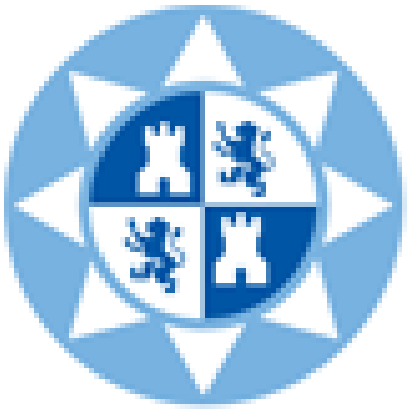
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE CARTAGENA



Trabajo Fin de Grado

Laboratorios docentes en la nube: un caso práctico con AWS

Autor: Alberto Martínez Costa
Director: María Dolores Cano Baños

Resumen

En este proyecto, vamos a explorar el desarrollo y la situación moderna del e-learning y de la computación en la nube. Con esa información crearemos un programa que genere laboratorios personalizados en la nube. Dichos laboratorios podrán ser usados por alumnos como una práctica lectiva dentro del e-learning.

Índice de contenidos.

Índice de figuras

Índice de tablas

Capítulo 1: Introducción y objetivos.

1.1. Introducción.

Este proyecto consiste en la creación dinámica de laboratorios virtuales en la nube, que será usada con el propósito de e-learning. Para ello haremos uso del sistema de programación en la nube AWS y del lenguaje de programación Java que nos permitirá comandar el comportamiento de AWS mediante la creación de archivos JSON.

1.2. Introducción a la computación en la nube. [1] [2]

El origen de la programación en la nube es el ARPANET (Advanced Research Projects Agency Network) en 1969 que – junto con la creación de las primeras máquinas virtuales en la década de los '70 – concedía a los usuarios el poder de acceder a recursos remotos virtualizados. A día de hoy, la nube es uno de los servicios más demandados y utilizados. Toda compañía tecnológica destacada ofrece estos servicios.

Hay varias formas de clasificar los tipos de programación en la nube. En nuestro caso, vamos a requerir de una nube pública (un proveedor externo mantiene la infraestructura y distribuye los servicios por internet) y del tipo IaaS (Infraestructure as a service; El proveedor distribuye virtualizaciones y el almacenamiento para mantenerlas).

Hemos elegido la programación en la nube por las siguientes ventajas:

- **Elasticidad**. Al tratarse de un laboratorio de prácticas podemos esperar mucha variabilidad. No solo de asignatura en asignatura, pero incluso de sesión en sesión.
- **Pago por uso**. Al ser los laboratorios necesarios durante solo unas horas a la semana, el tiempo de inactividad se puede utilizar para ahorrar en servicios por demanda.
- **Auto-Servicio**. Nos permite crear los diferentes laboratorios sin mucha dificultad.

Otras ventajas menos relevantes para nosotros son: redundancia para el almacenamiento más resiliente, flexibilidad en la migración, amplio acceso de red, la compartición de recursos…

La mayor desventaja es la seguridad. Es muy dependiente en una compañía externa y es poco transparente con respecto al almacenamiento de los datos. Sin embargo, como sólo queremos utilizar la nube para prácticas de universidad, este riesgo no es tan importante. Nos debería preocupar más:

- La **imprevisibilidad del coste**. En el capítulo 4 he calculado el coste estimado de toda la operación, pero todos los precios están sujetos a cambios y un despliegue muy ambicioso podría salirse de presupuesto también.
- El **desempeño de los servicios** puede ser variable dependiendo del estado de nuestro proveedor.

1.3. Introducción al e-learning. [3] [4]

El e-learning (aprendizaje electrónico) consiste en el uso de tecnologías informáticas para mejorar la experiencia lectiva. Una gran parte del e-learning se encuentra ligada a la educación a distancia y su propósito es crear la infraestructura necesaria para que un alumno pueda acceder

a material lectivo, resolver ejercicios en línea, examinarse… Este proyecto se encuentra dentro de esta área del e-learning. Su finalidad es crear redes virtuales que los alumnos puedan utilizar para hacer pruebas desde cualquier lugar, sea este un ordenador provisto por la universidad o uno propio.

Los proyectos de e-learning se distinguen principalmente entre síncronos y asíncronos. En los proyectos síncronos, la diseminación de información por parte del profesor y su adquisición por parte del alumnado es instantánea. Se basan en clases presenciales emitidas en videoconferencia a los alumnos, con chat en vivo. En los proyectos asíncronos, el profesor crea materiales educativos – videos o apuntes, principalmente – y los alumnos los estudian en su propio ritmo. Estos modelos priorizan la flexibilidad y normalmente se acompañan con ejercicios con auto-corregido o entregables que el profesor corregiría en su propio tiempo.

En el capítulo 4, cuando estimemos el precio del despliegue, va a resultar obvio que el precio de activar los ordenadores dos horas a la semana y mantenerlos inaccesibles el resto del tiempo es más económico que dejar que los alumnos se conecten cuando quieran. El despliegue es más económico de forma síncrona. Sin embargo, para determinados sistemas operativos, la diferencia de precio es mínima, por lo que, mediante la opción de terminar o detener (que se explicará más ampliamente en el capítulo 3), el programa dará la opción de elegir una forma u otra en cada despliegue.

Las ventajas del e-learning relevantes a nuestro proyecto son:

- La capacidad para la **gestión conjunta y eficaz** de un gran número de participantes de diferentes procedencias y perfiles. No siendo necesaria la asistencia física a la universidad, la capacidad y la diversidad del aula aumenta.
- Es posible adaptar el laboratorio a una rápida **actualización de los contenidos** del curso gracias a la interfaz gráfica.

Una de las desventajas del e-learning con la que nos encontramos de frente es la falta de control del profesor con respecto al alumno. Más allá de saber si el alumno está conectado a una u otra instancia, el maestro no tiene ningún conocimiento de las actividades de su alumno y tiene que conformarse con evaluar memorias. Por otra parte, en caso de que este proyecto se despliegue tal que los alumnos deban acceder al laboratorio virtual desde su propia casa, dejar la base de hardware en manos de los alumnos puede resultar en disparidad de calidad en función de la clase del alumno.

1.4. Objetivo del TFE.

El objetivo de este TFE es aplicar de forma práctica conocimientos relativos a la programación en la nube y la creación de redes virtuales. Además, supondrá un aprendizaje más profundo del lenguaje de programación JAVA, estudiando cómo se realizan aplicaciones gráficas y cómo se genera, se manipula, y se escribe en un archivo un programa JSON.

1.5. Introducción a los siguientes capítulos.

Los próximos capítulos consisten de:

- Capítulo 2: Computación en la nube. Una expansión de los conceptos teóricos vistos en esta introducción junto con una discusión sobre las tecnologías disponibles hoy en día.

- Capítulo 3: Propuesta de un laboratorio en la nube. El desarrollo práctico del trabajo.
- Capítulo 4: Estimación de costes. Un cálculo aproximado del coste necesario para llevar la operación al mundo real.
- Capítulo 5: Conclusión. Resumen y propuestas de mejora.

Capítulo 2: Computación en la nube.

2.1. La computación en la nube. [5] [6]

La computación en la nube consiste en el acceso remoto y bajo demanda a una red virtual o a cualquiera de los servicios que ofrece. De acuerdo a la definición del NIST [7] las características necesarias de un sistema de computación en la nube son:

- **AutoServicio bajo demanda.** Un usuario puede aumentar su acceso a recursos informáticos como el tamaño de almacenamiento sin necesidad de comunicarse con el proveedor de los servicios.
- **Amplio acceso remoto.** Las redes utilizarán tecnologías estándar para que se puedan conectar a ellas diversos tipos de dispositivos.
- **Pooling de recursos.** Los recursos del proveedor se encuentran dentro de una misma pool de la que se extraen para satisfacer las necesidades del cliente. Dicho cliente no suele tener información de dónde en la pool vienen sus recursos.
- **Elasticidad.** Los proveedores tienen que estar preparados a un aumento rápido de la demanda por parte de la clientela.
- **Servicio medido.** Es el proveedor de recursos en la nube el que controla y optimiza los recursos

Con esta definición vemos que la computación en la nube está basada principalmente en dos tecnologías: La computación de la utilidad (o computación bajo demanda) y los sistemas peer to peer.

En la década de 1960, Carl Joseph Licklider Robnett, jefe del grupo ARPANet, concebía para la computación un futuro en el que cualquiera podría tener acceso a servicios computacionales. Sus escritos han sido aclamados como uno de los primeros pasos hacia el modelo de internet, pero también son claramente aplicables e inferenciales en el desarrollo de la nube. Su propuesta implicaba el acceso compartido a una serie de recursos de forma remota.

Durante el mismo periodo, John McCarthy sería uno de los primeros en proponer una aproximación a la computación como un servicio público igual que la telefonía. En este caso, John McCarthy tenía una propuesta ya que poco antes había creado el primer sistema de tiempo compartido conocido en el MIT. Previamente, los computadores se operaban mediante cartas de instrucciones, que formaban un programa o algoritmo. McCarthy propuso conectar estos ordenadores a los teletipos de la facultad, lo que permitía el uso simultaneo del computador por varias personas. Unos años más tarde nació el Compatible Time Sharing System (CTTS).

Los sistemas peer to peer, por otra parte, son un tipo de sistema distribuido. Un sistema distribuido es un sistema en el cual los componentes localizados en computadores de una misma red se pueden comunicar mediante mensajes. [8]

Con la creación de redes distribuidas más complejas, las expectaciones crecieron, y el desarrollo de sistemas distribuidos se centró en garantizar ciertas prestaciones. Estos incluyen:

- **Heterogeneidad**. Es la cualidad de poder interactuar con la red desde varios computadores con diferente hardware. Es el hardware el que se ocupa de traducir su

lenguaje interno si quiere acceder a una red distribuida, tal como internet. Esta capa traductora se denomina middleware.

- **Escalabilidad**. La escalabilidad consiste en garantizar el posible crecimiento de una red sin sacrificar su efectividad.
- **Manejo de fallos**. Hay varios modos de interactuar con un fallo: prevenirlos, contenerlos (reducir su impacto) o recuperarse de ellos. La prevención se basa en la redundancia de datos para detectar un mensaje fallido. La contención consiste en asumir que el error es inevitable e intentar reducir su impacto lo máximo posible. La recuperación se basa en las copias de seguridad y el manejo de datos distribuidos.
- **Transparencia**. La transparencia consiste en esconder los procesos internos al usuario o al proceso final. Cuando un proceso accede a un recurso remoto tiene que creer que es local (transparencia de acceso), cuando accede a un recurso lo puede hacer sin saber dónde está (transparencia de localización) y sin ser interferido por otros procesos que quieren acceder al mismo recurso al mismo tiempo (transparencia de concurrencia)…

La apertura, la concurrencia y la seguridad son otros focos de desarrollo, pero arriba están los más relevantes a la nube. Mediante la heterogeneidad y la transparencia, un usuario con un hardware cualquiera puede acceder de manera transparente a un recurso sin importar su localización física. La escalabilidad cobra más importancia cuando, como veremos a continuación, el crecimiento de la nube en la última década ha sido espectacular. El manejo de fallos es necesario ya que cada vez más aplicaciones sensibles están acabando en la nube.

Existen varios modelos en la arquitectura de un sistema distribuido. En el principio, los más comunes eran las varias iteraciones de los cliente-servidor. Dónde el rol de intérprete y proveedor de la información está claramente definido, tal que el/los cliente(s) invocan procesos en el/los servidor(es) de manera heterogénea, transparente…

Las redes peer to peer (P2P) nacen en los ‘90. Consisten en crear un dispositivo “peer” que tenga la capabilidad de actuar tanto de cliente como de servidor. Estos peer pueden ser colecciones de varios dispositivos y recursos. En general, aumentaron considerablemente la flexibilidad en las redes distribuidas y facilitaron la creación de la nube. En las figuras 1 y 2 podemos ver las diferencias entre típicas arquitecturas cliente-servidor y arquitecturas P2P.

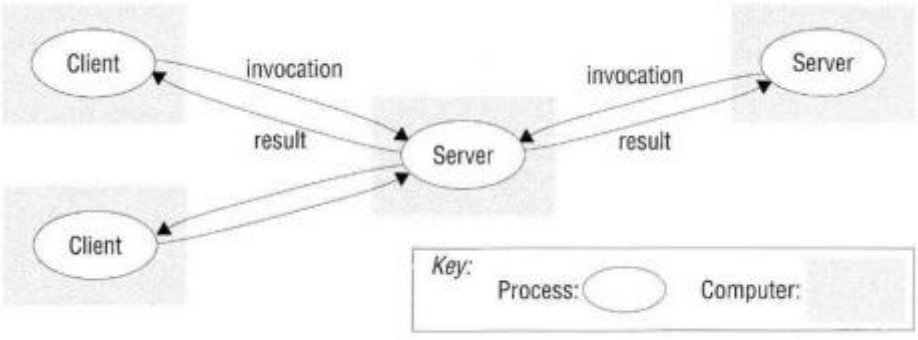


Fig. 1. Estructura cliente-servidor [8]

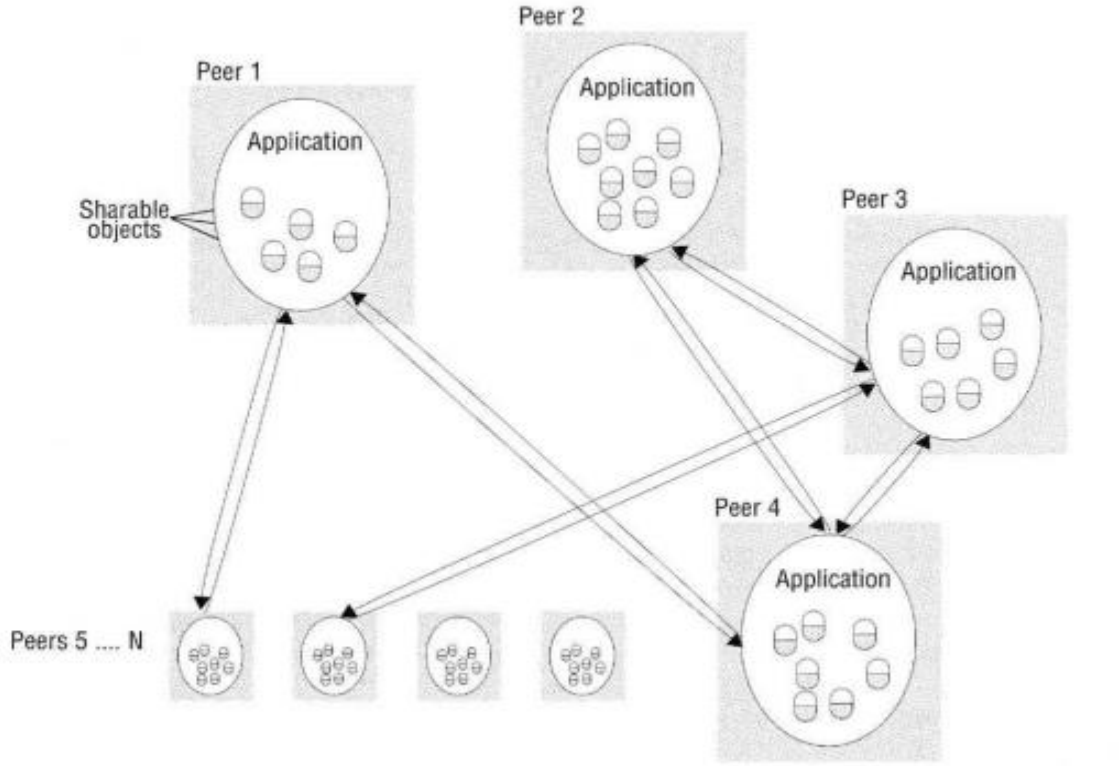


Fig. 2. Estructura p2p [8]

Sin embargo, esto no sería suficiente para que los servicios de computación en la nube se convirtieran en lo que son hoy en día. Según escribe V. Rajaraman en resonance magazine [6] los principales motivos de este rápido crecimiento son:

- **El aumento de ancho de banda y de velocidad de CPU**. En el caso del ancho de banda, es posible, ahora, enviar grandes cantidades de información a computadores remotos en países lejanos y casi no notar la diferencia. Se puede operar un ordenador remoto sin la cantidad de problemas de lag y stutter que nos encontrábamos en el pasado. Con respecto a la CPU, su mejora nos permite permutar las solicitudes de varios clientes tal que puedan compartir un mismo ordenador sin retrasos o problemas. Todos estos cambios han sido posibles sin un aumento en el precio de CPUs o ancho de banda.
- **Avances en sistemas operativos** que han hecho posible la fácil **virtualización** (simulación de su estructura desde otras máquinas) en la que se basa la nube. La posibilidad de crear diferentes cuentas en un mismo computador era un paso necesario para el correcto funcionamiento de la nube.
- Al mismo tiempo, el **coste de la infraestructura** computacional a nivel de empresa ha aumentado dada la digitalización de más y más ramas administrativas. Coste que se muestra tanto en la construcción como en el mantenimiento de estas redes. Contemporáneamente, la posición más común en las empresas medianas es dejar la computación a manos de terceros. A manos de la nube.
- Por último, la **consolidación** en el mundo tecnológico que ha dejado a Microsoft, Google y Amazon como los grandes ganadores. Todos ellos con los recursos para establecer grandes redes capaces de proveer estos servicios a precios razonables – Microsoft Azure, Google Cloud y AWS. Tanto los recursos como la confianza en estas empresas han hecho que más empresas medianas se decanten por servicios en la nube.

La virtualización, en concreto, se ha convertido en la tecnología principal de la nube. Mientras que otorgar acceso a ordenadores físicos de manera remota constituye una nube, este modelo es poco escalable y raramente rentable. Durante los últimos años, las empresas que ofrecen servicios cloud han optado por comprar hardware genérico sobre el cual emular cualquier sistema o servicio se requiera de ellos bajo demanda. A día de hoy, la computación en la nube es una forma de acceso a computadores o servicios virtuales.

2.2. Virtualización [9]

Al igual que la nube, la virtualización tiene precedentes en los ‘60. Tanta es la coincidencia que el CTTS mencionado anteriormente es considerado uno de los primeros ejemplos de virtualización. En los días tempranos de la computación, la computación en la nube y la virtualización eran concebidas como dos partes de un mismo proceso. Si el objetivo final era la creación de una red computacional como internet accesible públicamente, el proceso por el que se iba a conseguir era la partición de recursos de un grupo de computadoras universitarias. Cada partición sería accesible libremente por el público.

Sin embargo, con la llegada de los ordenadores personales en la década de los 80, la aproximación a internet cambió. Ahora que muchos ordenadores de bajo nivel computacional eran más baratos que pocos ordenadores de mucho nivel computacional estaba claro que en lugar de partir las grandes máquinas y ponerlas a disposición del público, era preferible conectar las pequeñas máquinas del público a internet. La virtualización pasó a un segundo plano.

Pero, como ya hemos visto, como consecuencia de la consolidación de las empresas tecnológicas, unas pocas han adquirido tal poder adquisitivo como para hacer este modelo rentable otra vez. La balanza ha tornado y, desde finales de los ‘90, vuelve a ser más barato comprar pocos ordenadores de mucho poder computacional. En contraste no a los ordenadores personales, sino a las infraestructuras de empresas menores, que son más ineficientes al ser más pequeñas.

Los dos conceptos sobre los que se basa la virtualización son la **máquina virtual** y el **hipervisor**. Existen dos tipos de máquinas virtuales: de sistema o de proceso. Las máquinas virtuales de proceso actúan como un proceso sobre el que se ejecuta una aplicación determinada. En el caso de nuestro proyecto estamos mirando a máquinas virtuales de sistema, y en ellas nos vamos a centrar.

Las máquinas virtuales de hardware o de sistema son aquellas que tienen acceso al hardware del anfitrión. En este caso, tienen acceso solo a una parte de todo ese hardware, pero viven bajo la ilusión de que eso es todo lo que hay. Se replica el sistema completamente y no solo una funcionalidad o proceso determinado. Una vez replicado, el sistema puede ejecutar esos procesos como si fueran nativos.

Los hipervisores se dedican a monitorear y controlar las máquinas virtuales que se construyen encima suyos. Son una capa intermedia de software entre el hardware del sistema anfitrión y el hardware simulado de la máquina virtual. Aunque recomendado, no es necesario incluirlo en todos los proyectos de virtualización. Sin embargo, es necesario para: Captar, responder e identificar operaciones CPU y manejar las colas, el envío y recibo de peticiones de acceso a los recursos de hardware. Las figuras 3 y 4 muestran la diferencia entre un ordenador típico y un ordenador usado como base para un sistema de virtualización.

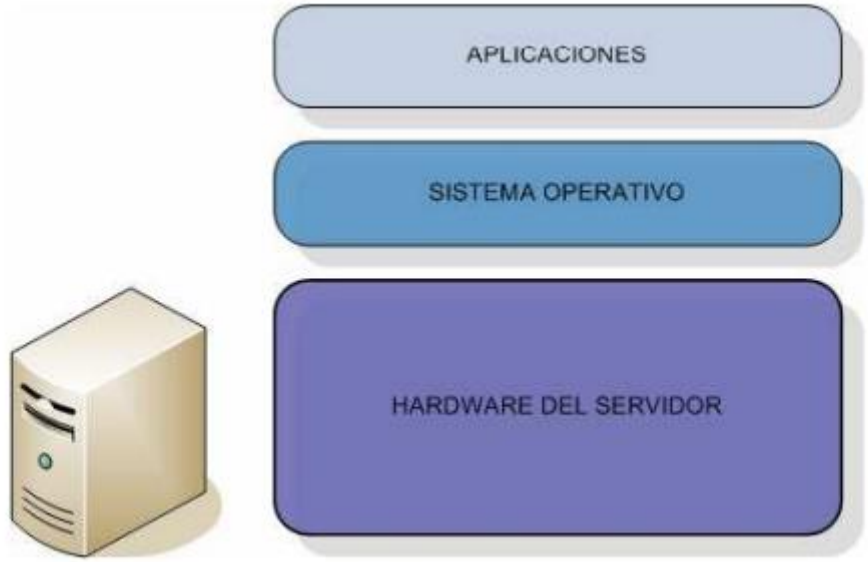


Fig. 3. Estructura de un computador normal [9]

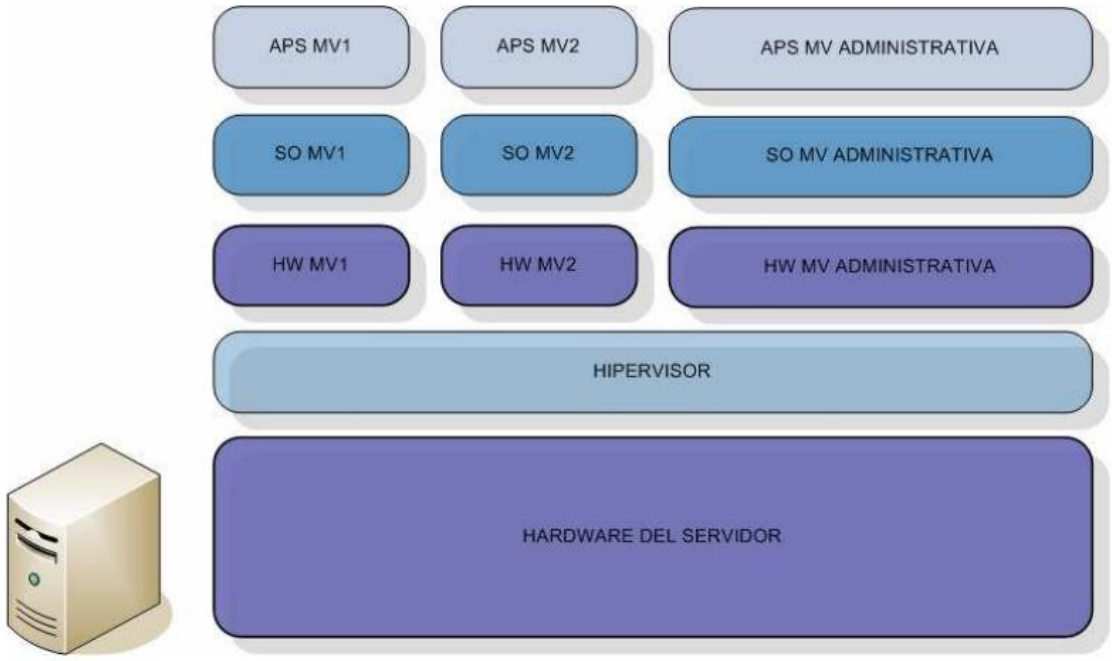


Fig. 4. Estructura de un computador con máquinas virtuales y un hipervisor [9]

Dependiendo del nivel en el que se virtualice un sistema, se pueden distinguir varios tipos de virtualización. Los de más baja abstracción, simularán todo un sistema operativo o toda una red. Cuanto más se abstraigan, más particulares se vuelven, hasta el punto en el que solo pueden simular un proceso concreto en un sistema. Esta clasificación está directamente correlada con la distinción entre máquinas virtuales de sistema y de proceso. También lo está con los distintos modos de funcionamiento de la nube que vimos en la introducción, XaaS.

Dependiendo del nivel de abstracción requerido, hay tres formas de clasificar un servicio en la nube (figura 5): IaaS (Infraestructura como servicio), PaaS (Plataforma como servicio) y SaaS (Software como servicio). Con IaaS nos encontramos directamente encima del hipervisor, con control sobre el sistema operativo y todas las funciones de nivel superior. PaaS y SaaS están construidas sobre IaaS. En el caso de PaaS, el proveedor ofrece tanto el sistema operativo como el entorno ya resuelto. El cliente, entonces, crea aplicaciones de software sobre ese entorno. En SaaS, el software ya está creado por el proveedor y el cliente solo tiene poder de usarlo. Este sería el paralelo a la virtualización de proceso.

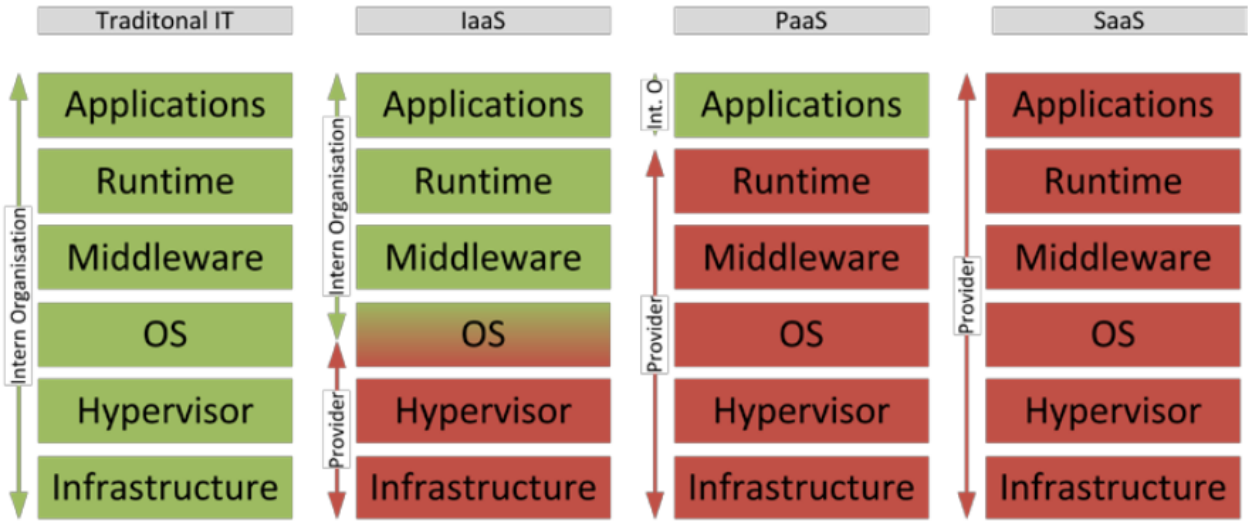


Fig. 5. Distintos modelos de computación en la nube [10]

En nuestro caso necesitamos manipular tanto el sistema operativo como la conexión entre las distintas virtualizaciones. Necesitamos un proveedor de IaaS.

2.3. Plataformas de computación en la nube. [10]

Las plataformas de computación en la nube están divididas según la relación del o de los clientes con el hardware sobre el que ejecutan sus virtualizaciones. Podemos distinguir cuatro tipos: [6]

- **Nubes públicas.** En este caso, el proveedor mantiene sus recursos independientemente de los clientes. Los clientes, entonces, comparten los recursos cuando quieran acceder a ellos. El modelo económico es el "bajo demanda" la mayoría de las veces. Son los más escalables y flexibles. Sin embargo, resulta en los clientes siendo muy dependientes de un proveedor externo.
- **Nubes privadas.** Son similares a las públicas en arquitectura y funcionalidad, pero en este caso la relación proveedor-cliente se rompe. Una nube privada es creada por una empresa con el propósito de ser utilizada por esa misma empresa. Al contrario que las redes públicas, tienen problemas de escalado y flexibilidad, pero otorgan más control e independencia.
- **Nubes híbridas.** Son una mezcla de públicas y privadas. La empresa divide sus necesidades en diferentes tareas independientes y las soluciona en nubes privadas o nubes públicas dependiendo de sus preferencias. Creando un sistema dónde el peso de la parte pública o la privada puede cambiar en cualquier momento. Este modelo contiene

los beneficios de los modelos anteriores, pero estructurar necesidades en tareas independientes puede ser complicado o imposible para ciertos procesos.

- **Nubes comunitarias.** Si las nubes híbridas implican una diversidad de proveedores; las nubes comunitarias implican una diversidad de clientes. En estos casos varias empresas comparten una misma nube. Dicha nube puede ser pública, privada o híbrida. Las nubes comunitarias suelen ayudar con el precio ya que muchas nubes públicas ofrecen descuentos por gran usaje.

En nuestro caso, necesitamos una nube pública ya que crear una nube privada está fuera de los límites de este trabajo y tanto las nubes híbridas como las comunitarias las requieren.

Los principales proveedores de computación en la nube de entre los que debemos elegir son: Amazon AWS, Microsoft Azure, IBM Cloud y Google Cloud.

IBM Cloud es, fundamentalmente, una infraestructura de comercio. La seguridad (tanto la privacidad como la recuperación de datos) es un foco principal de la plataforma y la mayoría de su matemática e IA (Inteligencia Artificial) está destinada a operaciones económicas. Por otra parte, la virtualización no es una prioridad, lo cual nos complicaría el trabajo. Debido a esta especialización, es también uno de los servicios más caros, por lo que lo descartamos en este trabajo. [11]

Similarmente, Google cloud ofrece una gran potencia y la mayor versatilidad de todos los servicios. Además, su tecnología Cloud Run permite generar redes automáticamente desde código Java, sin necesidad de crear un archivo JSON de intermediario. Sin embargo, los precios se disparan. En su familia VM Tau, la máquina N1 (la más barata) corre a 0,0042 US$/GB hora en almacenamiento. La instancia similar en AWS corre a 0,00013 US$/GB hora; en Microsoft Azure, a 0,00028 US$/GB hora. [12]

Microsoft Azure y AWS son las plataformas más cercanas. Ambas tienen un precio similar, interfaces accesibles y fáciles de entender y aplicaciones robustas para la creación automática de redes virtuales basados en JSON (ARM templates y CloudFromation). [13] [14]

La mayor diferencia la encuentro en el acercamiento al e-learning. Microsoft Azure dispone de Lab Services (un servicio de organización y control. Como Microsoft Teams, Zoom y otras aplicaciones educativas, pero más cercanas a las máquinas virtuales). AWS, además, incluye nodos para el rápido sreaming y distribución de videos y una amplia gama de herramientas IA y ML (Machine Learning) para personalizar la experiencia del estudiante. AWS también ofrece la ayuda AWS Cloud Credit for Research para investigadores. Ambas opciones están fuera de los límites de este trabajo. Pero, pensando a largo plazo, AWS ofrece más oportunidades de crecimiento.

Capítulo 3: Propuesta de laboratorio docente en la nube.

3.1. Amazon Web Services.

Nuestro objetivo final es crear, mediante Java, un programa que automatice la creación de archivos JSON que generen una cierta red virtual para ser usada por estudiantes. Para poder simular la red a partir de un archivo JSON utilizaremos la herramienta CloudFormation. Será necesario crear, aparte de los escritorios virtuales o instancias, una subred compartida por los PCs de un mismo equipo, un grupo de seguridad para aislarlos y un par de claves de acceso.

Subredes. [15]

En AWS, una subred es una partición de una VPC (Virtual Private Cloud), la cual simula una red física. La VPC de nuestra región (Norte de Virginia) es 172.31.0.0/16 y el rango de IPs 172.31.0.0 - 172.31.96.0 ya está en uso por otras subredes. En nuestro caso empezaremos a ocupar por 172.31.200.0. En AWS, además de las direcciones de red y de broadcast, las tres direcciones tras la IP de red están reservadas para varios aspectos. Así, la primera instancia de nuestro laboratorio deberá recibir la IP 172.31.200.4.

Ya que en nuestro proyecto solo es necesaria la conexión entre PCs de la misma subred, no hace falta configurar las tablas de enrutamiento. Todas las instancias tienen una norma default que los dirige hacia su router. En una subred lo único que vamos a configurar es la dirección IP de subred.

Grupos de seguridad. [16]

Un grupo de seguridad es un firewall virtual. Cada VPC incluye un grupo de seguridad por defecto, pero nosotros crearemos uno personalizado. Rechazando todo por defecto y permitiendo acceso para (1) tráfico desde y hacia internet (tanto http como https), (2) tráfico RDP desde el ordenador físico del alumno y (3) tráfico de cualquier tipo entre los ordenadores de la subred.

Instancias. [17]

Una instancia es una virtualización de un escritorio. En AWS son EC2 (Elastic Computer Cloud). Para estas debemos determinar el sistema operativo que queremos, la ip del dispositivo y el comportamiento de cierre (el cual será discutido más ampliamente en el capítulo 4). También debemos hacer referencias dinámicas a la subred y al grupo de seguridad creados arriba. Esto se realizará internamente con constantes de clase.

Pares de claves. [18]

Una clave pública es necesaria para generar una instancia, y una privada es necesaria para acceder a esta. AWS nos proporciona recursos para crear pares de claves fácilmente. En nuestro caso elegimos claves RSA por su facilidad y confiabilidad.

Mientras que es posible generar un par de claves mediante CloudFormation, la clave privada no se almacena en ningún sitio y desaparece. Es necesario crear las claves manualmente antes de la ejecución del programa. Una por cada grupo de prácticas.

CloudFormation. [19]

CloudFormation es un recurso que permite crear y manipular redes virtuales por medio de plantillas JSON o YAML. En nuestro caso, solo vamos a estar interesados en la creación. La documentación de AWS nos ofrece los modelos necesarios para crear los recursos necesarios para nuestro proyecto: subredes, grupos de seguridad e instancias.

La plataforma AWS.

Al entrar en AWS nos encontramos con una gran variedad de herramientas y menús de control de recursos. Los menús más importantes para nosotros son EC2 y VPC.

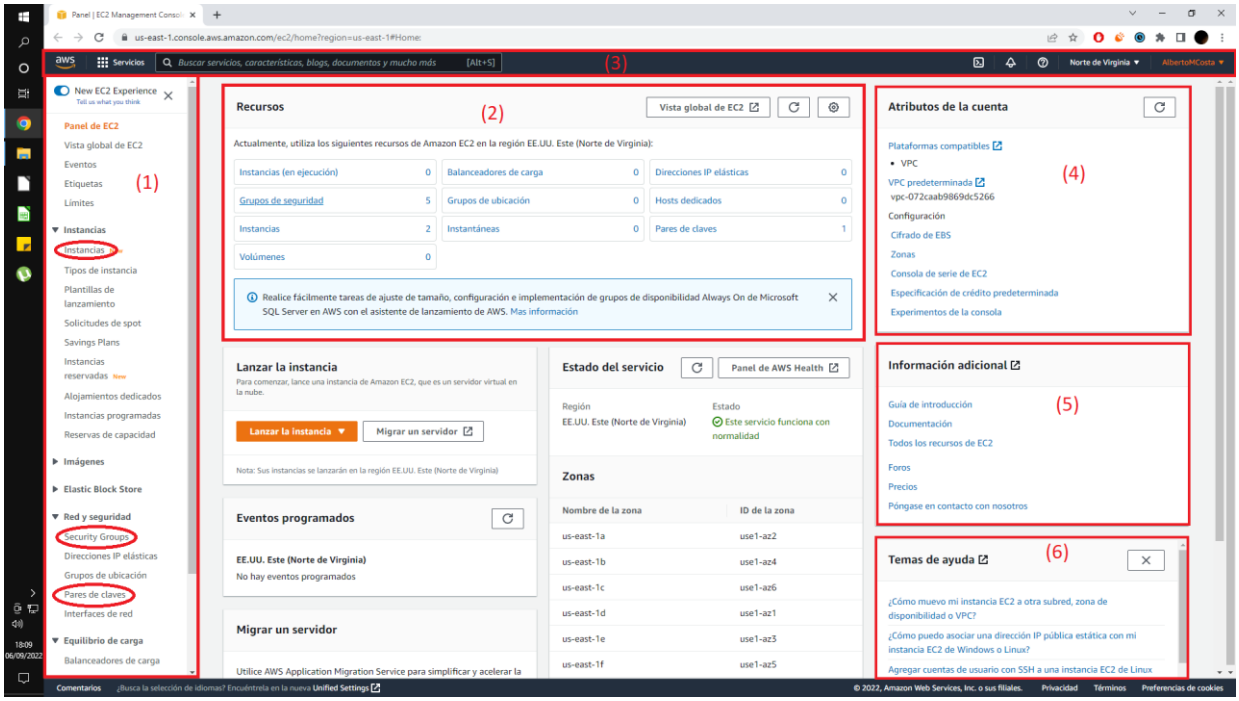


Fig. 6 Menú EC2 en AWS

En la vista EC2 (figura 6) podemos distinguir varias partes. A la izquierda, (1), tenemos el menú desplegable con los diferentes recursos que se monitorizan aquí. Incluyendo instancias, pares de claves y grupos de seguridad. Pulsando sobre ellas, llegamos a una página específica (vista en la figura 8) desde la que podemos monitorear más precisamente su comportamiento. En el centro (2) está el resumen general. AWS toma los recursos que considera más importantes y los muestra en primer plano. Este es también el caso para (4), solo que esta no se conforma con datos dentro de EC2 y ofrece también información sobre VPC. Debajo hay algunas acciones comunes como lanzar una instancia, pero estas van a estar siempre disponibles en las páginas personales de cada recurso. Arriba, (3), está la información de usuario. Los dos últimos cuadros bajo (4) son de ayuda. (5) tiene accesos directos a tutoriales relacionados, y (6) a foros.

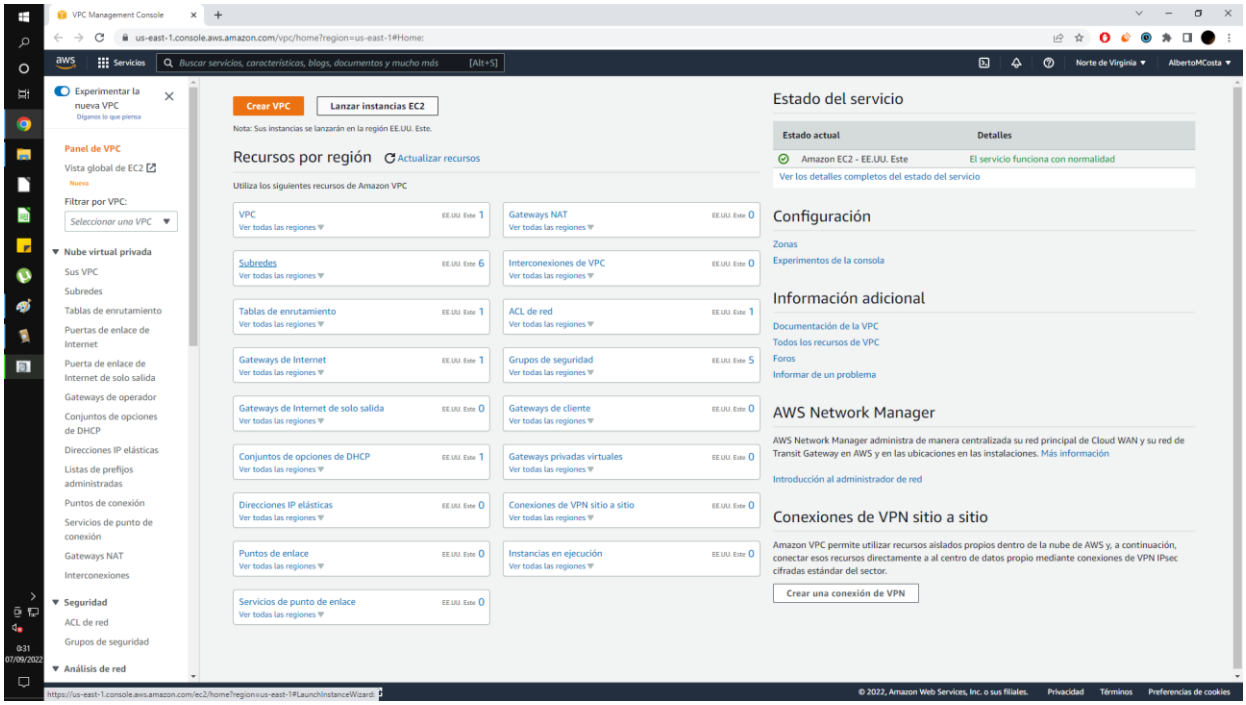


Fig. 7 Menú VPC en AWS

El menú de VPC (figura 7) está organizado de la misma manera: menús a la izquierda, resumen en el centro y ayuda a la derecha. En este menú nos encontraremos con las subredes.

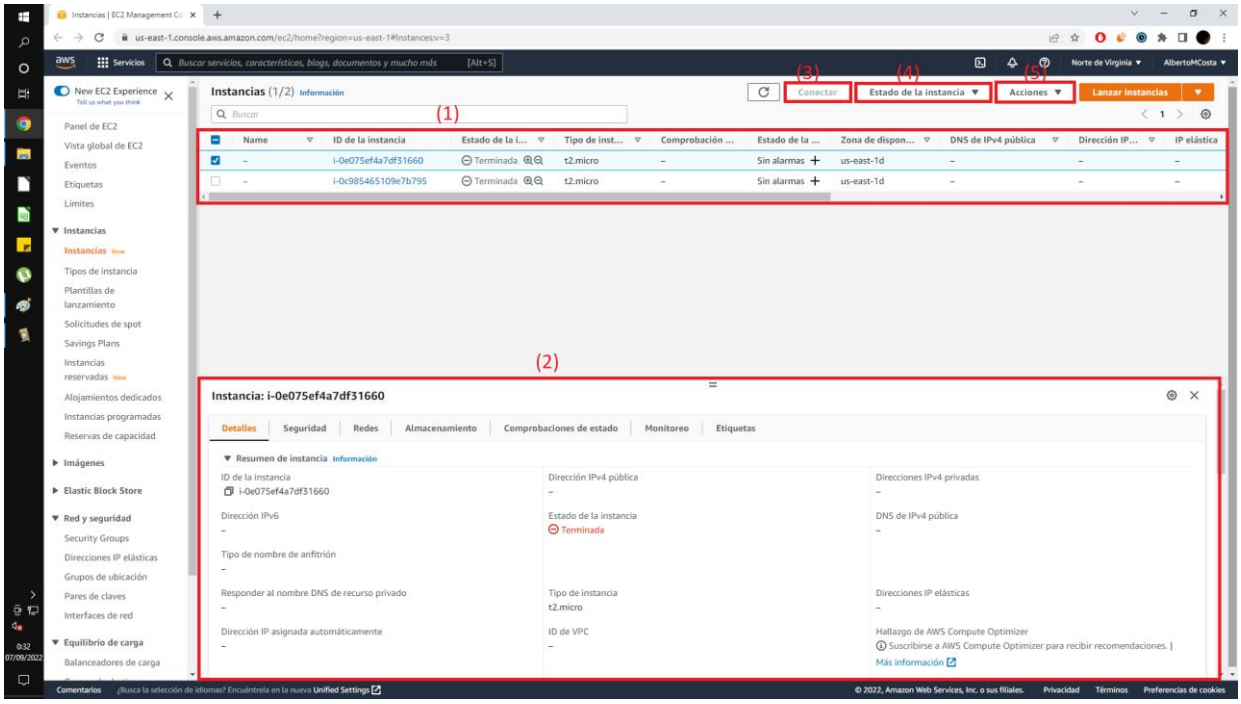


Fig. 8 detalles de instancia en AWS

Dentro de cada recurso nos encontramos con una página específica (figura 8) que enumera las instancias de ese recurso y su actividad. Así como otros datos más detallados. La enumeración (1) se sitúa en el medio de la pantalla. Junto con el nombre de cada recurso, podemos ver algunos de sus datos más importantes. Si pulsamos en alguno de ellos, aparecerá una pestaña

con un menú desplegable abajo (2) con datos más específicos. Si se selecciona más de un recurso a la vez, esta pestaña mostrará los datos que se puedan agregar como el consumo.

Arriba están los botones de acción. En este caso estamos en la página de las instancias, así que hay un botón de conexión (3), uno de estado (4) para activar, desactivar, poner a dormir... la instancia, y uno de acción (5) que se puede utilizar, entre otras cosas, para obtener la contraseña de Windows.

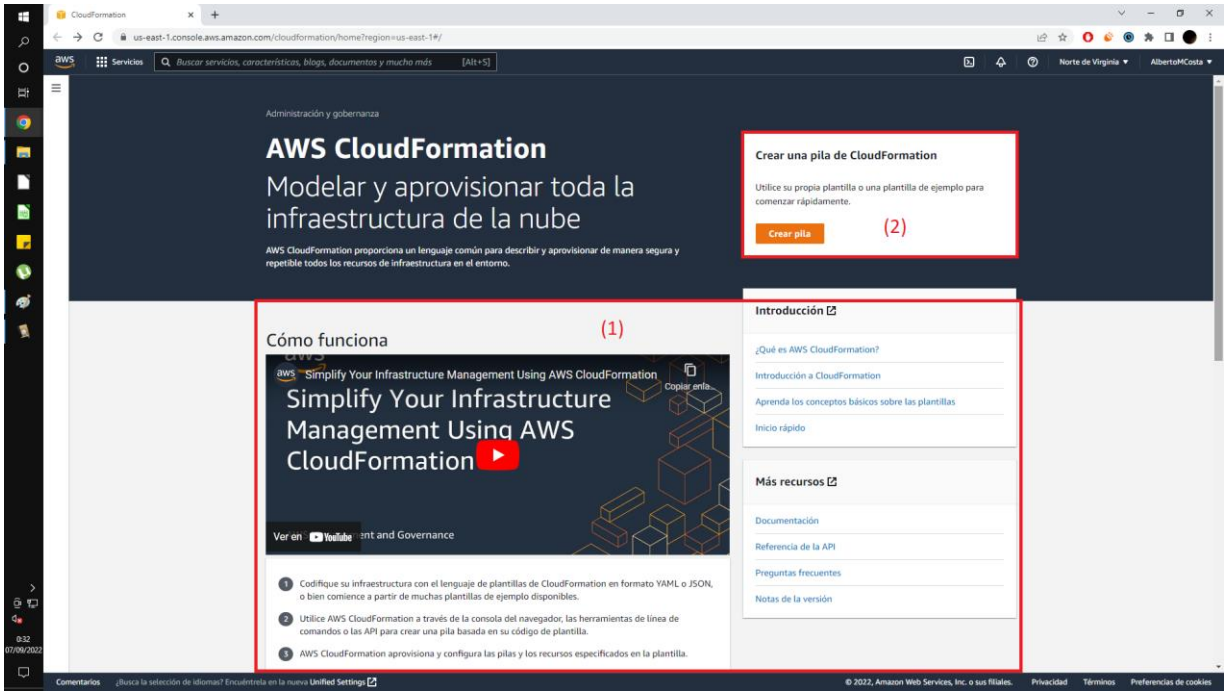


Fig. 9 Menú CloudFormation en AWS

En cuanto a los servicios, nos interesa CloudFormation. Podemos llegar de la misma manera que a los menús: buscando el nombre en el buscador en la parte superior. Normalmente, las páginas de servicios se basan en varios menús de ayuda (1) debajo y un botón de acceso al constructor del servicio (2) arriba a la derecha. La figura 9 nos muestra la página de CloudFormation.

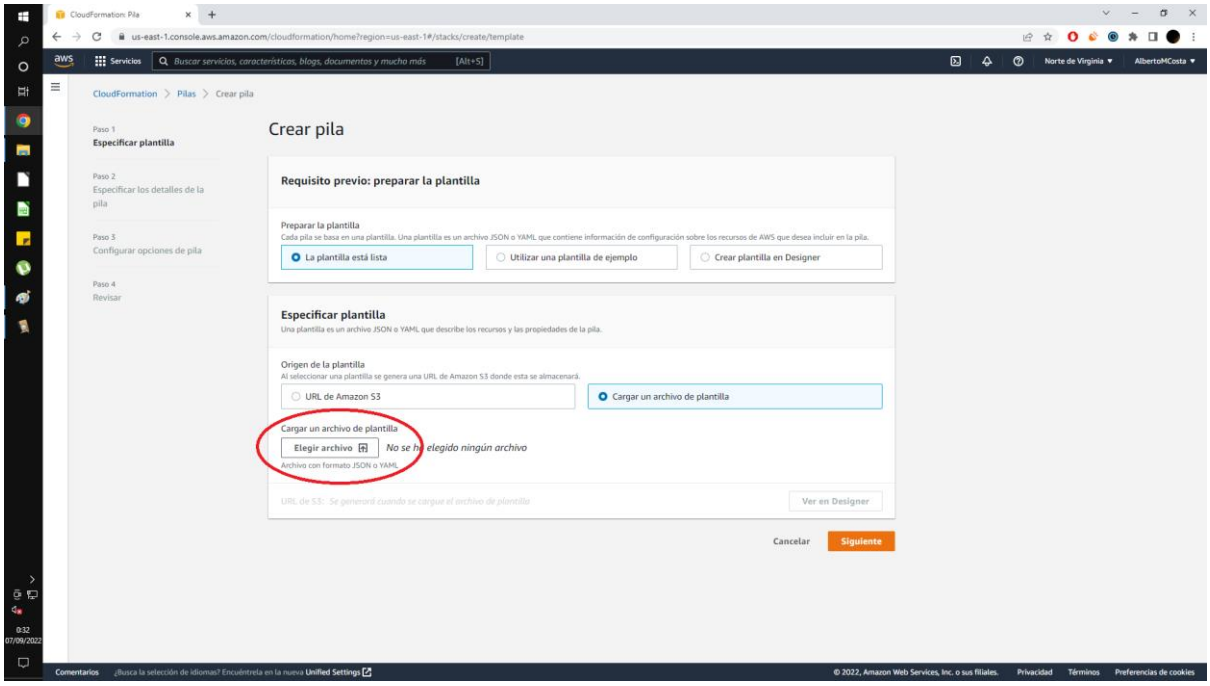


Fig. 10 Menú creación CloudFormation en AWS

Una vez dentro del constructor (figura 10), AWS requiere un fichero JSON para empezar a trabajar, que es lo que vamos a crear ahora.

3.2. Programa Java

El desarrollo del programa Java se divide en dos partes fundamentales: La generación del archivo JSON a partir de unos parámetros determinados, y la creación de una interfaz por la que el usuario pueda insertar dichos datos.

Generación de archivo JSON

En primer lugar, es necesario crear un archivo contenedor y escribir, al principio y al final, las llaves necesarias de un programa JSON como se ve en la figura 11.

```
try {
    FileWriter escritura = new FileWriter("aws.json");
    String opener = "{\n" +
                    "   \"Description\":\"generate the labs\",\n" +
                    "   \"Resources\":{\n";
    escritura.write(opener);

                    String closer = "   }\n" +
                                    "}";
                    escritura.write(closer);

                    escritura.close();
```

Fig. 11. Comienzo y final del JSON

En medio de estos dos puntos se desarrollará nuestro programa. Los parámetros recibidos desde la interfaz serían: nEquipos, un int que nos indica el número de equipos (subredes y grupos de seguridad) que hay que crear; el array nPC, que nos indica el número de PCs (instancias) que hay que simular por grupo; el array de Strings ipLab, con las IP físicas del laboratorio; el array de Strings sitemaOperativo, con el sistema operativo; el array de booleans terminar, que es true o false dependiendo de si las máquinas son terminadas o detenidas al final de una sesión; y el boolean configuracionIndividual, que nos dice si los valores de sistemaOperativo y terminar son distintos para cada instancia (en cuyo caso su longitud sería el sumatorio del array nPC), o si son comunes para todos los PC de un equipo (en cuyo caso su longitud sería de nEquipos)

También están definidos los valores internos de subredId, grupoSeguridadId e instanciaId para facilitar los nombramientos. Y los valores ipRed3 e ipRed4, que coinciden con los dos últimos números de una IP y que aumentarán en valor después de cada subred creada para darle una IP de red a la siguiente. Todo esto se ve en la figura 12.

```
class GenerarJson{

    private int subredId = 0;
    private int grupoSeguridadId = 0;
    private int instanciaId = 0;

    public GenerarJson(int nEquipos, int[] nPC, String[] ipLab, boolean configuracionIndividual,
            String[] sistemaOperativo, boolean[] terminar) {

        int ipRed4 = 0;
        int ipRed3 = 201;
```

```
   for(int i =0;i<nEquipos;i++) {
       int ac = 0;

       escritura.write(GenerarSubred(ipRed4, ipRed3));
       escritura.write(GenerarGrupoSeguridad(ipLab[i]));

       for (int j = 0;j<nPC[i];j++) {
           boolean ultimo=false;
           if(i==nEquipos-1 && j==nPC[i]-1) {
               ultimo=true;
           }
           if (configuracionIndividual) {
               escritura.write(GenerarInstancia(sistemaOperativo[ac],ipRed4+1+j+3, ipRed3, ultimo, terminar[ac]));
           }
           else {
               escritura.write(GenerarInstancia(sistemaOperativo[i],ipRed4+1+j+3, ipRed3, ultimo, terminar[i]));
           }
           ac++;
       }
```

Fig. 12 Lógica interna generador JSON

Dentro del bucle principal se crea una subred y un grupo de seguridad por equipo. Además, se ejecuta un bucle anidado, tan grande como el número indicado en nPC, para generar las instancias. Es necesario hacer una distinción entre configuracionIndividual true o false, para saber qué sistema operativo y configuración de cierre poner. Dado que el último elemento del bucle va a ser una instancia, hemos de determinar si es la última instancia, puesto que esta no necesitará una coma.

Al final, se aumenta el valor de ipRed4 y, si fuera necesario, ipRed3, como se ve en la figura 13.

```
            ipRed4+=16;
            if(ipRed4 >=256) {
                ipRed4 = 0;
                ipRed3++;
            }
        }
```

Fig. 13. Actualización de valores generador JSON

Los métodos del bucle principal funcionan de la siguiente manera:

```
public String GenerarSubred(int ipRed4, int ipRed3) {
    String subred =
            "       \"subred" + subredId + "\": {\n"+
            "           \"Type\": \"AWS::EC2::Subnet\",\n"+
            "           \"Properties\": {\n"+
            "               \"AvailabilityZone\": \"us-east-1d\",\n"+
            "               \"CidrBlock\": \"172.31." + ipRed3 + "." + ipRed4 + "/28\",\n"+
            "               \"EnableDns64\": \"false\",\n"+
            "               \"Ipv6Native\": \"false\",\n"+
            "               \"MapPublicIpOnLaunch\": \"true\",\n"+
            "               \"VpcId\": \"vpc-072caab9869dc5266\"\n"+
            "           }\n"+
            "       },\n";
    subredId++;

    return subred;
}
```

Fig. 14. Generador de subred

Para el método de generación de subredes (figura 14) simplemente tenemos que especificar la ip de red que nos será pasada como argumentos. Nuestra máscara es de /28 lo que significa que tendremos 14 IPs disponibles, 11 si restamos las tres que AWS necesita internamente.

```
public String GenerarGrupoSeguridad(String ipLab) {
    String grupoSeguridad=
            "       \"GrupoSeguridad" + grupoSeguridadId + "\": {\n" +
            "           \"Type\": \"AWS::EC2::SecurityGroup\",\n"+
            "           \"Properties\": {\n"+
            "               \"GroupDescription\": \"Reglas para las instancias del grupo " + grupoSeguridadId + "\
            "               \"SecurityGroupIngress\": [{\n"+
            "                       \"CidrIp\": \"" + ipLab + "\",\n"+
            "                       \"IpProtocol\": \"TCP\",\n"+
            "                       \"FromPort\": \"3389\",\n"+
            "                       \"ToPort\": \"3389\"\n"+
            "                   },\n"+
            "                   {\n"+
            "                       \"CidrIp\": \"0.0.0.0/0\",\n"+
            "                       \"IpProtocol\": \"TCP\",\n"+
            "                       \"FromPort\": \"80\",\n"+
            "                       \"ToPort\": \"80\"\n"+
            "                   },\n"+
            "                   {\n"+
            "                       \"CidrIpv6\": \"::/0\",\n"+
            "                       \"IpProtocol\": \"TCP\",\n"+
            "                       \"FromPort\": \"80\",\n"+
            "                       \"ToPort\": \"80\"\n"+
            "                   },\n"+
```

```
                "                    {\n"+
                "                        \"CidrIp\": \"0.0.0.0/0\",\n"+
                "                        \"IpProtocol\": \"TCP\",\n"+
                "                        \"FromPort\": \"443\",\n"+
                "                        \"ToPort\": \"443\"\n"+
                "                    },\n"+
                "                    {\n"+
                "                        \"CidrIpv6\": \"::/0\",\n"+
                "                        \"IpProtocol\": \"TCP\",\n"+
                "                        \"FromPort\": \"443\",\n"+
                "                        \"ToPort\": \"443\"\n"+
                "                    },\n"+
                "                    {\n"+
                "                        \"CidrIp\": \"172.31."+ ipRed3 +"."+ ipRed4 +"/28\",\n"+
                "                        \"IpProtocol\": \"-1\"\n"+
                "                    }\n"+
                "                ],\n"+
                "                \"VpcId\":\"vpc-072caab9869dc5266\"\n"+
                "            }\n"+
                "        },\n";

        grupoSeguridadId++;
        return grupoSeguridad;
    }
```

Fig. 15. Generador grupo de seguridad

Para el método de generación de grupos de seguridad (figura 15) empezamos definiendo la regla que nos permite acceder a la instancia mediante RDP con la IP que se pasa por argumentos. Tras las cuatro instrucciones para permitir el tráfico HTTP y HTTPs, tanto ipv4 como ipv5, la regla final abre todos los puertos para PCs en una misma subred.

```
public String GenerarInstancia(String sistemaOperativo, int ipRed4, int ipRed3, boolean ultimo, boolean terminar) {
    String instancia;
    String ter = terminar? "terminate" : "stop";
    String fin = ultimo? "       }\n" : "        },\n";

    instancia =
            "       \"instancia" + instanciaId + "\": {\n"+
            "           \"Type\": \"AWS::EC2::Instance\",\n"+
            "           \"Properties\": {\n"+
            "               \"AvailabilityZone\": \"us-east-1d\",\n"+
            "               \"ImageId\": \"" + sistemaOperativo +"\",\n"+
            "               \"InstanceInitiatedShutdownBehavior\": \"" + ter + "\",\n" +
            "               \"InstanceType\": \"t2.micro\",\n"+
            "               \"KeyName\": \"clave" + (subredId-1) + "\",\n"+
            "               \"PrivateIpAddress\": \"172.31." + ipRed3 + "." + ipRed4 + "\",\n"+
            "               \"SecurityGroupIds\": [{ \"Fn::GetAtt\" : [\"GrupoSeguridad" + (grupoSeguridadId-1) +"\"
            "               \"SubnetId\": { \"Fn::GetAtt\" : [\"subred" + (subredId-1) + "\", \"SubnetId\"] }\n"+
            "           }\n"+
            fin;

    instanciaId++;
    return instancia;
```

Fig. 16. Generador instancias

Ya que el último elemento va a ser una instancia, tenemos que definir si esta es la última instancia y añadir o no una coma a la última línea antes de escribirla. También usamos un operador ternario para escribir correctamente la característica de detención o terminación del proceso. Además, se pasan por argumentos la IP del equipo en concreto y su sistema operativo. El método que rige esta creación está en la figura 16.

Creación de la interfaz

La interfaz consiste en dos pantallas diferentes, una sale después de la otra. Pero antes, se pregunta al usuario mediante un JOptionPane, presente en la figura 17, el número de equipos y se utiliza esta información para construir el primer panel.

```
public class GeneradorInterfaz {
    public static void main(String[] args) {
        int nEquipos = Integer.parseInt(JOptionPane.showInputDialog("Selecciona el numero de equipos"));
        MarcoEquipos m = new MarcoEquipos(nEquipos);
    }

}
```

Fig. 17. Selección número de equipos

El primer marco, MarcoEquipos (figura 18), crea una ventana genérica en el medio de la pantalla y crea un JScrollPane que rodea al panel donde se encuentra la información, PanelEquipos.

```
class MarcoEquipos extends JFrame{

    public MarcoEquipos(int nEquipos) {
        Dimension size = Toolkit.getDefaultToolkit().getScreenSize();
        int width = (int)size.getWidth();
        int height = (int)size.getHeight();

        setTitle("seleccionar numero de equipos");
        setBounds((int) (0.3*width), (int) (0.3*height), (int) (0.4*width), (int) (0.4*height));
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JScrollPane scrollPane = new JScrollPane(new PanelEquipos(nEquipos, this));
        scrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
        scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);
        add(scrollPane);

        validate();
    }
```

Fig. 18. Creación marco Swing

PanelEquipos está contenido en un GridLayout. Tan alto como se le haya indicado en el JOptionPane, con una fila extra para determinar la configuración inicial, y otra para el botón. También pasamos como argumento una referencia al mismo marco para que el panel pueda cerrarlo una vez se pulse el botón.

Necesitamos definir la constante de clase nPCs, una lista de JTextFields que guardarán los números de PCs elegidos para cada grupo. Con esto, se inicializa el GridLayout y, con un bucle for, se llenan las primeras celdas. Todo esto está presente en la figura 19.

```
private class PanelEquipos extends JPanel{
    ArrayList<JTextField> nPCs = new ArrayList<JTextField>();

    PanelEquipos(int nEquipos, JFrame main){

        setLayout(new GridLayout(nEquipos+2,2));

        for(int i = 0; i<nEquipos; i++) {
            add(new JLabel("PCs para el equipo " + (i+1)));
            JTextField texto = new JTextField(10);
            nPCs.add(texto);
            add(texto);
        }
```

Fig. 19. Creación primer panel (parte 1)

Las últimas dos líneas se añaden fuera, primero la JCheckBox y el botón. La JCheckBox se acompaña de un campo en blanco para que el botón quede debajo suya y no al lado. Al botón se le añade la funcionalidad de recopilar los datos de nPCs y la JCheckBox y utilizarlos para crear un nuevo marco, MarcoGeneral. Tras ello, destruye su propio marco. Como vemos en la figura 20.

```
JCheckBox configuracionIndividual = new JCheckBox("configuracion individual", false);
add(configuracionIndividual);
add(new JLabel(""));

JButton siguiente = new JButton("Siguiente");
siguiente.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e) {
        int[] numeroPCs = new int[nPCs.size()];
        int count = 0;
        for(JTextField j : nPCs) {
            numeroPCs[count] = Integer.parseInt(j.getText());
            count++;
        }

        MarcoGeneral m = new MarcoGeneral(nEquipos, numeroPCs, configuracionIndividual.isSelected());

        main.dispose();
    }
});
add(siguiente);
```

Fig. 20. Creación primer panel (parte 2)

MarcoGeneral está dispuesto de la misma manera que MarcoEquipos. Es su contenido, PanelGeneral, que podemos ver en las figuras 21-24, el que es diferente.

Como en PanelEquipos, definimos listas de variables por cada argumento que queremos recoger, y un botón para terminar el programa. Además, definiremos longitud como una constante de clase ya que puede variar dependiendo de si el usuario elige configuraciónInicial o no. Además, para el caso del sistema operativo, debemos crear un array con todas las opciones del usuario para poder automatizar la creación de los JComboBoxes, y crear un mapa que convierta esas opciones en los códigos que entiende AWS

```
private class PanelGeneral extends JPanel{

    ArrayList<JTextField> IPs = new ArrayList<JTextField>();
    ArrayList<JComboBox> image = new ArrayList<JComboBox>();
    ArrayList<JCheckBox> terminar = new ArrayList<JCheckBox>();
    JButton fin = new JButton("finalizar");
    int longitud = -1;

    String[] sistemasDisponibles = {"Amazon Linux", "Ubuntu", "Windows", "Red Hat", "SUSE Linux", "Debian"};
    Map<String, String> decod = new HashMap<String, String>();

    public PanelGeneral(int nEquipos, int[] nPCs, boolean configuracionIndividual, JFrame main) {

        decod.put("Amazon Linux", "ami-0cff7528ff583bf9a");
        decod.put("Ubuntu", "ami-052efd3df9dad4825");
        decod.put("Windows", "ami-05912b6333beaa478");
        decod.put("Red Hat", "ami-06640050dc3f556bb");
        decod.put("SUSE Linux", "ami-08895422b5f3aa64a");
        decod.put("Debian", "ami-09a41e26df464c548");
```

Fig. 21. Creación segundo panel (parte 1)

En este caso va a haber una gran diferencia dependiendo de si el usuario selecciona configuraciónInicial o no. En caso de que lo haga, necesitaremos calcular el número total de instancias que se van a crear. En caso contrario, el número será igual a nEquipos.

```
if(configuracionIndividual) {
    longitud = 0;
    for(int i : nPCs) {
        longitud += i;
    }
}
else {
    longitud = nEquipos;
}

setLayout(new GridLayout(longitud+2,3));
```

Fig. 22. Creación segundo panel (parte 2)

El layout se rellena similarmente al panel pasado. Se crea la interfaz con un bucle for y, dentro del ActionListener, se transforman las listas del principio en Arrays de String y se pasan como parámetros a la clase definida en la pasada sección, dónde se generará el código JSON. Al final, se cierra el marco.

```
add(new JLabel(""));
add(new JLabel("¿Terminar?"));
add(new JLabel("Sistema operativo"));
add(new JLabel("IP de conexion"));

for(int i = 0; i<longitud; i++) {
    add(new JLabel("lab " + (i+1)));

    JCheckBox t = new JCheckBox();
    terminar.add(t);
    add(t);

    JComboBox os = new JComboBox();
    for(String s : sistemasDisponibles) {
        os.addItem(s);
    }
    os.setSelectedIndex(-1);
    image.add(os);
    add(os);

    JTextField texto = new JTextField(10);
    IPs.add(texto);
    add(texto);
}
```

Fig. 23. Creación segundo panel (parte 3)

```
add(new JLabel(""));
add(new JLabel(""));
fin.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e) {

        String[] ipLab = new String[longitud];
        String[] os = new String[longitud];
        boolean[] ter = new boolean[longitud];

        int i = 0;
        for(JComboBox j : image) {
            os[i] = decod.get((String) j.getSelectedItem());
            i++;
        }

        i=0;
        for(JCheckBox j : terminar) {
            ter[i] = j.isSelected();
            i++;
        }

        i=0;
        for(JTextField j : IPs) {
            ipLab[i] = (j.getText() + "/32");
            i++;
        }

        GenerarJson g = new GenerarJson(nEquipos, nPCs, ipLab, configuracionIndividual, os, ter);
        main.dispose();
    }
});
add(fin);
```

Fig. 24. Creación segundo panel (parte 4)

3.3. Despliegue en AWS

Para probar el programa vamos a hacer un despliegue con dos equipos y dos instancias por equipo. El sistema operativo es Windows, ya que AWS ofrece un escritorio remoto y es más fácil de visualizar. El primero de los equipos mostrará un comportamiento de cierre de terminación y, el segundo, de detención. La red se debería quedar como vemos en la figura 25

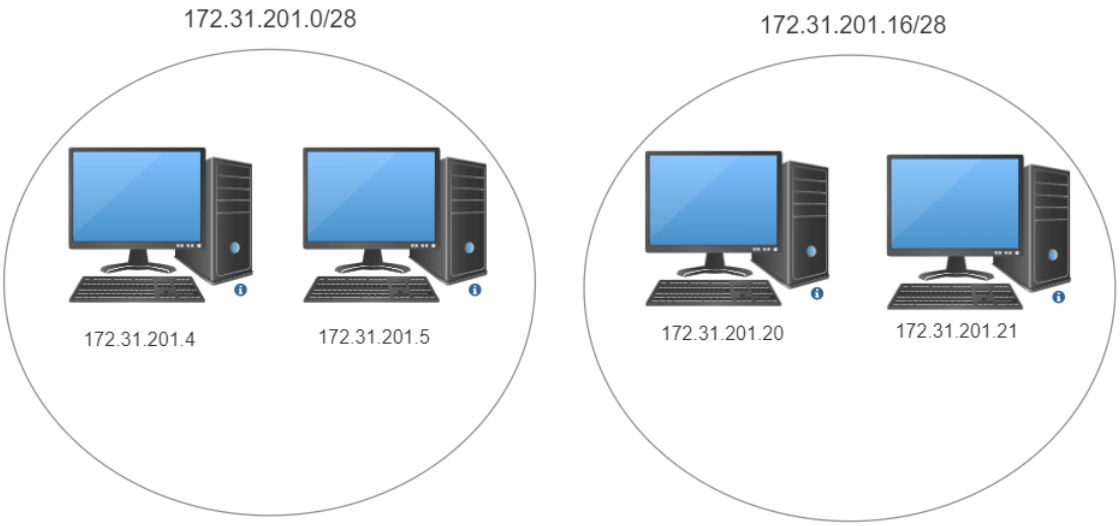


Fig. 25. Esquema de la prueba

Para generar el JSON, debemos operar la interfaz como se muestra en las figuras 26-28

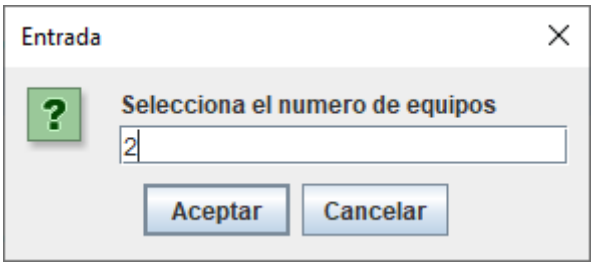


Fig. 26. Seleccionando número de equipos

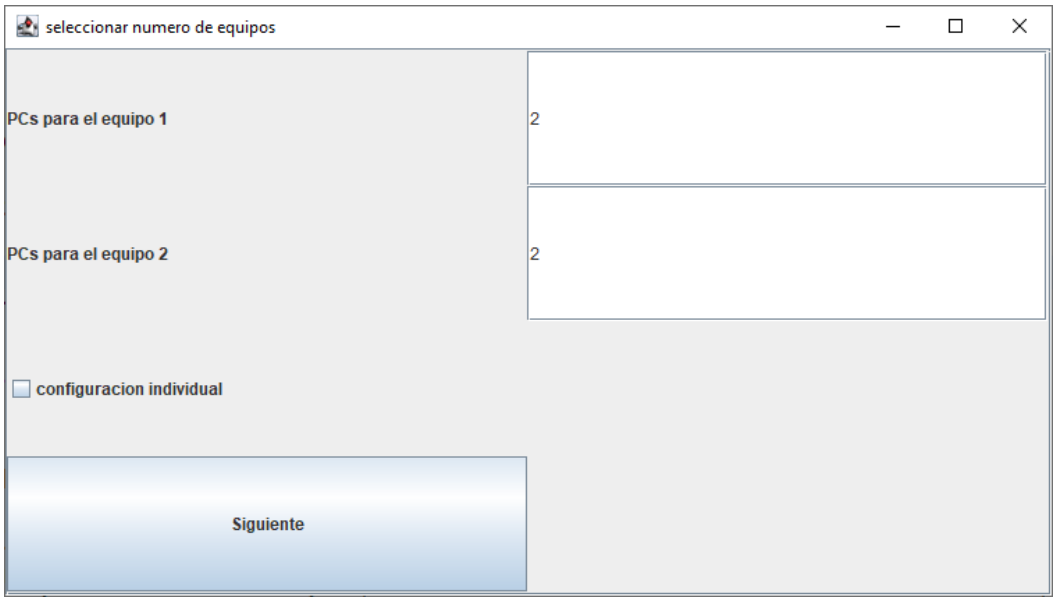


Fig. 27. Seleccionando PCs por grupo

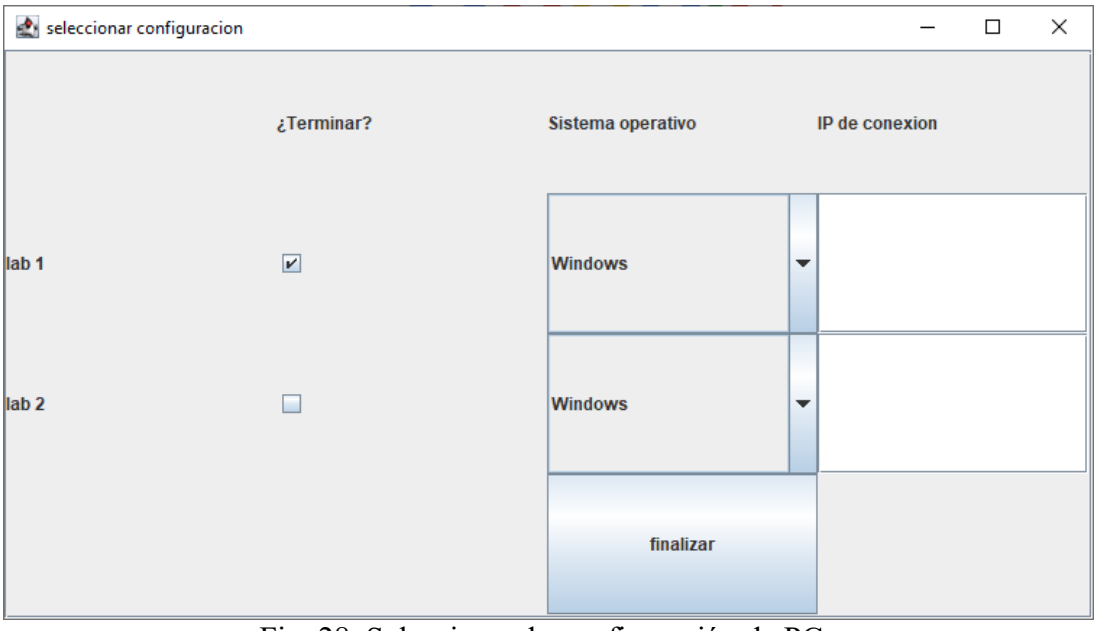


Fig. 28. Seleccionando configuración de PCs

El archivo resultante se guarda en la misma carpeta del programa bajo el nombre aws.json. Una vez con el archivo, simplemente nos tenemos que mover por los menús de CloudFormation hasta que nos lo pidan y llegaremos al log del proceso de creación (figura 29), donde tendremos que esperar a que se creen los recursos.

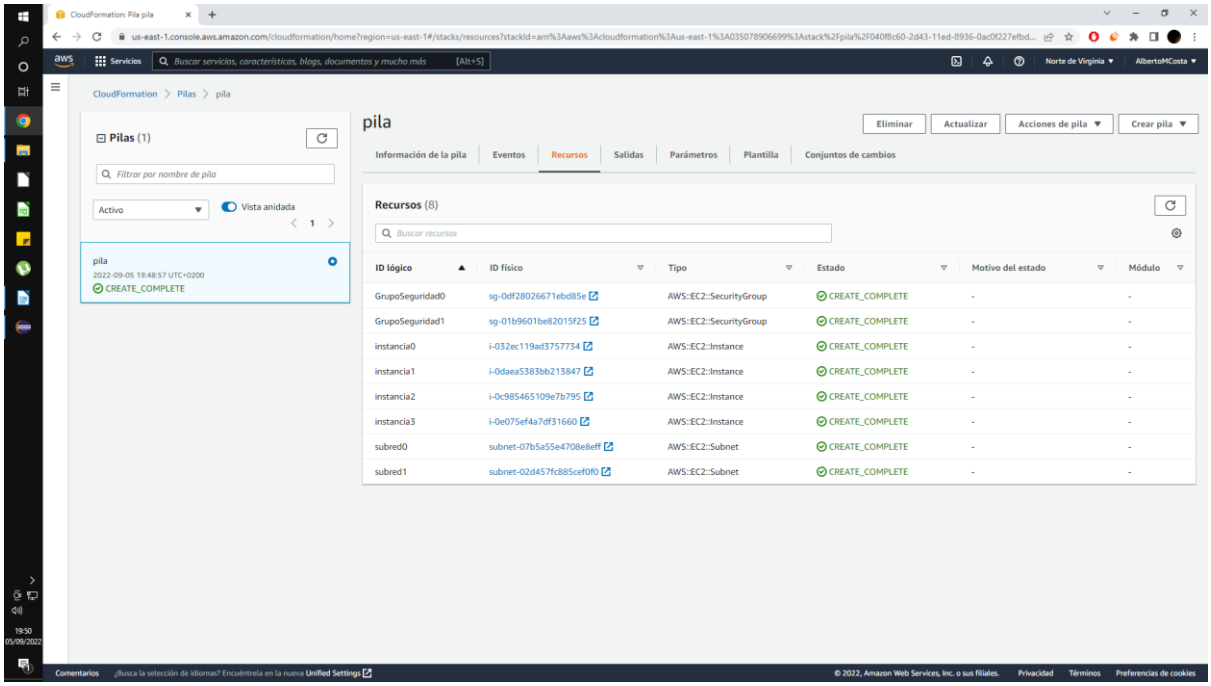


Fig. 29. Página de CloudFormation

Una vez finalizado podemos ir a la sección de instancias para ver que se han creado correctamente (figura 30). Una vez aquí, habrá que esperar otro tiempo hasta que la comprobación finalice.

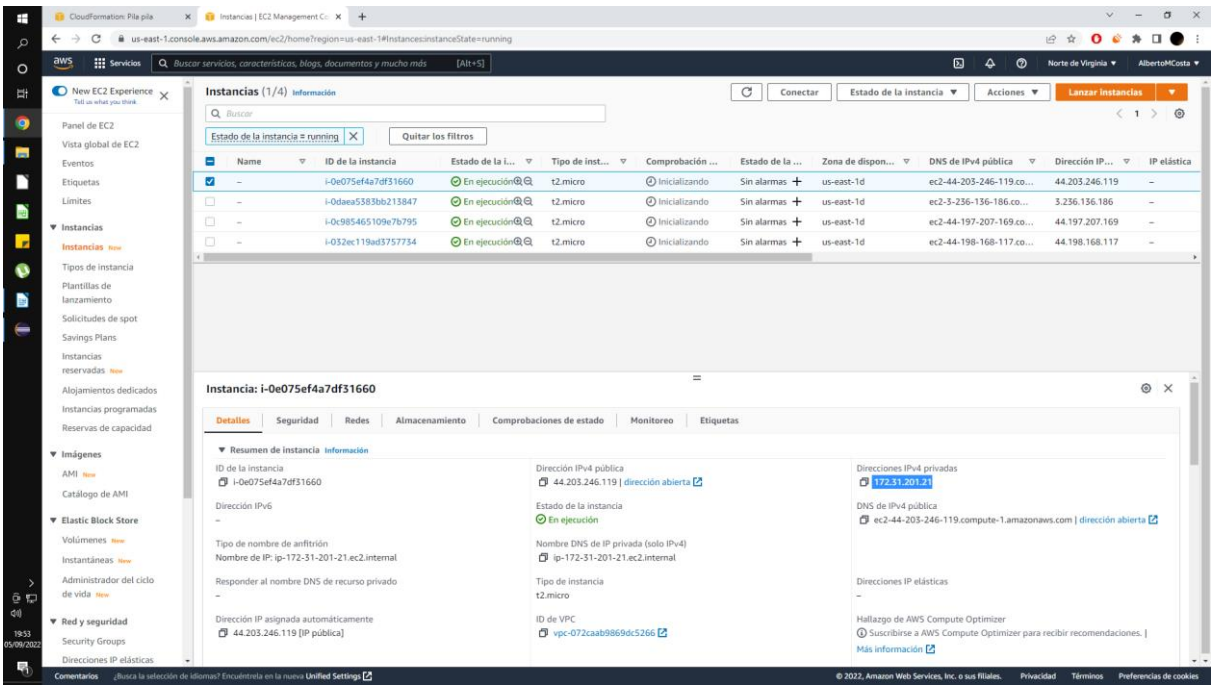


Fig. 30. Página de instancias

Para conectarnos a una instancia, en el caso de windows, es necesario primero obtener la contraseña de acceso, que está cifrada con nuestro par de claves. Para ello, pulsamos en acciones → Seguridad → Obtener contraseña de Windows. Una vez hecho esto, nos encontraremos con un menú (figura 31) que requiere de nuestra clave privada. Dicha clave estará en la siguiente pantalla y podremos copiarla.

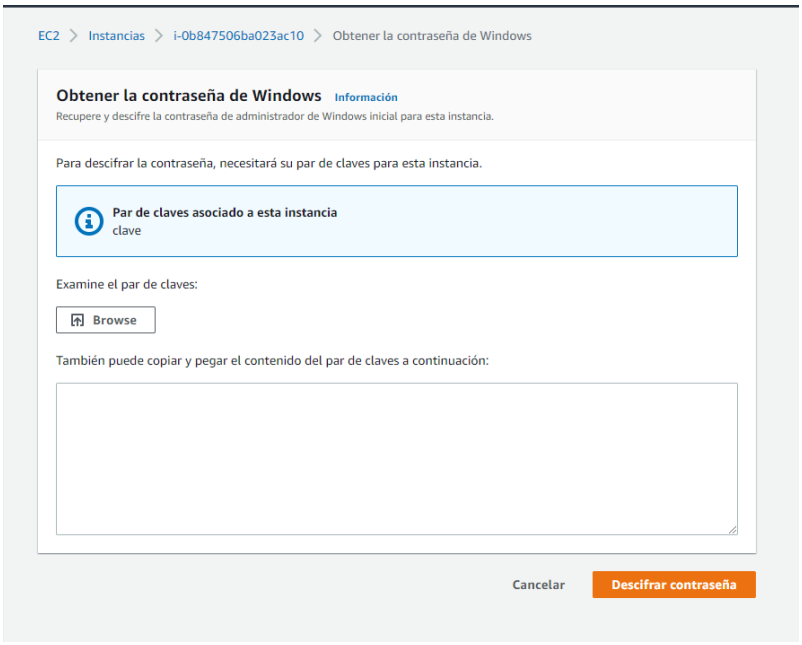


Fig. 31. Obtener Contraseña de Windows

Tras ello, pulsamos en Conectar y navegamos a la pestaña Cliente RDP (figura 32). Una vez ahí podemos descargar un escritorio remoto de nuestra instancia. Una vez descargado y ejecutado, nos pedirá nuestra contraseña de windows. Si la incluimos nos dejará acceder como se puede observar en la figura 33..

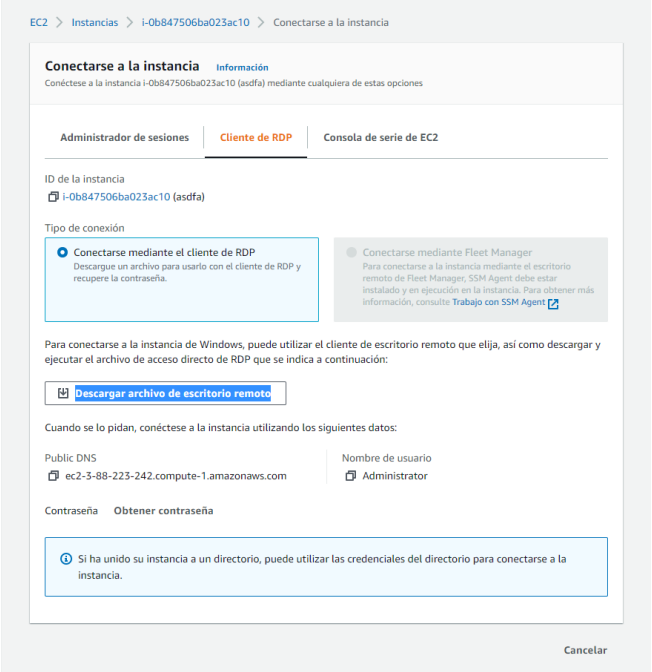


Fig. 32 Menú conexión

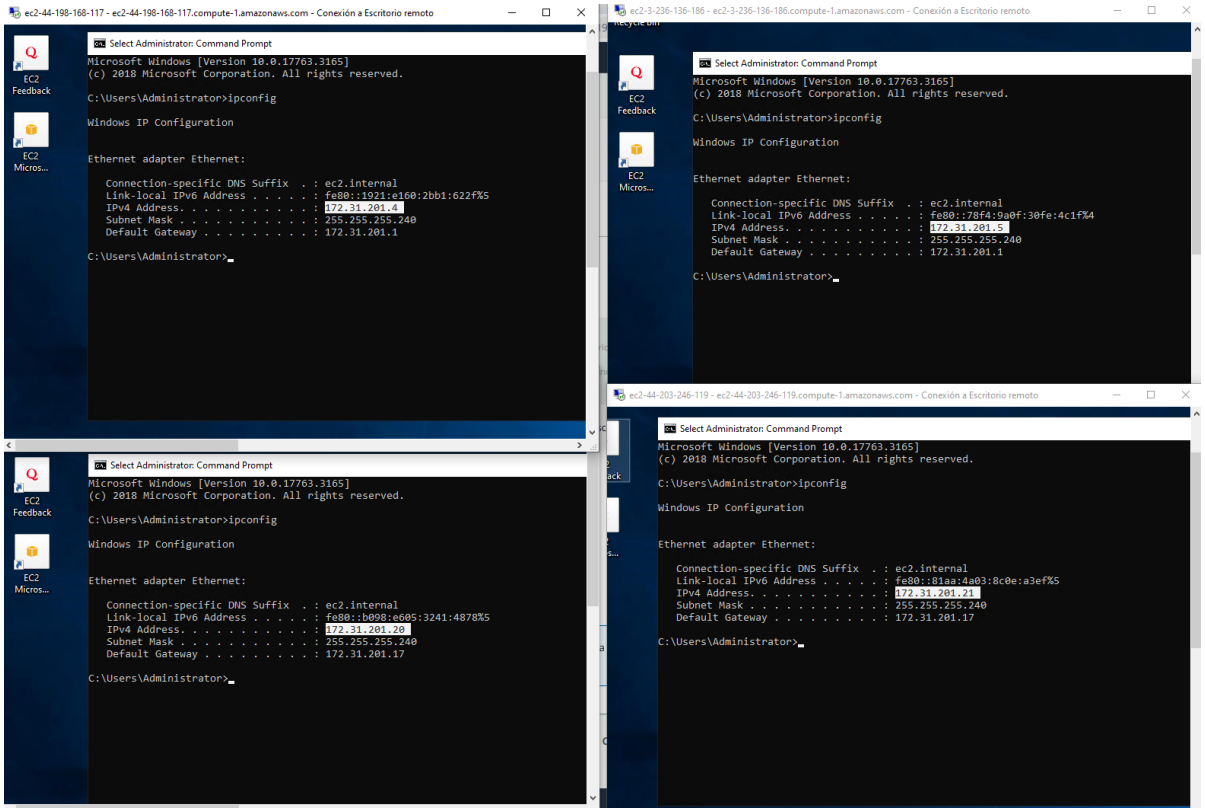


Fig. 33. Conexión con virtualizaciones exitosa

Para comprobar conexión, es necesario que el firewall de la simulación no se ponga en medio. Para windows, la forma de permitir el flujo de tráfico icmp es con el comando “netsh advfirewall firewall add rule name="ICMP Allow incoming V4 echo request" protocol=icmpv4:8,any dir=in action=allow”. Una vez introducido podemos comprobar la conexión entre instancias de una misma subred (figura 34).

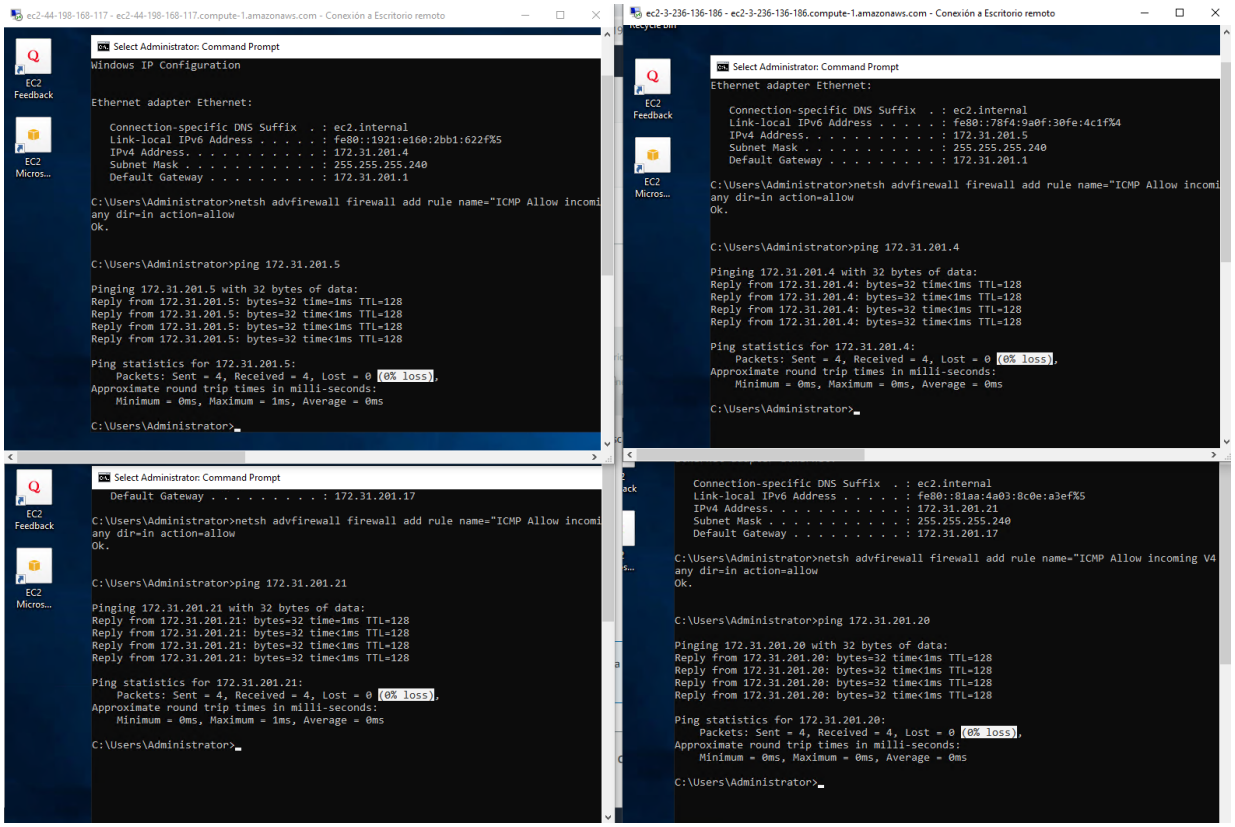


Fig. 34. Conexión entre instancias exitosa

La figura 35 muestra que la comunicación entre instancias de distinta subred es imposible.

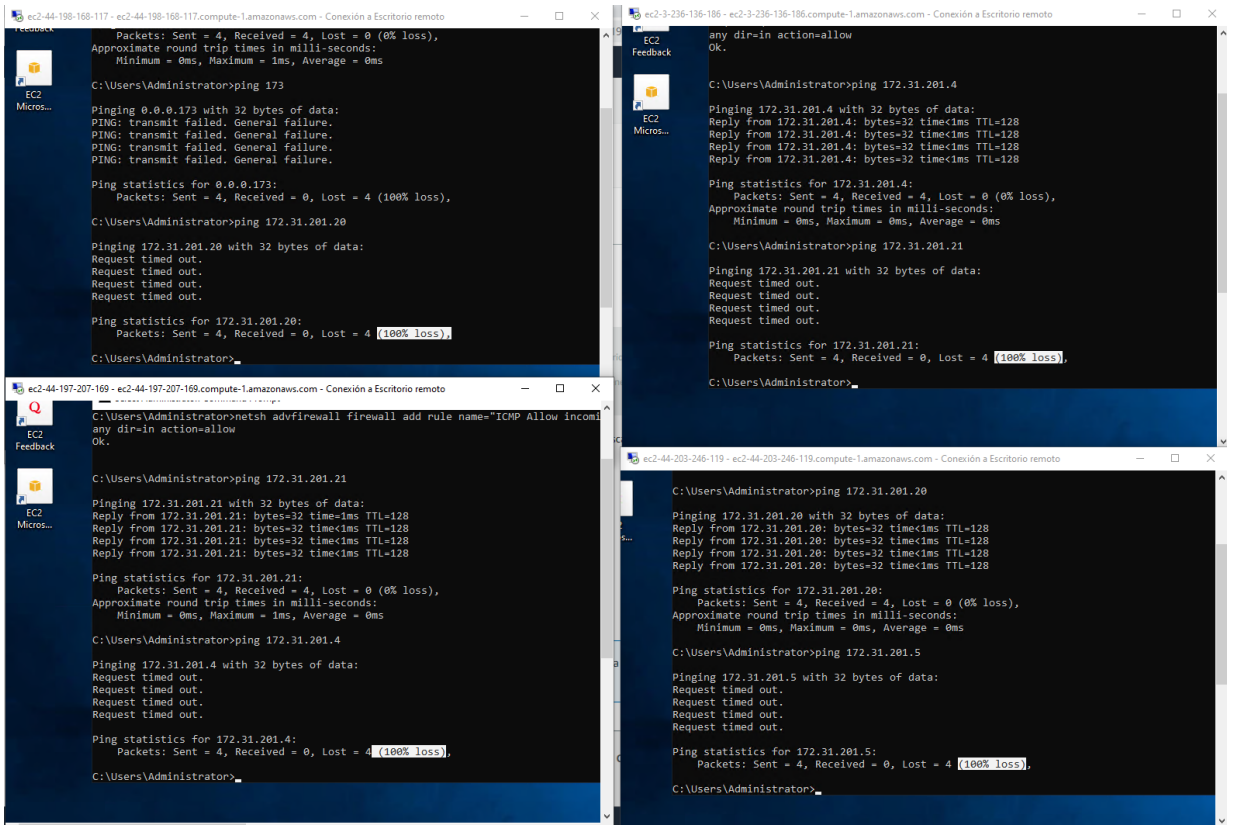


Fig. 35. Conexión entre instancias fallida

Por último, vamos a crear un archivo de texto “test.txt” para comprobar la persistencia de memoria de las dos instancias de abajo, las cuales están configuradas para detenerse y no terminarse. Creamos el archivo en todas y las cerramos.

Tras un tiempo, las dos primeras instancias han desaparecido de nuestra lista de instancias registradas, pero las segundas siguen preparadas para re-activarse (figura 36).

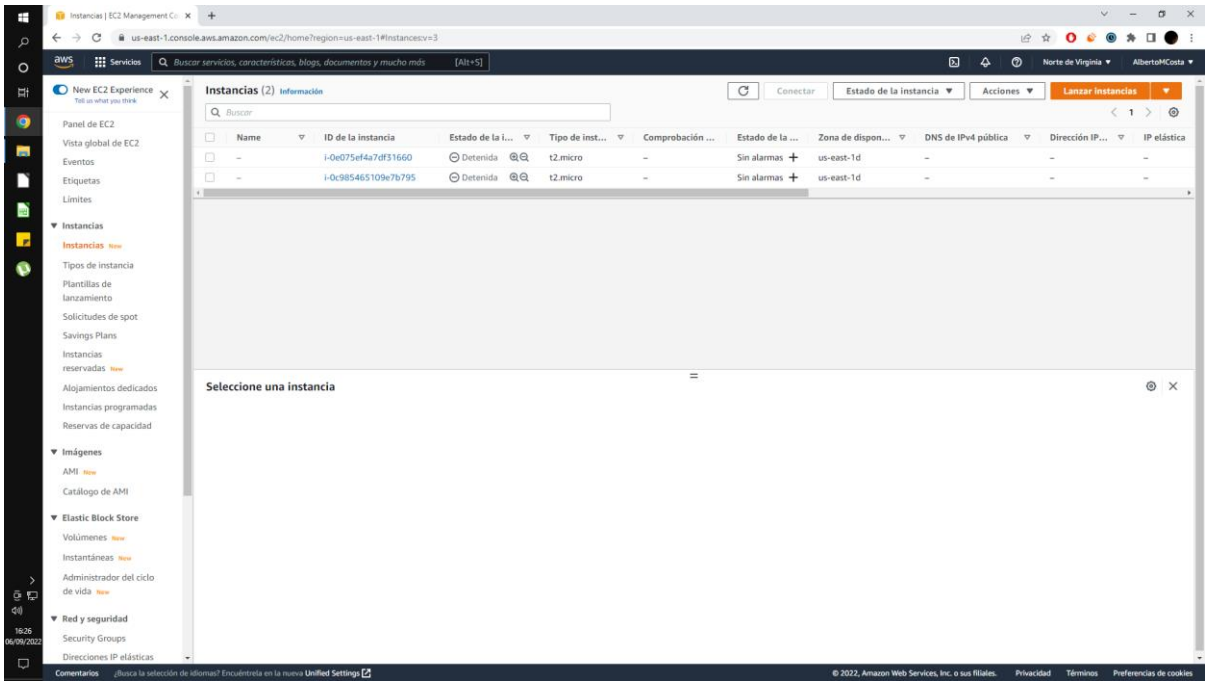


Fig. 36. Página de instancias un día después

Volvemos a conectarnos (tras descargar de nuevo el monitor remoto, ya que el DNS ha cambiado) y vemos que los archivos siguen ahí (figura 37).

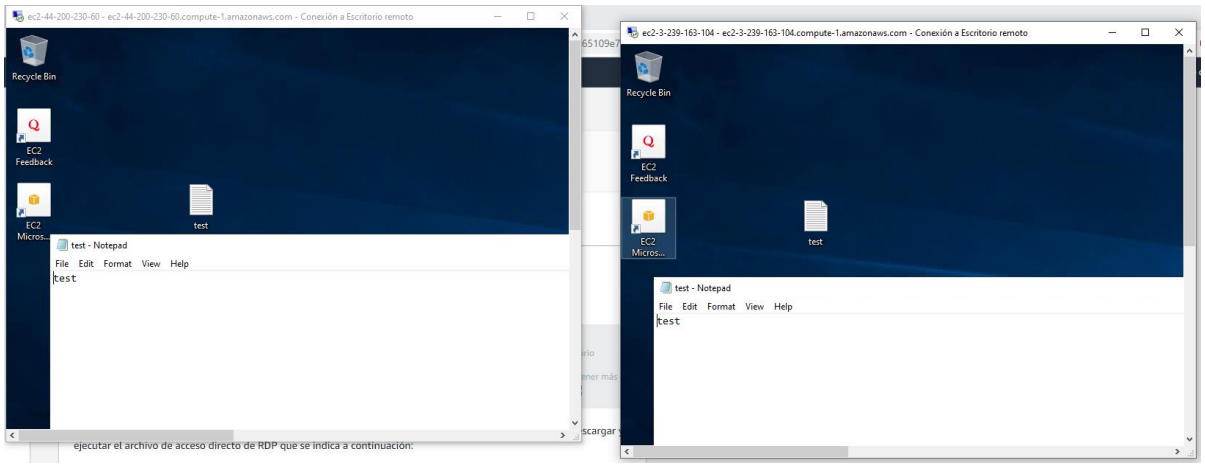


Fig. 37. Consistencia de los archivos

Capítulo 4. Análisis del coste.

Los costes de nuestra operación en Amazon AWS se dividirán en creación de la red con CloudFormation, generación y uso de las claves de seguridad, coste de procesado de las instancias y coste de almacenamiento. La estimación estará dividida por sistema operativo, ya que el precio del procesado y la capacidad de almacenamiento están dictadas por este. También habrá una división entre el comportamiento de cierre. Con Terminación y Detención siendo drásticamente distintos en su aproximación a los recursos.

4.1. Estimación del coste.

Cuando una instancia se cierra, hay dos comportamientos posibles. O bien esa instancia se detiene. Se detiene el procesado, almacenando el estado y la información de la máquina en memoria, pero manteniendo la máquina en la nube. O bien se termina. Se detiene tanto el procesado como el almacenamiento, pero la instancia se tiene que crear de nuevo si se quiere volver a usar.

Ambos casos suponen problemas. Tanto el almacenamiento con EBS como la creación de instancias con CloudFormation, suponen un precio añadido a nuestro proyecto. Esta estimación, en la que asumimos 20 grupos de estudiantes con 2 instancias cada uno durante el curso de 10 semanas con una clase semanal de 2 horas, está pensada para resolver este problema.

CloudFormation

CloudFormation cobra tanto por operaciones como por el tiempo en ejecutarlas. Las operaciones son 0,0009US$/operación y el tiempo 0,00008US$/s para cada operación. En nuestro caso es necesario un grupo de seguridad y una subred para cada laboratorio y tantas instancias como hagan falta.

Así, el precio por operación es de 0,0009*80 = 0,072US$. Por otra parte, el tiempo necesario para crear una subred o un grupo de seguridad es de aproximadamente 30 segundo, mientras que las instancias pueden llegar a durar hasta un minuto. El coste de tiempo sería 0,00008*(40*30+40*60) = 0,288US$

Tras el curso de 10 semanas, los laboratorios que se detienen tendrán que ejecutar el script una sola vez, por lo que pagarán 0,36US$. Los laboratorios que terminen tendrán que realizar la tarea cada semana, gastando 3,60US$

KeyPair

La creación de claves viene con una tarifa extra de 0,03US$ por cada 10000 operaciones de clave. Estas operaciones incluyen la creación y el uso de dichas claves. Se crearán tantas claves como laboratorios y se cifrarán todas las instancias. Una vez hecho eso, se utilizarán las claves para acceder a los laboratorios (supongamos una vez por semana) Nos da un precio total de:

0,03/10000 * (20+40) = 0,00018US$. Invariable a la configuración.

Instancias.

Utilizamos instancias EC2 t2.micro en la región Norte de Virginia, que tienen diferentes costes en función del sistema operativo, como nos muestra la tabla 1.

Tabla 1. Precio por hora del sistema operativo

OS	Amazon Linux	Ubuntu	Windows	Red Hat	SUSE Linux	Debian
Coste (US$/hr)	0,0166	0,0166	0,0162	0,0716	0,0166	0,0166

En este caso hay tres opciones: terminar las instancias una vez se finaliza la sesión, detenerlas o dejarlas activas. Si se terminan sólo se utilizarán durante las horas de prácticas, pero los habría que reiniciar cloudformation cada vez. Deteniendo las instancias se nos cobra el mismo tiempo en cuanto a la instancia, pero el almacenamiento nos sigue costando. Dejarlas activas hace que nos cobren tanto la instancia como el almacenamiento durante toda la semana. Aislando las instancias el coste de 10 semanas y 40 instancias se muestra en la tabla 2.

Tabla 2. Precio total del sistema operativo

OS	Amazon linux	Ubuntu	Windows	Red Hat	SUSE Linux	Debian
Terminar	13,28	13,28	12,96	57,28	13,28	13,28
Detener	13,28	13,28	12,96	57,28	13,28	13,28
Continuar	1115,60	1115,60	1088,00	4812,00	1115,60	1115,60

Almacenamiento.

El almacenamiento se paga por separado con las tarifas de Amazon EBS. El almacenamiento es independiente y esté una instancia parada o en marcha, se aplican las tarifas. El precio es de 0,10 US$/Gb al mes. El almacenamiento depende del sistema operativo, en nuestro caso, su coste se muestra en la tabla 3

Tabla 3. Precio del almacenamiento

OS	Amazon Linux	Ubuntu	Windows	Red Hat	SUSE Linux	Debian
GiB	8	8	30	10	10	8
Precio/hr	0,0011	0,0011	0,0042	0,0013	0,0013	0,0011
terminar	0,88	0,88	3,36	1,04	1,04	0,88
detener	7,392	7,392	282,24	87,36	87,36	7,392

Conclusión

Los precios de arriba están dados sin IVA incluido. Juntándolo todo y añadiendo impuestos tenemos, dependiendo del sistema operativo, se muestran en la tabla 4

Tabla 4. Precio total

OS	Amazon Linux	Ubuntu	Windows	Red Hat	SUSE Linux	Debian
Precio terminar	21,49	21,49	24,10	74,92	21,68	21,49
Precio detener	25,36	25,36	357,54	175,35	122,11	25,36
Precio continuar	1358,83	1358,83	1658,00	5929,00	1455,58	1358,83

La opción de terminar va a ser siempre una mejor opción a no ser que fuera necesaria la consistencia de los datos, o dejar acceso a las instancias fuera de las horas de clase.

Capítulo 5: Conclusión.

En este TFE, he estudiado las características de la programación en la nube y he contrastado las apuestas más populares para encontrar la más óptima. Con ello, he realizado un programa de Java que genera un archivo JSON a partir de los requerimientos expresados por el usuario a través de una interfaz Swing. Dicho JSON se ha utilizado luego para generar una red virtual en la nube de AWS. Tras haber probado la funcionalidad, he calculado el coste aproximado de llevar la operación al ámbito educacional estimando el coste de una clase sobre un trimestre de 10 semanas.

Para este trabajo he necesitado familiarizarme con la tecnología de AWS. Tanto su estructura, como sus posibilidades y opciones. También he necesitado ganar fluidez con el lenguaje de marcas JSON y he ampliado mis conocimientos en Java, estudiando la biblioteca Swing para la creación de interfaces.

Ha sido imposible probar la estimación de 40 instancias desde la capa gratuita de AWS, pero una simulación de dos grupos con dos instancias cada uno ha sido un éxito. Las instancias son accesibles con el nombre DNS y la contraseña, generados automáticamente.

Esta ha sido una base para crear la infraestructura de un laboratorio de prácticas en la nube. En el futuro, se puede hacer disposición del amplio repertorio de recursos e-learning que AWS contiene para mejorar y personalizar la experiencia lectiva aún más.

También se podría ampliar el trabajo haciendo uso de algunas de las otras funciones de AWS. AWS Network firewall, por ejemplo, se puede utilizar para simplificar el proceso de conexión entre equipos de una misma subred, como alternativa a la desactivación manual en el cmd de la instancia. De la misma manera, hay varios campos (como el nombre DNS, la ip pública...) que se han dejado en blanco para que AWS los automatice. Otro proyecto podría dar al usuario más control sobre estos aspectos que yo he considerado secundarios.

Referencias

[1] Techtarget (en línea) Consultado el 25/07/22
https://www.techtarget.com/searchcloudcomputing/definition/cloud-computing

[2] Techtarget (en línea) Consultado el 25/07/22
https://www.techtarget.com/searchitchannel/definition/user-self-provisioning

[3] Euroinnova (en línea) Consultado el 25/07/22 https://www.euroinnova.edu.es/blog/tipos-de-e-learning

[4] Gema Lopez Guerrero. Introducción al e-learning. Consultado el 25/07/22
https://archivos.csif.es/archivos/andalucia/ensenanza/revistas/csicsif/revista/pdf/Numero_27/GEMA%20LOPEZ%20GUERRERO_1.pdf

[5] Julián Andrés Toro Torres. Guía básica para entender la estructura y el funcionamiento de la computación en la nube. Consultado el 04/09/22
https://repositorio.utp.edu.co/items/e8807b24-1bc4-4f74-a532-6c70076e2944/full

[6] V. Rajaraman. Cloud Computing. Consultado el 29/08/22
https://www.ias.ac.in/article/fulltext/reso/019/03/0242-0258

[7] Peter Mell, Tim Grance. The NIST definition of Cloud Computing. Consultado el 29/08/22 https://www.nist.gov/system/files/documents/itl/cloud/cloud-def-v15.pdf

[8] George Coulouris, Jean Dollimore, Tim Kindberg. Sistemas distribuidos, conceptos y diseño. Cuarta edición.

[9] Eugenio Villar, Julio Gomez. Introducción a la virtualización. Consultado el 05/09/22
http://biblioteca.udgvirtual.udg.mx/jspui/handle/123456789/2273

[10] SATW. Cloud Computing. Consultado el 29/08/22
https://widmer.ch/fileadmin/templates/publikationen/20121106_SATW_WhitePaper_CloudComputing_EN.pdf

[11] IBM (en linea) Consultado el 01/08/22 https://www.ibm.com/es-es

[12] Google (en línea) Consultado el 01/08/22 https://cloud.google.com/?hl=es

[13] Amazon (en línea) Consultado el 01/08/22 https://aws.amazon.com/es/?nc2=h_lg

[14] Microsoft (en línea) Consultado el 01/08/22 https://azure.microsoft.com/es-es/

[15] Documentación AWS (en línea) Consultado el 05/08/22
https://docs.aws.amazon.com/es_es/vpc/latest/userguide/what-is-amazon-vpc.html

[16] Documentación AWS (en línea) Consultado el 05/08/22
https://docs.aws.amazon.com/es_es/vpc/latest/userguide/VPC_SecurityGroups.html

[17] Documentación AWS (en línea) Consultado el 05/08/22
https://docs.aws.amazon.com/es_es/AWSEC2/latest/WindowsGuide/concepts.html

[18] Documentación AWS (en línea) Consultado el 05/08/22
https://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/ec2-key-pairs.html

[19] Documentación AWS (en línea) Consultado el 05/08/22
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html