

TRABAJO PRACTICO FINAL PPII



Alumno: Manuel Nuñez

Fecha de entrega: 29/11/2023

Materia: Practicas Profesionalizantes 2

Docente: Gabriela Ruffini

- Introducción al Cloud Computing
 - ¿Qué es el CC?
 - ¿Cuáles son las plataformas más reconocidas a nivel mundial?
 - Tareas realizadas sobre la creación de una VM en Google Cloud Platform
 - Potencial de las VM de GCP
 - AppSheet
 - RAD Lab
- Introducción al proyecto VMCloud
- Desarrollo del proyecto
 - Preparación de Django
 - Preparación de la Base de Datos
 - Preparación de las URLs
 - Diseño y distribución de las vistas
 - Implementación
- Conclusión
- Bibliografía

Cloud Computing

- ¿Qué es el CC?: El Cloud Computing es un *modelo de servicios de infraestructura* que dispone recursos flexibles y escalables, es decir, que se adaptan a las necesidades del usuario o empresa, por medio de la internet. Esto evita que las empresas, las cuales necesitan una infraestructura óptima para funcionar, gasten en recursos físicos, servicios de mantenimiento y configuración de los mismos, además de tener que invertir cada cierta cantidad de tiempo en nuevos recursos actualizados. Con “CC” las empresas solo necesitan una *buena conectividad a internet para operar* ya que un servidor central gestiona toda la infraestructura solicitada por el usuario.
- Ventajas del CC:
 - Es flexible: Es posible acceder desde cualquier computadora conectada a internet, además de que permite escalar los recursos utilizados.
 - Es eficaz: Permite desarrollar aplicaciones y ponerlas en producción sin necesidad de infraestructura.
 - Ofrece un valor estratégico: Ofrece un mayor retorno de la inversión que dispone la utilización de los recursos, a diferencia de la adquisición de recursos físicos que pueden quedar obsoletos.
 - Es seguro: No hay necesidad de resguardar recursos físicos en la empresa, porque no los hay, en cambio los grupos de servicios de CC disponen de expertos en seguridad.
 - Es rentable: Las empresas solo pagan por los recursos utilizados del servicio.
- ¿En qué situación es necesario implementar CC?
 - Cuando el crecimiento de la empresa supera la capacidad de la infraestructura.
 - Baja utilización de los recursos de infraestructura disponible.
 - Cuando los datos almacenados superan los recursos de almacenamiento locales.
 - Cuando hay tiempos de respuesta lentos con la infraestructura local.
 - Cuando la empresa sufre retrasos en los ciclos de desarrollo debido a las limitaciones de la infraestructura local.
 - Cuando el gasto en inversiones de infraestructura local es muy grande.
 - Cuando hay usuarios movibles o distribuidos.
- ¿Para qué sirve el CC?:
 - Escalado de infraestructura: Se pueden adaptar los recursos a la variación de necesidades de la infraestructura.
 - Recuperación tras fallos: Las empresas pueden hacer copias de seguridad de sus recursos digitales.
 - Almacenamiento: Dispone grandes volúmenes de datos.

- Desarrollo de aplicaciones: Rápido acceso a herramientas de desarrollo para crear, probar, y lanzar.
- Análisis Big Data: Recursos casi ilimitados para procesar grandes volúmenes de datos y agilizar el tiempo para buscar información valiosa.
- ¿Cuáles son las plataformas CC más reconocidas a nivel mundial?:
 - Amazon Web Services: Es el servicio CC de Amazon. Ofrece *flexibilidad y facilidad de uso*, ya que permite elegir desde el sistema operativo, hasta el lenguaje de programación que se va a utilizar. *Es rentable* ya que se cobra por la potencia de los recursos utilizados y su tiempo de procesamiento. Puede ganarse *velocidad de organización*, dado que se reduce el tiempo de espera de disponibilidad de recursos para desarrollo. Permite *escalabilidad y elasticidad*, porque la aplicación puede ampliarse según la demanda de recursos y almacenamiento. En cuanto a *seguridad* cuenta con *certificaciones y acreditaciones* para administrar la infraestructura.
 - Google Cloud Platform: Ofrece *precios accesibles* y mejores que el de la competencia, ya que se paga solamente el tiempo que se utilizan sus servicios. El *rendimiento* de los recursos se da con respecto al importe que paguemos. Está al día con las últimas *novedades sobre herramientas* modernas. Tiene *seguridad cifrada de extremo a extremo*, utiliza redes privadas de Google, lo que hace muy poco probable los ciberataques. Dispone una *alta restauración de los datos*, porque reduce los tiempos del proceso.
 - Microsoft Azure: Es una nube *apta para todo tipo de empresas*, cuenta con soluciones focalizadas con combinaciones de productos y servicios. Corre en *cualquier servidor físico o en la nube*, dispone de una nube física y otra privada, lo que posibilita que las aplicaciones se compartan entre ellas, las necesidades se adecuan a la empresa que las utilice. La *seguridad se da con inteligencia artificial* proactiva, los datos se resguardan cifrados. Permite *ambientes multi-cloud*, es decir que, se pueden combinar soluciones AWS, GCP e infraestructura local en un solo lugar. Para el desarrollo facilita el camino, ya que *utiliza el mismo lenguaje para las aplicaciones de Windows*.
- Mientras que AWS ofrece amplia gama de recursos y ser *líder en la industria del CC*, siendo dominante en diversas cargas de trabajo y escenarios empresariales; *Google Cloud se especializa en la innovación* y es adecuado para proyectos centrados en datos y aprendizaje automático, ya que tiene un fuerte análisis de datos, machine learning, y servicios gestionados. Pero Azure no se queda atrás, dado que ofrece *servicios integrados con productos de Microsoft*, y está destinado a organizaciones que utilizan servicios de Microsoft.
- ¿Para qué tipo de usos es más adecuado cada plataforma?:
 - Grandes volúmenes de datos – AWS, GCP, Azure.

- Marketing digital – AWS, Azure, GCP.
 - Comercio electrónico – AWS, GCP, Azure.
 - Juegos – AWS, GCP.
 - Gobiernos – AWS, Azure (federal, Estado y locales)
 - Internet de las cosas (Internet of things) – AWS, GCP, Azure (mantenimiento predictivo, control remoto)
 - Nubes privadas – AWS, GCP, Azure.
 - Reseller Hosting – AWS, GCP, Azure.
- Tareas realizadas en GCP sobre la creación de una VM:
 - ❖ Primero hay que ingresar a la página de google cloud <https://cloud.google.com/?hl=es>
 - ❖ Luego debemos entrar en el área de *consola*.

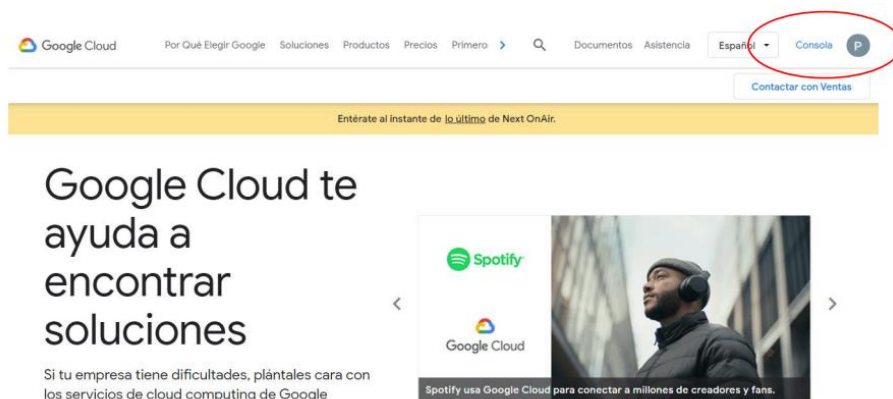


Ilustración 1: Demostración grafica para acceder a "Consola"

- ❖ Luego debemos seleccionar el proyecto PPII2023.

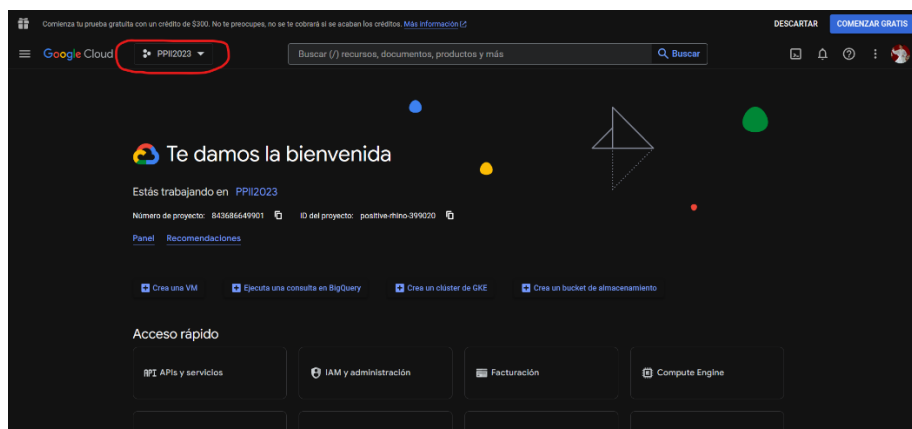


Ilustración 2: Demostración grafica para acceder al proyecto

- ❖ Una vez dentro del proyecto vamos al menú de la aplicación y seleccionamos la herramienta “*Compute Engine*”, y del menú desplegado luego de esta acción, seleccionamos “*Instancias de VM*”.

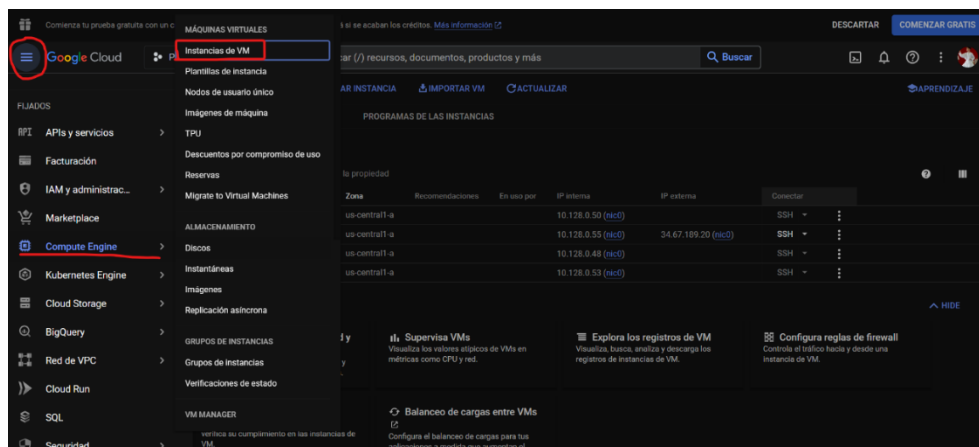


Ilustración 3: Demostración grafica para acceder a las "instancias de VM"

- ❖ Al ingresar a las instancias vamos a ver todas las creadas por el usuario o el grupo de usuarios del proyecto. Para crear una nueva instancia debemos seleccionar la opción de “*Crear Instancia*”

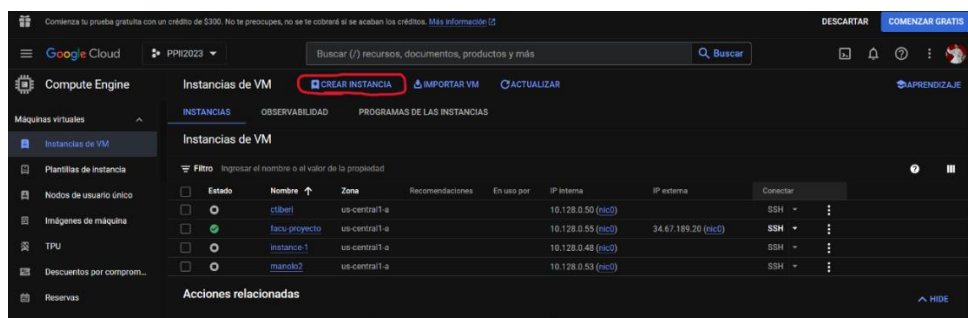


Ilustración 4: Visualización de las instancias creadas y ubicación de la creación de nuevas.

- ❖ Una vez dentro de la creación de la instancia le damos un nombre, configuramos la región, le asignamos VM configuración de “*uso general*”, procesamiento de bajo costo (E2), y el tipo de CPU elegimos “*e2-medium*”.

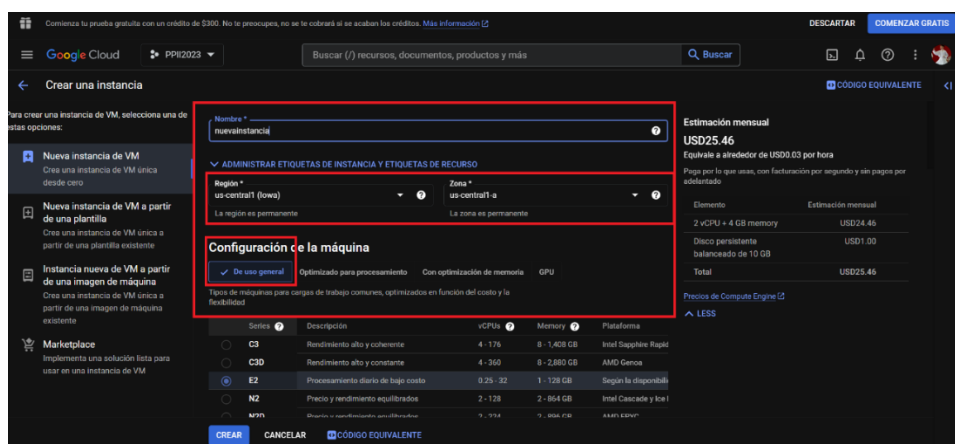


Ilustración 5: Asignación de nombre, región y configuración de la VM.

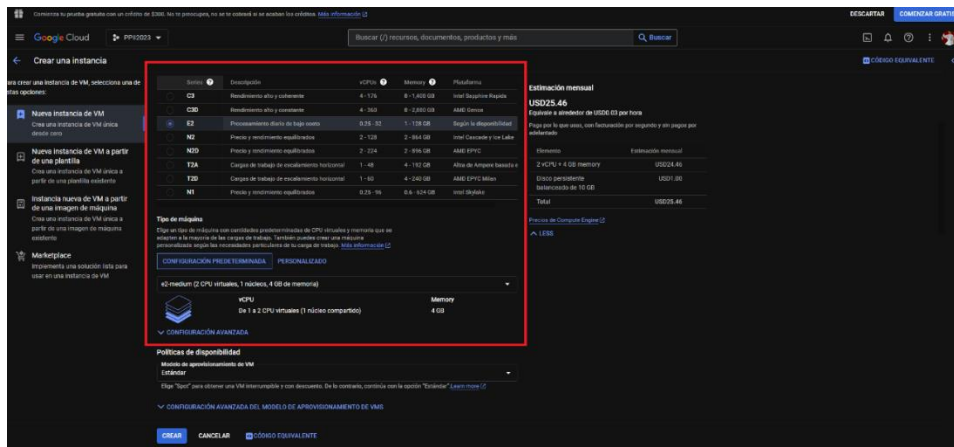


Ilustración 6: Asignación de procesamientos y tipos de CPU a la VM.

- ❖ Luego debemos abrir en otra pestaña la creación de los discos, para poder crear un disco el cual le vamos a asignar a la VM.

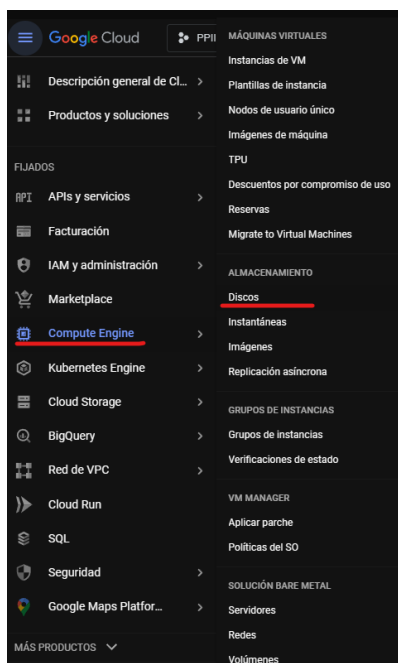


Ilustración 7: Ubicación de para la gestión de discos.

- ❖ Presionamos en crear, le damos un nombre, configuramos la misma región que tiene la VM, y le asignamos una imagen de S.O. (Ubuntu-2004).

MANUEL NUÑEZ

TRABAJO PRACTICO FINAL (PRACTICAS PROFESIONALIZANTES II)

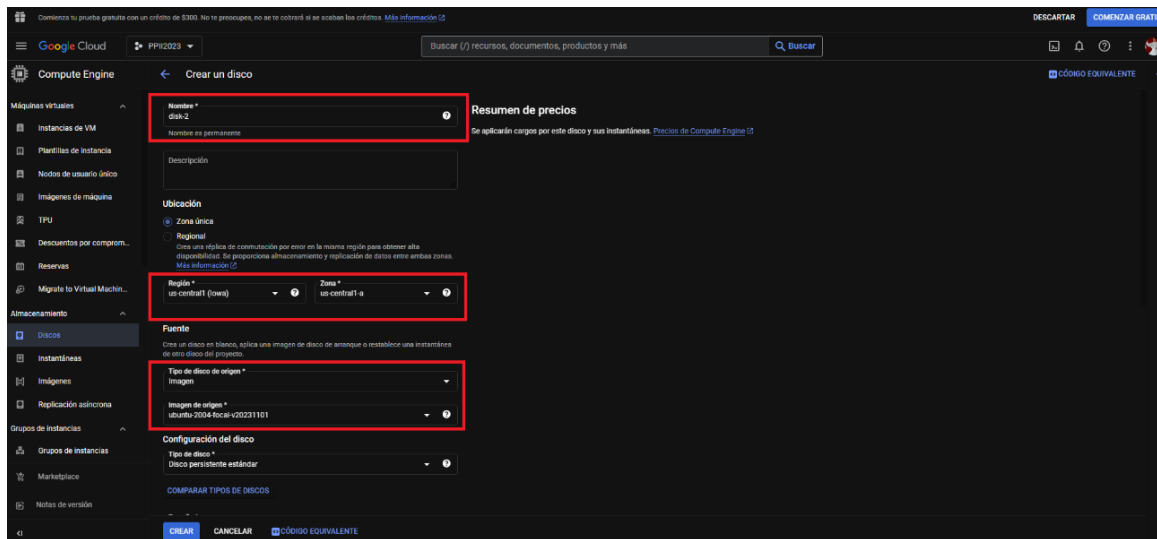


Ilustración 8: Detalles de la creación del disco.

- ❖ Una vez creado el disco, volvemos a la pestaña donde tenemos la creación de la VM, vamos al apartado de unidad de disco, y presionamos en cambiar.

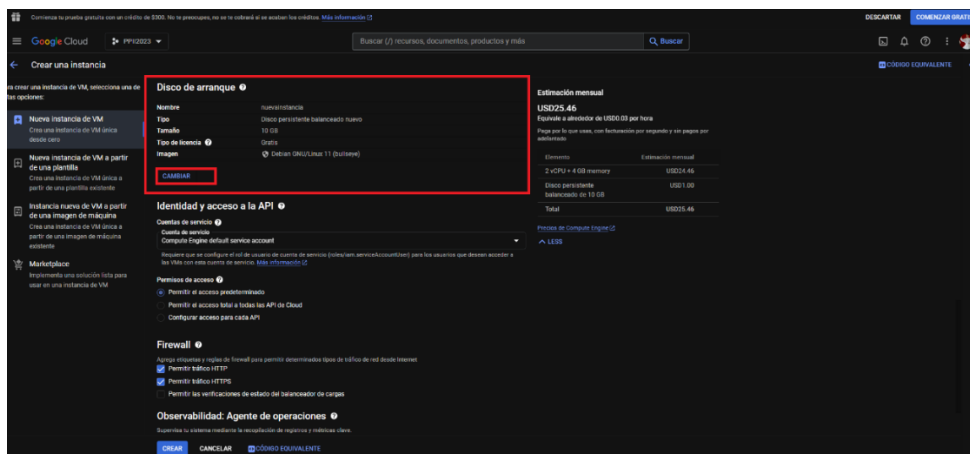


Ilustración 9: Visualización de configuración de disco, en la creación de la VM.

- ❖ Dentro de la selección del disco a utilizar, nos dirigimos a la solapa “Discos existentes”, y seleccionamos el disco creado.

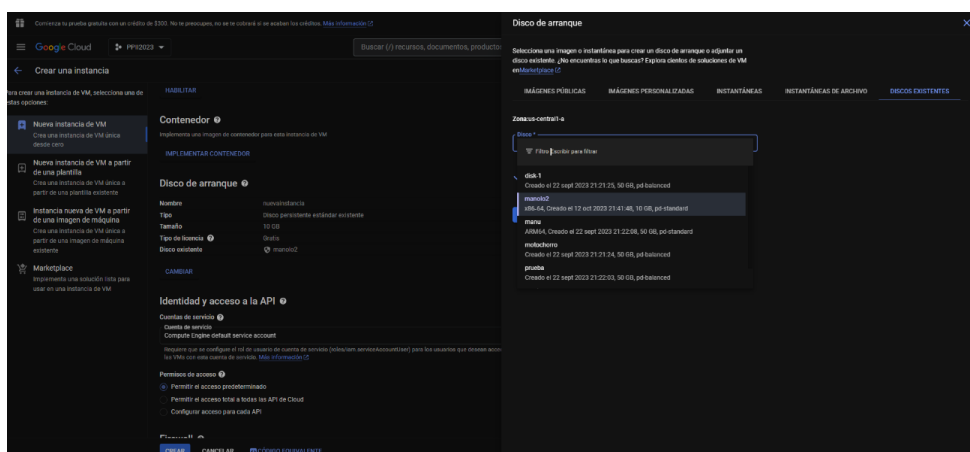


Ilustración 10: Asignación del disco creado a la VM

- ❖ Luego de generar el disco para la VM quedan algunas configuraciones antes de guardarla, en permisos de acceso debemos elegir “*acceso predeterminado*”. Y para el firewall debemos permitir los tráficos “*HTTP*” y “*HTTPS*”.

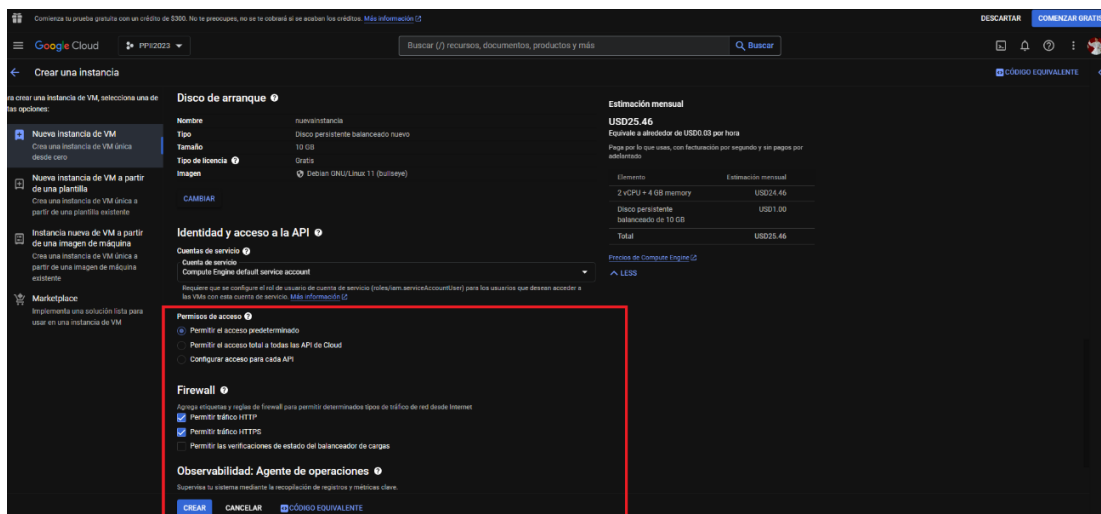


Ilustración 11: Asignación de permisos de acceso y trafico permitido en el firewall.

- ❖ Al crear la VM volvemos a las instancias y ubicamos nuestra nueva VM, debemos presionar en los tres puntos del registro de la VM (más opciones) para encenderla, y una vez esté encendida presionar en “*SSH*” para conectarse a la consola de la instancia.

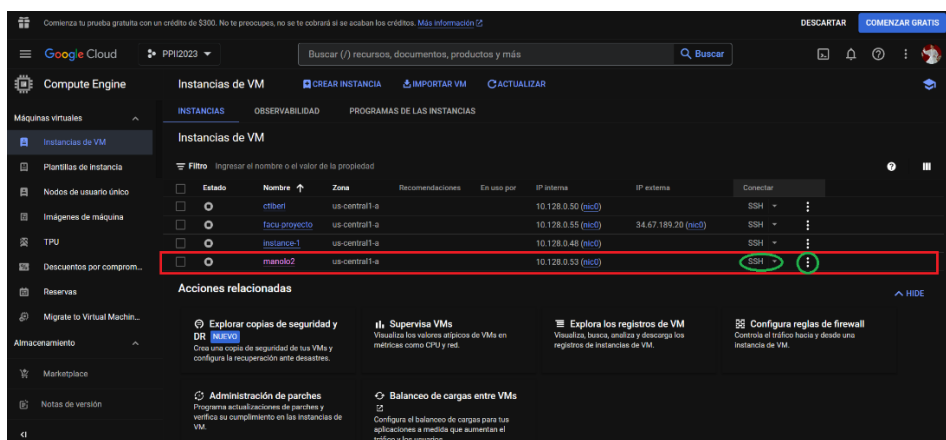





Ilustración 12: Encendido y control de la VM.

- ❖ Se nos abrirá una ventana emergente del sitio en la cual va a conectar a la consola de la VM y desde ahí vamos a poder manejarla. Si ingresamos el comando “*sudo apt-get install ubuntu-desktop*”, podremos instalarle la interfaz gráfica del S.O.



- Potencial de las VM de GCP:
 - El uso de las VM suele ser muy variado y dinámico, por lo que el potencial que tienen es muy grande, obviamente dependiendo de la cantidad y calidad de recursos que asignemos a las VM.
 - Si tomamos como ejemplo la VM creada en el apartado anterior, con una configuración muy pobre en cuanto a recursos, teniendo en cuenta que no tenemos ni siquiera interfaz gráfica. Por lo que podremos usarla de servidor de archivos pequeños, como archivos de texto o documentos no tan extensos.
 - Si configuramos una VM con suficiente procesamiento y memoria, podremos obtener una terminal para *desarrollos y pruebas*.
 - Si configuramos una gran capacidad de almacenamiento y procesamiento, podremos realizar tareas de *Análisis de datos*.
 - O si por ejemplo configuramos una VM con almacenamiento persistente y redes optimizadas, podremos utilizarla para realizar tareas con bases de datos distribuidas.
- AppSheet:
 - AppSheet es una plataforma de desarrollo sin código que permite que cualquier usuario pueda crear aplicaciones web y móviles sin tener experiencia en programación. Las aplicaciones de AppSheet se pueden crear a partir de fuentes de datos, como Hojas de cálculo de Google, Excel, Cloud SQL, Salesforce y otros conectores similares. La actividad de los usuarios de la aplicación se sincroniza con las fuentes de datos conectadas.
 - Las aplicaciones son dinámicas y se pueden utilizar en todos los dispositivos móviles o navegadores.
 - La interfaz visual se configura en base a plantillas que la misma aplicación proporciona.
 - Se pueden automatizar eventos por medio de una IA, por ejemplo, el envío por mail de un certificado, configurando un documento con variables, la IA genera automáticamente todos los datos en base a las Hojas de cálculo y lo envía al mail definido por el programador.
 - Antes de iniciar un nuevo proyecto, AppSheet proporciona “templates” de aplicaciones, para facilitar la interfaz en base a las necesidades del desarrollador.
 - La programación es sencilla, no es necesario saber de lenguajes de programación sofisticados, el lenguaje es muy similar a los comandos en las hojas de cálculo.
- RAD Lab de GCC:
 - A diferencia de AppSheet que es un entorno de desarrollo de aplicaciones sin código. RAD Lab es un entorno de sandbox basado en la nube que proporciona una infraestructura pre

configurada y segura para que los equipos puedan probar y desarrollar sus aplicaciones y servicios.




- El RAD Lab ofrece las siguientes ventajas:

-  **Aceleración:** El RAD Lab permite a los equipos desplegar una infraestructura de nube completa en minutos, lo que les permite comenzar a desarrollar sus aplicaciones más rápido.
-  **Seguridad:** El RAD Lab está diseñado para cumplir con los requisitos de seguridad más estrictos, lo que brinda a los equipos la tranquilidad de que sus datos están protegidos.
-  **Flexibilidad:** El RAD Lab es personalizable, por lo que los equipos pueden adaptarlo a sus necesidades específicas.

- Desventajas:

-  Requiere conocimientos de desarrollo de aplicaciones.
-  Puede ser complejo de configurar y administrar.

- Ejemplos de organizaciones y usos:

-  **Empresas de tecnología:** Las empresas de tecnología pueden utilizar RAD Lab para desarrollar nuevas aplicaciones y servicios, probar nuevas tecnologías y mejorar la eficiencia de sus operaciones.
-  **Organizaciones de investigación:** Las organizaciones de investigación pueden utilizar RAD Lab para desarrollar nuevas aplicaciones de aprendizaje automático, análisis de datos e IoT.
-  **Gobiernos:** Los gobiernos pueden utilizar RAD Lab para desarrollar aplicaciones gubernamentales seguras y eficientes.

Proyecto de virtualización VMCloud

Introducción:

- En este apartado se van a detallar los procedimientos realizados durante el desarrollo de la aplicación “VMCloud”, que se trata de una web que le permite al usuario *crear y utilizar máquinas virtuales sin necesidad de recursos físicos locales*. Pudiendo personalizar cada uno de sus recursos que se dividen en:
 - Máquinas Virtuales.
 - Discos.
 - Redes.
- La página integra *Python (Django)*, junto con la *interfaz de comandos* de consola que provee la aplicación de *Virtual Box*. Lo que permite programar una interfaz sencilla, en la web, para que

el usuario no tenga complicaciones gráficas y pueda comenzar a operar con sus VM en un lapso de tiempo corto.

- El alcance que tiene VMCloud, es variado. Porque puede ser utilizado e implementado tanto en un servidor local con buenos recursos, para compartir las VM a la red de toda la empresa. O también puede instalarse en un servidor web, para ofrecer un servicio de virtualización alternativo a las empresas de Cloud Computing ya existentes.
- Una de sus principales limitaciones, es que no cuenta con una visualización de las VM propia, sino que al encender cada una, le devuelve al usuario un *id de AnyDesk*, el cuál debe ser ingresado en ésta aplicación y así el usuario podrá tomar el control de la VM.

Virtual Box:

Desarrollado por Oracle Corporation, Virtual Box es una *herramienta de virtualización* potente y versátil que se utiliza comúnmente para crear entornos virtuales en *sistemas operativos de host*, lo que permite ejecutar múltiples sistemas operativos en una sola máquina física. Ésta utiliza los recursos físicos de la computadora en la que se encuentra instalada, y permite seleccionarlos y administrarlos según las necesidades de los usuarios. Es decir que, si instalamos Virtual Box en una computadora de bajos recursos, vamos a obtener máquinas virtuales limitadas en cuanto a procesamiento, memoria y almacenamiento.

AnyDesk:

AnyDesk es una aplicación de *software de escritorio remoto* que permite a los usuarios acceder y controlar computadoras de forma remota desde cualquier lugar. Es especialmente útil para soporte técnico, colaboración en equipo y acceso remoto a dispositivos. *Mediante un ID o un Alias*, el usuario tendrá acceso a la visualización y control del escritorio del destino deseado.

Django:

Django es un *framework de desarrollo web* de código abierto *escrito en Python*. Fue creado para facilitar el desarrollo rápido y limpio de aplicaciones web complejas, al proporcionar una serie de herramientas que permiten a los desarrolladores construir aplicaciones de manera eficiente. Entre ellas las que vamos a utilizar en este proyecto:

➤ Modelo-Vista-Controlador (MVC):

Django sigue el patrón de diseño Modelo-Vista-Controlador (MVC), aunque en Django se le conoce como *Modelo-Vista-Template (MVT)*. Este patrón ayuda a organizar el código de una

manera modular y escalable. El cuál dispone Vistas y Modelos codificados en Python, y templates codificados en HTML.

➤ ORM (Object-Relational Mapping):

Django incluye un ORM que permite a los desarrolladores *interactuar con la base de datos utilizando objetos de Python* en lugar de escribir SQL directamente. Esto facilita la manipulación de datos y reduce la complejidad en la interacción con la base de datos.

➤ Rutas y Vistas:

Django utiliza un enfoque basado en configuración para definir rutas y vistas, facilitando la *creación de URLs y la respuesta a las solicitudes del usuario*.

Desarrollo del proyecto:

Se comenzó la planificación del proyecto seleccionando los comandos por consola de Virtual Box necesarios para cumplir el objetivo del desarrollo: crear máquinas virtuales, y sus recursos, de una manera sencilla.

El módulo de comandos se llama *VBoxManage*, y se encuentra ubicado por defecto en la carpeta “C:\Program Files\Oracle\VirtualBox”. En el siguiente listado va a especificarse la acción de cada comando:

- `VBoxManage import "archivo.ova" --options=importtovdi --vsys=0 --vmname="Nombre de la VM" --unit=0`
- El comando importa desde un archivo *ova* una imagen guardada de una máquina virtual. En este caso las máquinas virtuales disponibles del proyecto, se pre configuraron con AnyDesk y se exportaron para que al momento de “crear” la VM, el usuario pueda acceder con la aplicación remota.
 - Esto le ahorra al usuario la instalación del sistema operativo y le provee una máquina virtual instantáneamente.
 - Existe un comando de creación de VM, pero éste se descartó, ya que crea una imagen vacía y el usuario debería instalarla desde cero.
 - Ésta importación viene junto al disco con el que se había configurado la VM previo a la exportación, *--importtovdi* convierte el disco en un formato legible para Virtual Box, para que la VM cree un disco automáticamente.
 - *--vsys=0* indica que la VM al importarse, debe utilizar el sistema operativo guardado y su configuración. *--vmname* permite indicar al usuario el nombre que va a tener la VM. *--unit=0* indica que la importación debe incluir el disco con el que fue guardada la VM.

- `VBoxManage modifyvm "nombre o id" --cpus 4 --memory 4500 --vram 80.`
 - Esta orden permite modificar algunas de las propiedades de la VM, especificando su nombre o id, podremos cambiar de manera flexible sus principales características.
 - `--cpus` indica el número de CPUs que la VM va a disponer.
 - `--memory` indica el número de memoria RAM (en Mb) que va a tener.
 - `--vram` indica el número de memoria de video que la VM puede brindar.
- `VBoxManage showvminfo "nombre o id"`
 - Especificando el nombre o id de la VM podremos ver información detallada sobre sus puertos de booteo, su ubicación en el host, sus adaptadores “físicos” tanto de red, como de discos, su sistema operativo, sus recursos como las CPUs, RAM, VRAM, entre otros.
- `VBoxManage modifyvm "nombre o id" --nic1=natnetwork --nic-type1=82540EM --cable-connected1=on --nic-boot-prio1=1 --nat-network1="nombre de la red"`
 - Indica el tipo de red y la red que la VM va a utilizar.
 - `--nic1=natnetwork` se traduce a que el adaptador 1 va a estar configurado con un tipo de red NAT.
 - `--nic-type1` indica el tipo de adaptador utilizado.
 - `--cable-connected1=on` significa que la red del adaptador 1 va a estar conectada.
 - `--nic-boot-prio1` indica el orden de arranque que tienen los cinco adaptadores de red que dispone la VM.
 - `--nat-network1` permite al usuario elegir un nombre para identificar qué red NAT va a utilizar la VM. La NAT debe estar previamente cargada.
- `VBoxManage debugvm "nombre o id" osdetect`
 - El comando muestra el sistema operativo de la VM, solo si ésta se encuentra encendida.
- `VBoxManage natnetwork add --enable --netname="nombre de la red" --network="prefijo ipv4" --dhcp=on/off --ipv6=on/off`
 - Es el comando que se utiliza para crear redes NAT y configurarlas. Una red NAT permite traducir desde la *ip privada del host*, una especie de subred para las VM, la ip de éstas va a depender del prefijo de ipv4 que tenga configurada la NAT.
 - `--enable` configura a la red como activada.
 - `--netname` le asigna un nombre a la red.
 - `--network` permite configurar un prefijo ipv4, por ejemplo *10.0.2.0/24*.
 - `--dhcp` dependiendo de si es on, indica que el dhcp está activo para la red, o si es off, el dhcp está inactivo y hay que configurar una ip estática desde la VM.
 - `--ipv6` activa si está en on, desactiva si está en off, el protocolo de ipv6 para la VM.

- `VBoxManage modifymedium disk "nombre o id" --resize=90100 --move="nuevo directorio"`
 - Para modificar las propiedades de los discos, utilizamos este comando, especificando el nombre o id del disco.
 - `--resize` reconfigura la capacidad de almacenamiento del disco, siempre debe ser ascendente, no se le puede quitar espacio.
 - `--move` se utiliza para cambiar el nombre del disco o el directorio en el que se ubica en el host.
- `VBoxManage showmediuminfo disk "nombre o id"`
 - Indicando el nombre o el id de algún disco, es posible ver información detallada del mismo. Ya sea su ubicación en el host, su capacidad, el tamaño que está en uso y la VM en la que está asignado.
- `VBoxManage natnetwork modify --dhcp=on/off --netname="nombre de la red" --network="prefijo ipv4"`
 - Sus atributos son iguales al de la creación de una red NAT, a diferencia que, en este caso, sirve para modificar las propiedades de una ya existente.
- `VBoxManage unregistervm "id o nombre" --delete --delete-all`
 - El comando junto con sus atributos, hacen un borrado completo a nivel archivo de la máquina virtual, incluido el disco.

Finalizado el análisis y las pruebas de función de cada comando, se verificó y se analizó la posibilidad de integrar estos comandos con Python. Al hacerlo se pudo descubrir la facilidad de la tarea, ya que Python tiene un módulo completo dedicado al manejo de línea de comandos de sistemas operativos. Por lo tanto, con, simplemente codificar:

- `import os` #Importa el módulo de Python especificado
- `os.chdir('directorio de vboxmanage')` #Dirige el código al directorio
- `os.system('VBoxManage etc. .')` #Ejecuta un comando

Ya tenemos comandos VBox integrados con Python, y con ello, la etapa de planificación finalizada.

Preparación del entorno Django:

Una vez finalizada la planificación, se procede a explicar brevemente, cómo fue la creación y configuración del entorno Django para comenzar a programar.

1. Verificamos que se encuentre Python instalado en la máquina, si no lo está, debemos dirigirnos al [sitio oficial](#).

2. Una vez instalado el lenguaje nos dirigimos a la consola “cmd” o “powershell”, e instalamos Django con cualquiera de los siguientes comandos:
 - pip install Django
 - python -m pip install Django
 3. Luego nos dirigimos a cualquier directorio en el sistema, por ejemplo, el “C:\”, y creamos el repositorio del proyecto ingresando este comando:
 - django-admin startproject VMCloud
- Se crea un directorio con el siguiente contenido:

```
Directory: C:\VMCloud

Mode                LastWriteTime         Length Name
----                -
d-----          28/11/2023    22:12             VMCloud
-a----          28/11/2023    22:12             685 manage.py

PS C:\VMCloud> cd VMCloud
PS C:\VMCloud\VMCloud> dir

Directory: C:\VMCloud\VMCloud

Mode                LastWriteTime         Length Name
----                -
-a----          28/11/2023    22:12              0 __init__.py
-a----          28/11/2023    22:12             407 asgi.py
-a----          28/11/2023    22:12            3347 settings.py
-a----          28/11/2023    22:12             770 urls.py
-a----          28/11/2023    22:12             407 wsgi.py
```

Ilustración 13: Contenido del proyecto creado "VMCloud"

4. El siguiente paso es crear la estructura en la que se va a subdividir la aplicación, para ello creamos 5 sub-aplicaciones, teniendo así el proyecto más organizado. Lo hacemos con estos comandos:
 - python manage.py startapp Modelos
 - python manage.py startapp Mails
 - python manage.py startapp Login
 - python manage.py startapp Recursos
 - python manage.py startapp Detalles

Cada una de ellas son carpetas, que dentro tienen la siguiente estructura:

```
Directory: C:\VMCloud\Modelos

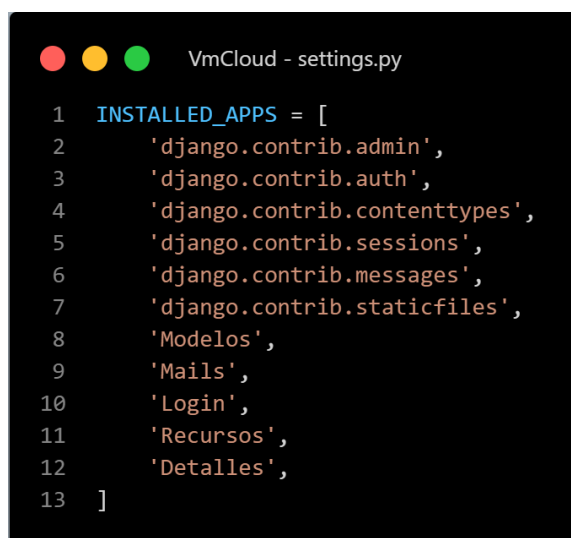
Mode                LastWriteTime         Length Name
----                -
d-----          28/11/2023    22:19             migrations
-a----          28/11/2023    22:19              0 __init__.py
-a----          28/11/2023    22:19             66 admin.py
-a----          28/11/2023    22:19            152 apps.py
-a----          28/11/2023    22:19             60 models.py
-a----          28/11/2023    22:19             63 tests.py
-a----          28/11/2023    22:30              0 urls.py
-a----          28/11/2023    22:19             66 views.py

PS C:\VMCloud\Modelos> |
```

Ilustración 14: Estructura de los directorios de las sub-aplicaciones

- a. La aplicación *Modelos*, contiene todo lo relacionado con la base de datos y el sector de *Admin* de la página web. En ella se encuentran todas las estructuras pertenecientes a la base de datos.

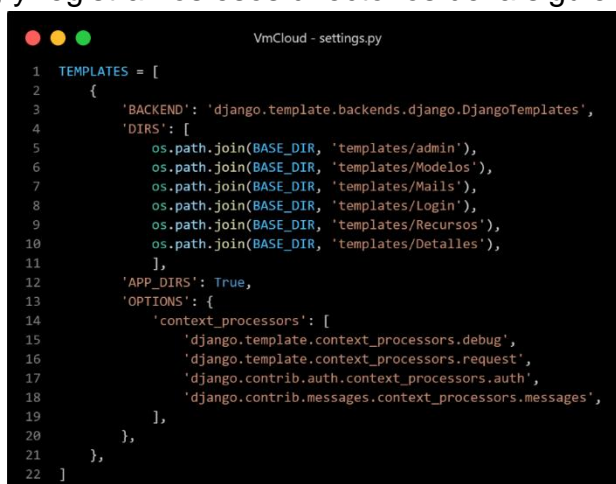
- b. La aplicación de *Login*, contiene todas las vistas y control de datos pertenecientes al *registro de usuarios* de la app y el *inicio de sesión* de la página.
 - c. La aplicación de *Mails*, dispone de la estructura para los envíos de mails de la app, entre ellas el *restablecimiento de la contraseña* de los usuarios y el *mail de bienvenida*.
 - d. La aplicación de *Recursos*, contiene las tablas de los *recursos de virtualización disponibles* (discos, VMs y redes), además de brindar la interfaz para añadir nuevos recursos.
 - e. Por último, la aplicación de *Detalles*, que dispone de las vistas necesarias para ingresar a cada recurso, *visualizarlo y gestionarlo*.
5. Luego incluimos las aplicaciones creadas en la lista de aplicaciones instaladas, ubicada en el `/VmCloud/settings.py`:



```
1  INSTALLED_APPS = [  
2      'django.contrib.admin',  
3      'django.contrib.auth',  
4      'django.contrib.contenttypes',  
5      'django.contrib.sessions',  
6      'django.contrib.messages',  
7      'django.contrib.staticfiles',  
8      'Modelos',  
9      'Mails',  
10     'Login',  
11     'Recursos',  
12     'Detalles',  
13 ]
```

Ilustración 15: Lista de aplicaciones instaladas en el código de settings.py

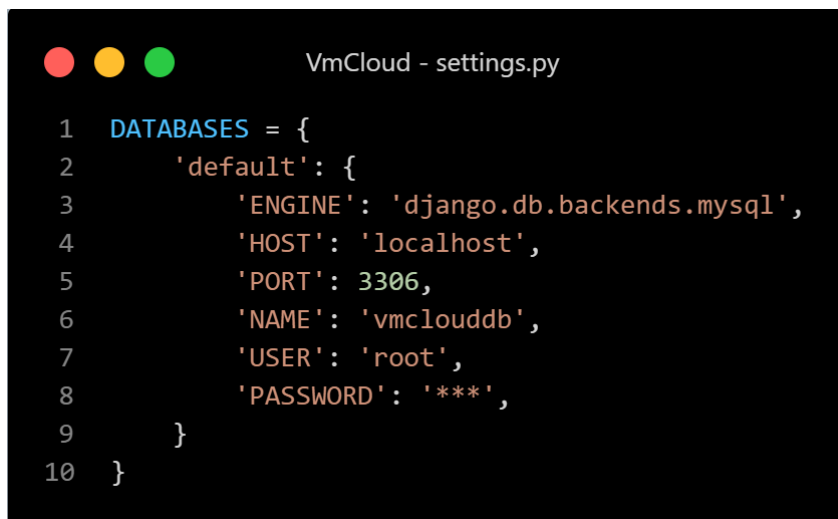
6. En el siguiente paso, creamos en la carpeta raíz del proyecto, una carpeta llamada *templates*, dentro de ella, más subcarpetas con los nombres de cada una de las sub-aplicaciones. En ellas se van a alojar los HTML que vamos a utilizar para darle forma a la página. Luego de crearlas, volvemos al settings.py, y registramos esos directorios de la siguiente forma:



```
1  TEMPLATES = [  
2      {  
3          'BACKEND': 'django.template.backends.django.DjangoTemplates',  
4          'DIRS': [  
5              os.path.join(BASE_DIR, 'templates/admin'),  
6              os.path.join(BASE_DIR, 'templates/Modelos'),  
7              os.path.join(BASE_DIR, 'templates/Mails'),  
8              os.path.join(BASE_DIR, 'templates/Login'),  
9              os.path.join(BASE_DIR, 'templates/Recursos'),  
10             os.path.join(BASE_DIR, 'templates/Detalles'),  
11          ],  
12          'APP_DIRS': True,  
13          'OPTIONS': {  
14              'context_processors': [  
15                  'django.template.context_processors.debug',  
16                  'django.template.context_processors.request',  
17                  'django.contrib.auth.context_processors.auth',  
18                  'django.contrib.messages.context_processors.messages',  
19              ],  
20          },  
21      },  
22  ]
```

Ilustración 16: Listado de templates configurados en el settings.py

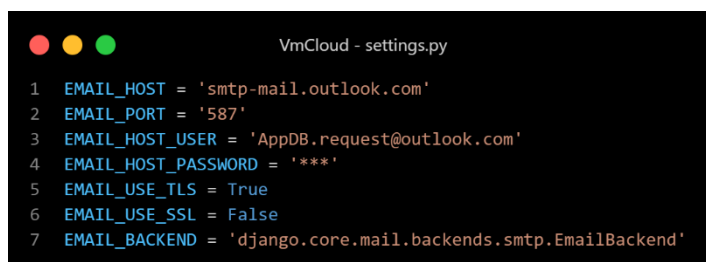
7. Ahora es momento de configurar la *base de datos* que va a utilizar el proyecto, para ello, en *settings.py* debemos dirigirnos al apartado correspondiente, e ingresar los datos de nuestro servidor SQL, en este caso se utilizó MySQL (Django tiene por defecto SQLite para las DB).



```
1 DATABASES = {
2     'default': {
3         'ENGINE': 'django.db.backends.mysql',
4         'HOST': 'localhost',
5         'PORT': 3306,
6         'NAME': 'vmclouddb',
7         'USER': 'root',
8         'PASSWORD': '***',
9     }
10 }
```

Ilustración 17: Configuración de la base de datos en settings.py

8. Por último, como la aplicación contiene envíos de mails, se le debe configurar una *cuenta como referencia* para realizar los correos.



```
1 EMAIL_HOST = 'smtp-mail.outlook.com'
2 EMAIL_PORT = '587'
3 EMAIL_HOST_USER = 'AppDB.request@outlook.com'
4 EMAIL_HOST_PASSWORD = '****'
5 EMAIL_USE_TLS = True
6 EMAIL_USE_SSL = False
7 EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
```

Ilustración 18: Configuración del email de la página.

Preparación de la base de datos:

La creación de la base de datos en Django es sencilla, con la ayuda de ORM, nos desligamos del lenguaje SQL y dejamos plasmada toda la estructura de la misma, en código Python.

Para ello nos dirigimos a la sub-app *Modelos*, y luego ingresamos en el archivo *models.py*, para definir las tablas utilizamos clases y las variables dentro de cada clase, son las columnas de cada tabla.

Para este proyecto, precisamos tablas:

- Usuarios: que contiene todos los datos de los usuarios que ingresan a la página.
- Tokens: la cual hace referencia a los tokens de recuperación de contraseñas.
- Discos: la cual identifica todos los discos que se crean y que debe tener como referencia al usuario que los crea, desde la tabla de usuarios.
- Redes: que contiene las redes creadas, y también debe referenciar el usuario que las crea.

- Máquinas Virtuales: la tabla principal de la aplicación, que va a contener toda la información de las VM generadas y va a referenciar tanto al usuario que las crea, como a los discos y redes que contiene en su configuración.

Teniendo esa información, deberíamos obtener un código de modelos como éste:

```
VmCloud - models.py
1 class PlanillaUsuarios(models.Model):
2     usuario = models.CharField(max_length=20,verbose_name='usuario')
3     nombre = models.CharField(max_length=60,verbose_name='nombre')
4     apellido = models.CharField(max_length=60,verbose_name='apellido')
5     dni = models.IntegerField(verbose_name='dni')
6     email = models.EmailField(max_length=255,verbose_name='email')
7     ip = models.CharField(max_length=15,verbose_name='ip')
8     web_ip = models.CharField(max_length=15,verbose_name='web_ip')
9     clave = models.CharField(max_length=255,verbose_name='clave')
```

Ilustración 19: Estructura de la tabla de Usuarios

```
VmCloud - models.py
1 class TokenRecuperacionTemp(models.Model):
2     usuario = models.CharField(max_length=20,verbose_name='usuario')
3     token = models.CharField(max_length=50,verbose_name='token')
4     fecha_gen = models.DateTimeField(verbose_name='fecha_gen')
5     usado = models.SmallIntegerField(verbose_name='usado')
```

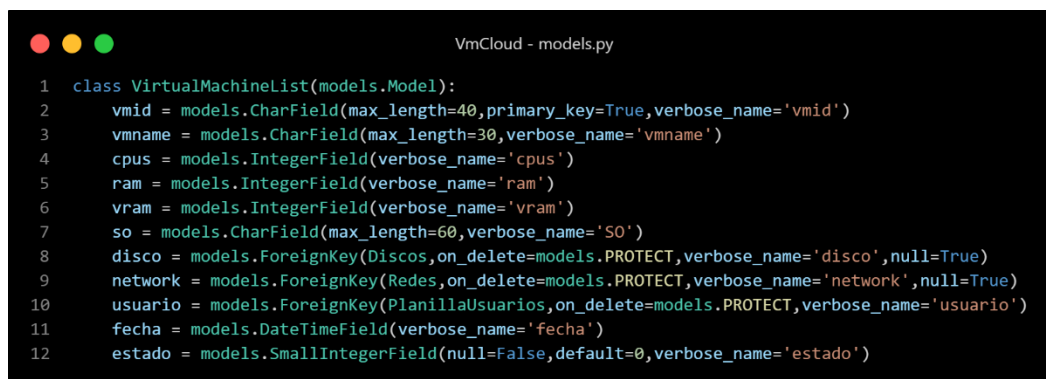
Ilustración 20: Estructura de la tabla de Tokens

```
VmCloud - models.py
1 class Discos(models.Model):
2     diskid = models.CharField(max_length=40,primary_key=True,verbose_name='diskid')
3     nombre = models.CharField(max_length=60,verbose_name='nombre')
4     size = models.BigIntegerField(verbose_name='size')
5     uso = models.BigIntegerField(verbose_name='uso',null=True)
6     usuario = models.ForeignKey(PlanillaUsuarios,on_delete=models.PROTECT,verbose_name='usuario')
7     fecha = models.DateTimeField(verbose_name='fecha')
```

Ilustración 21: Estructura de la tabla de Discos

```
VmCloud - models.py
1 class Redes(models.Model):
2     nombre = models.CharField(max_length=60,verbose_name='nombre')
3     network = models.CharField(max_length=30,verbose_name='network')
4     dhcp = models.CharField(max_length=3,verbose_name='dhcp',default=1)
5     usuario = models.ForeignKey(PlanillaUsuarios,on_delete=models.PROTECT,verbose_name='usuario')
6     fecha = models.DateTimeField(verbose_name='fecha')
```

Ilustración 22: Estructura de la tabla de Redes



```
1 class VirtualMachineList(models.Model):
2     vmid = models.CharField(max_length=40, primary_key=True, verbose_name='vmid')
3     vmname = models.CharField(max_length=30, verbose_name='vmname')
4     cpus = models.IntegerField(verbose_name='cpus')
5     ram = models.IntegerField(verbose_name='ram')
6     vram = models.IntegerField(verbose_name='vram')
7     so = models.CharField(max_length=60, verbose_name='SO')
8     disco = models.ForeignKey(Discos, on_delete=models.PROTECT, verbose_name='disco', null=True)
9     network = models.ForeignKey(Redes, on_delete=models.PROTECT, verbose_name='network', null=True)
10    usuario = models.ForeignKey(PlanillaUsuarios, on_delete=models.PROTECT, verbose_name='usuario')
11    fecha = models.DateTimeField(verbose_name='fecha')
12    estado = models.SmallIntegerField(null=False, default=0, verbose_name='estado')
```

Ilustración 23: Estructura de la tabla de Virtual Machines

Ahora bién, la base de datos ya se encuentra armada, pero no operativa. Es decir, ya escribimos toda su estructura, pero aún falta generarla en MySQL.

Para ello la tarea es sencilla. Debemos dirigirnos al directorio raíz del proyecto por en la consola, e ingresar:

```
python manage.py makemigrations Modelos
```

El comando anterior, va a generar automaticamente todos los scripts de bases de datos pendientes a realizar, leyendo el contenido de la base MySQL, y comparandolo con los modelos. Una vez hecho esto, ejecutamos:

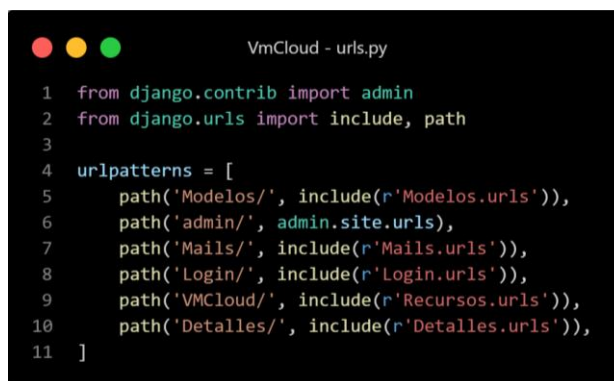
```
python manage.py migrate
```

Éste último, ejecutará los scripts anteriormente generados, teniendo ya operativa la base de datos para comenzar a trabajar.

Preparación de las URLs:

Para finalizar la etapa de preparación del proyector, debemos tener en cuenta que estamos trabajando sobre una pagina web, por lo tanto vamos a tener que configurar una serie de direcciones web (urls) para que las sub-aplicaciones, puedan operar en el proyecto.

Para esto vamos a trabajar con el archivo *urls.py*, teniendo en cuenta que cada sub-aplicación tiene este archivo, es necesario especificar la ruta al que nos referimos. Éste se encuentra en la carpeta de configuracion del proyecto “C:\VMCloud\VMCloud\urls.py”, en el mismo vamos a llamar a las sub-aplicaciones y le vamos a dar una ruta a cada una. Incluyendo las URLs que contengan dentro. El código se vería así:



```
1 from django.contrib import admin
2 from django.urls import include, path
3
4 urlpatterns = [
5     path('Modelos/', include(r'Modelos.urls')),
6     path('admin/', admin.site.urls),
7     path('Mails/', include(r'Mails.urls')),
8     path('Login/', include(r'Login.urls')),
9     path('VMCloud/', include(r'Recursos.urls')),
10    path('Detalles/', include(r'Detalles.urls')),
11 ]
```

Ilustración 24: Configuración de las URLs de las sub-aplicaciones

Diseño y distribución de las vistas:

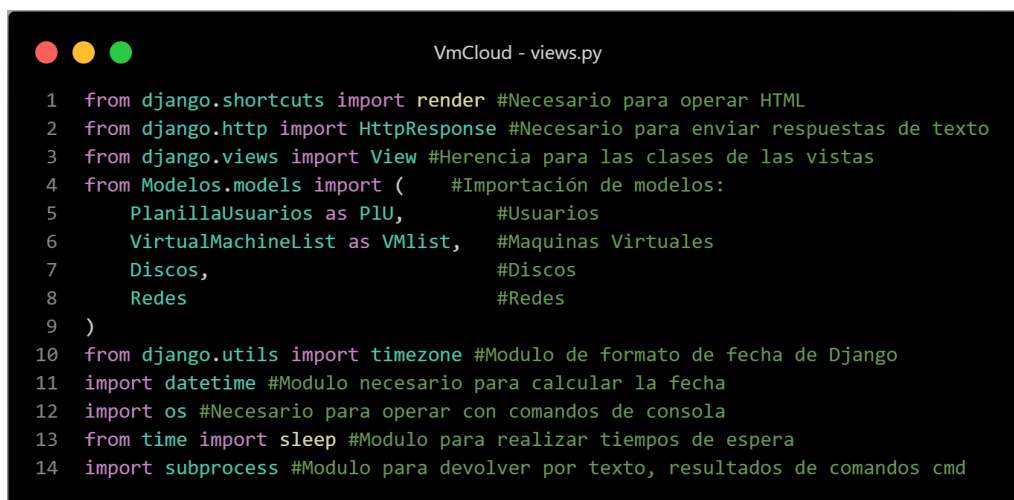
Para la etapa de diseño del proyecto, nos centraremos en lo que son las vistas, URLs de las vistas, y las plantillas HTML. Cómo operan en conjunto. El archivo para las vistas es *views.py* y se encuentra ubicado en cada una de las carpetas de las sub-aplicaciones.

En Django, una vista es una función de Python que procesa una solicitud web y devuelve una respuesta. Su función es tomar la información de la solicitud (como los parametros de la URL o los datos de los formularios) y devolver una respuesta, ya sea en formato HTML o JSON.

- ✚ Rutas y URLs: Las vistas estan asociadas a rutas específicas en la URL de la app. Es posible definir las en el archivo *urls.py* contenido, ahora si, en cada una de las sub-aplicaciones.
- ✚ Base de datos: Utilizando ORM, pueden interactuar con los modelos de bases de datos para realizar cálculos o procesar formularios.
- ✚ Plantillas (Templates): Pueden utilizar plantillas para generar contenido HTML de manera dinámica. Además, Django proporciona un sistema de plantillas que permite *combinar código Python en un documento HTML*.
- ✚ Respuestas HTTP: Los objetos devueltos son de respuesta HTTP, los cuales pueden ser *HttpResponse* (una respuesta de texto simple), *render* (una respuesta con página HTML), o incluso *JsonResponse* (una respuesta de JSON).

A continuación detallaremos el paso a paso utilizado para crear la vista del apartado de adición de máquinas virtuales.

- 1) El primer paso es dirigirse al archivo *views.py* de la sub-aplicación de *Recursos* (ubicación de la funcionalidad). Una vez dentro del archivo, debemos llamar a todos los módulos y funciones necesarias para añadir la maquina virtual, es decir, necesitamos principalmente el módulo *os* para poder importar las VM en el motor de Virtual Box. Y también necesitaríamos llamar al modelo de base de datos, que hace referencia a la tabla de las VM.



```
1 from django.shortcuts import render #Necesario para operar HTML
2 from django.http import HttpResponse #Necesario para enviar respuestas de texto
3 from django.views import View #Herencia para las clases de las vistas
4 from Modelos.models import ( #Importación de modelos:
5     PlanillaUsuarios as PLU, #Usuarios
6     VirtualMachineList as VMlist, #Maquinas Virtuales
7     Discos, #Discos
8     Redes #Redes
9 )
10 from django.utils import timezone #Modulo de formato de fecha de Django
11 import datetime #Modulo necesario para calcular la fecha
12 import os #Necesario para operar con comandos de consola
13 from time import sleep #Modulo para realizar tiempos de espera
14 import subprocess #Modulo para devolver por texto, resultados de comandos cmd
```

Ilustración 25: Importación de módulos necesarios para operar con las vistas de la app

2) Luego procedemos a crear la vista, la cuál puede ser:

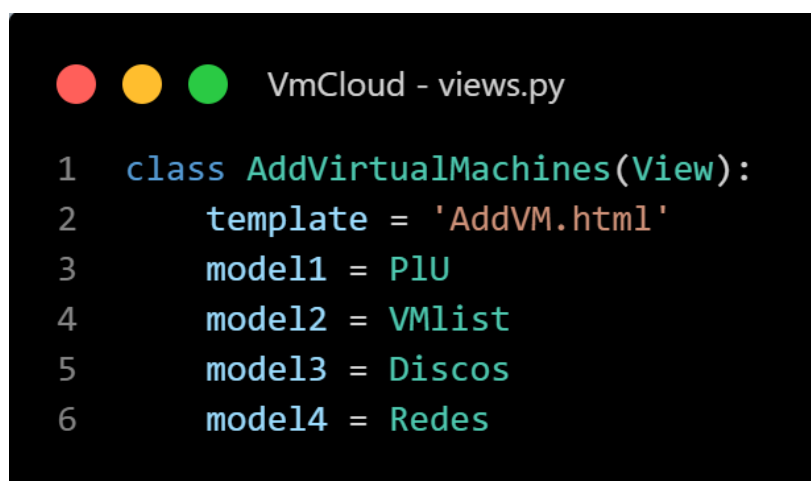
- Una función (def funcionVista(parametros)).
- Una clase (class VistaDeClases(View)).

Debido a la cantidad de variables y datos que se manejan, todas las vistas de VMCloud, operan con clases.

Las clases toman la herencia del padre *View* que convierte automáticamente nuestro código en una vista de Django.

Lo primero es definir las variables iniciales de la vista, que van a contener:

- El template que va a utilizar la vista.
- El modelo de los usuarios (ya que la gestión de VM es por usuarios).
- El modelo de las VM.
- El modelo de los Discos (necesario para relacionar Discos-VM)
- El modelo de las Redes (necesario para relacionar Redes-VM)



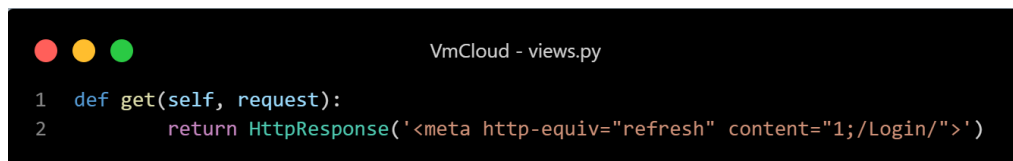
```
1 class AddVirtualMachines(View):
2     template = 'AddVM.html'
3     model1 = PLU
4     model2 = VMlist
5     model3 = Discos
6     model4 = Redes
```

Ilustración 26: Inicio de la vista y definición de las variables iniciales

3) Lo siguiente es definir las dos acciones principales para el procesamiento de las páginas web, el *get* y el *post*. Ambas son solicitudes HTTP, la primera sirve para solicitar los parámetros que

envia el servidor *desde la URL*. La otra sirve para enviar datos al servidor y colocar los mismos en el cuerpo de la solicitud.

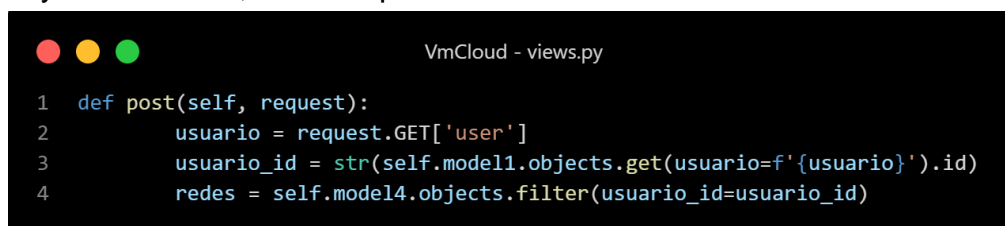
Debido a razones de seguridad, metodo *get* se va a utilizar solamente para tomar los parámetros que contienen las URL de las vistas. Pero van a estar bloqueados como acción, ya que sería inseguro, si la página permitiera ingresar a la visualización de las VM simplemente ingresando una URL en el navegador. Por lo que para esta acción se desarrolló una redirección automática hacia el *Login*. Es decir que, si ingresamos la URL de las VM, del usuario “manu”, nos va a redirigir al inicio de sesión para obligar al usuario a identificarse. Así fue como se desarrolló:



```
1 def get(self, request):
2     return HttpResponseRedirect('<meta http-equiv="refresh" content="1;/Login/">')
```

Ilustración 27: Función get de la vista de VMs, retorna por texto una redirección automática al login en formato HTML.

Luego de definir el *get* continuamos con el *post*. En el que necesitaremos los datos iniciales de *usuarios* y *redes*. Al usuario lo vamos a traer desde el GET de la URL (/url/?user=nombre de usuario). Luego vamos a procesarlo para traer su id de usuario, buscándolo en el modelo de los usuarios desde el nombre que recuperamos en el método GET. Y vamos a recuperar un resultado de query de las redes, filtrando por el id de usuario.



```
1 def post(self, request):
2     usuario = request.GET['user']
3     usuario_id = str(self.model1.objects.get(usuario=f'{usuario}').id)
4     redes = self.model4.objects.filter(usuario_id=usuario_id)
```

Ilustración 28: Comienzo del método post de la vista, recuperando usuario y redes.

Para realizar el procesamiento de los datos de la VM creada, la vista consta con un formulario realizado en HTML, éste va a enviar los datos necesarios para cargargar, tanto en base de datos, como en el motor de Virtual Box, las propiedades de la VM. Y de esto surge una problemática, cuando ingresamos inicialmente a la carga de una VM, nos va a estar pidiendo datos ya cargados, pero en este primer ingreso no los vamos a tener. Por lo que el *post* de la vista se divide en dos partes gestionadas con el manejo de excepciones de Python *try/exept*:

- El *try*: Va a intentar recuperar todos los datos enviados desde el formulario de las VM. Procesando así las variables POST enviadas desde el formulario, realizando una validación identificando si el nombre de la VM para el usuario ya está en uso, llamando a funciones para crear la VM en VBox, crear el disco relacionado a la misma y procesar los datos identificadores de cada uno de los recursos para luego enviarlos a la base de datos.


```

VmCloud - views.py

1  try:
2      vmname = request.POST['vmname']
3      cpus = request.POST['cpus']
4      ram = request.POST['ram']
5      vram = request.POST['vram']
6      so = request.POST['so']
7      network = request.POST['network']
8      fecha = datetime.datetime.now(tz=timezone.utc)
```

Ilustración 29: Desde él envió del formulario se recuperan las variables (nombre de VM, CPUs, RAM, VRAM, sist. operativo, red relacionada) y se calcula la fecha que identificaría la carga de la VM

```

VmCloud - views.py

1  try:
2      val_vm_usr = str(self.model2.objects.get(vmname=vmname).usuario_id)
3      if val_vm_usr == usuario_id:
4          return HttpResponse(f'El nombre de VM ({vmname}) ya está en uso')
5  except:
6      pass
```

Ilustración 30: Validación de la existencia del nombre de la vm para el usuario en la base de datos, manejado con excepciones. (Código incluido en el try principal)

```

VmCloud - views.py

1  vmid = CrearMaquinaVirtual(vmname,cpus,ram,vram,so,network,usuario)
2      #Crea la maquina virtual y disco, y devuelve el id de la VM.
3  diskname = CambiarNombreDisco(so,vmname,usuario)
4      #Cambia el nombre del disco, para identificar que pertenece a la VM, y lo retorna.
5  diskid = CalcularIdDisco(vmname,usuario)
6      #Calcula y retorna el id que tiene el disco.
7  ValoresDisco = CalcularValoresDisco(diskid,diskname)
8      #Devuelve un listado de las características del disco
```

Ilustración 31: Funciones de creación y procesamiento de los datos de la VM creada. (Incluido en el try principal)

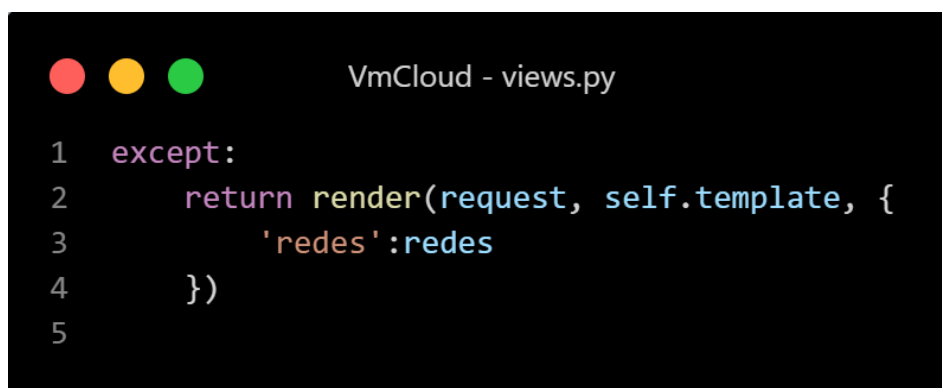
```

VmCloud - views.py

1  #Se graba el disco en base de datos (ORM)
2  grabar = self.model3(
3      diskid=ValoresDisco[0],
4      nombre=ValoresDisco[1],
5      size=ValoresDisco[2],
6      uso=ValoresDisco[3],
7      usuario_id=usuario_id,
8      fecha=fecha
9  )
10 grabar.save()
11
12 #Se graba la VM en base de datos (ORM)
13 grabar = self.model2(
14     vmname=vmname,
15     cpus=cpus,
16     ram=ram,
17     vram=vram,
18     so=so,
19     disco_id=diskid,
20     network_id=network,
21     fecha=fecha,
22     vmid=vmid,
23     usuario_id=usuario_id
24 )
25 grabar.save()
```

Ilustración 32: Inserción de los datos procesados dentro de la base de datos, manejado con ORM. Se insertan la VM y el Disco conjuntamente. (Incluido en el try inicial)

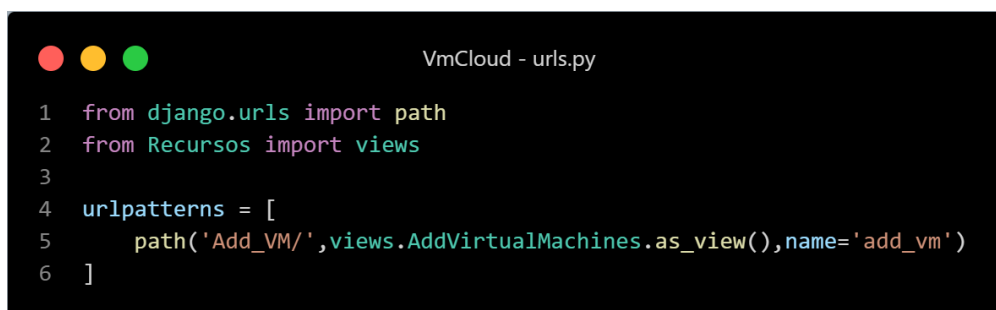
- El *except*: Si todo lo anterior descrito falló, es decir, si ingresamos inicialmente al formulario de creación de VM, obviamente no vamos a tener los datos del POST del formulario, ya que ingresamos por primera vez. Por lo que simplemente la vista va a devolver el formulario vacío esperando al ingreso de datos. La en esta parte, la vista retorna con la función *render* la respuesta desde el *template anteriormente definido*, y le pasa un *parámetro al diccionario*, que se trata del resultado de la query de *redes*, utilizado para mostrar datos en un desplegable que contiene las redes disponibles para la VM.

A screenshot of a code editor window titled "VmCloud - views.py". It shows a Python code snippet for an exception handler. The code is as follows:

```
1  except:
2      return render(request, self.template, {
3          'redes': redes
4      })
5
```

Ilustración 33: Codificación del except de la vista, devuelve un formulario vacío.

- 4) Finalizada la vista, necesitamos darle una URL. Este procedimiento es simple: Debemos dirigirnos al archivo de URLs de la sub-aplicación (*/Recursos/urls.py*), y configurar una URL, llamar a la vista, y darle un nombre de identificación.

A screenshot of a code editor window titled "VmCloud - urls.py". It shows Python code for configuring URL patterns. The code is as follows:

```
1  from django.urls import path
2  from Recursos import views
3
4  urlpatterns = [
5      path('Add_VM/', views.AddVirtualMachines.as_view(), name='add_vm')
6  ]
```

Ilustración 34: Configuración de la url para la vista de creación de VM, en el listado de urls de la sub-aplicación

- 5) Para finalizar el diseño de la vista debemos crear un documento HTML, ubicado, en este caso, en *templates/Recursos/AddVM.html*. En éste van a ir incluidas las variables que se solicitan desde Python identificadas entre dos llaves “*{{ variable }}*”, y las sentencias entre llaves y porcentajes “*{% if %}*”. Lo demás son etiquetas HTML.

```

VmCloud - AddVM.html
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <link rel="StyleSheet" type="text/css" href="/static/VMCloud/VMs.css">
5    </head>
6    <title>Añadir VM</title>
7    <body>
8      <h2>Añadir una VM:</h2>
9      <form action="" method="post">
10         {% csrf_token %}
11         Nombre de la VM: <input type="text" name="vmname" required>
12         <p></p>
13         Cantidad de cpus: <input type="number" name="cpus" required max="5" min="1" placeholder="Ej: 3">
14         <p></p>
15         Cantidad de RAM(Mb): <input type="number" name="ram" required min="1000" placeholder="Ej: 3000">
16         <p></p>
17         Cantidad de VRAM: <input type="number" name="vram" required min="30" placeholder="Ej: 55">
18         <p></p>
19         Sistema Operativo: <select name="so" id="so" required>
20           <option value="Windows10_64">Windows 10 Pro</option>
21           <option value="Ubuntu_64">Ubuntu</option>
22         </select>
23         <p></p>
24         Red NAT: <select name="network" id="network" required>
25           <option value="NULL">No especificado</option>
26           {% for vista in redes %}
27           <option value="{{ vista.id }}">{{ vista.nombre }}</option>
28           {% endfor %}
29         </select>
30         <p></p>
31         <input type="submit" value="Aceptar">
32       </form>
33     </body>
34   </html>

```

Ilustración 35: HTML de la vista de adición de VM.

De ésta manera finalizamos la etapa de la creación de vistas, ya que, las demás vistas utilizadas en la app, cumplen con el mismo formato, las cuales se diferencian solamente por las variables que contiene cada una al realizar acciones específicas.

Para finalizar con la etapa de diseño, se van a listar las funciones creadas con el objetivo de procesar los recursos de Virtual Box para traerlos a Django. Estas funciones son las que se van a necesitar en cada vista, para realizar las tareas provenientes de la virtualización, a partir de los comandos de consola.

❖ CalcularIdDisco: Necesaria para la creación del disco en la base de datos, solicita los parámetros del nombre de la VM, y el usuario que la creó.

Busca el disco en el directorio de la VM, el cuál es la combinación del usuario y el nombre de la misma, y lo almacena en una variable. Luego a partir del comando *showmediuminfo* y un flitro de cadena que busque "UUID:", procesa la información y convierte el resultado del comando en el *ID* del disco, posteriormente retornándolo para su uso.

```

VmCloud - views.py
1  def CalcularIdDisco(vmname,usuario):
2      vmname = f'{usuario}_{vmname}'
3      Directorio = os.path.join('C:\\', 'Users', 'manuq', 'VirtualBox VMs', f'{vmname}', f'{vmname}_dk.vdi')
4      comando = f'VBoxManage showmediuminfo disk "{Directorio}" | findstr /b "UUID:"'
5      ShellResult = subprocess.check_output(comando, shell=True, text=True)
6      comando = ShellResult.strip('\n').strip('UUID:').strip(' ')
7      return comando

```

Ilustración 36: Código de la función CalcularIdDisco

- ❖ CalcularValoresDisco: Utilizado en la vista de creación de VM, ésta función pide id y nombre del disco, crea una lista vacía para almacenar los valores del disco, para luego con el comando *showmediuminfo* y distintos filtros, poder ir iterando cada uno de las características del disco en la lista, posteriormente retornada.

```
VmCloud - views.py
1 def CalcularValoresDisco(diskid,diskname):
2     valores = []
3     valores.append(diskid)
4     valores.append(diskname)
5
6     comando = f'VBoxManage showmediuminfo disk "{diskid}" | findstr /b "Capacity:"'
7     ShellResult = subprocess.check_output(comando, shell=True, text=True)
8     comando = ShellResult.strip('Capacity:').strip(' ').strip('\n').strip('').strip('MBytes').strip(' ')
9     valores.append(comando)
10
11     comando = f'VBoxManage showmediuminfo disk "{diskid}" | findstr /b "Size on disk:"'
12     ShellResult = subprocess.check_output(comando, shell=True, text=True)
13     comando = ShellResult.strip('Size on disk:').strip(' ').strip('\n').strip('').strip('MBytes').strip(' ')
14     valores.append(comando)
15
16     return valores
```

Ilustración 37: Código de la función CalcularValoresDisco.

- ❖ CambiarNombreDisco: Solicitando de los parámetros *sistema operativo*, *nombre de la VM* y *usuario*, identifica el sistema operativo de la VM para buscar el nombre inicial del disco, ya que se importa con el nombre de VM con el que fue guardado. Luego define el nuevo directorio (que es el mismo, solo cambia el nombre del disco), dándole como nombre, la VM más la cadena “_dk”, así éste se relaciona con la VM a la que pertenece. Por último, utilizando el comando *modifymedium* y el parámetro *–move* se asigna el nuevo nombre y lo retorna.

```
VmCloud - views.py
1 def CambiarNombreDisco(so,vmname,usuario):
2     if so == 'Windows10_64':
3         DiskName = "W10PRO"
4     elif so == 'Ubuntu_64':
5         DiskName = "Ubuntu"
6     NombreDisco = f'{vmname}_dk'
7     vmname = f'{usuario}_{vmname}'
8     Directorio = os.path.join('C:\\', 'Users', 'manuq', 'VirtualBox VMs', f'{vmname}', f'{DiskName}-disk001.vdi')
9     NuevoDir = os.path.join('C:\\', 'Users', 'manuq', 'VirtualBox VMs', f'{vmname}', f'{vmname}_dk.vdi')
10    ModifyDisk = f'VBoxManage modifymedium disk "{Directorio}" --move="{NuevoDir}"'
11    os.system(ModifyDisk)
12    return NombreDisco
```

Ilustración 38: Código de la función CambiarNombreDisco

- ❖ ActualizarEspacioUsado: Utilizada para actualizar el tamaño de los discos en el detalle de los recursos, cada vez que se actualiza la página, no solicita parámetros, solo consulta con *showmediuminfo* el tamaño usado de los discos en un bucle *for* que itera todos los registros de discos disponibles, y luego con ORM, actualiza el dato para cada disco.

```
VmCloud - views.py
1 def ActualizarEspacioUsado():
2     discos = Discos.objects.values_list('diskid', flat=True)
3     for disco in discos:
4         comando = f'VBoxManage showmediuminfo disk "{disco}" | findstr /b "Size on disk:"'
5         ShellResult = subprocess.check_output(comando, shell=True, text=True)
6         comando = ShellResult.strip('Size on disk:').strip(' ').strip('\n').strip('').strip('MBytes').strip(' ')
7         actualizar = Discos.objects.get(diskid=f'{disco}')
8         actualizar.uso = comando
9         actualizar.save()
```

Ilustración 39: Código de la función ActualizarEspacioUsado

- ❖ CrearMaquinaVirtual: La función que se llama desde la vista de creación de máquinas virtuales, solicita los parámetros de nombre de la VM, CPUs, RAM, VRAM, red y usuario. Identifica el sistema operativo elegido de la VM, añade al nombre de la misma el nombre del usuario, y luego con el comando *import* busca el archivo de imagen de la VM dependiendo del S.O. elegido y comienza la importación, después con el comando *modifyvm* añade los valores de CPUs, RAM y VRAM. Con *showvminfo* y un filtro de cadena calcula el *id* de la VM, y por último añade la red relacionada si ésta fue configurada. Retornando como valor el *id* de la VM.

```
1 def CrearMaquinaVirtual(vmname,cpus,ram,vram,so,network,usuario):
2     if so == 'Windows10_64':
3         sistema = "W10PRO.ova"
4     elif so == 'Ubuntu_64':
5         sistema = "Ubuntu.ova"
6
7     vmname = f'{usuario}_{vmname}'
8     RutaOva = os.path.join('C:\\', 'Python', 'PY_DJANGO', 'VmCloud', f'{sistema}')
9     CrearVm = f'VBoxManage import {RutaOva} --options=importtovdi --vsys=0 --vmname={vmname} --unit=0'
10    RecursosVm = f'VBoxManage modifyvm "{vmname}" --cpus {cpus} --memory {ram} --vram {vram}'
11    os.system(CrearVm)
12    os.system(RecursosVm)
13    CalcularId = f'vboxmanage showvminfo "{vmname}" | findstr /b "UUID"'
14    ShellResult = subprocess.check_output(CalcularId, shell=True, text=True)
15    comando = ShellResult.strip('\n').strip('UUID:').strip(' ')
16
17    if network != None:
18        netname = Redes.objects.get(id=network).nombre
19        AddNtw = f'VBoxManage modifyvm "{vmname}" --nic1=natnetwork --nic-type1=82540EM --cable-connected1=on --nic-boot-prio1=1 --nat-network1="{usuario}_{netname}"'
20        os.system(AddNtw)
21
22    return comando
```

Ilustración 40:Codigo de la función CrearMaquinaVirtual

- ❖ CalcularEstadoVm: Esta función sirve para detectar si la VM está encendida o apagada. Realiza el control con el comando *debugvm* y valida que, si devuelve el sistema operativo, quiere decir que la VM se encuentra encendida, sino, la VM está apagada. Pide como único parámetro el id de la VM y retorna en 1 o 0 según corresponda el estado.

```
1 def CalcularEstadoVm(vmid):
2     comando = f'VBoxManage debugvm {vmid} osdetect'
3     try:
4         subprocess.check_output(comando, shell=True, text=True)
5         comando = 1
6     except:
7         comando = 0
8     return comando
```

Ilustración 41: Código de la función CalcularEstadoVm

- ❖ ActualizarEstadoVm: La siguiente función no solicita parámetros y va ubicada en la página general de los recursos, su objetivo es, utilizando la función *CalcularEstadoVm*, actualizar con ORM el dato del estado de las VM listadas.

```
1 def ActualizarEstadoVm():
2     vms = VMlist.objects.values_list('vmid',flat=True)
3     for vm in vms:
4         estado = CalcularEstadoVm(vm)
5         actualizar = VMlist.objects.get(vmid=vm)
6         actualizar.estado = estado
7         actualizar.save()
```

Ilustración 42: Código de la función ActualizarEstadoVm

- ❖ CambiarRed: Solicita los parámetros de id de la VM, el nombre de la red, y el usuario. Sirve para alternar entre las redes disponibles para la VM y opera con el comando *modifyvm*.

```
1 def CambiarRed(vmid,network,usuario):
2     RemoveNat = f'VBoxManage modifyvm "{vmid}" --nic1=none --nic-type1=82540EM --cable-connected1=off --nic-boot-prio1=1 --nat-network1=none'
3     os.system(RemoveNat)
4     if network != None:
5         network = Redes.objects.get(id=network).nombre
6         AddNtw = f'VBoxManage modifyvm "{vmid}" --nic1=natnetwork --nic-type1=82540EM --cable-connected1=on --nic-boot-prio1=1 --nat-network1="{usuario}_{network}"'
7         os.system(AddNtw)
```

Ilustración 43: Código de la función CambiarRed

- ❖ ModificarVirtualMachine: Solicita los parámetros de id de la VM, CPUs, RAM, VRAM, red y usuario. Sirve para modificar los recursos de la VM incluida la red utilizando la función *CambiarRed*.

```
1 def ModificarVirtualMachine(vmid,cpus,ram,vram,network,usuario):
2     CambiarRed(vmid,network,usuario)
3     ModifyVM = f'VBoxManage modifyvm "{vmid}" --memory={ram} --vram={vram} --cpus={cpus}'
4     os.system(ModifyVM)
```

Ilustración 44: Código de la función ModificarVirtualMachine

- ❖ BorrarMaquinaVirtual: Con el objetivo de borrar la VM seleccionada, utiliza el parámetro de id de la misma, y utiliza el comando *unregistervm* junto con los atributos *--delete --delete-all*, que realizan un borrado completo de todos los archivos de la VM, incluido el disco.

```
1 def BorrarMaquinaVirtual(vmid):
2     UnregisterVm = f'VBoxManage unregistervm {vmid} --delete --delete-all'
3     os.system(UnregisterVm)
```

Ilustración 45: Código de la función BorrarMaquinaVirtual

Implementación y puesta en marcha de la app:

Para implementar la app, primero hay que tener instalados y configurados todos los recursos anteriormente nombrados, en el servidor local o web en el que se va a ejecutar. Para la ejecución es sencilla, hay que dirigirse al directorio raíz del proyecto mediante la consola y escribir lo siguiente:

- `Python manage.py runserver ip_del_servidor:puerto`

Si todo está correcto, debería dar como resultado una ruta para ingresar en el navegador:

```
PS C:\Python\PY_DJANGO\VmCloud> python manage.py runserver 10.11.21.126:8000
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
November 29, 2023 - 14:46:49
Django version 4.1.2, using settings 'VmCloud.settings'
Starting development server at http://10.11.21.126:8000/
Quit the server with CTRL-BREAK.
```

Ilustración 46: Inicio del servidor de la aplicación

Luego debemos ingresar la URL que nos devuelve el resultado del comando en el navegador, añadiendo luego de la barra */VmCloud*. Nos va a ingresar a una especie de menú en el que hay una opción para iniciar sesión, y otra para registrar un usuario. Debemos registrarnos para poder ingresar a la página.

Se nos va a abrir el registro de usuario, en el que debemos ingresar datos como, *nombre de usuario, nombre completo, apellido, DNI, email, contraseña y confirmación*. Una vez le damos a “Aceptar”, la página va a validar los datos y nos va a devolver en la parte superior si los datos son válidos o no, si lo son, debemos volver a aceptar. Posteriormente la página nos registrará y enviará un mail de registro satisfactorio. Por último, nos va a redireccionar al Login para que ya ingresemos con nuestro nuevo usuario.

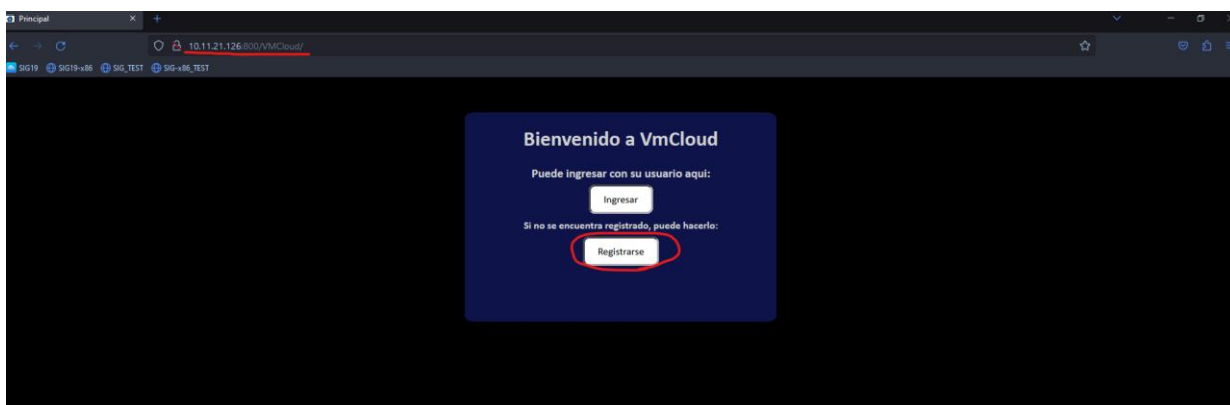


Ilustración 47: Página principal de VMCloud

The screenshot shows a registration form on a dark blue background. The title is 'Ingrese sus datos para registrarse:'. The form contains the following fields and labels: 'Usuario:' with the value 'ProyectoCC', 'Nombre:' with 'Manuel', 'Apellido:' with 'Nuñez', 'DNI:' with '44299355', 'Email:' with 'ejemplo@gmail.com', 'Contraseña:' with a masked password of 8 dots, and 'Confirmar contraseña:' with a masked password of 8 dots. At the bottom, there is a 'REGISTRAR' button and a '>Volver<' link.

Ilustración 48: Ingreso del usuario a la página

This screenshot shows the same registration form as Illustration 48, but with a green banner at the top that reads 'Datos ingresados correctamente, quiere continuar?'. The 'Email' field now contains 'a@test3'. The 'ACEPTAR' button is highlighted, and the '>Cancelar<' link is visible next to it. The '>Volver<' link remains at the bottom.

Ilustración 49: Datos de registración de usuario aceptados

The screenshot shows a login form on a dark blue background. The title is 'Ingrese sus datos si ya esta registrado:'. It contains two fields: 'Usuario:' with the value 'ProyectoCC' and 'Clave:' with a masked password of 8 dots. Below these is an 'INGRESAR' button. At the bottom, there is a link that says '>Click Aquí para registrarse<'. The text 'Todavía no se ha registrado?:' is also present above the link.

Ilustración 50: Interfaz de registración de usuario

Una vez el usuario ingresa a la página, se va a encontrar con tres listados vacíos, que hacen referencia a VM, discos y redes. Lo recomendable, antes que nada, es que el usuario configure al menos una red NAT para asociar durante la creación de la VM. Para ello deberá presionar en el botón “+” del ultimo listado (redes).

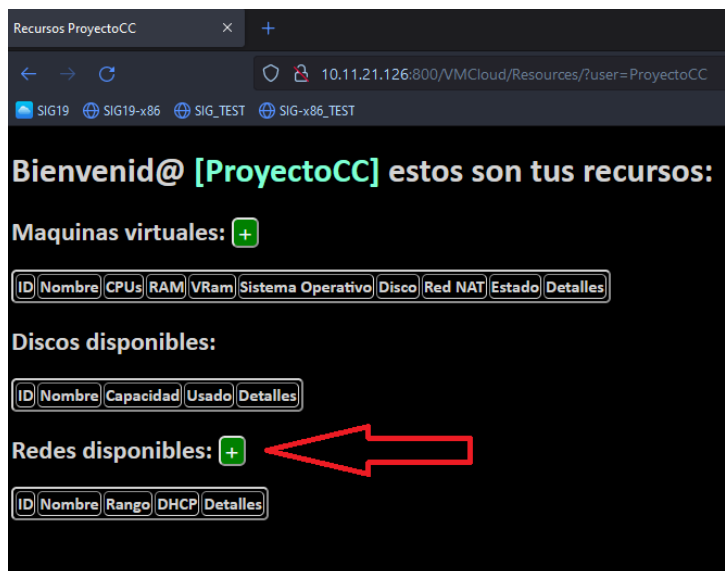


Ilustración 51: Interfaz de recursos y selección para añadir una red

Luego simplemente se debe escribir un nombre de red, un prefijo ipv4 y tildar o no el dhcp.

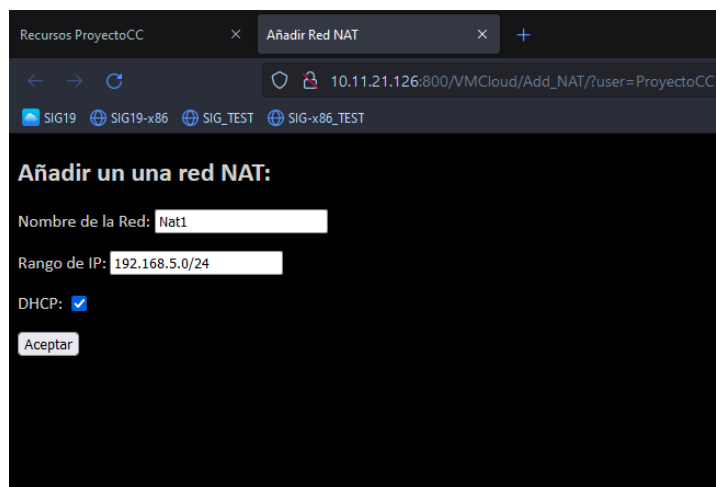


Ilustración 52: Interfaz de creación de redes

Al darle a “Aceptar”, el foco volverá a la página principal, y al refrescar, se visualizará el registro de la primera red. Posterior a esto (o no), es posible crear una VM.

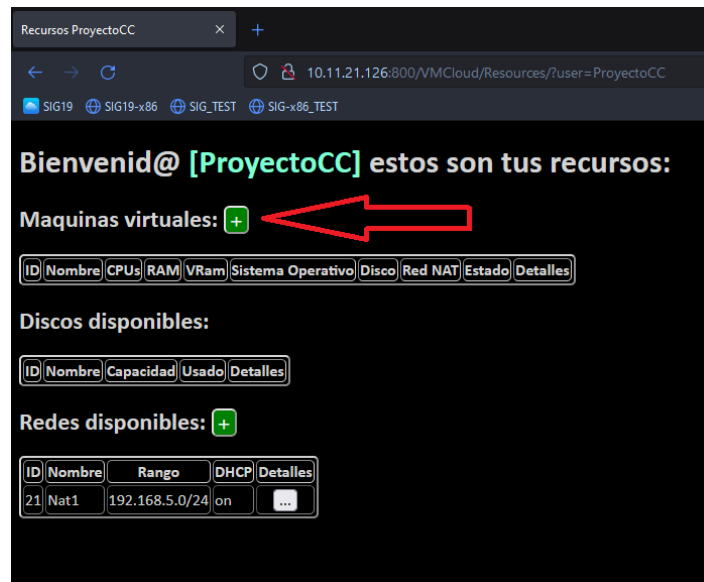


Ilustración 53: Primer registro de red creada y selección de creación de VM.

En la creación de la VM se debe ingresar, su nombre, CPUs, RAM, VRAM, sistema operativo (la página dispone de Ubuntu y Windows10), y seleccionar del desplegable alguna red.

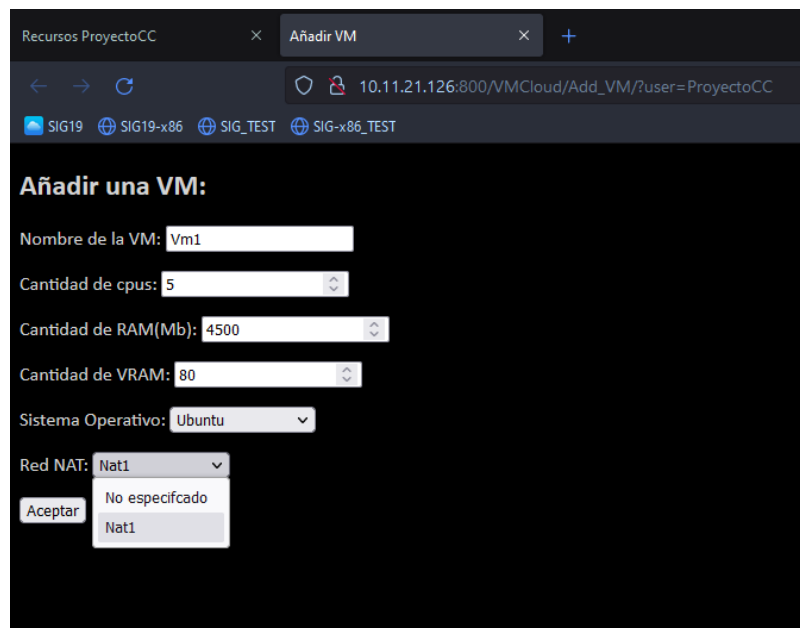


Ilustración 54: Interfaz de creación de VM

Y listo, con esto tendríamos completa la carga de los datos, ya que, el disco se configura automáticamente durante la creación de la VM.

La página principal se verá así:

Recursos ProyectoCC

10.11.21.126:800/VMCloud/Resources/?user=ProyectoCC

Bienvenid@ [ProyectoCC] estos son tus recursos:

Maquinas virtuales: +

ID	Nombre	CPUs	RAM	VRam	Sistema Operativo	Disco	Red NAT	Estado	Detalles
1d4ea1fd-7c50-4271-a207-d6428af18a7e	Vm1	5	4500	80	Ubuntu_64	Vm1_dk	Nat1		

Discos disponibles:

ID	Nombre	Capacidad	Usado	Detalles
266df8d4-9934-4d43-ac18-6c721974620d	Vm1_dk	21340(Mb)	12767(Mb)	

Redes disponibles: +

ID	Nombre	Rango	DHCP	Detalles
21	Nat1	192.168.5.0/24	on	

Ilustración 55: Página principal, con todos los recursos cargados.

Se recomienda acceder a los detalles de los nuevos discos creados, y agrandar su capacidad, ya que por defecto vienen configurados con espacio suficiente para contener el sistema operativo y su configuración básica.

Recursos ProyectoCC

10.11.21.126:800/VMCloud/Resources/?user=ProyectoCC

Bienvenid@ [ProyectoCC] estos son tus recursos:

Maquinas virtuales: +

ID	Nombre	CPUs	RAM	VRam	Sistema Operativo	Disco	Red NAT	Estado	Detalles
1d4ea1fd-7c50-4271-a207-d6428af18a7e	Vm1	5	4500	80	Ubuntu_64	Vm1_dk	Nat1		

Discos disponibles:

ID	Nombre	Capacidad	Usado	Detalles
266df8d4-9934-4d43-ac18-6c721974620d	Vm1_dk	21340(Mb)	12767(Mb)	

Redes disponibles: +

ID	Nombre	Rango	DHCP	Detalles
21	Nat1	192.168.5.0/24	on	

Ilustración 56: Selección para abrir los detalles del disco

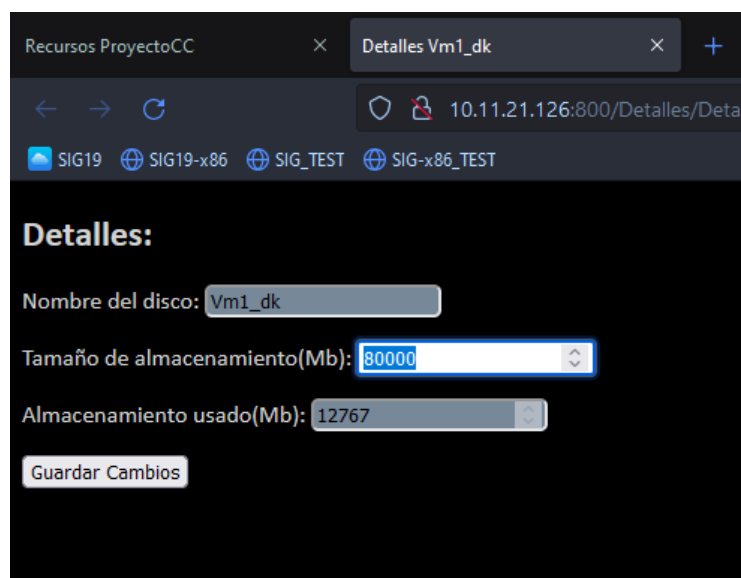


Ilustración 57: Interfaz de detalles del disco, modificación del tamaño del almacenamiento a 80GB

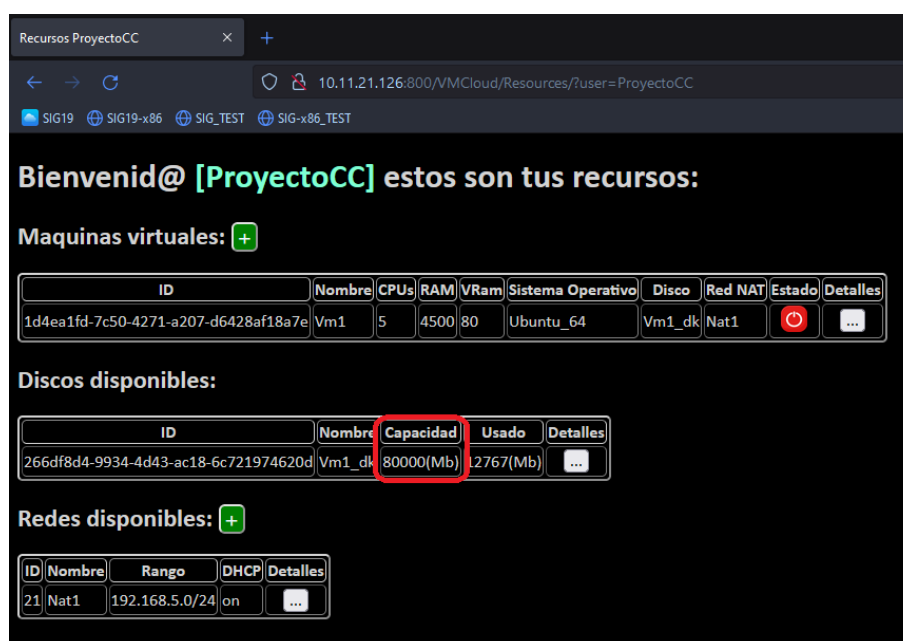


Ilustración 58: Visualización del aumento del tamaño del disco en la pantalla principal.

De igual manera que accedimos al disco para editarlo, podemos acceder a las redes, en las cuales podremos editar el prefijo y activar/desactivar el dhcp, además de tener una opción de borrado. En las VM, podemos editar CPUs, RAM, VRAM y red, tenemos una opción de borrado, la cual además de borrar la VM borra también su disco de los registros. Y disponemos del encendido o apagado de la VM, dependiendo de su estado.

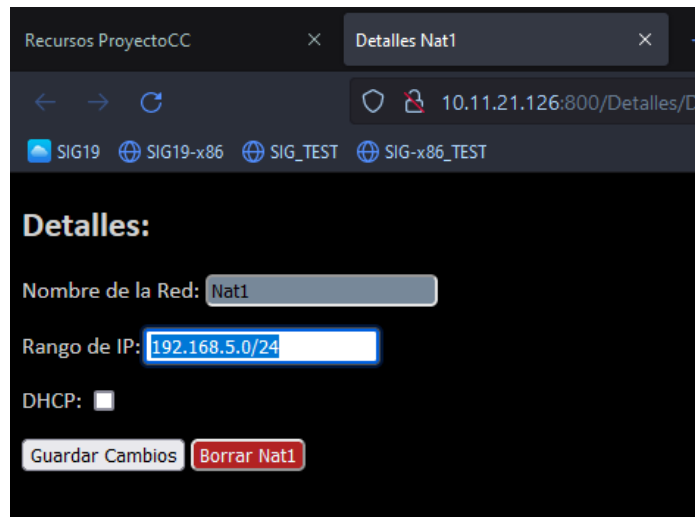


Ilustración 59: interfaz de visualización de los detalles de la red, su edición y borrado.

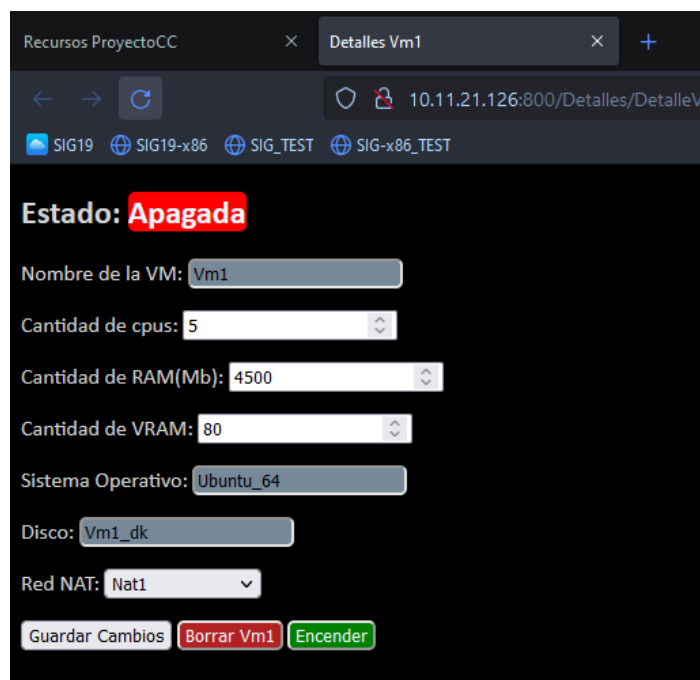


Ilustración 60: Interfaz de visualización de los detalles, edición, borrado y encendido de las VM.

Al *encender* las VM, el ícono de la pantalla principal, se pintará de verde y al ingresar a los detalles de la VM encendida, veremos que nos informa de su estado y no nos permitirá modificar ninguna de sus características. También el botón de *encendido* se convertirá en el botón de *apagado*.

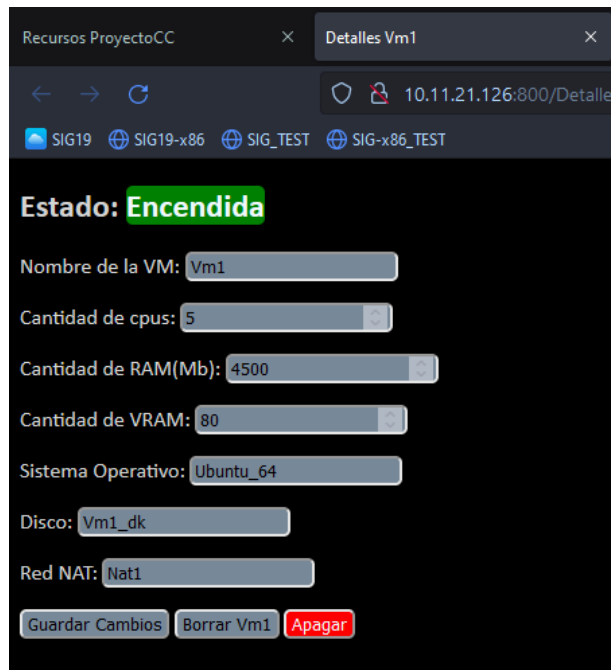


Ilustración 61: Interfaz de detalles de la VM, mientras se encuentra encendida

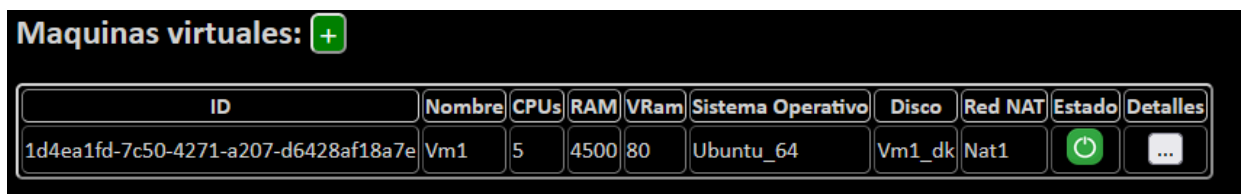
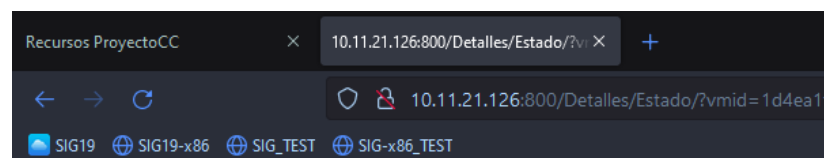


Ilustración 62: Visualización de la VM encendida en la página principal de los recursos

Al momento del encendido, la página nos abrirá una página emergente, con un mensaje que avisa que la VM realizó la acción de manera satisfactoria, y nos dispondrá un id de AnyDesk, que utilizaremos para conectarnos a ella.



Equipo encendido...
Inicie AnyDesk e ingrese el siguiente codigo: 1958095200

Ilustración 63: Mensaje de encendido y código de anydesk de la VM

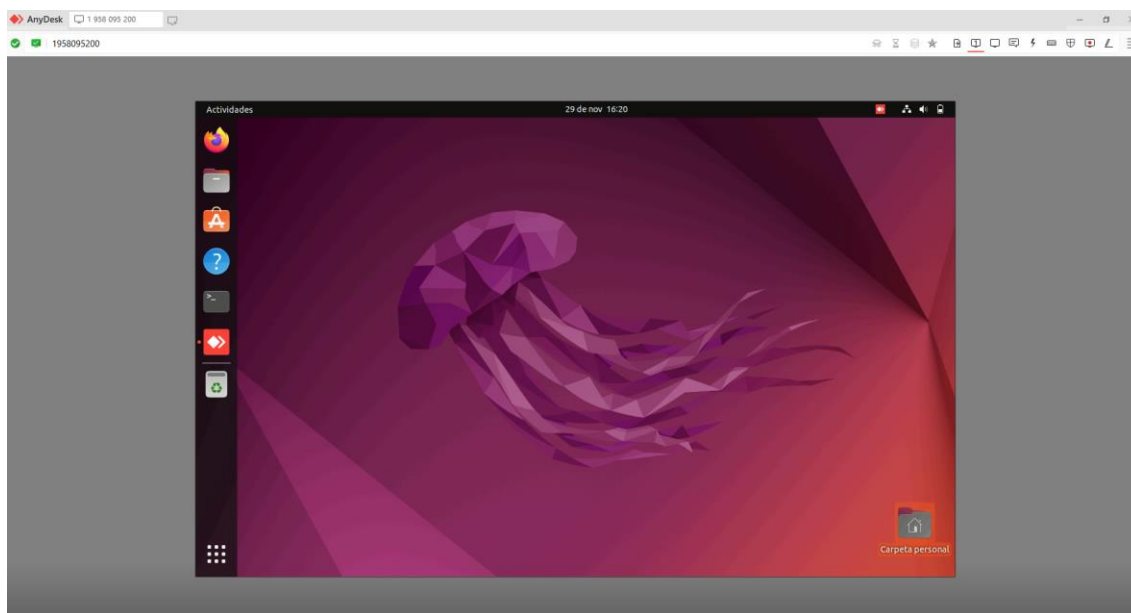


Ilustración 64: Ingreso a la VM por medio de AnyDesk

Conclusión:

Para finalizar éste proyecto, cabe destacar que la página fue realizada con el objetivo de explicar, entender, y estudiar, cómo funciona un servicio de virtualización en la nube, a escalas menores. Basado en la funcionalidad de los servicios que presta *Google Cloud Platform*.

Se espera para el futuro, que todas las empresas, chicas o grandes, se desliguen completamente de la inversión en hardware, como también, que los usuarios finales de dispositivos, no comprometan su propio almacenamiento para guardar archivos importantes, y que toda ésta información, esté resguardada en una nube segura, flexible, e intuitiva.

En conjunto, Cloud Computing no solo ha cambiado la forma en que se administran los recursos informáticos, sino que también ha abierto nuevas oportunidades para la innovación y el crecimiento empresarial. A medida que la tecnología evoluciona, la nube seguirá desempeñando un papel fundamental en la evolución de la infraestructura tecnológica global.

Bibliografía utilizada:

<https://docs.djangoproject.com/en/4.1/>

<https://cloud.google.com/learn/what-is-cloud-computing?hl=es>

<https://www.clarcat.com/comparativa-aws-vs-microsoft-azure-vs-google-cloud-platform/>

<https://cloud.google.com/?hl=es>

<https://support.google.com/a/answer/10100275?hl=es>

<https://bard.google.com/>

<https://www.oracle.com/co/virtualization/virtualbox/>

<https://www.virtualbox.org/manual/ch08.html>

<https://www.oracle.com/mx/technical-resources/articles/it-infrastructure/admin-manage-vbox-cli.html>