UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

ORQUESTADOR DE RECURSOS DE INFRAESTRUCTURA CLOUD

Tesis de Licenciatura
Presentada en la Escuela Técnica Superior de Ingeniería de Telecomunicación de Barcelona
Universidad Politécnica de Cataluña

Daniel Estirado Morales

En cumplimiento parcial de los requisitos para el grado en INGENIERÍA DE SISTEMAS DE TELECOMUNICACIÓN

Asesor: Anna Umbert Juliana

Barcelona, Junio 2019

Abstract

The world of technology is advancing at a great pace and we must move forward at the same pace if we want to keep up. In the case of cloud technology, it must be said that it is certainly part of the present, but still has much to say in the future, so it is important to know more in depth what it consists of and what the differences are with its predecessors.

As I have always thought that the best way to know a new technology is to get involved directly and work with it, I decided to carry out this project, in which one of the main objectives will be to learn the particularities of the cloud world.

Resum

El món de la tecnologia avança a grans passos i nosaltres hem d'avançar al mateix ritme si volem mantenir-nos a l'alçada. En el cas de la tecnologia cloud, cal dir que sens dubte és part del present, però encara té molt a dir en el futur, pel que és important conèixer més en què consisteix i quines són les diferencies amb els seus antecessors.

Com sempre he pensat que la millor manera de conèixer una nova tecnologia és implicar-nos directament i treballar amb ella, vaig decidir realitzar aquest projecte, en el qual un dels objectius principals serà poder aprendre les particularitats del món cloud.

Resumen

El mundo de la tecnología avanza a grandes pasos y nosotros debemos avanzar al mismo ritmo si queremos mantenernos a la altura. En el caso de la tecnología *cloud*, hay que decir que sin duda es parte del presente, pero aún tiene mucho que decir en el futuro, por lo que es importante conocer más en profundidad en que consiste y cuáles son las diferencias con sus antecesores.

Como siempre he pensado que la mejor manera de conocer una nueva tecnología es implicarnos directamente y trabajar con ella, decidí realizar este proyecto, en el cual uno de los objetivos principales será poder aprender las particularidades del mundo *cloud*.

Agradecimientos

En primer lugar me gustaría agradecer a las personas que han hecho posible que pueda participar en este proyecto. A mi tutora de proyecto en la universidad Anna Umbert y a mis compañeros en la empresa Isidro Ingles, Juanma Rodríguez, Alfonso Maria Franch y Francisco Javier Gallardo.

También a los compañeros que he conocido en la facultad durante estos años. Con ellos he pasado momentos de todo tipo, pero es en los difíciles donde realmente te das cuenta que son personas excepcionales y amigos para toda la vida.

Por último y no menos importante, agradecer el apoyo de mi familia y mi pareja, quienes no han dudado de mí en ningún momento. Sin ellos no podría haber llegado hasta aquí.

Historial de revisiones y registro de aprobación

Revisión	Fecha	Propósito
0	02/05/2019	Creación del documento
1	15/05/2019	Versionado de documento
2	22/05/2019	Versionado de documento

Nombre	E-mail
Daniel Estirado Morales	esti_dani@hotmail.com
Anna Umbert Juliana	annau@tsc.upc.edu

Escrito por:		Revisado y aprobado por:	
Fecha 02/05/2019			
Nombre	Daniel Estirado Morales	Nombre	Anna Umbert Juliana
Posición	Autor del proyecto	Posición	Supervisor del proyecto

Índice

Lista de figuras

Lista de tablas:

1. Introducción

Este proyecto nace de la oportunidad que se me ofreció desde la empresa *Everis*, donde estoy realizando prácticas desde hace unos meses. Les comenté que tenía que realizar el proyecto final de grado y que me gustaría que fuera participando en un proyecto real dentro de la empresa. Les pareció bueno idea y me ofrecieron este proyecto.

Se trata de un proyecto dentro del área de infraestructura de una aplicación de tesorería para banca que se creó y que está en funcionamiento actualmente en la empresa. En concreto se trabaja con infraestructura *cloud* y el proyecto consiste en crear un software que permita conectar con los elementos que forman la infraestructura e incluso llegar a interactuar con ellos.

Esto permitiría a los operadores que trabajan con estos elementos y controlan el correcto funcionamiento de la infraestructura el llevar a cabo parte de su trabajo de manera más eficiente.

El punto de partida será conocer el contexto en el que se trabaja. Por lo que es necesaria una gran labor de documentación sobre las tecnologías y herramientas que posteriormente se utilizarán.

Un pilar fundamental del proyecto es el analizar diversas herramientas con las que interactuar con la infraestructura, para poder decidir cuál es la óptima. Para ello, muchas de las pruebas a realizar se efectúan en paralelo con varias de estas herramientas.

Una vez planifiquemos cuales son las funcionalidades que queremos probar, trabajar en un entorno de test adecuado será fundamental, ya que se trata de parte de una aplicación real y actualmente en funcionamiento, con la que muchos equipos de profesionales trabajan a diario para su correcto funcionamiento.

1.1. Objetivos

Los objetivos planteados para este proyecto son los siguientes:

- Aprender el contexto de infraestructura *cloud*
- Analizar y utilizar nuevas herramientas
- Conocer el funcionamiento de un proyecto real dentro del mundo laboral
- Definir y diseñar las funcionalidades o interacciones necesarias
- Llegar a conectar con los recursos de la infraestructura
- Interactuar con los recursos de la infraestructura según lo definido
- Diseñar y trabajar en un entorno de test dentro de una aplicación real

Tenerlos presentes durante todo el transcurso del proyecto es vital, ya que son la base sobre la que trabajar. También es importante que una vez finalice el proyecto, se analicen estos objetivos y ver hasta donde hemos podido llegar.

1.2. Plan de trabajo

Para la elaboración de un correcto plan de trabajo, se ha creado la siguiente tabla, en la cual se pueden ver los diferentes bloques de trabajo y la planificación que se ha realizado de ellos en el tiempo. Es necesario seguir un esquema ordenado de trabajo, por lo que se ha seguido un sistema de dependencias entre bloques.

ID	Nombre	Inicia	Terminado	Predecesor
1	Información previa (Cloud)	15/02/2019	22/02/2019	
2	Información previa (OCI)	22/02/2019	01/03/2019	1
3	Información previa (*Terraform*)	15/02/2019	01/03/2019	
4	Análisis Python SDK	01/03/2019	29/03/2019	3
5	Análisis *Ansible*	01/03/2019	29/03/2019	3
6	Planificación	01/03/2019	15/03/2019	5
7	Diseño	15/03/2019	12/04/2019	6
8	Implementación	12/04/2019	13/05/2019	7
9	Test	26/04/2019	31/05/2019	
10	Propuesta de proyecto	25/02/2019	04/03/2019	
11	Revisión crítica de proyecto	25/04/2019	02/05/2019	
12	Memoria final de proyecto	06/05/2019	25/06/2019	

Tabla 1: Lista de bloques de tareas con su planificación.

Por otro lado se ha hecho un diagrama de Gantt, una herramienta con la que mostrar de manera visual una previsión de las horas dedicadas a cada bloque de trabajo definido.

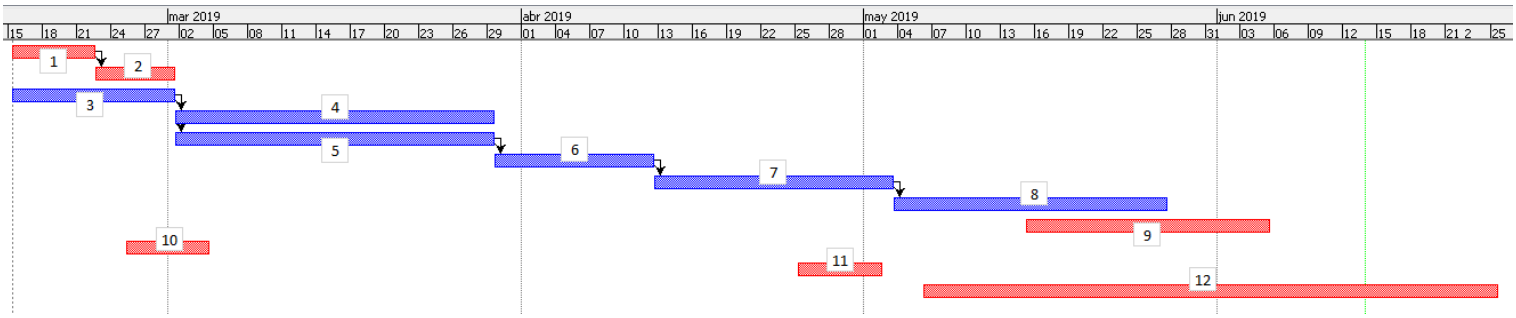


Figura 1-1: Diagrama de Gantt del proyecto

1.3. Desviaciones del plan inicial

Existen diversos riesgos que pueden modificar el plan de trabajo inicialmente propuesto. Sabíamos de la posibilidad de que esto ocurriera, ya que parte del proyecto consiste en analizar a medida que avanzamos, qué camino seguir y que herramientas utilizar.

En este caso el plan de trabajo inicial se ha visto afectado por la necesidad de cambiar la herramienta propuesta (*Terraform*) por otras que pudieran satisfacer las necesidades. Debido a esta decisión, ha sido necesario adaptarse y seguir con el análisis de herramientas. Esto nos ha llevado a la creación de los dos paquetes de trabajo llamados "Análisis *Python SDK*" y "Análisis *Ansible*" vistos en el punto anterior, los cuales no existían en el plan inicial.

2. Estado de la tecnología

2.1. Infraestructura *cloud*

Para empezar, es necesario que conozcamos el contexto tecnológico sobre el que nos vamos a basar. Por una parte tenemos el concepto de infraestructura tecnológica y por el otro la computación a través de la nube. Es con la combinación de estos dos conceptos tecnológicos que vamos a trabajar.

Para explicar el concepto de infraestructura *cloud*, es necesario tener una idea clara de que es la infraestructura tecnológica de una empresa. Podemos definir este concepto como el conjunto de elementos necesarios para ofrecer un servicio. Entre estos se encuentran el hardware y el software además de todos los servicios necesarios para la monitorización, la gestión interna o la seguridad entre otros.

El concepto de *cloud* se refiere a servidores a los cuales un usuario puede acceder a través de internet en cualquier momento, y que pueden resolver peticiones de información o servicio. Estos servidores se encuentran localizados en cualquier región del mundo.

La computación en la nube es un modelo de prestación de servicio, que ofrece al usuario la capacidad de acceder a diferentes soluciones e información de manera ágil y adaptativa. El trabajar con esta solución aporta ventajas tanto para el proveedor, que puede ofrecer más servicios además de tener la capacidad de adaptarlos de manera más ágil, como para el usuario, que disfruta de transparencia e inmediatez.

Figura 2-1: Descripción gráfica de conexión con la nube mediante diferentes dispositivos.

Si nos basamos en estos dos conceptos, la infraestructura cloud no es más que la virtualización de los recursos que forman la infraestructura y que se ofrece como servicio para ser consumida a través de internet.

Los componentes que la forman son:

- Front-end: Son los dispositivos físicos a los que el usuario accede. Podría ser un PC, una Tablet o un Smartphone.
- Back-end: En esta agrupación entraría toda la infraestructura física y las aplicaciones de software a las que se accede a través de internet.

Los usuarios consumidores, acceden a través de una consola de gestión en navegador o mediante una API y se benefician de la transparencia y la eficiencia que les otorga este tipo de modelo.

2.1.1. Ventajas

Sin duda este modelo de infraestructura basado en la computación cloud, proporciona una serie de ventajas respecto al modelo clásico, por eso muchas empresas tienden ya a trabajar con él. Algunas de ellas son las siguientes:

- **Facilidad de acceso:** El poder acceder a todos los documentos y la información desde cualquier punto teniendo acceso a internet, proporciona una flexibilidad sin precedentes. A nivel empresarial aún más, ya que puedes organizar tu actividad sin depender de una oficina.
- **Disminución de costes:** La empresa prescinde de una infraestructura física y de todos los gastos en mantenimiento y personal que esto conlleva. Además este modelo utiliza el *Pay as you go,* concepto con el que solo se paga por el tiempo de uso realizado.
- **Software actualizado:** El usuario siempre podrá tener a acceso a tecnología actualizada.
- **Capacidad de almacenamiento:** El almacenamiento en la nube es “ilimitado”, por lo que el usuario no se ve obligado a realizar ampliaciones en su propio equipo.

2.1.2. Inconvenientes

Pese a ser una tecnología cada día más presente, no podemos pasar por alto las desventajas que se derivan de su implantación:

- **Necesidad de internet:** El poder acceder mediante internet, a priori es una ventaja, pero frente a una caída del servicio, nos encontraríamos con la imposibilidad de acceder, por lo que es un punto importante a tener en cuenta.
- **Dependencia del proveedor:** Las actualizaciones, política de backups u otros aspectos, siempre dependerán del proveedor.
- **Privacidad:** Si no se toman las medidas preventivas necesarias, siempre existe un riesgo de privacidad de toda la información que enviamos.

2.1.3. Proveedores

Para una empresa que se decide a trabajar con infraestructura cloud, es importante saber que existen diversos proveedores de servicios cloud y que cada uno de ellos tendrá sus particularidades. Si vamos a trabajar con una de ellos, es imprescindible conocerlo en profundidad.

Algunos de los más relevantes proveedores actuales de cloud son los siguientes:

- Microsoft Azure
- Amazon web services
- IBM
- Oracle
- SAP

Figura 2-2: Logos de tres de los proveedores Cloud mencionados.

En este proyecto, la infraestructura que se utiliza es del proveedor Oracle, por lo debo conocer la infraestructura cloud de Oracle (OCI), para conocer que herramientas pueden utilizarse y que particularidades tienen los recursos cloud entre otras cosas.

2.2. OCI - Oracle Cloud Infraestructure

La infraestructura *cloud* de Oracle es el conjunto de servicios en la nube que permiten el crear, gestionar y ejecutar una serie de aplicaciones y servicios. Permite trabajar con diferentes elementos (como instancias de hardware físico) además de un ofrecer un almacenamiento en una red virtual a la que acceder vía internet desde una red local.

Estos son algunos de los servicios que ofrece son:

- **Compute:** Crear y gestionar instancias de una máquina virtual de manera casi inmediata. Esto permite trabajar tal como lo haríamos desde un centro de datos local, pero sin la necesidad de utilizar servidores físicos.
- **Auditoria:** Permite visibilidad de los actividades realizada en los recursos de la infraestructura cloud.
- **Migración:** Posibilidad de transferir grandes volúmenes de datos a la infraestructura cloud.
- **Networking:** Mediante el uso de redes virtuales en la nube (VCN), gestionar el acceso a sus recursos, permitiendo el tráfico privado y público desde internet.

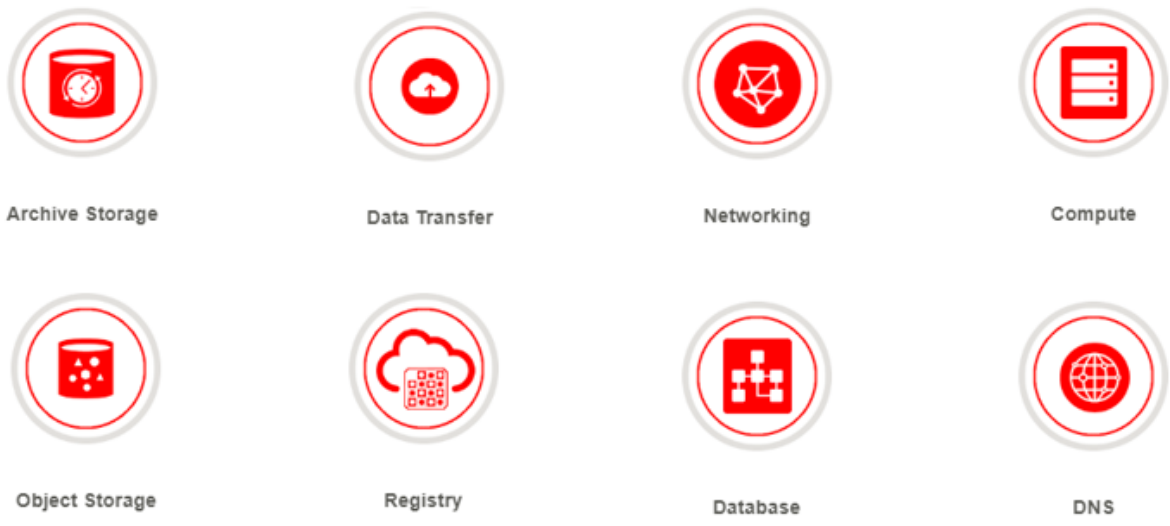


Figura 2-3: Ejemplos de los diferentes servicios que ofrece la Infraestructura Cloud de Oracle.

2.2.1. Herramientas de gestión del OCI

Para la gestión de todo la infraestructura, existen diversas herramientas que podemos utilizar, dependiendo de cuales sean las necesidades. Estos son algunos de los ejemplos, aunque parte del proyecto consiste en utilizar algunas de estas herramientas, por lo que más adelante se darán más detalles de varias de ellas:

- **Consola OCI:** Interfaz gráfica muy intuitiva que permite crear y gestionar recursos en función de los permisos del usuario.
- **APIs:** Las APIs de la OCI son APIs convencionales que utilizan peticiones y respuestas HTTPS.
- **SDKs:** Herramientas de desarrollo software que permiten la conexión con las APIs de la OCI. Utilizan varios lenguajes de programación como son Java, Python o Ruby.
- **Command Line Interface (CLI):** Línea de comandos disponible únicamente para algunos servicios.
- **Terraform:** Software “infraestructura as code” que permite definir tu infraestructura mediante la generación de ficheros. Oracle es compatible con Terraform.
- **Ansible:** Herramienta para crear, gestionar y orquestar los recursos OCI de la infraestructura. Oracle es compatible con Ansible.
- **Resource Manager:** Servicio OCI que permite crear y aprovisionar recursos OCI. Sigue un modelo de “infraestructura as code”.

2.2.2. Recursos OCI

Como ya se ha comentado, la infraestructura cloud, está formada de una serie de recursos y aplica una terminología que la caracteriza. Al tratarse de infraestructura cloud de Oracle, se comentarán estos elementos para el caso del proveedor Oracle.

- **Regiones y dominios:** Las regiones son localizaciones geográficas donde se encuentra físicamente la infraestructura. En una región, puede haber uno o varios centros de datos que forman un dominio. Estos dominios de una misma región se encuentran conectados entre ellos y pueden verse afectados por fallas ajenas.
  Las regiones si se encuentran aisladas y puede ser conveniente escoger en que región realizar ciertas tareas.

Nombre de la región	Localización
ap-seoul-1	Seoul, South Korea
ap-tokyo-1	Tokyo, Japan
ca-toronto-1	Toronto, Canada
eu-frankfurt-1	Frankfurt, Germany
uk-london-1	London, United Kingdom
us-ashburn-1	Ashburn, VA
us-phoenix-1	Phoenix, AZ

Tabla 2: Regiones existentes y su localización física.

- **Tenancy:** Partición aislada dentro de la OCI, que Oracle facilita a una empresa para crear, organizar y administrar los recursos cloud.
- **Compartimentos:** Agrupaciones lógicas de recursos, a los que se debe dar credenciales de acceso a un grupo o usuario. Todos los compartimentos se encuentran dentro del tenancy o “root compartement”.
- **Virtual cloud networks (VCN):** Versión definida por software de una red física convencional que incluye subredes, tablas de enrutamiento y getaways donde corren las instancias.
  Pertenece a una región y engloba todos los dominios que esta contenga. Utiliza notación CIDR (Classless Inter-Domain Routing) para la distribución de las diferentes IP’s de las subredes mediante tablas de enrutamiento.

- **Instancias:** Se trata de un host alojado en la nube, que de acceso a un hardware físico. Permite un gran rendimiento y seguridad.
- **Imagen:** Plantilla que define ciertas características del software de una instancia, como por ejemplo su sistema operativo.
- **Shape:** Define características de una instancia como la CPU o la memoria.
- **Key pair:** Mecanismo de autenticación de utiliza la OCI. Consiste en dos claves, una de ellas privada y que se mantiene en el equipo del usuario y otra pública que debe ser cargada en la consola OCI.
- **Block volume:** Disco duro virtual que proporciona almacenamiento para las instancias OCI. Se usa como un disco duro físico, proporciona almacenamiento además de ser intercambiable entre instancias sin suponer perdida de datos.
- **Oracle Cloud Identifier (OCID):** ID asignada a cada uno de los recursos OCI.

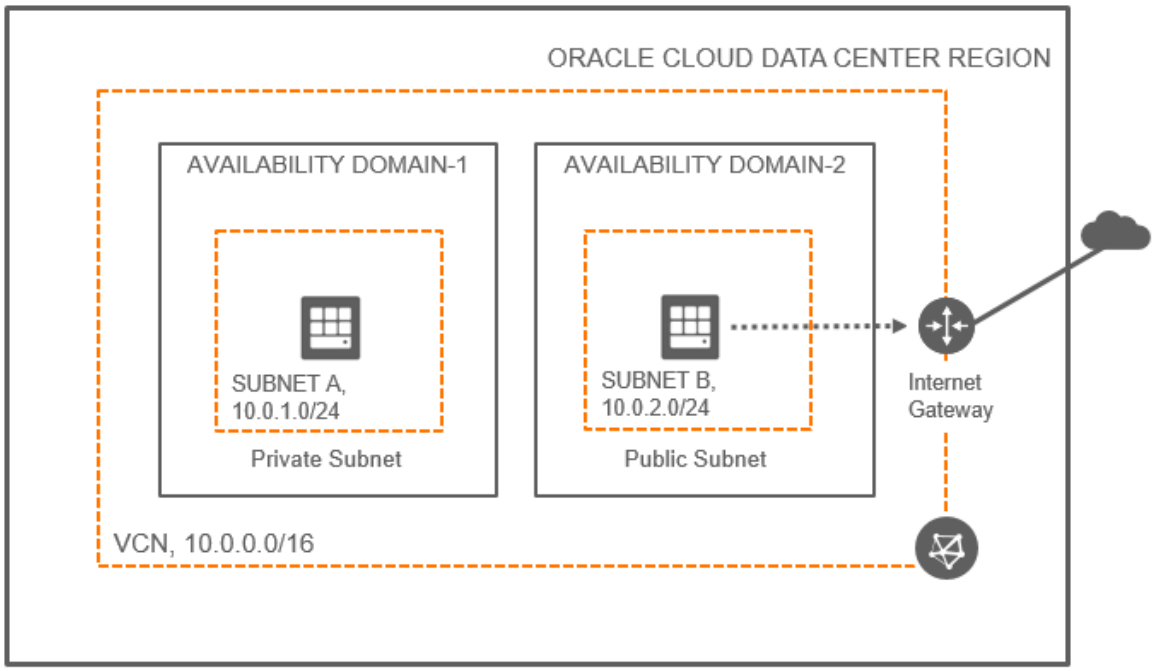


Figura 2-4: Ejemplo de estructura y relaciones entre recursos OCI.

2.3. Terraform

Terraform es un software de infraestructura de código abierto, desarrollado por HashiCorp, que permite definir nuestra infraestructura como código, lo que quiere decir que es posible definir en un fichero de texto las características de nuestra infraestructura con un lenguaje de alto nivel.

Terraform tiene soporte y puede trabajar simultáneamente con diversos proveedores de infraestructura cloud, como pueden ser Azure, AWS o en el caso que nos interesa, Oracle.

Figura 2-5: Logo de Terraform.

Las principales ventajas de la utilización de Terraform serían:

- Posibilidad de trabajar con gran cantidad de proveedores
- Trabajar con ficheros facilita el compartir y reutilizar
- Capacidad de gestionar infraestructuras de gran tamaño

Terraform es la herramienta con la que inicialmente decidimos trabajar en el proyecto. Nos parecía una buena opción por las posibilidades que ofrecía. Fue en el momento de definir las principales funcionalidades en las que queríamos centrarnos, cuando se decidió explorar alternativas, ya que la principal función de Terraform es el definir tu infraestructura en ficheros, lo cual te proporciona grandes beneficios, pero no son los buscados.

Decidimos centrarnos en el poder conectar, arrancar y parar instancias de nuestra infraestructura, lo que a través de otras herramientas orientadas a gestión de infraestructura cloud, parecía más accesible que con Terraform. Fue el momento en el que se decidió analizar Ansible y el SDK del OCI y dejar Terraform.

Este imprevisto ha requerido de una actualización en el plan de trabajo inicial, pero pensamos se trate de una buena decisión a largo plazo, ya que aunque todos los caminos nos llevaran a un mismo destino, es importante analizar cuál es el que puede hacer llegar en menos tiempo y de forma más cómoda.

2.4. Python SDK

La infraestructura cloud de Oracle proporciona kits de desarrollo de software (SDK) para permitir el desarrollo de aplicaciones y soluciones personalizadas. Todo lo desarrollado puede integrarse con los servicios que ofrece la OCI. Existen varios tipos de SDK, cada uno con sus características, pero todos ofrecen herramientas, ejemplos de código y documentación necesaria para poder trabajar. Estos kits consiguen conectar con los recursos del cloud a través de la API del OCI, por lo que será necesario una configuración previa.

Existen varios SDK's, principalmente diferenciados por el lenguaje de programación con el que trabajan. Para este proyecto se realizarán pruebas con Python, pero sería posible utilizar los basados en Java o en Ruby.

Los requisitos necesarios para poder trabajar con SDK son los siguientes:

1. Tener una cuenta OCI válida (la que utilizamos para acceder a la consola de nuestra infraestructura.
2. Un usuario en esa cuenta con los permisos necesarios para acceder a la API. Estos permisos los otorga el administrador de la infraestructura.
3. Una par de claves. Una pública subida al OCI y una privada que solo debe conocer el usuario que está conectando con la API.

2.5. Ansible

Ansible es una herramienta de automatización de TI. Puede configurar sistemas, implementar software y organizar tareas. Una de sus principales características es el permitir trabajar con simplicidad y facilidad de uso. Puede utilizarse tanto para sistemas donde solo hay implicadas unas pocas instancias, hasta grandes configuraciones donde intervienen miles de ellas. Al trabajar con OpenSSH en el aspecto de la comunicación cifrada del sistema, su seguridad es considerablemente elevada.

Los objetivos de una herramienta de gestión de la configuración como es Ansible se pueden resumir de la siguiente manera:

- Implementación de funciones que ayuden en la creación y la gestión de los recursos.
- Dar soporte a los servicios más comunes como pueden ser Compute, Storage o Networking.

Figura 2-6: Logo de Ansible.

Ansible trabaja con varios sistemas de la infraestructura al mismo tiempo. Para ello, hace uso del concepto de inventario. Se trata de una selección de los recursos sobre los que aplicará las acciones. Este inventario se especifica con un fichero dentro de la máquina del usuario. Ansible puede hacer uso de varios archivos de inventario al mismo tiempo o incluso extraer este inventario de fuentes dinámicas o en la nube. La opción de hacer dinámico el inventario nos permite mantenerlo actualizado de forma automática. Esto nos ayudará en el caso que se prevea un inventario fluctuante en el tiempo por encendido y apagados de host de la infraestructura o por posibles caídas por exceso de demandas.

Por otra parte, Ansible hace uso de los llamados Playbooks. Se trata del lenguaje que utiliza Ansible para realizar las acciones de gestión, orquestación o configuración. Están desarrollados en un lenguaje de texto y diseñados de manera que sean legible para el humano.

Por último, Ansible hace uso de una herramienta más, los módulos. Los módulos son funciones que incluye Ansible por defecto y que se pueden ejecutar a través de un

Playbook. Un usuario puede además, diseñar sus propios módulos en función de las necesidades.

Los módulos se basan en una serie de parámetros a especificar, que hacen referencia a recursos del cloud con los que vamos a interactuar. Adaptando estos parámetros de la manera en que la documentación de Ansible nos expone, pueden realizarse unas funciones u otras.

Dependiendo del proveedor con el que trabajemos, los módulos variarán. En la siguiente imagen podemos observar algunos ejemplos de módulos para dos tipos de proveedores diferentes. En este caso utilizaremos los destinados a Oracle (utilizan el prefijo oci en el nombre):

Figura 2-7: Ejemplos de algunos de los módulos que Ansible ofrece para OCI y para AWS.

La opción de implementar soluciones utilizando Ansible, nos la darán muchos proveedores de infraestructura cloud, pero en el caso que nos ocupa, nos centraremos en Oracle. El esquema de funcionamiento es el siguiente:

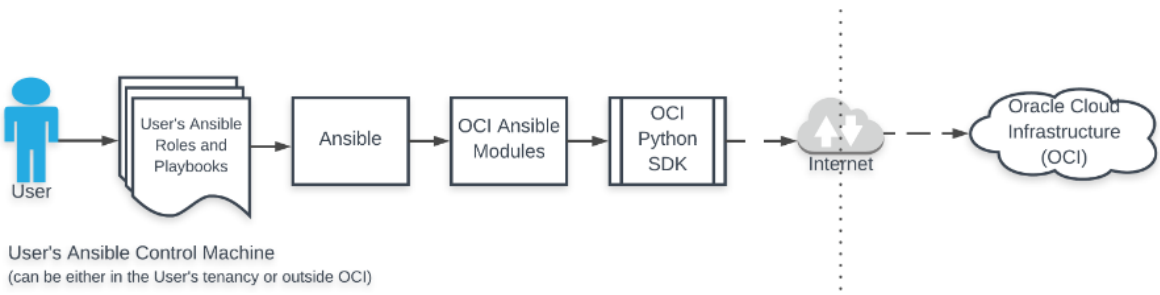


Figura 2-8: Esquema de funcionamiento de Ansible.

Ansible hace uso de las dos herramientas antes mencionadas, los playbooks y los módulos, para definir cuáles son los estados y las acciones a ejecutar.

Como se puede observar, Ansible necesita del Python SDK (software development kit), el cual a través de la API puede acceder a los recursos de la OCI sobre los que ejecutas las acciones definidas.

3. Desarrollo del proyecto

En este punto, se detallará todo el desarrollo realizado en el transcurso del proyecto. Todo lo especificado se sostiene por los conceptos previamente explicados, los cuales nos permiten profundizar en las diversas herramientas que se utilizan.

El desarrollo consta de diversas partes. Se detallará primero la manera en como definimos el entorno de test, para la seguridad de nuestras pruebas y seguirá con el análisis y las ejecuciones realizadas con diversas herramientas sobre este entorno.

La manera en que se ha procedido refleja el plan de trabajo definido para el proyecto y busca consolidar los objetivos que nos planteamos inicialmente.

3.1. Entorno de test

Como ya se ha comentado, estamos trabajando dentro de una infraestructura actualmente operativa, con la cual no podemos interactuar salvo durante pruebas controladas y teniendo claras las reacciones esperadas. Debido a estas restricciones, nos es obligado el definir un entorno en el que si poder realizar nuestras ejecuciones. Este entorno debe ser una simulación del ya existente, donde aparezcan recursos y elementos que se comporten de manera idéntica a como lo harían en la realidad. Por otra parte, no es necesario que sea idéntico, no se crearan más que los elementos justos y necesarios para las pruebas, hay que tener claro que no es un entorno que se utilizará proporcionar un servicio.

Este entorno constará de dos partes. La primera, una serie de máquinas virtuales que crearemos con la herramienta VirtualBox, en las que instalaremos todo lo necesario y desde las que conectaremos con el cloud. La segunda es el definir un área de trabajo en la nube, independiente de la ya creada para la infraestructura real de la aplicación.

Figura 3-1: Logo de VirtualBox.

3.1.1. Creación de máquinas virtuales

Para la creación de una máquina virtual, se seguirá el siguiente procedimiento:

- Descargar e instalar VitualBox, se trata de un software libre.
- Crear nueva máquina virtual e indicar nombre, sistema operativo y versión. En nuestro caso se trata de dos máquinas con sistema operativo Linux con versión Oracle-64, ya que se trata de un sistema sencillo que ya nos proporciona lo necesario y nos da la familiaridad de trabajar en Linux.

Figura 3-2: Pantalla inicial y módulo de creación de máquina virtual de VirtualBox.

- Definir el resto de parámetros que aparecen en las siguientes pestañas. Los parámetros indicados ya son válidos, no es necesario excederse en las características.

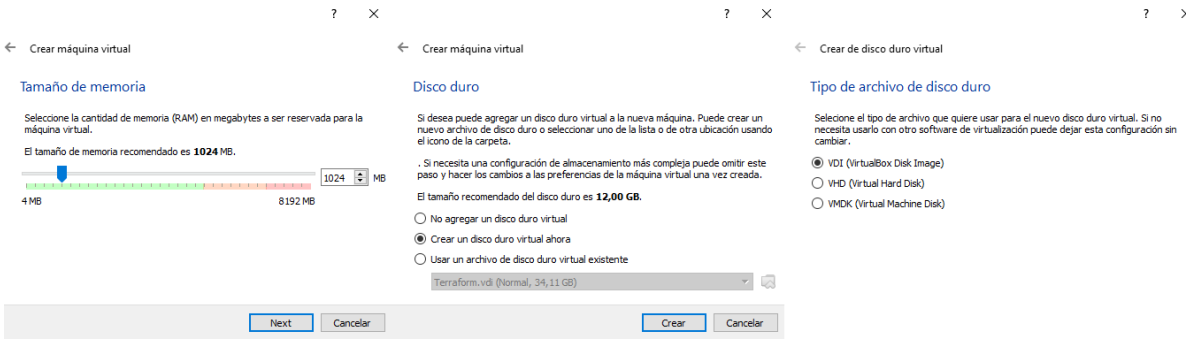


Figura 3-3: Seguimiento de pantallas para la creación de una máquina virtual en VirtualBox.

Trabajaremos con dos máquinas virtuales ya que como se verá más adelante, ciertas ejecuciones se realizan de una a otra o por el hecho de tener la opción de aplicar a varias máquinas.

3.1.2. Consola OCI

Es necesario generar un área de trabajo ajena a la infraestructura real, para ello utilizamos una herramienta que Oracle proporciona, con la cual gracias a su interfaz gráfica, podemos diseñar nuestra infraestructura cloud de forma muy visual.

Los elementos mínimos que se necesitan para generar este entorno de trabajo son los siguientes:

- VCN (virtual cloud network): En la tabla de enrutamiento solo es necesario incluir el getaway por defecto a internet, ya que no habrá subredes entre las que interactuar.

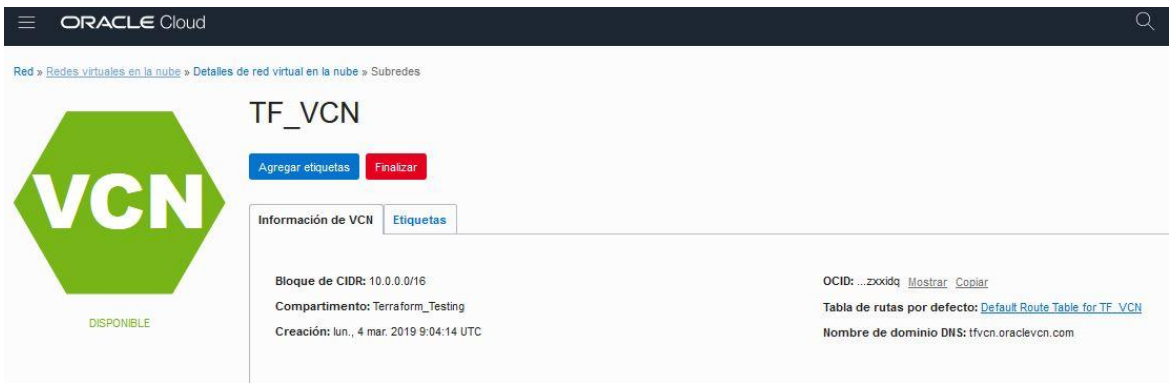


Figura 3-4: Consola OCI. Detalles de la VCN "TF_VCN".

- Compartimento: Este compartimento se creará dentro del propio tenancy o root compartment de la infraestructura real. Al ser elementos aislados, no hay ningún riesgo.

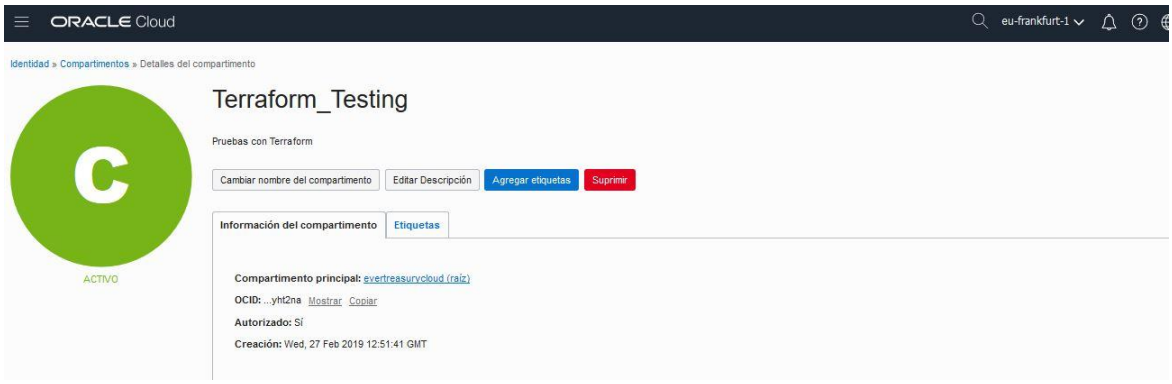


Figura 3-5: Consola OCI. Detalles del compartimento "Terraform_testing".

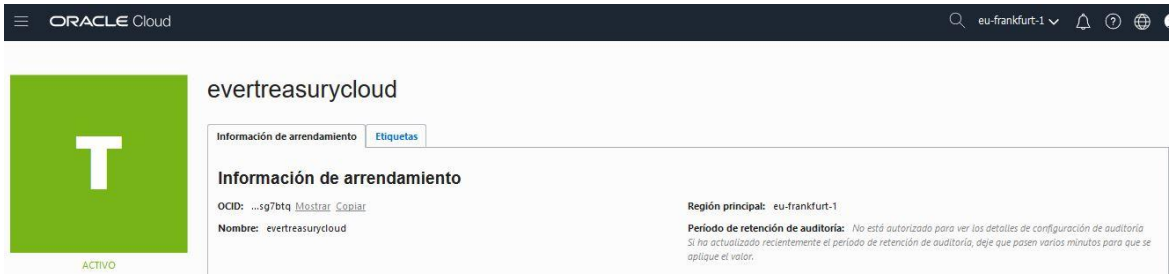


Figura 3-6: Consola OCI. Detalles del Tenancy "evertreasurycloud".

- Instancia: Únicamente es necesaria la creación de una instancia. Será sobre la que conectaremos.

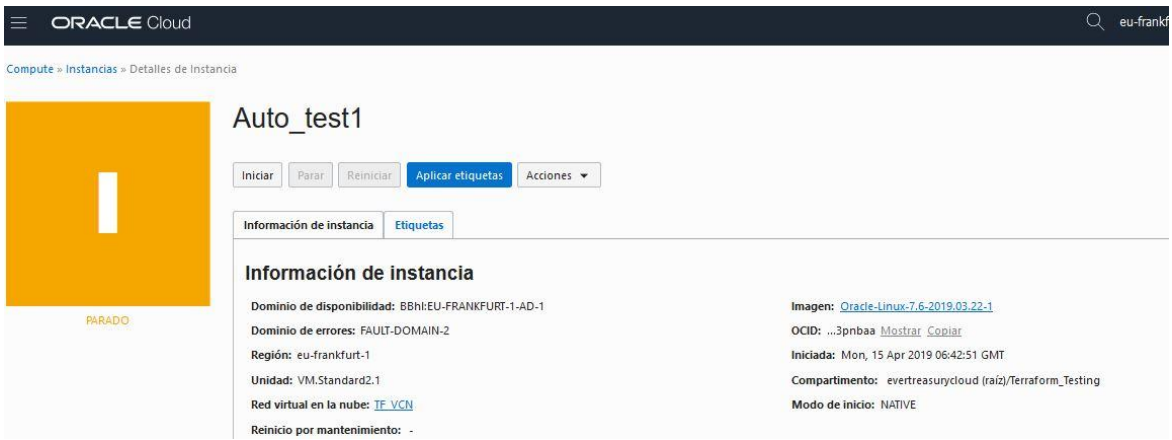


Figura 3-7: Consola OCI. Detalles de la instancia "Auto_test1".

Como se puede observar, no ha sido necesario el crear otros recursos OCI como son la región y dominio (no se crean, ya que están definidas por Oracle) o tenancy. Esto es debido a que se pueden crear los elementos que queremos aprovechando los ya existentes, manteniendo el aislamiento y por tanto la seguridad.

3.2. Análisis Python SDK

Como alternativa a Terraform, vamos a trabajar con dos herramientas y analizar si pueden ofrecer más profundidad. La primera de ellas es el kit de desarrollo de software (SDK) basado en Python.

Como ya se ha comentado, las funcionalidades en las que nos centraremos para comprobar la potencia de estas herramientas son el poder contactar con una instancia de la Oracle cloud infraestructure y conseguir arrancarla y pararla de forma satisfactoria.

3.2.1. Instalación de componentes

Para trabajar con esta herramienta, trabajaremos desde una de las máquinas virtuales que hemos creado y seguiremos el proceso de instalación de software siguiente:

1. Instalación de Python 2.7 o superior

```
[root@terraform ~]# sudo yum install -y yum-utils
[root@terraform ~]# sudo yum-config-manager --enable *EPEL
[root@terraform ~]# sudo yum install -y python2.7
```

Figura 3-8: Comandos instalación de Python.

```
[root@terraform ~]# python -V
Python 2.7.5
[root@terraform ~]#
```

Figura 3-9: Comandos para la consulta de la versión de Python instalada.

2. Instalación de PIP (Paquete de instalación PIP) para realizar instalaciones:

```
[root@terraform ~]# yum install python36-setuptools
[root@terraform ~]#  yum install python-pip
[root@terraform ~]#  yum install python36-pip
```

Figura 3-10: Comandos instalación de PIP.

```
[root@terraform bin]# ls pip*
pip  pip2  pip2.7  pip3.6
```

Figura 3-11: Comandos para la consulta de la versión de PIP instalada.

3. Instalación de OCI SDK con el comando indicado o descargando de GIT y su configuración:

```
[root@terraform bin]# pip install oci==2.1.3
```

Figura 3-12: Comandos para instalación del SDK.

```
[root@terraform bin]# python -c "import ssl; print(ssl.OPENSSL_VERSION)"
OpenSSL 1.0.2k-fips  26 Jan 2017
```

Figura 3-13: Configuración de SDK.

4. Instalación de herramienta VIM, para editar ficheros de texto cómodamente.

```
[root@terraform oci-ansible-modules]# sudo apt-get install vim
```

Figura 3-14: Comandos para instalación de VIM.

```
[root@terraform oci-ansible-modules]# vi --version
VIM - Vi IMproved 7.4 (2013 Aug 10, compiled Aug 24 2018 23:43:55)
```

Figura 3-15: Comandos para la consulta de la versión de VIM instalada.

3.2.2. Pruebas realizadas

Una vez realizado todo el proceso de instalación y actualización de los diferentes componentes necesarios, debemos proceder a la creación de varios ficheros. Se trata de los ficheros de configuración y el par de claves que se utiliza para acceder a nuestra Oracle cloud infraestructure.

3.2.2.1. Claves pública y privada

Estas dos claves están vinculadas la una a la otra, pero separadas en dos ficheros. Tienen un formato PEM y un tamaño máximo de 2048 caracteres. El comando para la creación de la clave privada es el siguiente:

```
[root@terraform .oci]# ls
config  config.ini
[root@terraform .oci]# openssl genrsa -out /root/.oci/oci_api_key.pem 2048
Generating RSA private key, 2048 bit long modulus
.....................................................................+++
.............+++
e is 65537 (0x10001)
[root@terraform .oci]# ls
config  config.ini  oci_api_key.pem
```

Figura 3-16: Comandos para la generación de la clave privada.

Para más seguridad, una buena opción es el eliminar permisos de lectura, escritura y ejecución a todos los usuarios menos al administrador.

```
[root@terraform .oci]# chmod go-rwx /root/.oci/oci_api_key.pem
```

Figura 3-17: Comandos para la eliminación de permisos de lectura, escritura y ejecución sobre la calve privada.

Generamos ahora la clave pública, la cual estará vinculada a la privada en su creación, permitiendo únicamente el acceso al OCI a través de esta.

```
[root@terraform .oci]# ls
config  config.ini  oci_api_key.pem
[root@terraform .oci]# openssl rsa -pubout -in /root/.oci/oci_api_key.pem -out /root/
.oci/oci_api_key_public.pem
writing RSA key
[root@terraform .oci]# ls
config  config.ini  oci_api_key.pem  oci_api_key_public.pem
```

Figura 3-18: Comandos para la generación de la clave pública.

Si inspeccionamos el contenido de los dos ficheros creados, podemos ver el contenido en detalle de la clave y su formato PEM.

```
[root@terraform .oci]# cat oci_api_key.pem
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA0CymL9NWQzvud+7b0t/TXDAzwGazDiKclN3WHJ2bGfVNHuPv
X14fRUUaX2c6GSHgY/i/iIL3EVwp5EotBTp845xv0eW/Um72WYxB9YD85By/rhna
rHaXLOSWd+my4XNAzZkjocW7u/Zk1VrxOdySvv/D9q9MPvV4iuZWsjvkO2LuPMSd
4U1XBqAIw6a7EM+x4dg56WZbCHwGIeYkaoC4vcxm4zwNbQ40paH6lZCtF51OSYb5
8TsqsbAxvqYxfR85d55xiMIdrOp5LJ4HFi9usjQumTcL1zaG6+vOxFsofa6hXsfS
2i00z4cVQxDWEeNsqDiff7yMm7bq23hAOKpCYwIDAQABAoIBAQDI2XtSA2ZkIiu3
nUaMU5W6iWs1BCFXzv8iBlKxc0KVLf5FgoeOjOJA1CZYy+DbJLs52WL6k9lK75Af
iY2cUMz0Uhf0/OS15QdcaJ2u/89/S1mI4bsA0L6U+stOULVdHfT1KP6e84dpF5Y7
362ENROEooJwvLskt3OhGBGnj29p9w733wwoMDf6Ox+PvWAv+ilWerZ6LEmTEjIY
SWzOWxBTdHS8SeCmv5fC/6CMfbZ4Ay9IQ1YEoBP1IuNhIF9USaBTx9bKIxSyFgBD
3E8X01vk6DlIb0Z1tnzrJCWOcge9uBdTQG7rnUVAAFK2eiHc7ksXy9ClNxlb+jV4
wMpRNLkhAoGBAP4/CV2aG71+vzS0s568vdTdkwpYy9NNuNIyiGhnrEFDkUyzobJh
x6R6IhPeAj/qGmGn1TJJPQUu/297fWrJ+qu6R68iCy8+iaFEWD2Zya9+qaywOGRK
m6sZngs9JsZcDw+Smqm3q1BJu6G6kGorxyqX5pcA87MqtWNkhXzviA/xAoGBANGc
QZO1JkE3P7/t2BgmFX2cpFM74OiqM08qDaCkoW9PKpI1/S3nNItsKbxX6PmBR4Je
d9zlRMXXEaYi5pIjN7WZ1vYmsJZ7UrM2ABDED4t5JfmIaHAeh8+wb8LxV3GgZCSv
ZUMYMuJFPq1M1Eq4TrIts6wwpY8dtfrGOUo8NcuTAoGAHcdMVYZDUYKS6p1vnFwG
7/Oz0T2tSSaFSzlX+MKLzmkXj6BKhBYnx//2FdWe9dV5mY6KzgT9/KlwiaIxIlzd
21dKJh0aqkHs5NKYcz6b/M4dGXjaYgTLDtNFS5OyAvGtYemN6+rzScgZDexIFedi
wH5iJ7WY1Zde/3E6oVN84EECgYApkAVEx5+tQMpIQV4e87ZBMQU4Z+Ah4lXINbt8
BgjorFbxoczBBF5b4venm/rbNnI6fIbh6ygm3o1mVvdHNOp//OAt8OAFBrEQTf3X
VTkL1B8pio3C5ZriaMbQbdB7/GS/DXRTyIGWckqKTHiv1DnuhLdixkadyTNjZVjI
yoWUNwKBgAvInal5TiHW9N6ypTejTqobSf7O8h7B50DxfDoW74hPRlvOuNf68wpJ
LGdMLmDnG35blleJef6uU8cByVc54FzYm2FNukF1CIKux+NxOEix2GiNQJugooVg
mx1dxaKD4JH9Tg5lALG3RoJyubia9VkAjp3LLiJV4+fbkvAItwQv
-----END RSA PRIVATE KEY-----
```

Figura 3-19: Detalles de la clave privada en formato PEM.

```
[root@terraform .oci]# cat oci_api_key_public.pem
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0CymL9NWQzvud+7b0t/T
XDAzwGazDiKclN3WHJ2bGfVNHuPvX14fRUUaX2c6GSHgY/i/iIL3EVwp5EotBTp8
45xv0eW/Um72WYxB9YD85By/rhnarHaXLOSWd+my4XNAzZkjocW7u/Zk1VrxOdyS
vv/D9q9MPvV4iuZWsjvkO2LuPMSd4U1XBqAIw6a7EM+x4dg56WZbCHwGIeYkaoC4
vcxm4zwNbQ40paH6lZCtF51OSYb58TsqsbAxvqYxfR85d55xiMIdrOp5LJ4HFi9u
sjQumTcL1zaG6+vOxFsofa6hXsfS2i00z4cVQxDWEeNsqDiff7yMm7bq23hAOKpC
YwIDAQAB
-----END PUBLIC KEY-----
```

Figura 3-20: Detalles de la clave pública en formato PEM.

Una vez creada las claves, el último paso consiste en cargar en nuestra OCI la clave pública, en concreto se vinculará con el usuario con el que nos conectaremos. Para hacerlo trabajaremos a través de la consola OCI. Solo es necesario entrar en la consola y escoger la opción de agregar clave pública dentro del perfil del usuario desde el que conectamos y escribir la clave en la pantalla que aparece.

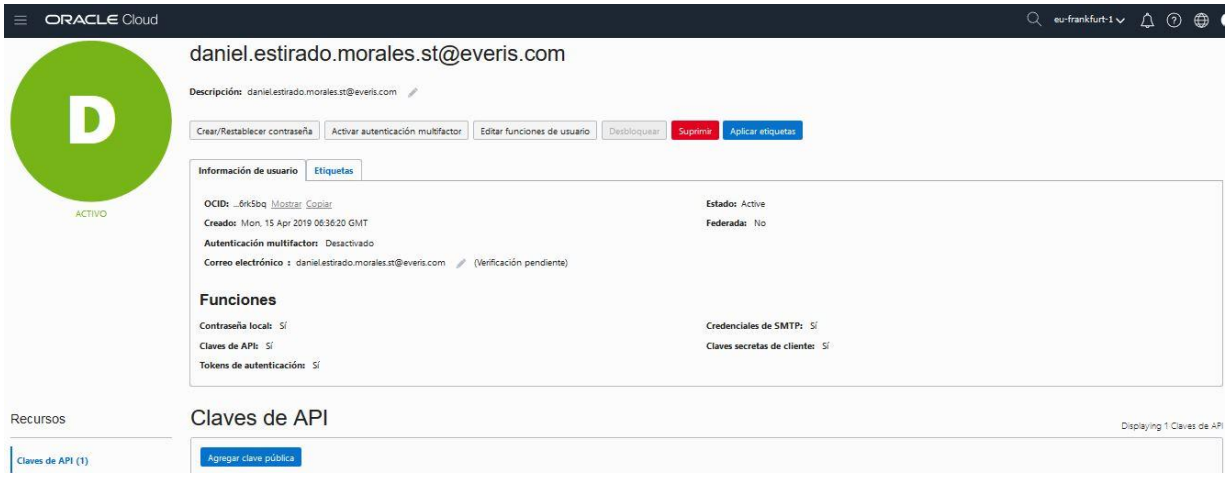


Figura 3-21: Consola OCI: Detalles del usuario utilizado para realizar al conexión.

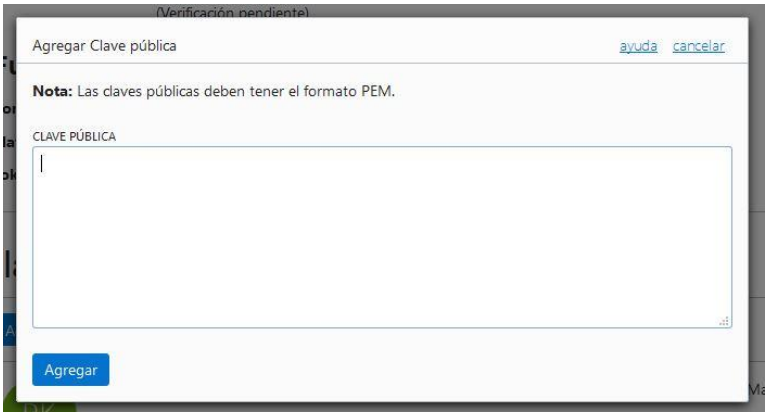


Figura 3-22: Detalles de la opción "Agregar clave pública" al usuario.

3.2.2.2. Fichero de configuración

Como se ha explicado anteriormente, el fichero de configuración consiste en un fichero que crearemos, imprescindible para el acceso a la OCI. En este fichero se especifica cierta información referente al usuario de la OCI con el que se accederá, el par de claves de acceso previamente generadas e información característica de la OCI como es la región y el tenancy.

Para la creación del fichero se ha utilizado la herramienta VIM anteriormente instalada.

```
[root@terraform .oci]# cat config
[DEFAULT]
user=ocid1.user.oc1..aaaaaaaasrzrq5ix5ayi26cmxorgbqr57uuyesrhshpmkia2oxontm5s6rsq
fingerprint=d2:39:60:c0:1f:66:80:9c:e0:39:32:02:a0:7e:39:4d
key_file=/root/base/.oci/oci_api_key.pem
tenancy=ocid1.tenancy.oc1..aaaaaaaaz5a27xj3w532ljlo6sknozc4wmu37wr7mmlmhrst3d2sozsg7b
tq
region=eu-frankfurt-1
```

Figura 3-23: Contenido del fichero de configuración utilizado en las pruebas.

Una vez más obtendremos la información que se requiere de la consola OCI. Tanto los OCID correspondientes al usuario de conexión y al tenancy como el nombre de la región donde se encuentra la infraestructura los obtendremos consultando en la consola estos recursos.

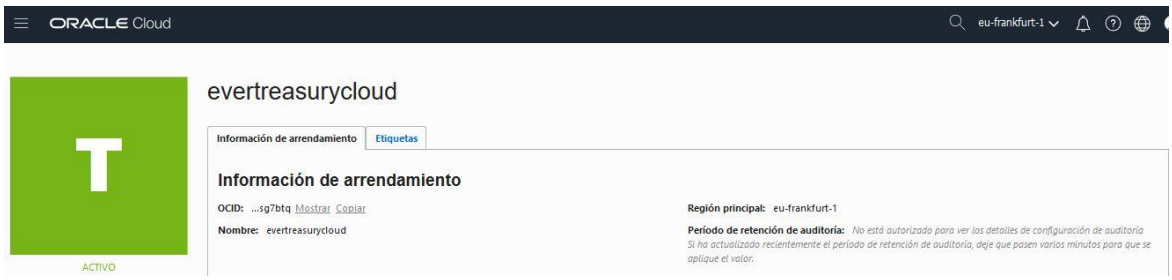


Figura 3-24: Consola OCI. Detalles del tenancy, en concreto su OCID.

Los otros dos campos a detallar son los correspondientes a las claves pública y privada. La privada, al deberse mantener en la máquina que estamos utilizando, indicamos el directorio en el que se encuentra y la especificaremos mediante su huella o fingerprint. La huella es la secuencia de letras y números usada para identificar una clave. Su utilidad es el poder diferenciar entre claves que tienen unas mismas propiedades o características, ya que al igual que la huella dactilar, se trata de un identificador único.

Para obtener la huella de la clave existen dos opciones:

1. Consultar en la consola OCI, donde se ha subido anteriormente. Al vincularse a un usuario, la podemos encontrar en el detalle de este recurso.

Figura 3-25: Consola OCI. Huella de la calve pública anteriormente vinculada al usuario.

2. Mediante la consola de comandos en la máquina virtual donde estamos trabajando.

```
[root@ansible .oci]# ls
config            oci_api_key3.pem  oci_api_key_public2.pem  oci_api_key_public.pem
oci_api_key2.pem  oci_api_key.pem   oci_api_key_public3.pem
[root@ansible .oci]# openssl rsa -pubout -outform DER -in ~/.oci/oci_api_key2.pem | openss
l md5 -c
writing RSA key
(stdin)= 30:d9:cc:65:cf:bc:1c:4f:25:ea:54:81:88:96:7e:4a
```

Figura 3-26: Comandos para la generación de la huella de la clave pública.

3.2.2.3. Conexión con la instancia

Teniendo todos las herramientas necesarias instaladas en nuestra máquina virtual y toda la configuración sea correcta, procedemos a conectar con las instancia creada en nuestra infraestructura cloud.

Para ellos debemos trabajar mediante Python, el cual ya hemos instalado. El código utilizado para la conexión es el siguiente:

```
[root@ansible .oci]# python
Python 2.7.5 (default, Nov  1 2018, 03:12:47)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36.0.1)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figura 3-27: Comando para trabajar con la pantalla de trabajo de Python.

```
>>> import oci
>>> from oci.config import from_file
>>> config = from_file(file_location="/root/.oci/config")
>>> compute_client = oci.core.ComputeClient(config, retry_strategy=oci.retry.DEFAU
LT_RETRY_STRATEGY)
>>> instance_id = 'ocid1.instance.oc1.eu-frankfurt-1.abtheljsja2spygagks3uooz5p5os
fd7oad4mov47ildjie4k7errj3pnbaa'
>>> instance = compute_client.get_instance(instance_id).data
>>> print(instance.display_name,instance.lifecycle_state )
(u'Auto_test1', u'STOPPED')
```

Figura 3-28: Líneas de código Python empleado para la conexión con la OCI.

Como se puede observar, los pasos a seguir son crear un variable de configuración, en la cual se especifica el directorio del nuestro fichero de configuración y se crea otra variable de instancia, en la que se especifica su OCID. De esta variable instancia, para probar la conexión, indicamos que nos pinte por pantalla el nombre y su estado actual (se encuentra apagada).

```
>>> print(instance.display_name,instance.lifecycle_state )
(u'Auto_test1', u'RUNNING')
```

```
>>> print(instance.display_name,instance.lifecycle_state )
(u'Auto_test1', u'STOPPED')
```

Figura 3-29: Ejemplos de la línea de código encargada de la conexión, para los casos de la máquina encendida y apagada.

A partir de que podemos conectar con la instancia de prueba que hemos creado en la infraestructura, probamos las dos funcionalidades principales, que consisten en arrancar y apagar la instancia. Para ello utilizaremos las siguientes líneas de código Python siempre después de realizar una conexión.

```
>>> compute_client.instance_action(instance_id, 'START')
<oci.response.Response object at 0x7fb2b5430850>
```

```
>>> compute_client.instance_action(instance_id, 'STOP')
<oci.response.Response object at 0x7fb2b5430490>
```

Figura 3-30: Código para arrancar y para parar la instancia.

Vamos a ver cuáles serían las secuencias de código correctas para pasar de un estado a otro:

```
>>> print(instance.display_name,instance.lifecycle_state )
(u'Auto_test1', u'STOPPED')
>>> compute_client.instance_action(instance_id, 'START')
<oci.response.Response object at 0x7fb2b5430850>
>>> print(instance.display_name,instance.lifecycle_state )
(u'Auto_test1', u'STOPPED')
>>> print(instance.display_name,instance.lifecycle_state )
(u'Auto_test1', u'STOPPED')
>>> import oci
>>> from oci.config import from_file
>>> config = from_file(file_location="/root/.oci/config")
>>> compute_client = oci.core.ComputeClient(config, retry_strategy=oci.retry.DEFAU
LT_RETRY_STRATEGY)
>>> instance_id = 'ocid1.instance.oc1.eu-frankfurt-1.abtheljsja2spygagks3uooz5p5os
fd7oad4mov47ildjie4k7errj3pnbaa'
>>> instance = compute_client.get_instance(instance_id).data
>>> print(instance.display_name,instance.lifecycle_state )
(u'Auto_test1', u'RUNNING')
```

Figura 3-31: Ejemplo de intentar visualizar el estado de la instancia, sin realizar la conexión previa. Para el caso de arrancarla.

```
>>> print(instance.display_name,instance.lifecycle_state )
(u'Auto_test1', u'RUNNING')
>>> compute_client.instance_action(instance_id, 'STOP')
<oci.response.Response object at 0x7fb2b5430490>
>>> print(instance.display_name,instance.lifecycle_state )
(u'Auto_test1', u'RUNNING')
>>> import oci
>>> from oci.config import from_file
>>> config = from_file(file_location="/root/.oci/config")
>>> compute_client = oci.core.ComputeClient(config, retry_strategy=oci.retry.DEFAU
LT_RETRY_STRATEGY)
>>> instance_id = 'ocid1.instance.oc1.eu-frankfurt-1.abtheljsja2spygagks3uooz5p5os
fd7oad4mov47ildjie4k7errj3pnbaa'
>>> instance = compute_client.get_instance(instance_id).data
>>> print(instance.display_name,instance.lifecycle_state )
(u'Auto_test1', u'STOPPED')
```

Figura 3-32: Ejemplo de intentar visualizar el estado de la instancia, sin realizar la conexión previa. Para el caso de apagarla.

En ambas estructuras podemos ver como partiendo de un estado, no es posible realizar un cambio sin la conexión con la instancia previa. Al realizar dicha conexión y con las líneas de código pertinentes para el cambio de estado, sí que conseguimos la reacción esperada.

Otra manera de comprobar que hemos puesto en marcha una instancia o que por el contrario la hemos detenido, es mediante la consola OCI. Una vez más, podemos acceder a los detalles de la instancia para comprobar su estado:

Figura 3-33: Consola OCI: Visualización del estado de la instancia "Auto_test1".

3.3. Análisis Ansible

Para empezar con el análisis de esta herramienta, es necesario instalar el software necesario, el cual será complementario al ya instalado en la sección anterior (Python SDK), debido a que trabajar con Ansible implica el utilizar de forma indirecta el SDK, por lo que es necesario el tenerlo disponible en nuestra máquina.

Una vez tengamos los requisitos básicos de uso, pasamos trabajar con los conceptos de inventario y playbooks. Como se ha explicado, el inventario consiste en la definición de las máquinas o instancias sobre las que vamos a actuar y los playbooks son las instrucciones que deberá seguir Ansible y que aplicará sobre el inventario especificado.

Por último, una vez hayamos analizado las posibilidades de estos conceptos de Ansible, pasamos a realizar la conexión con nuestra infraestructura cloud Oracle y a aplicar una arrancada y una parada sobre nuestra instancia de prueba, de igual manera que realizamos con el SDK.

3.3.1. Instalación de Ansible

De igual manera que con SDK, son útiles las herramientas como VIM o PIP, por lo que es recomendable el instalarlas como se especificó en la anterior sección. Por otra parte, es necesario el instalar Python en una versión igual o superior a 2.7 y el SDK tal y como se indicó anteriormente.

El añadido que instalaremos será únicamente Ansible, a través de GIT (Repositorio de software del que podemos descargar código fuente de diversas aplicaciones):

- Instalación de GIT:

```
[root@ansible ~]# yum  install git
```

Figura 3-34: Comando para la instalación de GIT.

```
[root@ansible ~]# git --version
git version 1.8.3.1
```

Figura 3-35: Comando para la consulta de la versión de GIT instalada.

- Descarga e instalación de Ansible:

```
[root@ansible ~]# git clone https://github.com/oracle/oci-ansible-modules.git
```

Figura 3-36: Comando para la descarga de Ansible desde GIT.

```
[root@ansible ~]# cd oci-ansible-modules/
[root@ansible oci-ansible-modules]# ./install.py
```

Figura 3-37: Comandos para la instalación de Ansible.

```
[root@ansible oci-ansible-modules]# ansible --version
ansible 2.7.7
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/root/.ansible/plugins/modules', u'/usr/share/ansible/plu
gins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.5 (default, Nov  1 2018, 03:12:47) [GCC 4.8.5 20150623 (Red Hat 4.8.5-3
6.0.1)]
```

Figura 3-38: Comandos para la consulta de la versión de Ansible instalada.

3.3.2. Pruebas de playbooks

Los playbooks funcionan como un listado de tareas a realizar por Ansible sobre un inventario definido.

En este caso, el inventario definido consiste en las dos máquinas virtuales que estamos utilizando. El aspecto del inventario es el siguiente:

```
[root@ansible proyecto]# cat inventario
[all]
ansible
terraform

[grupo1]
terraform

[grupo2]
ansible
```

Figura 3-39: Detalles del inventario que contiene las máquinas virtuales para las pruebas.

Como se observa en la imagen, se han definido las dos máquinas virtuales (sus nombres son ansible y terraform). Además se han incluido dos subgrupos además el [all] por defecto que incluye todas. Se trata de los grupos [grupo1] y [grupo2] que pueden ser utilizados para aplicar ejecuciones a un subgrupo de máquinas.

Se han generado varios playbooks para comprobar su funcionalidad. Para crearlos se ha hecho uso de la herramienta VIM, con la cual podemos crear y editar un fichero de texto en el formato especificado, en este caso se utiliza extensión yml (extensión definida para playbooks). Estos son los detalles de tres de ellos:

```
[root@ansible proyecto]# cat playbook1.yml
---
- hosts: all
  tasks:
    - name: Instalar Apache
      yum: name=httpd state=latest
    - name: Instalar Telnet
      yum:
       name: telnet
       state: latest
```

```
[root@ansible proyecto]# cat playbook2.yml
---
- hosts: all
  tasks:
    - name: Comprovar conexión con máquinas virtuales
      ping:
```

```
[root@ansible proyecto]# cat playbook3.y
---
- hosts: all
  tasks:
    - name: Desinstalar Apache
      yum:
       name=httpd
       state=absent
    - name: Desinstalar Telnet
      yum:
       name=telnet
       state=absent
```

Figura 3-40: Detalles de 3 playbooks creados para las pruebas.

Se trata de tres playbooks de estructura sencilla, donde se distingue claramente entre el grupo especificado de máquinas y las diferentes acciones a realizar sobre ellas.

El playbook1, instala dos componentes (Apache y Telnet) sobre el grupo de máquinas [all] del inventario (que incluye ambas máquinas). Para realizarlo, únicamente indicamos el nombre de cada una de las tareas, la acción a realizar (yum para instalar) y el estado en que queremos que finalice (latest para indicar que queremos que finalice la instalación).

El playbook2, realiza un ping, también sobre ambas máquinas, para comprobar si puede conectar con ellas (en caso de no estar arrancadas no podrá conectar).

El playbook3, al contrario que el primero, desinstala los dos componentes (Apache y Telnet). Se observa que es necesario indicar un nuevo estado para dejar claro el punto de la instalación al que queremos llegar.

Para ejecutar los playbooks utilizamos el siguiente comando:

```
[root@ansible proyecto]# ansible-playbook playbook3.yml
```

Figura 3-41: Comando para la ejecución de un playbook.

A continuación se muestran las ejecuciones de los diferentes playbooks y el cómo interactúan con nuestras máquinas virtuales:

- Playbook2: Siempre que tengamos las dos máquinas en ejecución, Ansible podrá realizar un ping sobre ellas correctamente:
  La ejecución de este mismo playbook con una de las dos máquinas apagada, significaría el no poder conectar con ella y por lo tanto no realizar la instalación.

```
[root@ansible proyecto]# ansible-playbook playbook2.yml

PLAY [all] *********************************************************************

TASK [Gathering Facts] *********************************************************
ok: [ansible]
ok: [terraform]

TASK [Comprovar conexión con máquinas vituales] ********************************
ok: [terraform]
ok: [ansible]

PLAY RECAP *********************************************************************
ansible                    : ok=2    changed=0    unreachable=0    failed=0
terraform                  : ok=2    changed=0    unreachable=0    failed=0
```

Figura 3-42: Ejecución del playbook 2, con las dos máquinas virtuales de pruebas encendidas.

```
[root@ansible proyecto]# ansible-playbook playbook2.yml

PLAY [all] *********************************************************************************************

TASK [Gathering Facts] *********************************************************************************
ok: [ansible]
fatal: [terraform]: UNREACHABLE! => {"changed": false, "msg": "Failed to connect to the host via ssh:
ssh: connect to host terraform port 22: Connection timed out", "unreachable": true}

TASK [Comprovar conexión con máquinas vituales] ********************************************************
ok: [ansible]
        to retry, use: --limit @/root/proyecto/playbook2.retry

PLAY RECAP *********************************************************************************************
ansible                    : ok=2    changed=0    unreachable=0    failed=0
terraform                  : ok=0    changed=0    unreachable=1    failed=0
```

Figura 3-43: Ejecución del playbook 2, con la máquina virtual "Terraform" apagada.

- Playbook1: Siempre y cuando no estén ya instalados en las máquinas, la ejecución del playbook1, lo hará y nos lo describirá como modificaciones en las máquinas (marcado en amarillo):

```
[root@ansible proyecto]# ansible-playbook playbook1.yml

PLAY [all] *********************************************************************

TASK [Gathering Facts] *********************************************************
ok: [ansible]
ok: [terraform]

TASK [Instalar Apache] *********************************************************
changed: [terraform]
changed: [ansible]

TASK [Instalar Telnet] *********************************************************
changed: [terraform]
changed: [ansible]

TASK [Iniciando Servicios] *****************************************************
changed: [terraform] => (item=httpd)
changed: [ansible] => (item=httpd)

PLAY RECAP *********************************************************************
ansible                    : ok=4    changed=3    unreachable=0    failed=0
terraform                  : ok=4    changed=3    unreachable=0    failed=0
```

Figura 3-44: Ejecución del playbook 1.

- Playbook3: Sabiendo que los componentes están instalados, se puede ejecutar el playbook3, el cual realizará la desinstalación en las dos máquinas y nos lo mostrará de igual manera en el número de cambios realizados.

```
[root@ansible proyecto]# ansible-playbook playbook3.yml

PLAY [all] *********************************************************************

TASK [Gathering Facts] *********************************************************
ok: [ansible]
ok: [terraform]

TASK [Desinstalar Apache] ******************************************************
changed: [ansible]
changed: [terraform]

TASK [Desinstalar Telnet] ******************************************************
changed: [terraform]
changed: [ansible]

PLAY RECAP *********************************************************************
ansible                    : ok=3    changed=2    unreachable=0    failed=0
terraform                  : ok=3    changed=2    unreachable=0    failed=0
```

Figura 3-45: Ejecución del playbook 3.

3.3.3. Conexión la infraestructura cloud

Para conectar con la infraestructura cloud, utilizaremos el concepto de inventario dinámico. El inventario dinámico no es más que un script que nos proporciona Ansible, con que podemos acceder a la información del cloud. Para ello debemos indicarle por los parámetros del script (más detalles en el anexo 1.1), información de nuestra infraestructura cloud. La información escogida, entre los diversos parámetros que el script permite, se ha escogido el OCID del compartimento en el que se encuentra nuestra instancia (disponible en la propia consola OCI).

Figura 3-46: OCID del compartimento donde se encuentra la instancia (extraído de la consola OCI) y comando para la creación del inventario dinámico especificando ese OCID.

Como resultado de esta ejecución, obtenemos un gran número de datos referentes a los recursos que intervienen con este compartimento. Se puede consultar la ejecución completa en el anexo 1.3. En las siguientes imágenes se puede ver algunos de la información más relevante, comparada con la información real extraída de la consola OCI:

1- Instancias existentes en el compartimento con sus OCID correspondientes:

Figura 3-47: Comparativa de la información de las instancias del inventario dinámico generado y la información real (extraída de la consola OCI). En concreto sus nombres y OCIDs.

2- Dominio donde se encuentran los recursos:

Figura 3-48: Comparativa de la información de las instancias del inventario dinámico generado y la información real (extraída de la consola OCI). En concreto el dominio en el que se encuentra.

3- Región a la que pertenece la infraestructura:

Figura 3-49: Comparativa de la información de las instancias del inventario dinámico generado y la información real (extraída de la consola OCI). En concreto la región en la que se encuentra.

Una vez se ha comprobado que la conexión mediante el inventario dinámico definido es correcta, podemos proceder a arrancar y parar una instancia. Para ello trabajaremos con un playbook en el que incluiremos un módulo. Los módulos son funciones que proporciona Ansible o que pueden ser prediseñadas por usuarios.

En este caso trabajaremos con el módulo: “oci_instance-Launch. Una vez creemos el playbook con la estructura del módulo dentro, únicamente será necesario ejecutarlo contra el inventario dinámico definido:

```
[root@ansible inventory-script]# cat playbook4.yml
- hosts: all
  tasks:
  - name: Stop an instance
    oci_instance:
      id: "ocid1.instance.oc1.eu-frankfurt-1.abtheljsja2spygagks3uooz5p5osfd7oad4
mov47ildjie4k7errj3pnbaa"
      state: "stopped"
```

Figura 3-50: Playbook creado con la estructura del módulo para parar la instancia.

```
[root@ansible inventory-script]# cat playbook5.yml
- hosts: all
  tasks:
  - name: Launch an instance
    oci_instance:
       id: "ocid1.instance.oc1.eu-frankfurt-1.abtheljsja2spygagks3uooz5p5osfd7oad4
mov47ildjie4k7errj3pnbaa"
       state: "running"
```

Figura 3-51: Playbook creado con la estructura del módulo para arrancar la instancia

Como se observa, únicamente se necesita indicar el OCID de la instancia con la que trabajar, en este caso el de la instancia “Auto_test1”, y jugar con el estado en que queremos que acabe la instancia tras la ejecución.

Para ejecutar este playbook creado contra el inventario que conecta con la infraestructura, ejecutamos el siguiente comando:

```
[root@ansible inventory-script]# ansible-playbook /oci_inventory.py playbook5.yml
```

Figura 3-52: Comando para la ejecución de un playbook sobre las instancias definidas en un inventario dinámico.

4. Presupuesto

Este proyecto consiste en el desarrollo de software, por lo que no es posible hacer una valoración a nivel componentes o coste de un prototipo.

En cuanto al gasto derivado de licencias de software utilizado, es nulo. Se ha podido trabajado en todo momento con herramientas OpenSource, las cuales están disponibles en el dominio público y que por lo tanto no añaden coste al proyecto.

Para calcular el coste de la mano de obra, se utilizaría la siguiente ecuación:

$$Coste = nº\ trabajadores * \frac{salario\ (€)}{h} * horas\ dedicadas$$

Pese a que he realizado consultas a compañeros del equipo y han dedicado horas, vamos a considerarlas despreciables y a contar un único trabajador. En este caso, un estudiante. Para dicho estudiante se aplicaría el sueldo bruto de 8 euros a la hora que la universidad pacta con las empresas como ayuda al estudiante.

Por último, en cuanto a las horas de dedicadas al proyecto, se hará una estimación teniendo en cuenta que se trata de un proyecto final de grado que consta de 24 créditos, los cuales están valorados en 25 horas cada uno.

Teniendo en cuenta todas estas consideraciones, el coste de la mano de obra sería el siguiente:

$$Coste\ mano\ de\ obra = nº\ trabajadores * \frac{salario\ (€)}{h} * horas\ dedicadas$$

$$Coste\ mano\ de\ obra = 1 * \frac{8€}{h} * (24\ créditos * 25\ h) = \mathbf{4800}€$$

Es cierto que la aplicación general de la empresa Everis en la cual se ha basado este proyecto, tiene contratada una gran infraestructura formada de gran número de recursos en la nube, los cuales significan para la empresa un elevado coste. Pero no es necesario contarlos dentro del presupuesto de este proyecto, ya que únicamente hemos trabajado con una pequeña área, creada específicamente para este proyecto y en la cual solo se han realizado interacciones con una instancia.

La infraestructura Cloud de Oracle se basa en un modelo *Pay as you Go*, lo que significa que únicamente se cobra por el tiempo de uso de un recurso o lo que es lo mismo, significará un coste el tiempo que tengamos una instancia en ejecución.

Debido a que precisamente las pruebas que se han realizado consistían en arrancar la instancia de test, habría que incluir una estimación del coste que esto ha conllevado.

$$Coste\ pruebas = \frac{precio(€)}{tiempo\ ejecución\ (h)} * nº\ horas\ en\ ejecución(h)$$

Shape	Instance type	OCPU	RAM (GB)	PAYG (OCPU/Hr.)
VM.Standard2.1 – VM.Standard2.24*	Gen 2 Standard VMs	1-24	15-320	$0.0638
VM.Standard1.1 – VM.Standard1.16	Gen 1 Standard VMs	1-16	7-112	$0.0638
VM.DenseIO2.8 – VM.DenseIO2.24	Gen 2 Dense IO VMs	8-24	60-320	$0.1275
VM.DenseIO1.4 VM.DenseIO1.16	Gen 1 Dense IO VMs	4-16	60-240	$0.1275
Compute Windows OS	1 OCPU Windows Server Standard			$0.0204

Tabla 3: Resumen del precio por PayAsYouGo dependiendo de la instancia y de sus características.

En la tabla se puede observar el precio Pay as you Go de una instancia de infraestructura cloud de Oracle, dependiendo del tipo que sea y de su capacidad de procesador y de RAM. Una OCPU se define como la capacidad CPU equivalente a un procesador Intel Xeon con hyper threading habilitado. Se utiliza como una unidad básica de procesado.

Nuestra instancia de test, debido a la finalidad que tiene, se ha diseñado como una VM.Standard2.1, con 1 OCPU y 48GB de RAM. Estas especificaciones corresponderían a la primera fila de la tabla.

Además hay que tener en cuenta que por el simple hecho de arrancar una máquina, ya se cobraría como una hora de uso.

Por último, consideraremos de forma aproximada, que se han realizado 50 arrancadas de la instancia, en el transcurso de todas las pruebas realizadas con las diferentes herramientas.

Teniendo en cuenta estas consideraciones, el coste estimado de las pruebas realizadas resultante es el siguiente:

$$Coste\ pruebas = \frac{precio(€)}{tiempo\ ejecución\ (h)} * nº\ horas\ en\ ejecución(h)$$

$$Coste\ pruebas = \frac{0.0638\$}{1\text{h}} * \frac{0.89€}{1\$} * 50h = \mathbf{2.83}€$$

Por lo tanto en el coste total del proyecto tiene un peso considerablemente mayor la mano de obra:

$$Coste\ Total = Coste\ mano\ de\ obra + Coste\ pruebas = 4800€ + 2.83€ = \mathbf{4802.83}€$$

5. Conclusiones y futuro del proyecto

5.1. Conclusiones análisis Terraform

Al documentarme sobre Terraform, comprobé que se trataba de una herramienta muy útil, con la que definir la infraestructura y poder aplicar soluciones de gestión de la misma. El problema fue al pensar en cómo implementar las funciones de conexión con una instancia, puesta en marcha y parada, que eran las funciones en las que centrarse. Terraform no era la herramienta que podía realizarlo de la manera más sencilla y directa.

Esto no quiere decir que no sea una herramienta potente, todo lo contrario, pero es debido a las especificaciones concretas que se buscan, que la conclusión a la que se llegó, fue el buscar alternativas más óptimas.

5.2. Conclusiones análisis Python SDK

Con el kit de desarrollo de software que proporciona Oracle, sí que vi que era una herramienta con la cual se podía implementar las funciones requeridas. A diferencia de Terraform, el objetivo hacia el que está orientada la herramienta, es el poder conectar con recursos cloud de nuestra infraestructura, por lo que es sin duda una opción más óptima.

El poder trabajar con diferentes lenguajes de programación, le proporciona además una facilidad de adaptación bastante importante ya que no obliga al usuario a conocer una nomenclatura en particular.

Con Python SDK si se consiguieron llevar a cabo las pruebas y las funciones planificadas obteniendo resultados satisfactorios.

5.3. Conclusiones análisis Ansible

Con la utilización de Ansible, de igual manera que con SDK, he tenido la sensación de estar trabajando con un software con el que poder resolver las especificaciones de conexión, apagado y encendido de instancias de forma óptima. En este caso, gracias a los conceptos de playbook y módulo veo que es la opción más cómoda para hacerlo, ya que proporcionan una versatilidad muy grande, además de que los módulos son funciones ya implementadas que en muchos casos ya proporcionan los resultados buscados.

Por otra parte está el concepto de inventario dinámico. Se trata de una manera perfecta de mantener actualizado el conjunto de recursos cloud de tu infraestructura, lo cual es vital en una infraestructura que sufre constantes cambios, actualizaciones o creaciones de nuevos recursos.

Hay que dejar claro, que de igual manera que Terraform y SDK, se trata de una herramienta muy potente, con la que trabajar en muchos aspectos distintos de la infraestructura cloud. Pero Ansible lo hace posible de la manera más cómoda y más accesible para el usuario.

Es por esto que Ansible es la herramienta que tiene más posibilidades y que ha sido escogida para la continuación del proyecto.

5.4. Conclusiones objetivos propuestos

Quería analizar los objetivos que plantee en el inicio del proyecto, de manera que se vea de qué manera se han trabajado cada uno de ellos.

En primer lugar, mi intención era el conocer la infraestructura cloud. Pienso que se trata de un concepto muy amplio y complejo, del que aún tengo mucho que aprender, pero del que sin duda he incrementado considerablemente mis conocimientos. No solo a nivel teórico, si no trabajando directamente con herramientas y aplicaciones que manejan recursos de infraestructura reales.

En cuanto al analizar y conocer nuevas herramientas, pienso que es uno de los temas más trabajados, ya que el proyecto se ha basado en gran parte en la utilización de tres herramientas (Terraform, Ansible y SDK) a priori destinadas a un objetivo parecido pero que resultan ser totalmente diferentes.

Llegar a conocer un proyecto real, es quizá un punto que considero no alcanzado del todo. Es cierto que he trabajado junto a profesionales, que me han ayudado mucho y he conocido una aplicación real dentro de una empresa. Pero es quizá por ser un proyecto que podía desarrollarse de forma aislada a otros equipos y departamentos que participan en la aplicación, que quizá no haya llegado al punto de conocimiento que me habría gustado.

En cuanto al diseño de funcionalidades y su posterior implementación, creo que ha sido un objetivo alcanzado satisfactoriamente. Definimos trabajar en tres funcionalidades principales, el poder conectar, encender y apagar una instancia y he podido implementarlas con las Ansible y el SDK.

Por último, comentar que el entorno creado y su posterior utilización en el testeo, ha sido muy útil para conocer un entorno de estas características y su gran utilidad.

5.5. Futuro del proyecto

Desde un inicio este proyecto ha consistido en definir unas bases de cara al futuro del proyecto. El analizar las diferentes herramientas ha servido para llegar a la conclusión de que Ansible será en futuros desarrollos del proyecto, el software utilizado.

El haber podido interactuar con un recurso del cloud, abre las posibilidades a que se definan nuevas funcionalidades y se llegue a concretar una aplicación con la que poder orquestar el conjunto de la infraestructura cloud de Oracle.

Un punto a tener en cuenta de cara al futuro es que, debido a que este software está destinado a ser usado por los operarios que controlan el correcto funcionamiento de la infraestructura, es importante hacer que incorpore una interface, la cual debe ser accesible para usuarios que no conozcan el funcionamiento interno de la misma.

Bibliografía:

Lorin Hochstein. ”Ansible:Up and Running. Automating configuration management and deployment the easy way”. May 2015: First Edition.

Michael Heap. “Ansible, From beginner to Pro”. 2016. DOI: 10.1007.

Guido van Rossum, Fred L. Drake, Jr. “El manual de Python” September 2009

Web Terraform.io

Web docs.cloud.oracle.com

Web Oracle.com

Web blog.desdelinux.net

Web openwebinars.net

Web docs.ansible.com

Anexos:

Anexo 1: Ansible

Anexo 1.1: Contenido del script de creación del inventario dinámico

```
[root@ansible inventory-script]# cat oci_inventory.py
#!/usr/bin/env python
# Copyright (c) 2018, 2019, Oracle and/or its affiliates.
# This software is made available to you under the terms of the GPL 3.0 license or the Ap
ache 2.0 license.
# GNU General Public License v3.0+ (see COPYING or https://www.gnu.org/licenses/gpl-3.0.t
xt)
# Apache License v2.0
# See LICENSE.TXT for details.

"""
Oracle Cloud Infrastructure(OCI) Inventory Script
=================================================
This script generates Ansible dynamic inventory for OCI by using the OCI Python SDK.

The order of precedence for reading parameters is command line arguments, then environmen
t variables followed by options
in inventory settings file. The dynamic inventory settings file defaults to "./oci_invent
ory.ini" file.
The config file defaults to "~/.oci/config" file. The script reads "DEFAULT" profile from
 the config file if no profile
name is specified.

This script accepts following command line arguments:

usage: oci_inventory.py [-h] [--list] [--host HOST] [-config CONFIG_FILE]
                        [--profile PROFILE] [--tenancy TENANCY]
                        [--compartment-ocid COMPARTMENT_OCID]
                        [--compartment COMPARTMENT]
                        [--parent-compartment-ocid PARENT_COMPARTMENT_OCID]
                        [--fetch-hosts-from-subcompartments] [--refresh-cache]
                        [--debug] [--auth {api_key,instance_principal}]
                        [--enable-parallel-processing]
                        [--max-thread-count MAX_THREAD_COUNT]
                        [--freeform-tags FREEFORM_TAGS]
                        [--defined-tags DEFINED_TAGS] [--regions REGIONS]
                        [--exclude-regions EXCLUDE_REGIONS]
```

Figura 7-1: Script oci_inventory.py

Anexo 1.2: Información del comando de ejecución de playbooks

```
[root@ansible proyecto]# ansible-playbook
Usage: ansible-playbook [options] playbook.yml [playbook2 ...]

Runs Ansible playbooks, executing the defined tasks on the targeted hosts.

Options:
  --ask-vault-pass      ask for vault password
  -C, --check           don't make any changes; instead, try to predict some
                        of the changes that may occur
  -D, --diff            when changing (small) files and templates, show the
                        differences in those files; works great with --check
  -e EXTRA_VARS, --extra-vars=EXTRA_VARS
                        set additional variables as key=value or YAML/JSON, if
                        filename prepend with @
  --flush-cache         clear the fact cache for every host in inventory
  --force-handlers      run handlers even if a task fails
  -f FORKS, --forks=FORKS
                        specify number of parallel processes to use
                        (default=5)
  -h, --help            show this help message and exit
  -i INVENTORY, --inventory=INVENTORY, --inventory-file=INVENTORY
                        specify inventory host path or comma separated host
                        list. --inventory-file is deprecated
  -l SUBSET, --limit=SUBSET
                        further limit selected hosts to an additional pattern
  --list-hosts          outputs a list of matching hosts; does not execute
                        anything else
  --list-tags           list all available tags
  --list-tasks          list all tasks that would be executed
  -M MODULE_PATH, --module-path=MODULE_PATH
                        prepend colon-separated path(s) to module library
                        (default=[u'/root/.ansible/plugins/modules',
                        u'/usr/share/ansible/plugins/modules'])
  --skip-tags=SKIP_TAGS
                        only run plays and tasks whose tags do not match these
                        values
  --start-at-task=START_AT_TASK
                        start the playbook at the task matching this name
  --step                one-step-at-a-time: confirm each task before running
  --syntax-check        perform a syntax check on the playbook, but do not
                        execute it
  -t TAGS, --tags=TAGS  only run plays and tasks tagged with these values
  --vault-id=VAULT_IDS  the vault identity to use
  --vault-password-file=VAULT_PASSWORD_FILES
                        vault password file
  -v, --verbose         verbose mode (-vvv for more, -vvvv to enable
                        connection debugging)
  --version             show program's version number and exit

  Connection Options:
    control as whom and how to connect to hosts

    -k, --ask-pass      ask for connection password
    --private-key=PRIVATE_KEY_FILE, --key-file=PRIVATE_KEY_FILE
                        use this file to authenticate the connection
    -u REMOTE_USER, --user=REMOTE_USER
                        connect as this user (default=root)
    -c CONNECTION, --connection=CONNECTION
```

Figura 7-2: Información del comando ansible-playbook. Parte 1.

```
                        connection type to use (default=smart)
  -T TIMEOUT, --timeout=TIMEOUT
                        override the connection timeout in seconds
                        (default=10)
  --ssh-common-args=SSH_COMMON_ARGS
                        specify common arguments to pass to sftp/scp/ssh (e.g.
                        ProxyCommand)
  --sftp-extra-args=SFTP_EXTRA_ARGS
                        specify extra arguments to pass to sftp only (e.g. -f,
                        -l)
  --scp-extra-args=SCP_EXTRA_ARGS
                        specify extra arguments to pass to scp only (e.g. -l)
  --ssh-extra-args=SSH_EXTRA_ARGS
                        specify extra arguments to pass to ssh only (e.g. -R)

Privilege Escalation Options:
  control how and which user you become as on target hosts

  -s, --sudo            run operations with sudo (nopasswd) (deprecated, use
                        become)
  -U SUDO_USER, --sudo-user=SUDO_USER
                        desired sudo user (default=root) (deprecated, use
                        become)
  -S, --su              run operations with su (deprecated, use become)
  -R SU_USER, --su-user=SU_USER
                        run operations with su as this user (default=None)
                        (deprecated, use become)
  -b, --become          run operations with become (does not imply password
                        prompting)
  --become-method=BECOME_METHOD
                        privilege escalation method to use (default=sudo),
                        valid choices: [ sudo | su | pbrun | pfexec | doas |
                        dzdo | ksu | runas | pmrun | enable | machinectl ]
  --become-user=BECOME_USER
                        run operations as this user (default=root)
  --ask-sudo-pass       ask for sudo password (deprecated, use become)
  --ask-su-pass         ask for su password (deprecated, use become)
  -K, --ask-become-pass
                        ask for privilege escalation password
```

Figura 7-3: Información del comando ansible-playbook. Parte 2.

Anexo 1.3: Detalles completo de inventario dinámico generado

```
{
  "BBhl_EU-FRANKFURT-1-AD-1": {
    "children": [
      "ocid1.subnet.oc1.eu-frankfurt-1.aaaaaaaao734zu3juwkcdoy4aqmo6rjkfut2yrbkf
xq3rplmwkazgaus77ma"
    ],
    "hosts": [
      "130.61.85.64",
      "130.61.47.134",
      "130.61.113.89"
    ]
  },
  "Schedule#AnyDay=0_0_0_0_0_1_1_1_1_1_1_1_1_1_0_0_0_0_0_0_0_0": {
    "hosts": [
      "130.61.113.89"
    ]
  },
  "Terraform_Testing": {
    "hosts": [
      "130.61.85.64",
      "130.61.47.134",
      "130.61.113.89"
    ]
  },
  "VM.Standard2.1": {
    "hosts": [
      "130.61.85.64",
      "130.61.47.134",
      "130.61.113.89"
    ]
  },
  "_meta": {
    "hostvars": {
      "130.61.113.89": {
        "availability_domain": "BBhl:EU-FRANKFURT-1-AD-1",
        "compartment_id": "ocid1.compartment.oc1..aaaaaaaa7emdewvipspas3h6on2a57
saf2m3cs2yahrt4ffb4qe6ziyht2na",
        "defined_tags": {
          "Schedule": {
            "AnyDay": "0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0"
```

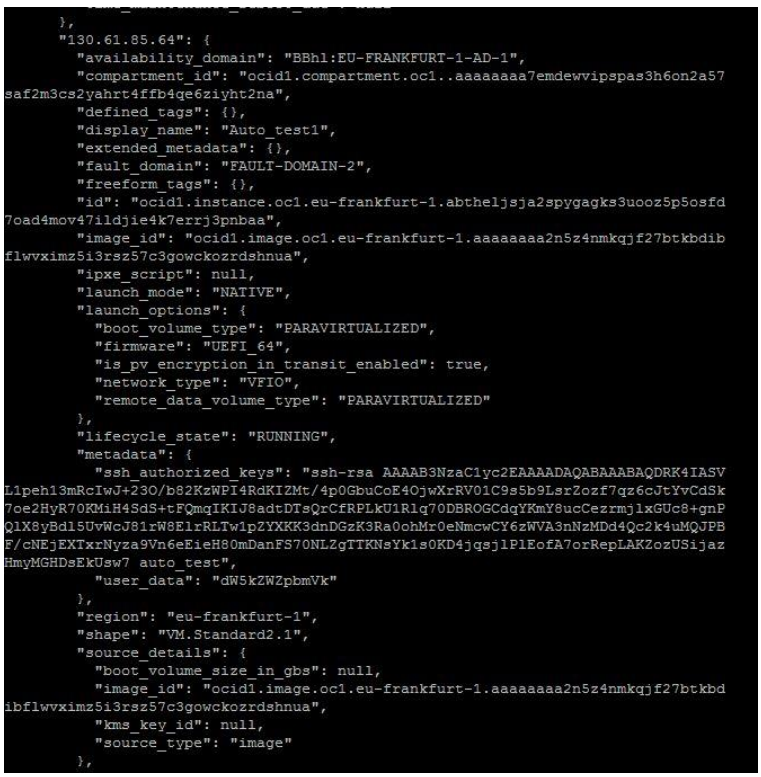

Figura 7-4: Inventario dinámico generado. Parte 1.

```
        }
      },
      "display_name": "TEST_JP2",
      "extended_metadata": {},
      "fault_domain": "FAULT-DOMAIN-2",
      "freeform_tags": {},
      "id": "ocid1.instance.oc1.eu-frankfurt-1.abtheljsffgqbisp54mwjppxf45ujvt
z36y4idrtxgj7uukcyqnkrccqtiia",
      "image_id": "ocid1.image.oc1.eu-frankfurt-1.aaaaaaaandqh4s7a3oe3on6osdbw
ysgddwqwyghbx4t4ryvtcwk5xikkpvhq",
      "ipxe_script": null,
      "launch_mode": "NATIVE",
      "launch_options": {
        "boot_volume_type": "PARAVIRTUALIZED",
        "firmware": "UEFI_64",
        "is_pv_encryption_in_transit_enabled": true,
        "network_type": "VFIO",
        "remote_data_volume_type": "PARAVIRTUALIZED"
      },
      "lifecycle_state": "RUNNING",
      "metadata": {
        "ssh_authorized_keys": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCy6MME1a
iY+WywYdViC3G/kuVuv00tVVtB70d3WZdqoj5lCWGzZq4O9foQHP0QcwZZV32eOtoSa39sb5oCbswWCq
Bj1poQrspwef24AwKWUJYneOwTZYsLtTZ/0w8yQ98wRepkaUgKHqY4ukdc3uklWlzcbucsU15ittFyBf
+QHv+BphWlacWE271ciEPq5uEwUG+l6sglzZJZOdSPKwaWbr5+gdhPHlvsd9lzK/DEUa6onmJW3Filmv
Bj7gGJ1wG3GCn1sfNSba+IrLz5aPkPLo+d3TCS240ttkH89jLylTRMrlSsdkuEkpz4x5lovBRwn0/cJV
WfN34404v6giD1 afranchg@BCN-HT2NVF2"
      },
      "region": "eu-frankfurt-1",
      "shape": "VM.Standard2.1",
      "source_details": {
        "boot_volume_size_in_gbs": null,
        "image_id": "ocid1.image.oc1.eu-frankfurt-1.aaaaaaaandqh4s7a3oe3on6osd
bwysgddwqwyghbx4t4ryvtcwk5xikkpvhq",
        "kms_key_id": null,
        "source_type": "image"
      },
      "time_created": "2019-05-21T07:30:46.839000+00:00",
      "time_maintenance_reboot_due": null
    },
    "130.61.47.134": {
```


Figura 7-5: Inventario dinámico generado. Parte 2.

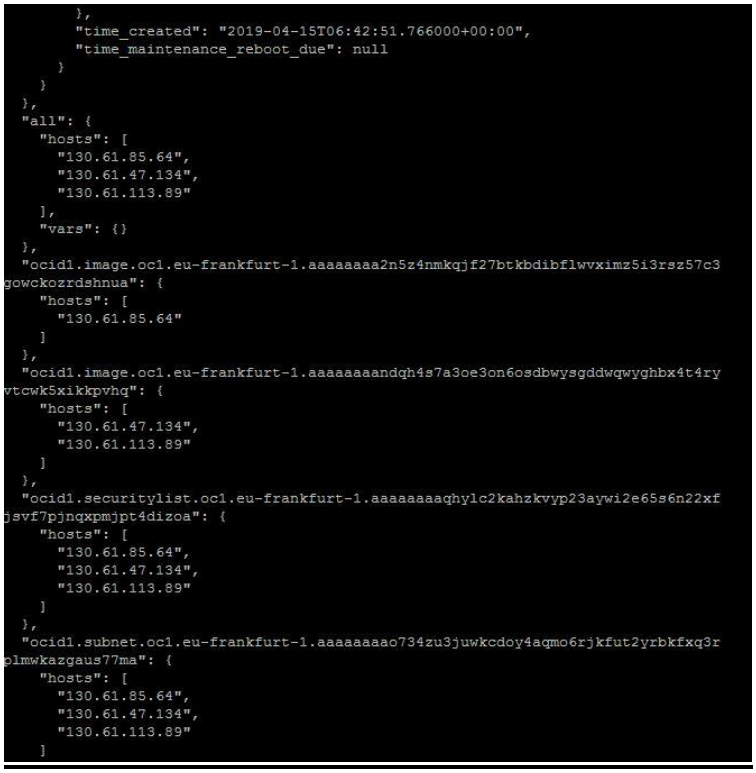
```
        "availability_domain": "BBhl:EU-FRANKFURT-1-AD-1",
        "compartment_id": "ocid1.compartment.oc1..aaaaaaaa7emdewvipspas3h6on2a57
saf2m3cs2yahrt4ffb4qe6ziyht2na",
        "defined_tags": {},
        "display_name": "TEST_JP1",
        "extended_metadata": {},
        "fault_domain": "FAULT-DOMAIN-2",
        "freeform_tags": {},
        "id": "ocid1.instance.oc1.eu-frankfurt-1.abtheljs7pjm7qh532bef6ekw6dww5d
gl5tax5uns3gjcu5l4rk6h5w7xq3q",
        "image_id": "ocid1.image.oc1.eu-frankfurt-1.aaaaaaaandqh4s7a3oe3on6osdbw
ysgddwqwyghbx4t4ryvtcwk5xikkpvhq",
        "ipxe_script": null,
        "launch_mode": "NATIVE",
        "launch_options": {
          "boot_volume_type": "PARAVIRTUALIZED",
          "firmware": "UEFI_64",
          "is_pv_encryption_in_transit_enabled": true,
          "network_type": "VFIO",
          "remote_data_volume_type": "PARAVIRTUALIZED"
        },
        "lifecycle_state": "RUNNING",
        "metadata": {
          "ssh_authorized_keys": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCy6MMEla
iY+WywYdViC3G/kuVuv00tVVtB70d3WZdqoj5lCWGzZq4O9foQHP0QcwZZV32eOtoSa39sb5oCbswWCq
Bj1poQrspwef24AwKWUJYneOwTZYsLtTZ/0w8yQ98wRepkaUgKHqY4ukdc3uklWlzcbucsU15ittFyBf
+QHv+BphWlacWE271ciEPq5uEwUG+16sglzZJZOdSPKwaWbr5+gdhPHlvsd9lzK/DEUa6onmJW3Filmv
Bj7gGJlwG3GCn1sfNSba+IrLz5aPkPLo+d3TCS240ttkH89jLylTRMrlSsdkuEkpz4x5lovBRwn0/cJV
WfN34404v6giD1 afranchg@BCN-HT2NVF2"
        },
        "region": "eu-frankfurt-1",
        "shape": "VM.Standard2.1",
        "source_details": {
          "boot_volume_size_in_gbs": null,
          "image_id": "ocid1.image.oc1.eu-frankfurt-1.aaaaaaaandqh4s7a3oe3on6osd
bwysgddwqwyghbx4t4ryvtcwk5xikkpvhq",
          "kms_key_id": null,
          "source_type": "image"
        },
        "time_created": "2019-05-21T07:27:08.194000+00:00",
        "time_maintenance_reboot_due": null
```


Figura 7-6: Inventario dinámico generado. Parte 3.



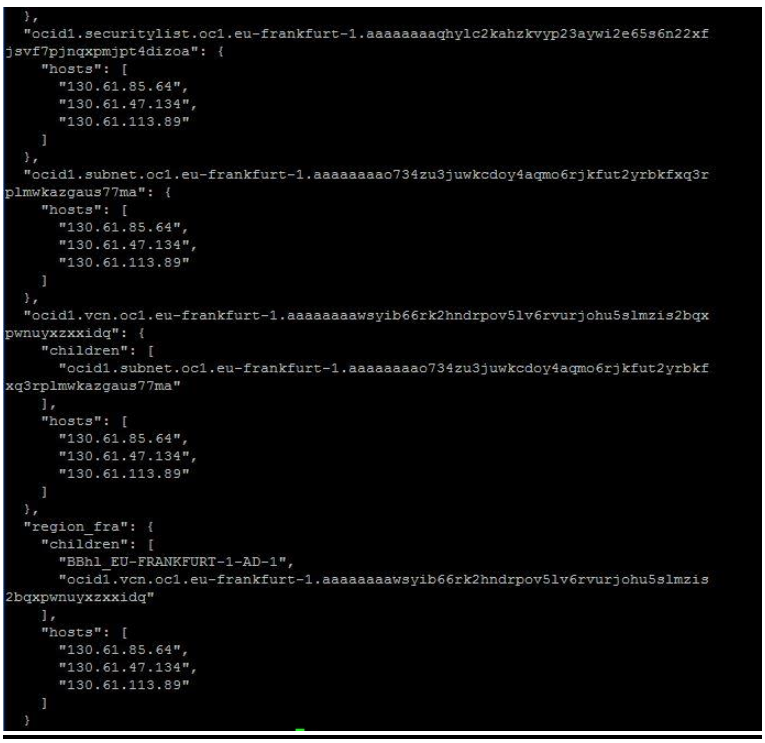
```
      },
      "130.61.85.64": {
        "availability_domain": "BBhl:EU-FRANKFURT-1-AD-1",
        "compartment_id": "ocid1.compartment.oc1..aaaaaaaa7emdewvipspas3h6on2a57
saf2m3cs2yahrt4ffb4qe6ziyht2na",
        "defined_tags": {},
        "display_name": "Auto_test1",
        "extended_metadata": {},
        "fault_domain": "FAULT-DOMAIN-2",
        "freeform_tags": {},
        "id": "ocid1.instance.oc1.eu-frankfurt-1.abtheljsja2spygagks3uooz5p5osfd
7oad4mov47ildjie4k7errj3pnbaa",
        "image_id": "ocid1.image.oc1.eu-frankfurt-1.aaaaaaaa2n5z4nmkqjf27btkbdib
flwvximz5i3rsz57c3gowckozrdshnua",
        "ipxe_script": null,
        "launch_mode": "NATIVE",
        "launch_options": {
          "boot_volume_type": "PARAVIRTUALIZED",
          "firmware": "UEFI_64",
          "is_pv_encryption_in_transit_enabled": true,
          "network_type": "VFIO",
          "remote_data_volume_type": "PARAVIRTUALIZED"
        },
        "lifecycle_state": "RUNNING",
        "metadata": {
          "ssh_authorized_keys": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDRK4IASV
L1peh13mRcIwJ+23O/b82KzWPI4RdKIZMt/4p0GbuCoE4OjwXrRV01C9s5b9LsrZozf7qz6cJtYvCdSk
7oe2HyR70KMiH4SdS+tFQmqIKIJ8adtDTsQrCfRPLkU1R1q70DBROGCdqYKmY8ucCezrmjlxGUc8+gnP
QlX8yBdl5UvWcJ81rW8ElrRLTw1pZYXKK3dnDGzK3Ra0ohMr0eNmcwCY6zWVA3nNzMDd4Qc2k4uMQJPB
F/cNEjEXTxrNyza9Vn6eEieH80mDanFS70NLZgTTKNsYk1s0KD4jqsjlPlEofA7orRepLAKZozUSijaz
HmyMGHDsEkUsw7 auto_test",
          "user_data": "dW5kZWZpbmVk"
        },
        "region": "eu-frankfurt-1",
        "shape": "VM.Standard2.1",
        "source_details": {
          "boot_volume_size_in_gbs": null,
          "image_id": "ocid1.image.oc1.eu-frankfurt-1.aaaaaaaa2n5z4nmkqjf27btkbd
ibflwvximz5i3rsz57c3gowckozrdshnua",
          "kms_key_id": null,
          "source_type": "image"
        },
```


Figura 7-7: Inventario dinámico generado. Parte 4.



```
        },
        "time_created": "2019-04-15T06:42:51.766000+00:00",
        "time_maintenance_reboot_due": null
      }
    }
  },
  "all": {
    "hosts": [
      "130.61.85.64",
      "130.61.47.134",
      "130.61.113.89"
    ],
    "vars": {}
  },
  "ocid1.image.oc1.eu-frankfurt-1.aaaaaaaa2n5z4nmkqjf27btkbdibflwvximz5i3rsz57c3
gowckozrdshnua": {
    "hosts": [
      "130.61.85.64"
    ]
  },
  "ocid1.image.oc1.eu-frankfurt-1.aaaaaaaandqh4s7a3oe3on6osdbwysgddwqwyghbx4t4ry
vtcwk5xikkpvhq": {
    "hosts": [
      "130.61.47.134",
      "130.61.113.89"
    ]
  },
  "ocid1.securitylist.oc1.eu-frankfurt-1.aaaaaaaaqhylc2kahzkvyp23aywi2e65s6n22xf
jsvf7pjnqxpmjpt4dizoa": {
    "hosts": [
      "130.61.85.64",
      "130.61.47.134",
      "130.61.113.89"
    ]
  },
  "ocid1.subnet.oc1.eu-frankfurt-1.aaaaaaaao734zu3juwkcdoy4aqmo6rjkfut2yrbkfxq3r
plmwkazgaus77ma": {
    "hosts": [
      "130.61.85.64",
      "130.61.47.134",
      "130.61.113.89"
    ]
```


Figura 7-8: Inventario dinámico generado. Parte 5.



```
  },
  "ocid1.securitylist.oc1.eu-frankfurt-1.aaaaaaaaqhylc2kahzkvyp23aywi2e65s6n22xf
jsvf7pjnqxpmjpt4dizoa": {
    "hosts": [
      "130.61.85.64",
      "130.61.47.134",
      "130.61.113.89"
    ]
  },
  "ocid1.subnet.oc1.eu-frankfurt-1.aaaaaaaao734zu3juwkcdoy4aqmo6rjkfut2yrbkfxq3r
plmwkazgaus77ma": {
    "hosts": [
      "130.61.85.64",
      "130.61.47.134",
      "130.61.113.89"
    ]
  },
  "ocid1.vcn.oc1.eu-frankfurt-1.aaaaaaaawsyib66rk2hndrpov5lv6rvurjohu5slmzis2bqx
pwnuyxzxxidq": {
    "children": [
      "ocid1.subnet.oc1.eu-frankfurt-1.aaaaaaaao734zu3juwkcdoy4aqmo6rjkfut2yrbkf
xq3rplmwkazgaus77ma"
    ],
    "hosts": [
      "130.61.85.64",
      "130.61.47.134",
      "130.61.113.89"
    ]
  },
  "region_fra": {
    "children": [
      "BBhl_EU-FRANKFURT-1-AD-1",
      "ocid1.vcn.oc1.eu-frankfurt-1.aaaaaaaawsyib66rk2hndrpov5lv6rvurjohu5slmzis
2bqxpwnuyxzxxidq"
    ],
    "hosts": [
      "130.61.85.64",
      "130.61.47.134",
      "130.61.113.89"
    ]
  }
```


Figura 7-9: Inventario dinámico generado. Parte 6.

Anexo 2: Comandos de Terraform

```
[root@terraform ~]# terraform
Usage: terraform [-version] [-help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.

Common commands:
    apply              Builds or changes infrastructure
    console            Interactive console for Terraform interpolations
    destroy            Destroy Terraform-managed infrastructure
    env                Workspace management
    fmt                Rewrites config files to canonical format
    get                Download and install modules for the configuration
    graph              Create a visual graph of Terraform resources
    import             Import existing infrastructure into Terraform
    init               Initialize a Terraform working directory
    output             Read an output from a state file
    plan               Generate and show an execution plan
    providers          Prints a tree of the providers used in the configuration
    push               Upload this Terraform module to Atlas to run
    refresh            Update local state file against real resources
    show               Inspect Terraform state or plan
    taint              Manually mark a resource for recreation
    untaint            Manually unmark a resource as tainted
    validate           Validates the Terraform files
    version            Prints the Terraform version
    workspace          Workspace management

All other commands:
    debug              Debug output management (experimental)
    force-unlock       Manually unlock the terraform state
    state              Advanced state management
```

Figura 7-10: Funciones de Terraform.

Anexo 3: Componentes instalados:

Anexo 3.1: Opciones GIT

```
[root@ansible inventory-script]# git --help
usage: git [--version] [--help] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

The most commonly used git commands are:
   add        Add file contents to the index
   bisect     Find by binary search the change that introduced a bug
   branch     List, create, or delete branches
   checkout   Checkout a branch or paths to the working tree
   clone      Clone a repository into a new directory
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   fetch      Download objects and refs from another repository
   grep       Print lines matching a pattern
   init       Create an empty Git repository or reinitialize an existing one
   log        Show commit logs
   merge      Join two or more development histories together
   mv         Move or rename a file, a directory, or a symlink
   pull       Fetch from and merge with another repository or a local branch
   push       Update remote refs along with associated objects
   rebase     Forward-port local commits to the updated upstream head
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index
   show       Show various types of objects
   status     Show the working tree status
   tag        Create, list, delete or verify a tag object signed with GPG
```

Figura 7-11: Detalles GIT.

Anexo 3.2: Opciones PIP

```
[root@ansible inventory-script]# pip -h

Usage:
  pip <command> [options]

Commands:
  install                     Install packages.
  download                    Download packages.
  uninstall                   Uninstall packages.
  freeze                      Output installed packages in requirements format.
  list                        List installed packages.
  show                        Show information about installed packages.
  check                       Verify installed packages have compatible dependencies.
  config                      Manage local and global configuration.
  search                      Search PyPI for packages.
  wheel                       Build wheels from your requirements.
  hash                        Compute hashes of package archives.
  completion                  A helper command used for command completion.
  help                        Show help for commands.
```

Figura 7-12: Detalles PIP.