i }}
8  {% endfor %}
```

Listado 11. Contenido del fichero crm_configure_fence.j2

ANEXO D. TESTING CLÚSTER

Para probar dicho clúster se ha decidido realizarlo sobre 3 nodos, ya que realizarlo sobre 1 sería absurdo, por lo que 3 puede ser un buen número para comprobar que todo funciona bien.

Comprobamos que la configuración del clúster esté correcta:

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo corosync-cmapctl | grep members
runtime.totem.pg.mrp.srp.members.1.config_version (u64) = 0
runtime.totem.pg.mrp.srp.members.1.ip (str) = r(0) ip(192.168.128.101)
runtime.totem.pg.mrp.srp.members.1.join_count (u32) = 1
runtime.totem.pg.mrp.srp.members.1.status (str) = joined
runtime.totem.pg.mrp.srp.members.2.config_version (u64) = 0
runtime.totem.pg.mrp.srp.members.2.ip (str) = r(0) ip(192.168.128.102)
runtime.totem.pg.mrp.srp.members.2.join_count (u32) = 1
runtime.totem.pg.mrp.srp.members.2.status (str) = joined
runtime.totem.pg.mrp.srp.members.3.config_version (u64) = 0
runtime.totem.pg.mrp.srp.members.3.ip (str) = r(0) ip(192.168.128.103)
runtime.totem.pg.mrp.srp.members.3.join_count (u32) = 1
runtime.totem.pg.mrp.srp.members.3.status (str) = joined
```

Comprobamos los recursos del clúster, incluidas las *fences*, que serán las encargadas de monitorizar los nodos.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo crm cib cibstatus simulate

Current cluster status:
Online: [ nodo1 nodo2 nodo3 ]

 IP-nginx       (ocf::heartbeat:IPaddr2):       Started nodo1
 Nginx-rsc      (ocf::heartbeat:nginx): Started nodo2
 fence_nodo1    (stonith:fence_virsh):  Started nodo3
 fence_nodo2    (stonith:fence_virsh):  Started nodo1
 fence_nodo3    (stonith:fence_virsh):  Started nodo2

Transition Summary:

Executing cluster transition:

Revised cluster status:
Online: [ nodo1 nodo2 nodo3 ]

 IP-nginx       (ocf::heartbeat:IPaddr2):       Started nodo1
 Nginx-rsc      (ocf::heartbeat:nginx): Started nodo2
 fence_nodo1    (stonith:fence_virsh):  Started nodo3
 fence_nodo2    (stonith:fence_virsh):  Started nodo1
 fence_nodo3    (stonith:fence_virsh):  Started nodo2
```

Migración de recursos

Como podemos apreciar, partimos de que el recurso *IP-nginx* se encuentra ejecutado sobre el *nodo1*, para realizar una migración y el clúster decide reasignarlo al nodo2.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo su
root@vagrant-ubuntu-trusty-64:/home/vagrant# crm resource
crm(live)resource# status IP-nginx
resource IP-nginx is running on: nodo1
crm(live)resource# migrate IP-nginx
WARNING: Creating rsc_location constraint 'cli-ban-IP-nginx-on-nodo1'
        This will prevent IP-nginx from running on nodo1 until the cor
ith cibadmin
        This will be the case even if nodo1 is the last node in the cl
        This message can be disabled with --quiet
crm(live)resource# status IP-nginx
resource IP-nginx is running on: nodo2
```

En la siguiente imagen vemos la configuración del clúster y una restricción nueva, al haber realizado dicha migración. El significado es que evite que en el *nodo1* se ejecute *IP-nginx*. La restricción es la primera de la categoría *location*.

```
root@vagrant-ubuntu-trusty-64:/home/vagrant# crm configure show
node $id="1" nodo1
node $id="2" nodo2
node $id="3" nodo3
primitive IP-nginx ocf:heartbeat:IPaddr2 \
        params ip="192.168.128.110" nic="eth1" cidr_netmask="24" \
        meta migration-threshold="2" \
        op monitor interval="20" timeout="60" on-fail="restart"
primitive Nginx-rsc ocf:heartbeat:nginx \
        meta migration-threshold="2" \
        op monitor interval="20" timeout="60" on-fail="restart"
primitive fence_nodo1 stonith:fence_virsh \
        params ipaddr="192.168.128.101" port="22" action="off" login="vagrant" passwd="vagrant" \
        op monitor interval="60s"
primitive fence_nodo2 stonith:fence_virsh \
        params ipaddr="192.168.128.101" port="22" action="off" login="vagrant" passwd="vagrant" delay="15" \
        op monitor interval="60s"
primitive fence_nodo3 stonith:fence_virsh \
        params ipaddr="192.168.128.101" port="22" action="off" login="vagrant" passwd="vagrant" delay="15" \
        op monitor interval="60s"
location cli-ban-IP-nginx-on-nodo1 IP-nginx -inf: nodo1
location l_fence_nodo1 fence_nodo1 -inf: nodo1
location l_fence_nodo2 fence_nodo2 -inf: nodo2
location l_fence_nodo3 fence_nodo3 -inf: nodo3
order lb-ord inf: IP-nginx Nginx-rsc
property $id="cib-bootstrap-options" \
        dc-version="1.1.10-42f2063" \
        cluster-infrastructure="corosync" \
        stonith-enabled="yes" \
        no-quorum-policy="ignore" \
        default-resource-stickiness="100"
```

Procedemos a la eliminación de dicha restricción.

```
root@vagrant-ubuntu-trusty-64:/home/vagrant# crm configure delete cli-ban-IP-nginx-on-nodo1
root@vagrant-ubuntu-trusty-64:/home/vagrant# crm configure show
node $id="1" nodo1
node $id="2" nodo2
node $id="3" nodo3
primitive IP-nginx ocf:heartbeat:IPaddr2 \
        params ip="192.168.128.110" nic="eth1" cidr_netmask="24" \
        meta migration-threshold="2" \
        op monitor interval="20" timeout="60" on-fail="restart"
primitive Nginx-rsc ocf:heartbeat:nginx \
        meta migration-threshold="2" \
        op monitor interval="20" timeout="60" on-fail="restart"
primitive fence_nodo1 stonith:fence_virsh \
        params ipaddr="192.168.128.101" port="22" action="off" login="vagrant" passwd="vagrant" \
        op monitor interval="60s"
primitive fence_nodo2 stonith:fence_virsh \
        params ipaddr="192.168.128.101" port="22" action="off" login="vagrant" passwd="vagrant" delay="15" \
        op monitor interval="60s"
primitive fence_nodo3 stonith:fence_virsh \
        params ipaddr="192.168.128.101" port="22" action="off" login="vagrant" passwd="vagrant" delay="15" \
        op monitor interval="60s"
location l_fence_nodo1 fence_nodo1 -inf: nodo1
location l_fence_nodo2 fence_nodo2 -inf: nodo2
location l_fence_nodo3 fence_nodo3 -inf: nodo3
order lb-ord inf: IP-nginx Nginx-rsc
property $id="cib-bootstrap-options" \
        dc-version="1.1.10-42f2063" \
        cluster-infrastructure="corosync" \
        stonith-enabled="yes" \
        no-quorum-policy="ignore" \
        default-resource-stickiness="100"
```

Simulación la caída del servidor Nginx

```
root@vagrant-ubuntu-trusty-64:/home/vagrant# crm status
Last updated: Fri May  8 00:21:37 2020
Last change: Fri May  8 00:20:19 2020 via cibadmin on nodo2
Stack: corosync
Current DC: nodo2 (2) - partition with quorum
Version: 1.1.10-42f2063
3 Nodes configured
5 Resources configured


Online: [ nodo1 nodo2 nodo3 ]

 IP-nginx       (ocf::heartbeat:IPaddr2):       Started nodo2
 Nginx-rsc      (ocf::heartbeat:nginx): Started nodo2
 fence_nodo1    (stonith:fence_virsh):  Started nodo3
 fence_nodo2    (stonith:fence_virsh):  Started nodo1
 fence_nodo3    (stonith:fence_virsh):  Started nodo2
root@vagrant-ubuntu-trusty-64:/home/vagrant# ssh root@nodo2
```

Tiramos el servicio, ejecutamos el comando para saber si lo hemos tirado, comprobamos que no nos devuelve nada y por tanto el servicio ha sido tirado. Esperamos alrededor de 1 minuto hasta que el servicio se levanta solo automáticamente.

```
root@vagrant-ubuntu-trusty-64:~# sudo killall -9 nginx
root@vagrant-ubuntu-trusty-64:~# pgrep -a nginx
root@vagrant-ubuntu-trusty-64:~# pgrep -a nginx
18280 nginx: master process /usr/sbin/nginx -c /etc/nginx/nginx.conf
18281 nginx: worker process
18282 nginx: worker process
18283 nginx: worker process
18284 nginx: worker process
```

Volvemos a tirar el servicio. El sistema al detectar que el servicio se ha caído 2 veces en el mismo nodo, decide migrarlo a otro nodo.

```
root@vagrant-ubuntu-trusty-64:~# sudo killall -9 nginx
root@vagrant-ubuntu-trusty-64:~# crm_mon -rfn1
Last updated: Fri May  8 00:26:23 2020
Last change: Fri May  8 00:20:19 2020 via cibadmin on nodo1
Stack: corosync
Current DC: nodo2 (2) - partition with quorum
Version: 1.1.10-42f2063
3 Nodes configured
5 Resources configured


Node nodo1 (1): online
        Nginx-rsc       (ocf::heartbeat:nginx): Started
        fence_nodo2     (stonith:fence_virsh):  Started
Node nodo2 (2): online
        IP-nginx        (ocf::heartbeat:IPaddr2):       Started
        fence_nodo3     (stonith:fence_virsh):  Started
Node nodo3 (3): online
        fence_nodo1     (stonith:fence_virsh):  Started

Inactive resources:


Migration summary:
* Node nodo2:
   Nginx-rsc: migration-threshold=2 fail-count=2 last-failure='Fri May  8 00:25:48 2020'
* Node nodo3:
* Node nodo1:

Failed actions:
    Nginx-rsc_monitor_20000 (node=nodo2, call=69, rc=7, status=complete, last-rc-change=Fri May  8 00:25:48 2020
, queued=0ms, exec=0ms
): not running
```

Procedemos a comprobar el contador de los fallos que ha registrado el nodo2 y ha reiniciado dicho contador, para que el servicio en un futuro pueda volver a instanciarse en dicho nodo en caso de que fuera necesario.

```
root@vagrant-ubuntu-trusty-64:~# crm resource
crm(live)resource# failcount Nginx-rsc show nodo2
scope=status  name=fail-count-Nginx-rsc value=2
crm(live)resource# cleanup Nginx-rsc
Cleaning up Nginx-rsc on nodo1
Cleaning up Nginx-rsc on nodo2
Cleaning up Nginx-rsc on nodo3
Waiting for 1 replies from the CRMd. OK
crm(live)resource# failcount Nginx-rsc show nodo2
scope=status  name=fail-count-Nginx-rsc value=0
```

Simulación del fallo de un nodo

Estado previo al fallo del nodo

```
root@vagrant-ubuntu-trusty-64:~# crm status
Last updated: Fri May  8 00:29:42 2020
Last change: Fri May  8 00:28:11 2020 via crmd on nodo3
Stack: corosync
Current DC: nodo2 (2) - partition with quorum
Version: 1.1.10-42f2063
3 Nodes configured
5 Resources configured


Online: [ nodo1 nodo2 nodo3 ]

 IP-nginx       (ocf::heartbeat:IPaddr2):       Started nodo2
 Nginx-rsc      (ocf::heartbeat:nginx): Started nodo1
 fence_nodo1    (stonith:fence_virsh):  Started nodo3
 fence_nodo2    (stonith:fence_virsh):  Started nodo1
 fence_nodo3    (stonith:fence_virsh):  Started nodo2
```

Desde la máquina anfitriona decidimos apagar un nodo virtual.

```
(base) martinez@martinez-PC:~/Escritorio/MASTER/Gestion_Infraestructuras_TIC/MIGUEL/2/pgitic_corosync_pacemaker/src$ VBoxManage controlvm nodo1
 poweroff soft --type headless
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
```

Como podemos observar, nos indica el fallo en dicho nodo, ya que no se encuentra disponible y los recursos que había en dicho nodo.

```
root@vagrant-ubuntu-trusty-64:/home/vagrant# crm status inactive bynode
Last updated: Fri May  8 00:33:24 2020
Last change: Fri May  8 00:28:11 2020 via crmd on nodo3
Stack: corosync
Current DC: nodo2 (2) - partition with quorum
Version: 1.1.10-42f2063
3 Nodes configured
5 Resources configured


Node nodo1 (1): UNCLEAN (offline)
        fence_nodo2     (stonith:fence_virsh):  Started
        Nginx-rsc       (ocf::heartbeat:nginx): Started
Node nodo2 (2): online
        IP-nginx        (ocf::heartbeat:IPaddr2):       Started
        fence_nodo3     (stonith:fence_virsh):  Started FAILED
Node nodo3 (3): online
        fence_nodo1     (stonith:fence_virsh):  Started FAILED

Inactive resources:


Failed actions:
    fence_nodo3_monitor_60000 (node=nodo2, call=40, rc=1, status=Error, last-rc-change=Fri May  8 00:31:56 2020
, queued=9669ms, exec=0ms
): unknown error
    fence_nodo1_monitor_60000 (node=nodo3, call=29, rc=1, status=Error, last-rc-change=Fri May  8 00:31:50 2020
, queued=11984ms, exec=0ms
): unknown error
```

Como podemos observar, los servicios del *nodo1* se han migrado al *nodo3*.

```
root@vagrant-ubuntu-trusty-64:/home/vagrant# crm cib cibstatus simulate

Current cluster status:
Node nodo1 (1): UNCLEAN (offline)
Online: [ nodo2 nodo3 ]

 IP-nginx       (ocf::heartbeat:IPaddr2):       Started nodo2
 Nginx-rsc      (ocf::heartbeat:nginx): Started nodo1
 fence_nodo1    (stonith:fence_virsh):  Started nodo3 FAILED
 fence_nodo2    (stonith:fence_virsh):  Started nodo1
 fence_nodo3    (stonith:fence_virsh):  Started nodo2 FAILED

Transition Summary:
 * Move    Nginx-rsc    (Started nodo1 -> nodo3)
 * Recover fence_nodo1  (Started nodo3)
 * Move    fence_nodo2  (Started nodo1 -> nodo3)
 * Recover fence_nodo3  (Started nodo2)

Executing cluster transition:
 * Resource action: fence_nodo1     stop on nodo3
 * Resource action: fence_nodo3     stop on nodo2
 * Fencing nodo1
 * Pseudo action:   stonith_complete
 * Pseudo action:   Nginx-rsc_stop_0
 * Resource action: fence_nodo1     start on nodo3
 * Resource action: fence_nodo1     monitor=60000 on nodo3
 * Pseudo action:   fence_nodo2_stop_0
 * Resource action: fence_nodo3     start on nodo2
 * Resource action: fence_nodo3     monitor=60000 on nodo2
 * Pseudo action:   all_stopped
 * Resource action: Nginx-rsc       start on nodo3
 * Resource action: fence_nodo2     start on nodo3
 * Resource action: Nginx-rsc       monitor=20000 on nodo3
 * Resource action: fence_nodo2     monitor=60000 on nodo3

Revised cluster status:
Online: [ nodo2 nodo3 ]
OFFLINE: [ nodo1 ]

 IP-nginx       (ocf::heartbeat:IPaddr2):       Started nodo2
 Nginx-rsc      (ocf::heartbeat:nginx): Started nodo3
 fence_nodo1    (stonith:fence_virsh):  Started nodo3
 fence_nodo2    (stonith:fence_virsh):  Started nodo3
 fence_nodo3    (stonith:fence_virsh):  Started nodo2
```

Procedemos a levantar de nuevo la máquina que hemos tirado, para que vuelva a formar parte del clúster.

```
(base) martinez@martinez-PC:~/Escritorio/MASTER/Gestion_Infraestructuras_TIC/MIGUEL/2/pgitic_corosync_pacemaker/src$ VBoxManage startvm nodo1
--type headless
Waiting for VM "nodo1" to power on...
VM "nodo1" has been successfully started.
```

Mantenimiento

```
root@vagrant-ubuntu-trusty-64:/home/vagrant# crm node standby nodo3
root@vagrant-ubuntu-trusty-64:/home/vagrant# crm node show
nodo1(1): normal
nodo2(2): normal
nodo3(3): normal
        standby: on
root@vagrant-ubuntu-trusty-64:/home/vagrant# crm node online nodo3
root@vagrant-ubuntu-trusty-64:/home/vagrant# crm node show
nodo1(1): normal
nodo2(2): normal
nodo3(3): normal
        standby: off
```

Para parar la monitorización de un servicio para realizar mantenimiento del mismo:

```
root@vagrant-ubuntu-trusty-64:/home/vagrant# crm resource unmanage Nginx-rsc
root@vagrant-ubuntu-trusty-64:/home/vagrant# crm status
Last updated: Fri May  8 00:43:24 2020
Last change: Fri May  8 00:43:15 2020 via cibadmin on nodo2
Stack: corosync
Current DC: nodo2 (2) - partition with quorum
Version: 1.1.10-42f2063
3 Nodes configured
5 Resources configured


Online: [ nodo1 nodo2 nodo3 ]

 IP-nginx       (ocf::heartbeat:IPaddr2):       Started nodo2
 Nginx-rsc      (ocf::heartbeat:nginx): Started nodo1 (unmanaged)
 fence_nodo1    (stonith:fence_virsh):  Started nodo2
 fence_nodo2    (stonith:fence_virsh):  Started nodo1
 fence_nodo3    (stonith:fence_virsh):  Started nodo2
```

Reactivamos la monitorización:

```
root@vagrant-ubuntu-trusty-64:/home/vagrant# crm resource manage Nginx-rsc
root@vagrant-ubuntu-trusty-64:/home/vagrant# crm status
Last updated: Fri May  8 00:44:00 2020
Last change: Fri May  8 00:43:59 2020 via cibadmin on nodo2
Stack: corosync
Current DC: nodo2 (2) - partition with quorum
Version: 1.1.10-42f2063
3 Nodes configured
5 Resources configured


Online: [ nodo1 nodo2 nodo3 ]

 IP-nginx       (ocf::heartbeat:IPaddr2):       Started nodo2
 Nginx-rsc      (ocf::heartbeat:nginx): Started nodo1
 fence_nodo1    (stonith:fence_virsh):  Started nodo2
 fence_nodo2    (stonith:fence_virsh):  Started nodo1
 fence_nodo3    (stonith:fence_virsh):  Started nodo2
```