Apuntes 1: Martes 5 de setiembre

Diego Granados Retana 2022158363

Bases de datos II

9 de setiembre, 2023

Apuntes de la clase:

Proyecto: Todos tenemos una cuenta de Oracle, pero debemos intentar utilizar una cuenta por grupo. Cuando entramos a la cuenta, en el top right hay un profile. En profile, hay tenancy. La parte importante es entrar y extraer el OCID. Todo entre el Oracle Cloud tiene un ID el OCID. Esto está en my profile. En tenancy tenemos otro id que tenemos que copiar.

Una vez que tengamos esa información, netramos al GitHub del profe.

El archivo que se llama provider le dice a terraform que use un servicio que se llama OCI. Nos están proporcionando códigos de infraestructura para que nos quede igual a la del profe.

Tenemos que ajustar las configuraciones para que hagan match a las configuraciones que vamos a tener.

En el ReadMe, está cómo setupear el proyecto. primero ejecutamos el OCI Setup.

Hay un ReadMe,

Cuando corremos oci setup config, nos van a hacer varias preguntas.

Luego ingresamos el OCID del usuario, encontrado en my profile. Luego nos pide el tenancy id. Luego seleccionamos la región donde creamos la cuenta, como us-chicago-1. Creo que la de Diego es Arizona

Le voy a decir que sí para que me cree un keeper, que son pares de llaves públicas y privadas para una autenticación.

Una vez que haya hecho eso, ejecutamos .oci/oci/oci_api_key_public Ingresamos al API Keys y le de paste public key. Pego el contenido que se generó ahí y le doy agregar. Me genera una configuración, que es idéntica a la que el comando el pidió. Ya con esto deberíamos podernos conectar a Oracle Cloud, con el comando.

Para correr el oci, tenemos que instalar algo ahí.

Tenemos que configurar lo que dice changee me en group.tfvars.

El network tf no tenemos que tocarlo, sino que lo que hace es que nos crea máquinas virtuales con 6GB de RAM y recursos para jugar.

Nosotros podemos crear 4 máquinas virtuales con 6 gb de ram y un procesador.

Uno especifica el tipo de instancia. Todo esto está configurado praa que lo hacemos. El availability domain es el zone donde se va a hacer. El display name es el nombre de la máquina virtual que se va a usar. El shape es

flexible, que podemos especificar la cantidad de cpus que queremos usar. El source details es que se va a crear a partir de una imagen.

Source id: id del sistema operativo que vamos a instalar.

vnic_details: se crea en una red pública y que agarra un ip público.

En los metadatos, nos va a generar un ssh authorized keys, que es la forma segura de conectarse a máquinas en el cloud. El user data corre un script que se conecta.

En el templates hay un script que instala docker, los certificates para configurar los repositorios, vamos a configurar todo para poder instalar docker y correrlo en un contenedor, etc.

Para instalarle más cosas a la máquina, vamos agregando instrucciones en la máquina. Esto es en el vm01.sh

Las unidades de lectura y escritura especifican la cantidad de lecturas y escrituras que se pueden hacer. Es un feature secundario en nuestra configuración.

El storage bucket crea como un Google rive dentro de Oracle donde podemos subir archivos. En el readme, podemos lo que tenemos que cambiar, los comandos que tenemos que seguir. Una vez que terminamos, ejecutamos una instrucción terraform init que descargan los paquetes de software para poder hablar con Oracle Cloud.

terraform apply se pega al oracle cloud, verifica qué está creado y qué no, y lo crea.

Después de que corremos eso, nos conectamos a lo que tira y estaríamos adentro de la máquina virtual del Oracle Cloud.

Para correr otra máquina virtual, reemplazamos el vm1 por vm2. Luego corremos terraform apply para que detecte los cambios y levante la otra máquina.

Cuando creamos otra máquina, nos metemos a oracle cloud y le doy overview, vemos cómo se empieza a crear. Podemos conectarnos con ModuExtern, SSH, etc.

La configuración que nos pasaron no llevaban los arcihvos de terraform tfstate y backup, ya que esto se genera contra la estructura y contra la cuenta. Siempre de correr terraform, hay que hacer un push de esos archivos. El comando para entrar a la máquina empieza con ssh -i …

El script que estamos ejecutando dura un poco en correr, entonces hay que darle unos cuantos minutos. Para ver por dónde va corriendo sudo -i tail -f /var/log/cloud-init-output.log

Una vez que termina de ejecutarse, cuando entremos al IP, la máquina va a responder un puerto 80, pero para que haga eso tiene que terminar de hacer todo.

Lo del puerto 80 es que corriera el archivo de Flask en ese puerto. Cuando uno hace eso, se mappea a todos los ips asociados.=, por lo que se mappea al puerto público también.

Si tenemos que borrarla, nos metemos al archivo y comentamos todo desde el resource instance y hacemos un terraform apply.

Hay que procurar apagar la infraestructura para que no se gasten los $300 de crédito gratis que tenemos.

Para borrar todo, hacemos terraform destroy.

En esta etapa, no impronta la seguridad. No nos vamos a cuidar con manejar la forma correcta de manejar las llaves de bases de datos, etc. Podemos empaquetar las credenciales para conectarse a la base dentro de la Docker image. Los passwords pueden estar quemados. Las IPs de las máquinas virtuales están permitidas de entrar a la base de datos.

Donde se crea la base de datos es en db.tf, ponemos un whitelisted ips.

Para ver nuestras computadoras, cogemos el ip público en whatismyip y lo ponemos en la configuración pública de bases de datos.

Podemos tener 4 máquinas virtuales y cada uno trabaja en su máquina virtual.

En el db.py, nos metemos a la página de Oracle y lo metemos al db.py

En MongoAtlas, nos registramos, signup with Google.

Creamos un cluster, free, aws, lo dejamos todo por defecto y le damos create cluster. Luego, se empieza a crear el cluster. Ponemos un username y un password. Guardamos la información y se nos va a estar creando un cluster nuevo. Ya con eso vamos a tener un Mongo Atlas corriendo en la aplicación. Como es un free tier, no va a ser muy bueno, pero es suficiente.

Para lo de Firebase, es gratuito. Nos metemos a go to console con la cuenta de Google. Le damos a project. Le damos continue, le quitamos lo primero que sale, y en 5 minutos crea toda la infraestructura para Firebase. Tenemos una base de datos de realtime y le damos create, next, create, el lock mode,

En autenticación, activamos el email/password. Lo importante es que trabajemos con las bases de datos. A lo menos que se le da peso es a lo de React. VuJS trae cosas nativas para conectarse con React, entonces puede ser una buena opción. La infraestructura como código ahí estás, entonces debería funcionar.

Cuando Mongo Atlas termine, le damos a create index en Atlas Search. Le damos load sample data y cuando lo carguemos, le damos browse collections. Creamos una nueva colección para Wikipedia y empezamos a meter datos. Si quiero crear un índice de búsqueda sobre la colección, le doy create search index, visual editor, agarro cualquiera y está todo l oque hay que hacer. Crea el índice de búsqueda. El índice no hay que refrescarlo. Nosotros metemos documentos, tienen consistencia eventual y en algún momento el índice lo agarra y lo indexa.

Para esta vez, podemos poner el private key del oci.

Siempre que tenemos bases de datos, tenemos que buscar un dataset. Vamos a usar los dumps de wikipedia. En los dumps de Wikipedia, hay archivos bastante grande. Toda la informacio´n de wikipedia se encuentra en archivos multistring. El tamaño alrededor es de 20 gigas compresos. Si descomprimimos es ocmo 80 gigas de texto. No vamos a cargar todo el texto. Como es texto se presta mucho para hacer algo interesante con Bds como Mongo. Vamos a bajar algunos de estos archivos para procesarlos. Estos archivos tienen todos los artículos de Wikipedia. Como tenemos suficiente memoria en Oracle, deberíamos poderlos procesar bien. Si abrimos el archivo y vemos una parte,

Hay otro pedazo de código que dice que utilicemos otras bibliotecas. La de wiki processor lee los archivos y se encarga de parsear. Agarra los archivos y saca la información que nosotros ocupemos. Tiene clases. Wiki.py saca los títulos, redirect, id y namespace. Agarra el título de cada una de las páginas.

SiteInfo: Saca el nombre del sitio, la db, los namespaces, etc. Page: Revision: última modificación que se le hizo a la página. Sale el user, el timestamp, la página, el comentario, el texto final, los bytes,

Tenemos que agarrar la biblioteca y parsear esos archivos. Al parsear esos archivos generamos un documento que representa la página. Luego cargamos el documento que representa esa página.

Descargamos los archivos descomprimidos. Los cargamos manualmente en object storage. Esto se hace en Oracle. Abrimos Oracle, ponemos Object storage, nos metemos a buckets, click en el que creamos, upload y agarramos el archivo descomprimido de Wikipedia. Una vez que subimos el archivo, queda en Oracle Storage. Los archivos van quedando arriba y después el loader va a estar revisando el object storage cada cierto tiempo. Parsea el archivo con la biblioteca.

Luego, el loader va revisando el object storage. Cuando detecta un archivo nuevo, lo parsea con la biblioteca. Luego genera un documento que lo guarda en mongo atlas y lo guarda en autonomous database. Esto es hacer un insert igual a ambas bases de datos. Tenemos que implementar un mechanismo para no procesar un archivo otra vez. El api se conecta a las bases de datos y hace una búsqueda. Con grandes cantidades de datos, el rendimiento de mongo atlas debería ser mejor a Autonomous Database. Autonomous con un full text index debería ser muy rápido. Cuando hacemos la búsqueda contra Oracle, mientras resolvemos todas las llaves foráneas y todo, las búsqeudas son muy lentas. Mongo normalmente es mejor que una base de datos SQL porque a Mongo le pegan mongo search que utiliza una tecnología que se llama Lucene. Le meten un índice a la par. Hay muchos factores que afectan. No estamos midiendo manzanas con manzanas porque estamos trabajadores con todo gratis.

Al API hace búsquedas sobre el Mongo Atlas y el autonomous database.

El ui nos permite ingresar con el usuario y password en firebase. cuando entro me muestra una interfaz, seleccion en qué tipo de motor voy a buscar. Pongo la información de búsqueda, me salen los facets, que son características de la búsqueda. Es un feature de Mongo Atlas. El texto en amarillo es un highlight del match. Mongo Atlas también tiene un endpoint que nos permite hacer eso. Es simplemente hacer una llamada a los endpoints de Mongo Atlas. Mongo tiene stages. Es meter un stage más que nos devuelve en qué desplazamiento está eso. Otra pantalla tiene un texto con toda la información.

En oracle nos permite meter 25 gigas. Podemos usar un par de archivos. Lo que importa es las conexiones entre bases de datos.

Tenemos que crear el dockerfile, agregar una nueva máquina y en el template, al puro final especificamos el nombre de la imagen que tenemos que ejecutar. Eso automáticamente levanta la máquina.

Hay facets de fechas, numéricas, textuales, etc. Por strings puede ser clasificar los productos por Marca.

Las máquinas de Oracle no utilizan una arquitectura x64. Si a alguien se le ocurre utilizar otra cosa que no sea Python con flask, hay que asegurarse de usar la versión de Docker correcta.

Debe estar instrumentada con Prometheus. Sería interesante ver cuál es el tiempo promedio cuando utilizamos Autonomous database y cuando utilizamos Mongo Atlas. Es simplemente que expongamos el endpoint, podamos hacer un curl y extraer eso. De puntos extras, le podemos poner certificados extras, etc.

El link directo a wikipedia se genera muy fácil al copiar un URL y pegarle el número de artículo. Si yo quiero pegar el title con la finromacion del archivo, tengo que quitarle toda la información al Title y cuando lo busco, me debería salir la página.

Con el archivo de multistring index podemos hacer el match. La cantidad de page links se encuentra en el mismo archivo. Tenemos que contar la cantidad de links. Toda la información que vemos la mostramos en el documento. Hacemos un modelo de entidad relación pequeño y lo insertamos en Autonomous database. El texto que tiene WIkipedia por detrás es Wikitext. El texto que tenemos en el archivo es lo que se renderiza. hay bibliotecas de react que uno le mete el wiki text y nos genera el HTML que genera el wikitext de la página.

Le tenemos que hacer pruebas unitarias a los endpoints, eso es todo lo que ocupa. El mapping de Mongo Atlas es decir porqué escogimos los tipos así.

El código que tenemos que crear es bastante básico. Es más difícil pegar lo de Spacy. La infraestructura que se genera automáticamente no nos debería causar problemas.

Deberíamos empezar con procesar los archivos de Wikipedia. Es el heavy lifting de la configuración. Crearía el Python que genera procesa los cosos y lo sube a las bases de datos y luego lo subimos a Docker. Creamos un Python Virtual Environment y lo corremos localmente.