Openstackdroid

Consola de administración de Openstack para tabletas con sistema operativo Android

Ricardo Carrillo Cruz
Ingeniería Técnica en Informática de Sistemas

Consultor: Roberto Ramírez Vique

Marzo-Junio de 2013

A Luisa

A mis padres

A mis perros, Duque y Tommy

Índice de contenidos

Plan de trabajo

Objetivos del proyecto

El objetivo de este proyecto es coger experiencia en dos áreas que están en pleno auge, y las cuales me gustaría introducirme laboralmente a medio plazo: Computación en la nube (cloud computing) y desarrollo de aplicaciones móviles.

Para ello, pretendo programar una consola de administración del software Openstack para tabletas con sistema operativo Android.

Openstack es un proyecto de software libre de computación en la nube, en particular IAAS (Infrastructure As A Service). Este tipo de computación en la nube consiste en prestar servicios de computación, almacenamiento y comunicaciones bajo demanda, adaptable a las necesidades de cada momento y facturando solo el uso que se le da a los recursos.
El proyecto Openstack está gobernado por una fundación de la cuál son miembros muchas empresas líderes del sector, como HP, Dell, IBM, Rackspace, RedHat, Novell, Intel, AMD, etc.

Android es un sistema operativo para dispositivos móviles, como tabletas y smartphones, desarrollado y mantenido por Google.
Es también un software abierto, pudiendo ser utilizado libre de costes y sin licencias de uso.

Dado mi interés en las dos tecnologías expuestas anteriormente (Openstack y Android), me puse a investigar si existía algún tipo de cliente de Openstack para el sistema operativo Android.
La respuesta fue negativa.
El proyecto de Openstack se hospeda en la plataforma de desarrollo colaborativo Launchpad. Como se puede ver en este enlace no hay ninguna aplicación oficial de Android para Openstack hasta el momento:

https://launchpad.net/openstack-clients
https://launchpad.net/openstack-android

Por tanto, me pareció ideal hacer mi trabajo de fin de carrera sobre estas dos tecnologías, para poderme especializar en ellas a medio-largo plazo.

Tecnología y herramientas escogidas

- Sistema operativo Android, versión Ice Cream Sandwich. En este momento es la versión más extendida en los dispositivos, y facilita la programación de aplicaciones en tabletas y smartphones, en gran parte a la inclusión de Fragments (pequeñas porciones pertenecientes a una Activity, que permiten más flexibilidad a la hora de crear interfaces de usuario)
- Openstack, versión Folsom (a día de publicación de este documento es la última versión estable)
- Lenguaje de programación Java
- REST API de Openstack, para interactuar desde la aplicación Android con el cloud controller de Openstack
- Entorno de desarrollo Eclipse
- Librería GSON para serialización y deserialización de JSON
- GIT para el control de versiones de software y el correspondiente plugin EGit para Eclipse
- ProjectLibre para diagramas de Gantt
- Libreoffice para creación de documentos, presentación y diagramas
- Tableta BQ Edison para pruebas reales en dispositivo hardware

Listado de funcionalidades

1. Autenticación de cualquier usuario, sea Administrator o no, por HTTP
2. Autorización y visualización de los recursos de Openstack acorde a los permisos que tenga el usuario autenticado. La aplicación sólo debe visualizar los objetos para los que el usuario tiene permisos
3. Selección de cualquier proyecto (nomenclatura usada en Openstack para “tenant”) que el usuario tenga acceso. La autorización y visualización posterior de los objetos del proyecto seleccionado serán acordes a los permisos que tenga el usuario autenticado en ese momento
4. Listado de todas las instancias (máquinas virtuales o unidades de computación)
5. Información detallada de cualquier instancia (identificador, nombre, estado, interfaces de red, volúmenes adjuntados, etc)
6. Listado de todos los volúmenes (unidades de almacenamiento)
7. Información detallada de cualquier volumen (identificador, nombre, estado, tamaño, etc)
8. Listado de sabores (configuraciones predefinidas para crear instancias (número de VCPUs, RAM, almacenamiento, etc)
9. Información detallada de cualquier sabor
10. Listado de imágenes (imágenes virtuales de disco, con un sistema operativo, kernel y ramdisk específicos)
11. Información detallada de cualquier imagen (identificador, nombre, estado, tipo de contenedor, etc)
12. En caso de que el usuario sea Administrador, la aplicación también debe poder listar y obtener información detallada sobre los usuarios del sistema y proyectos (o *tenants)* de Openstack
13. La aplicación debe ser fácil de usar, y seguir los patrones de diseño recomendados en la documentación oficial de Android

Riesgos del proyecto

Los mayores riesgos de este proyecto es la falta de experiencia en desarrollo de aplicaciones y desconocimiento de las APIs de Android disponibles.
No tengo experiencia programando aplicaciones de Android ni usando algunas herramientas esenciales para desarrolladores, como por ejemplo el software de control de versiones Git.

En el caso de Openstack, si bien no he utilizado las APIs disponibles para crear aplicaciones clientes, tengo un poco de experiencia en cómo funciona internamente y los distintos componentes, ya que he contribuido pequeños parches en el pasado para arreglar bugs.

Tengo experiencia programando en algunos proyectos de software libre y como hobby, por tanto la tarea de programación no debe ser un problema que ponga en riesgo el proyecto.

En conclusión, es necesario un esfuerzo adicional para ir aprendiendo sobre la marcha, con la ayuda de la documentación oficial de android, video-tutoriales, libros de consulta y distintos foros de Internet.

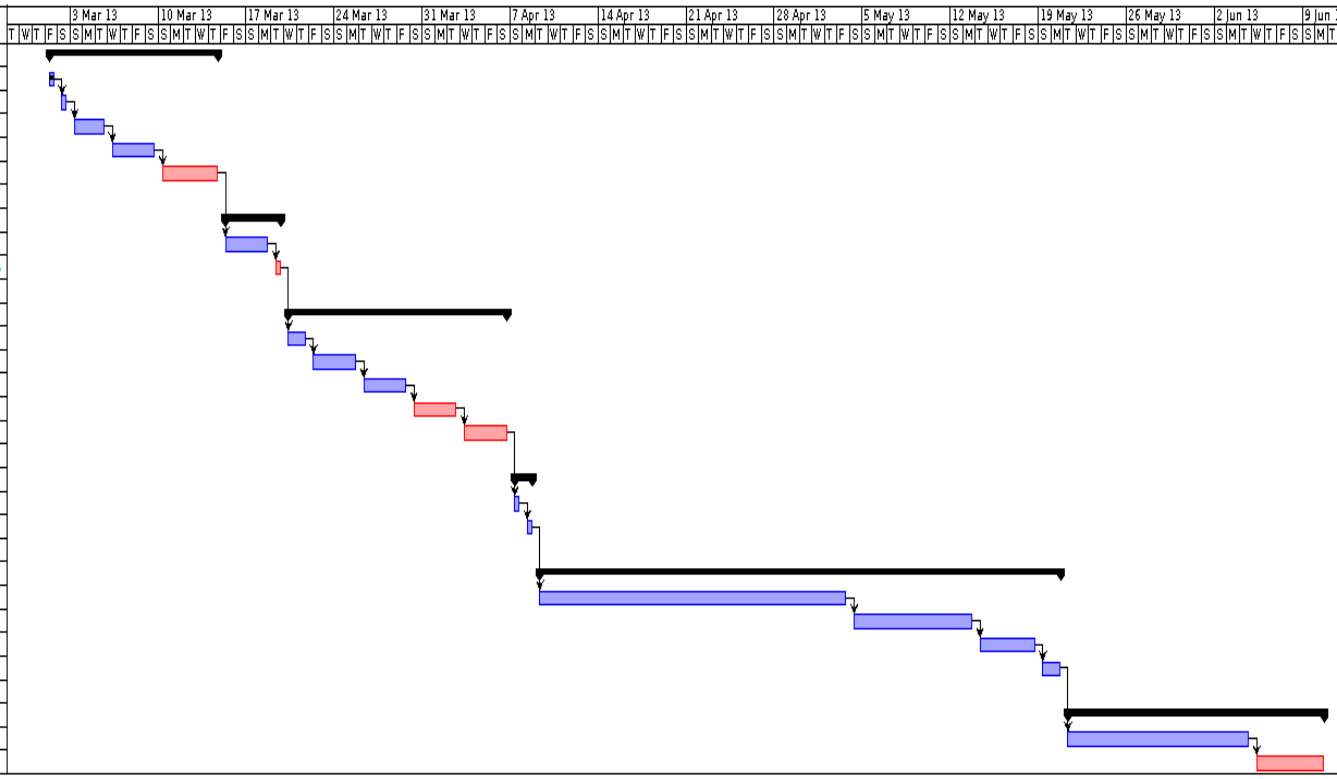
Todas las fuentes utilizadas las listaré debidamente en la memoria final.

Calendario del proyecto

La siguiente tabla muestra el calendario del proyecto, con las diferentes tareas y las fechas estimadas de realización del proyecto:

Análisis previo y planificación	14 días	1/03/2013	14/03/2013
Selección del proyecto	1 día	1/03/2013	1/03/2013
Propuesta y justificación del proyecto	1 día	2/03/2013	2/03/2013
Definición de funcionalidades del proyecto	3 días	3/03/2013	5/03/2013
Planificación del proyecto	4 días	6/03/2013	9/03/2013
Creación del documento	5 días	10/03/2013	14/03/2013
Entrega PEC1 (14/03/2013)	Hito		
Análisis funcional	5 días		
Especificación de requisitos funcionales	4 días		
Especificación de requisitos no funcionales	1 día		
Diseño	18 días		
Diseño conceptual	2 días		
Diagramas de casos de uso	4 días		
Diagramas de clases	4 días		
Diagramas de secuencias	4 días		
Diseño de interfaz de usuario	4 días		
Prototipo	2 días		
Diseño y realización del prototipo	1 día		
Preparación del documento	1 día		
Entrega PEC2 (8/04/2013)	Hito	8/04/2013	
Implementación y pruebas	42 días		
Desarrollo de programa principal	25 días		
Desarrollo de interfaz	10 días		
Pruebas	5 días		
Creación de instrucciones	2 días		
Entrega PEC3 (20/05/2013)	Hito	20/05/2013	
Entrega de proyecto	21 días		
Redacción de memoria final	15 días		
Creación de presentación	6 días		
Entrega final (10/06/2013)	Hito	10/06/2013	

	Name
1	**Análisis previo y planificación**
2	Selección del proyecto
3	Propuesta y justificación del proyecto
4	Definición de funcionalidades del proyecto
5	Planificación del proyecto
6	Creación del proyecto
7	**Análisis funcional**
8	Especificación de requisitos funcionales
9	Especificación de requisitos no funcionales
10	**Diseño**
11	Diseño conceptual
12	Diagramas de casos de uso
13	Diagramas de clases
14	Diagramas de secuencias
15	Diseño de interfaz de usuario
16	**Prototipo**
17	Diseño y realización del prototipo
18	Preparación del documento
19	**Implementación y pruebas**
20	Desarrollo de programa principal
21	Desarrollo de interfaz
22	Pruebas
23	Creación de instrucciones
24	**Entrega de proyecto**
25	Redacción de memoria final
26	Creación de presentación



Análisis, diseño y prototipo

Escenario de partida

Sunny Hosting S.L. es una empresa nacional que ofrece a sus clientes servicios de registro de dominio (compra de nombres de Internet con terminación .com, .net, ...), housing (hospedaje en sus datacenters de servidores propios de clientes para utilizar mayor ancho de banda, reducción de costes en facturas de luz, …), VPS (acrónimo de Virtual Private Server, servidores virtualizados en los cuales los clientes comparten instancias con otros clientes sobre el mismo servidor físico) y servidores dedicados (servidores físicos completamente a disposición del cliente, los cuales pueden instalar cualquier sistema operativo y aplicaciones que deseen).

Después de haber pasado la pasada década vendiendo estos servicios y obteniendo buenos resultados, los clientes empiezan a demandar a Sunny Hosting más flexibilidad: con la democratización de Internet en todos los hogares y países, las aplicaciones Web sirven cada vez más y más usuarios, en todo tipo de dispositivos. Aumentar las capacidades de los servidores que hospedan estas aplicaciones es muy caro, y migrar a otros servidores más potentes conlleva mucho tiempo y errores, debido al típico ciclo hacer backup/provisionar nuevo servidor/instalar sistema operativo/instalar middleware/restaurar backup.
Los clientes necesitan soluciones mucho más dinámicas y elásticas, necesitan recursos de computación que puedan crecer bajo demanda el tiempo que sea necesario y adaptarse a la carga real necesaria para satisfacer las peticiones de sus aplicaciones.

El lanzamiento por parte de la compañía Amazon de AWS (Amazon Web Services) en el año 2006 supone un cambio en el paradigma de prestación de servicios de computación, ofreciendo un amplio abanico de servicios y aplicaciones en la nube que solucionan muchos de los problemas de clientes mencionados en el párrafo anterior.
Este nuevo modelo supone una amenaza a las empresas de hosting tradicionales.

Sunny Hosting necesita vender servicios de computación en la nube para no quedarse atrás, y por ello empieza a buscar alternativas. De todas ellas, se decanta por Openstack, un software de IAAS (Infrastructure As A Service) de código abierto que ofrece funcionalidades muy similares a AWS.

Sunny Hosting implanta Openstack en un proyecto piloto y empieza a ver el gran potencial que tiene para proveer servicios de computación en la nube con calidad empresarial.
Es fácil de usar, escala de manera horizontal, y con la licencia open-source es posible mejorar funcionalidades.

La aplicación web que trae Openstack de serie para administrar el sistema es funcional. Sin embargo, Sunny Hosting observa que no hay aplicaciones móviles para administrar Openstack en tabletas Android, dispositivos que tienen todos sus empleados por la gran autonomía que ofrecen, pequeño tamaño y facilidad de uso.

Es por ello que se estima la conveniencia de crear una aplicación de gestión de Openstack en Android que tenga las típicas funcionalidades que se pueden encontrar en la aplicación web:

- Autenticar usuarios
- Administrar servidores/instancias
- Administrar volúmenes
- Administrar repositorio de objetos/ficheros
- Administrar plantillas de hardware/sabores
- Administrar plantillas de software/imágenes
- Administrar usuarios
- Monitorizar servicios y procesos
- Realizar informes de uso

Los beneficios de tener una aplicación para tabletas que pueda administrar Openstack son:

- Empleados no necesitan cargar con un portátil, pueden usar su tableta en cualquier lugar para administrar la nube de Sunny Hosting
- Mejora la imagen de la empresa, por utilizar las últimas tendencias tecnológicas, en este caso aplicaciones móviles en sistema operativo Android
- Posibilidad de vender esta aplicación a terceros en programas de resellers
- Posibilidad de vender esta aplicación a clientes finales en caso de implantación de Openstack en los datacenters propios de clientes (nube privada)

Usuarios del sistema

El sistema tendrá los siguientes tipos de usuarios:

- Usuario
- Usuario Gestor
- Usuario Administrator

El tipo Usuario es el más básico de todos. Tiene acceso a las funcionalidades de manejo de perfiles de conexión, como crear perfiles, borrarlos y clonarlos.
También tiene acceso a autenticarse en Openstack mediante el uso éstos perfiles.

El tipo Usuario Gestor es la clase de usuario más habitual de nuestra aplicación.
Tiene acceso a todas las funcionalidades propias de la clase Usuario por herencia, y además puede gestionar servidores y volúmenes.
Las operaciones típicas sobre estos recursos de Openstack serán tanto listarlos como obtener detalle de cualquier recurso.

El tipo Usuario Administrador es la clase de usuario que tiene acceso a todas las funcionalidades de la aplicación.
Aparte de las propias heredadas de Usuario y Usuario Gestor, también es capaz de gestionar imágenes , sabores , usuarios y proyectos.

El tipo de operaciones y relaciones de estos tres tipos de usuarios se verán con más detalle en el diagrama de casos de uso más adelante.

Requisitos funcionales

Siguiendo la división de funcionalidades requiridas en el apartado de escenario de partida, los requisitos funcionales se agrupan en bloques:

- Funcionalidades de autenticación
- Funcionalidades de servidores
- Funcionalidades de volúmenes
- Funcionalidades de imágenes
- Funcionalidades de proyectos
- Funcionalidades de sabores
- Funcionalidades de usuarios

Funcionalidades de autenticación

Las siguientes funcionalidades van dirigidas a perfiles de conexión y autenticación de usuarios con los mismos:

Crear perfil de conexión
Permite a los usuarios crear perfiles de conexión, con ternas de usuario/contraseña/proyecto, para posterior autenticación.

Borrar perfil de conexión
Permite a los usuarios borrar perfiles de conexión creados con anterioridad.

Clonar perfil de conexión
Permite a los usuarios clonar o duplicar perfiles de conexión ya existentes. Esto permite reutilización de parámetros de otros perfiles, por ejemplo crear perfiles de conexión con mismo usuario y contraseña y solo modificar el proyecto.

Autenticar en sistema con perfil de conexión
Permite a los usuarios autenticarse en Openstack, utilizando los parámetros de conexión especificados en el perfil

Funcionalidades de servidores

Las siguientes funcionalidades van dirigidas a la gestión de servidores o instancias de Openstack:

Obtener lista de servidores
Permite al usuario gestor obtener una lista completa de los servidores correspondientes al proyecto al cual está autenticado.

Obtener información detallada de servidor
Permite al usuario gestor obtener información detallada de cualquier servidor que seleccione de la lista obtenida en la funcionalidad anterior.

Funcionalidades de volúmenes

Las siguientes funcionalidades van dirigidas a la gestión de volúmenes de Openstack:

Obtener lista de volúmenes
Permite al usuario gestor obtener una lista completa de los volúmenes correspondientes al proyecto al cual está autenticado.

Obtener información detallada de volumen
Permite al usuario gestor obtener información detallada de cualquier volumen que seleccione de la lista obtenida en la funcionalidad anterior.

Funcionalidades de imágenes

Las siguientes funcionalidades van dirigidas a la gestión de plantillas de software o imágenes de Openstack:

Obtener lista de imágenes
Permite al usuario gestor obtener una lista completa de las imágenes correspondientes al proyecto al cual está autenticado.

Obtener información detallada de imagen
Permite al usuario gestor obtener información detallada de cualquier imagen que seleccione de la lista obtenida en la funcionalidad anterior.

Funcionalidades de proyectos

Las siguientes funcionalidades van dirigidas a la gestión de proyectos de Openstack:

Obtener lista de proyectos
Permite al usuario administrador obtener una lista completa de los proyectos disponibles en Openstack.

Obtener información detallada de proyecto
Permite al usuario gestor obtener información detallada de cualquier proyecto que seleccione de la lista obtenida en la funcionalidad anterior.

Funcionalidades de sabores

Las siguientes funcionalidades van dirigidas a la gestión de plantillas de hardware o sabores de Openstack:

Obtener lista de sabores
Permite al usuario administrador obtener una lista completa de los sabores disponibles en Openstack.

Obtener información detallada de sabor
Permite al usuario administrador obtener información detallada de cualquier sabor que seleccione de la lista obtenida en la funcionalidad anterior.

Funcionalidades de usuarios

Las siguientes funcionalidades van dirigidas a la gestión de usuarios de Openstack:

Obtener lista de usuarios
Permite al usuario administrador obtener una lista completa de los usuarios disponibles en Openstack.

Obtener información detallada de usuario
Permite al usuario administrador obtener información detallada de cualquier usuario que seleccione de la lista obtenida en la funcionalidad anterior.

Requisitos no funcionales

Los requisitos no funcionales de la aplicación los agrupo en varios bloques:

- Requisitos de interfaz
- Requisitos de seguridad
- Requisitos de información
- Requisitos de control de errores

Requisitos de interfaz

- Debe ser específica para Android ICS 4.0 o superior
- Tiene que estar optimizada para tabletas
- La navegación entre los distintos recursos de Openstack será mediante pestañas
- Debe ser fácil de usar
- Debe seguir los patrones de diseño recomendados por la documentación de Android

Requisitos de seguridad

Antes de poder acceder y utilizar de manera efectiva la aplicación con Openstack, es preciso que el usuario crea un perfil de conexión. Este perfil de conexión contendrá un usuario, una contraseña y un proyecto al cual el usuario tenga las debidas credenciales.
En el momento de autenticar al usuario en Openstack, si los datos proporcionados por el perfil de conexión son incorrectos, la aplicación retornará un error, dando las debidas explicaciones del fallo.

La aplicación mostrará información y habilitará acciones propias de las credenciales del correspondiente usuario del perfil de conexión, y por tanto no violará estas credenciales mostrando información a la cual el usuario no tenga acceso ni mostrando acciones a las cuales el usuario no esté autorizado.

Requisitos de información

Para cada tipo de recurso de Openstack que la aplicación administra, la aplicación mostrará una serie de atributos que se corresponde con el modelo detallado en las diferentes APIs de Openstack. En los siguientes párrafos se muestra en detalle los requisitos de información que la aplicación debe mostrar para cada uno de los diferentes recursos.

La información de los servidores que debe mostrarse es:

- Nombre
- ID
- Estado
- Sabor
- RAM
- VCPUs
- Tamaño de disco
- Direcciones IP
- Grupos de seguridad
- Imágenes
- Volúmenes conectados

La información de los volúmenes que debe mostrarse es:

- Nombre
- ID
- Estado
- Tamaño
- Fecha de creación
- Servidor conectado

La información de las imágenes que debe mostrarse es:

- Nombre
- ID
- Estado
- Pública
- Checksum
- Fecha de creación
- Fecha de actualización
- Tamaño
- Formato de contenedor
- Formato de disco
- ID de kernel
- ID de ramdisk

La información de los sabores que debe mostrarse es:

- Nombre
- ID
- VCPUs
- RAM
- Tamaño de disco root
- Tamaño de disco efímero
- Tamaño de disco swap

La información de los proyectos que debe mostrarse es:

- Nombre
- ID
- Descripción
- Habilitado
- Usuarios miembros

La información de los usuarios que debe mostrarse es:

- Nombre
- ID
- Email
- Habilitado

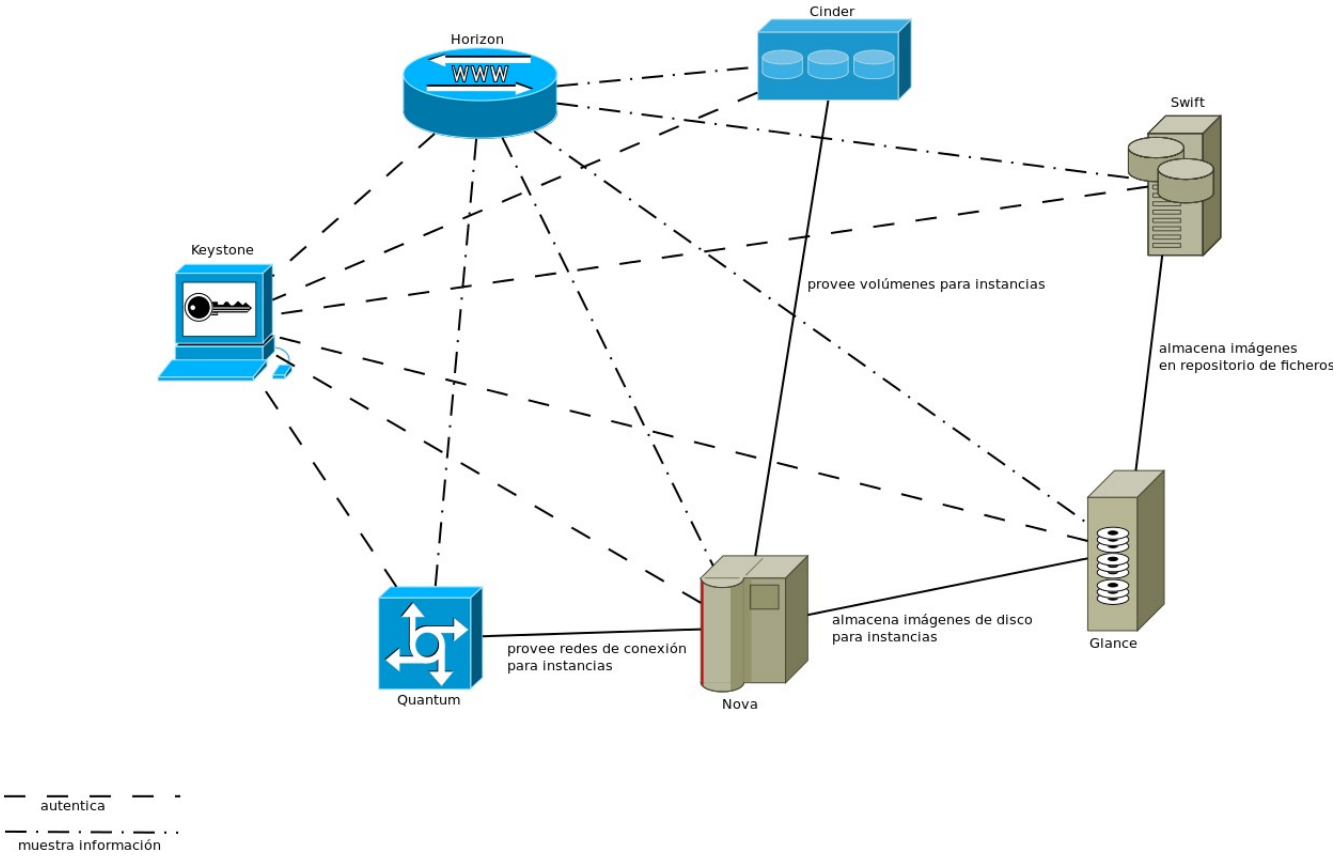
Requisitos de control de errores

La aplicación debe mostrar al usuario una notificación al usuario en caso de cualquier error, y no cerrarse abruptamente con una excepción.

Arquitectura de Openstack

Antes de entrar en materia en la arquitectura de la aplicación, es bueno tener una introducción a qué es Openstack y cuales son sus componentes.

A continuación, un diagrama que muestra los principales componentes de Openstack con las diversas interacciones entre ellos:



Openstack es una colección de componentes que provee servicios de computación en la nube.
Está compuesto de siete componentes principales que se comunican entre ellos para suministrar IAAS (Infrastructure as a service) o infraestructura como servicio.
Cada uno de ellos consta de su propia API basada en servicios web REST, para interectuar entre ellos o para dar servicios a terceras aplicaciones o clientes como en nuestro caso.

Está diseñado para escalar de forma masiva horizontalmente. Eso quiere decir que no es necesario aumentar las capacidades de hardware de los servidores que hospedan los componentes (a esta técnica se le denomina *scale-up*), sino que se pueden añadir tantos nodos a la funcionalidad que se desea escalar de forma horizontal sin ningún tipo de límite.

Esto es posible ya que Openstack está diseñado siguiendo un tipo de arquitectura software llamada *shared nothing*, en la que cada nodo tiene toda la información para funcionar de manera autónoma, de manera que si se cae un nodo la nube sigue funcionando y si se añaden nodos la carga se balancea de manera autómatica.
Toda la información del estado global de Openstack se almacena en una base de datos.

A continuación, una descripción detallada de las funciones de cada uno de los componentes

Keystone
Este componente se encarga de dar servicios de autenticación y autorización a todos los demás componentes. También contiene un catálago de los servicios disponibles en la nube de Openstack, así por ejemplo se puede extraer información de Keystone sobre cuales son los puntos de entrada de API de los demás componentes (URLs, puertos, …), servicios instalados, etc

Horizon
Este componente es una aplicación web que proporciona una interfaz de usuario para administrar Openstack. Con ella se pueden hacer todas las típicas tareas de usuarios de Openstack, como crear instancias, pararlas, administrar volúmenes, etc.
Necesita conexión directa con los demás componentes para administrarlos.
Es una aplicación escrita con el framework Django.

Swift
Este componente es un repositorio de objetos. Permite almacenar y extraer ficheros, y soporta versionado de ficheros.
Este componente proporciona el mismo tipo de servicios que servicios populares como Dropbox, Box.com , Google Drive, …

Nova
Este componente es el que proporciona servicios de computación. Proporciona instancias de máquinas virtuales bajo demanda, mediante la conexión con hipervisores.
Soporta los hipervisores más populares del mercado, com KVM, Xen, ESXi, …

Quantum
Este componente (de reciente inclusión desde la versión Folsom) proporciona NAAS (Network As A Service) o redes de conexión como servicio.
Permite crear redes IP, VLANs, políticas de seguridad en las redes, etc.
Típicamente, se crean las redes con Quantum y luego se conectan a interfaces de red de instancias manejadas por Nova.
Tiene una arquitectura basada en plugins, permite usar muchos tipos de tecnologías de red como Cisco, Nicira, Linux bridges, …

Glance
Este componente es un catálogo y repositorio de plantillas de software o imágenes de instancias, que son usadas por Nova para crear instancias.
Soporta diferentes métodos para almacenar estas imágenes, como Amazon S3 y sistemas de ficheros Unix/Linux, aunque típicamente se utiliza Swift como backend.

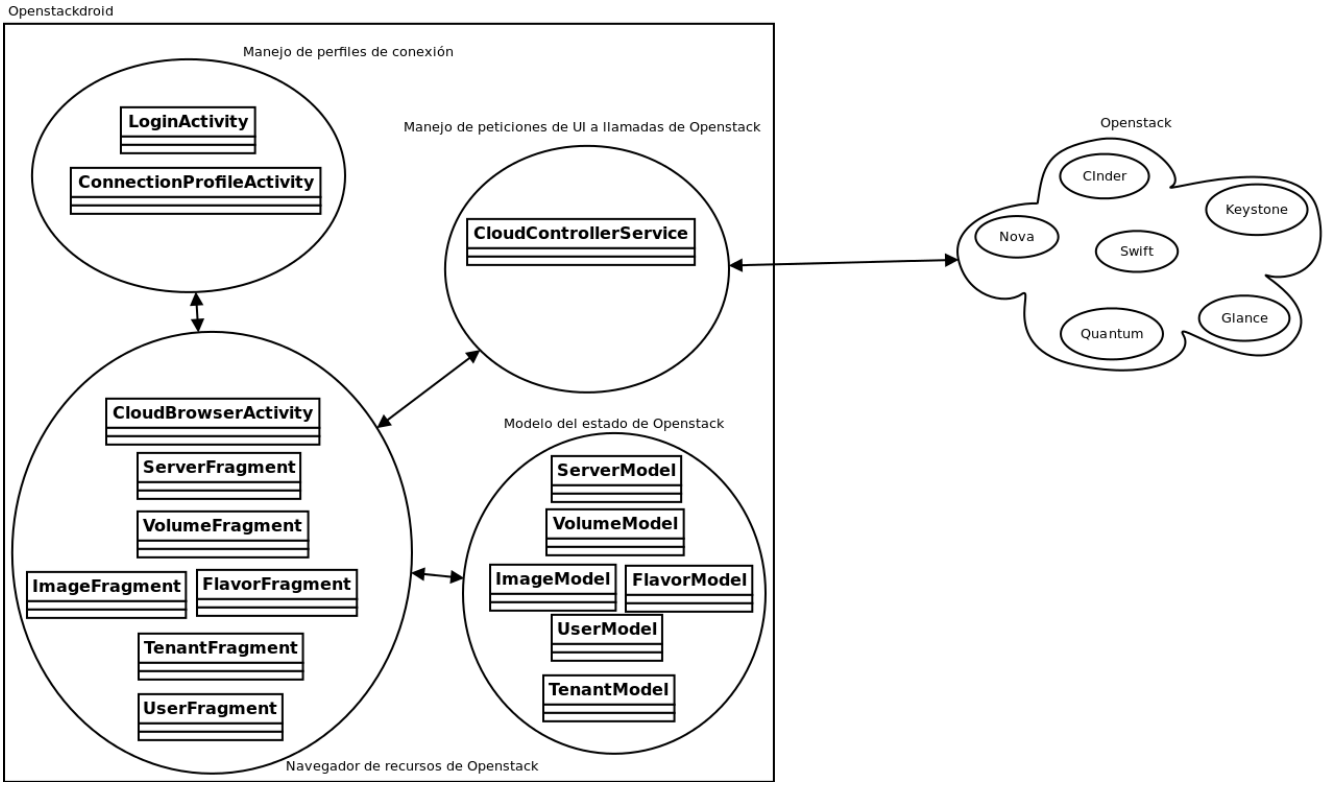
Cinder
Este componente (de reciente inclusión desde la versión Folsom) proporciona volúmenes persistentes para instancias manejadas por Nova. Son persistentes ya que sobreviven a la destrucción de instancias, se pueden conectar a cualquier instancia bajo demanda.
Típicamente esta almacenamiento se usa en instancias para almacenar datos de aplicaciones web (como la partición /var/www) , que son comunes en clusters web, o para guardar datos de bases de datos.

Arquitectura global de la aplicación

La arquitectura global de la aplicación de Openstackdroid la he dividido en cuatro componentes funcionales:

- Manejo de perfiles de conexión
- Modelo del estado de Openstack
- Navegador de recursos de Openstack
- Manejo de peticiones de interfaz de usuario a llamadas de Openstack

A continuación, un diagrama que describe esta arquitectura de alto nivel:



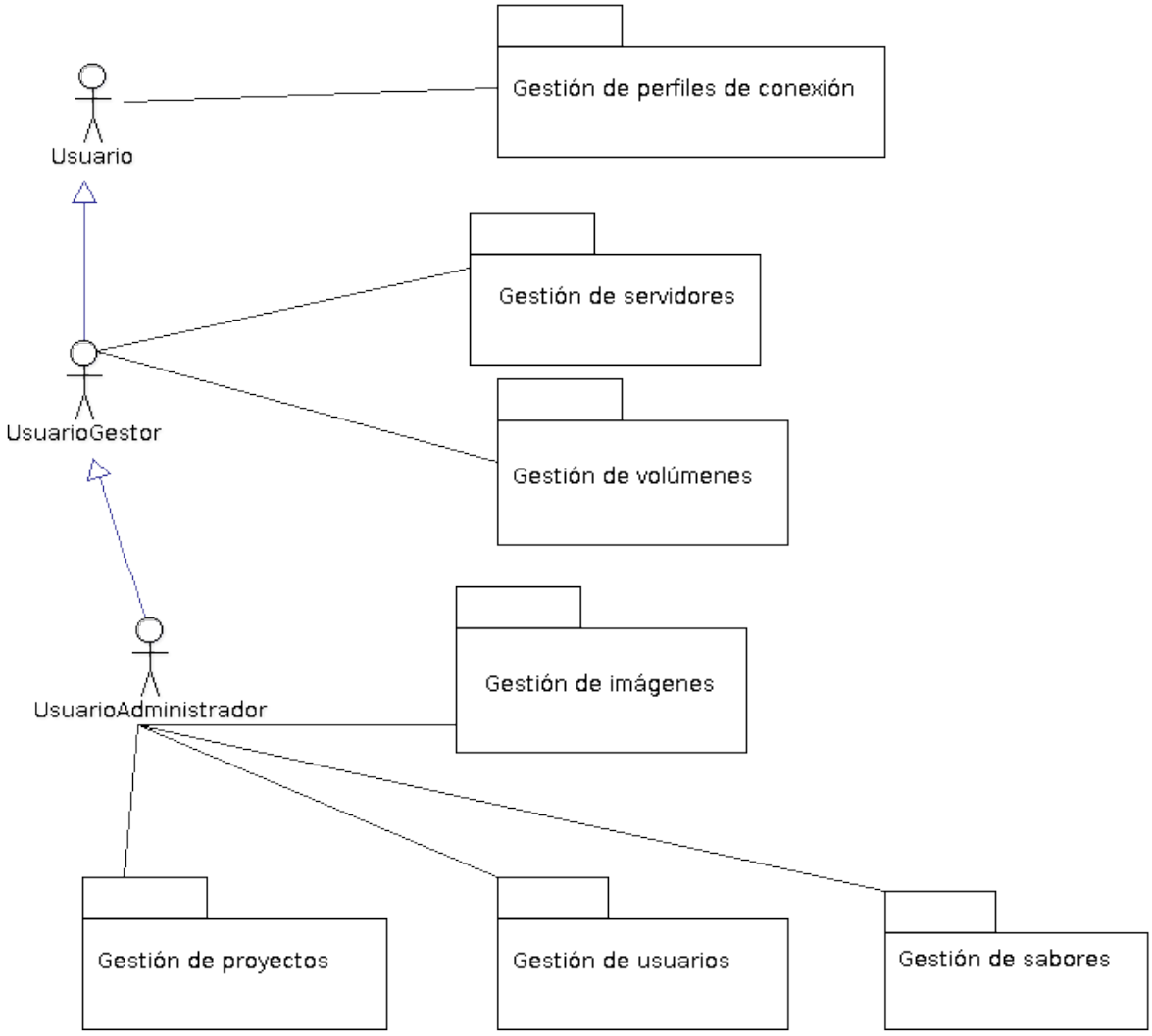
Es importante subrayar que las APIs de Openstack implementa todos sus métodos con servicios web REST (REpresentational State Transfer), con lo cual las diferentes interacciones desde la aplicación a Openstack es mediante verbos HTTP debidamente codificados, según las especificaciones de las diferentes APIs de Openstack. Para más información, consultar las guías de desarrollo de Openstack que añadiré en la memoria final en el apartado de bibliografía o fuentes de información.

Siguiendo el patrón de diseño MVC (Modelo-Vista-Controlador), se comprueba que en el diagrama el modelo se corresponde a “Modelo del estado de Openstack”, la vista a “Navegador de recursos de Openstack” y el controlador a “Manejo de peticiones de interfaz de usuario a llamadas de Openstack” .

Dentro de cada unidad funcional, he incluído las principales clases que implementan las funcionalidades de la aplicación.
Estas clases las explico con detalle más adelante en el diagrama de clases.

Diagramas de casos de uso

A continuación, el diagrama de casos de uso principal de la aplicación Openstackdroid:

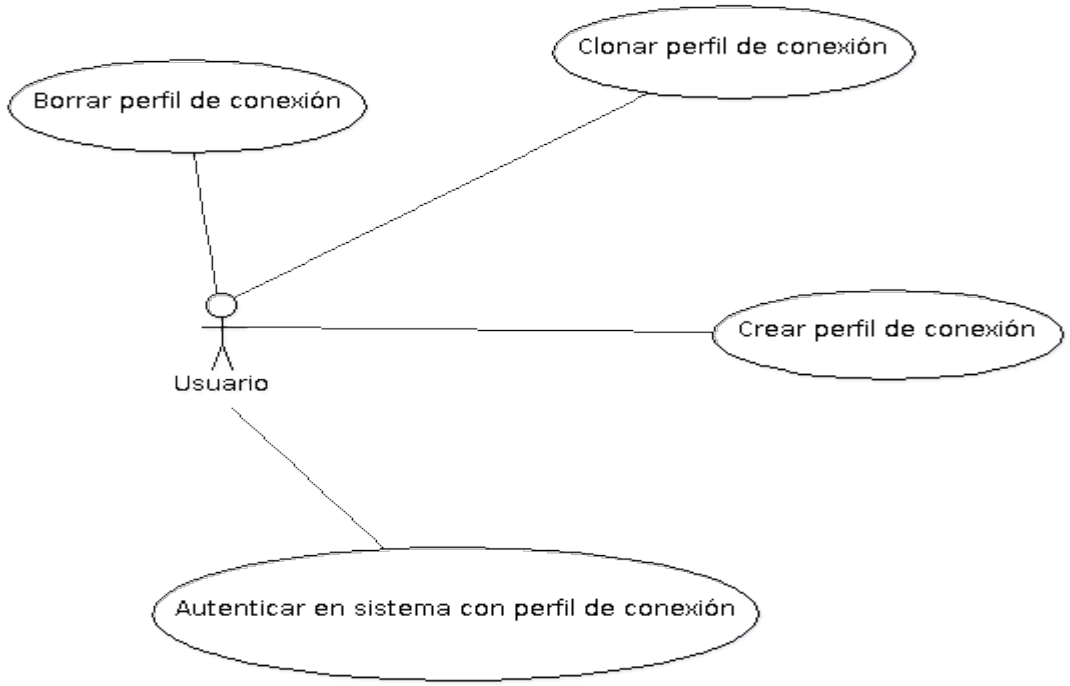


Como se puede observar, los usuarios de los casos de uso son los mismo que se explicaron en detalle en el apartado anterior “Usuarios del sistema”.

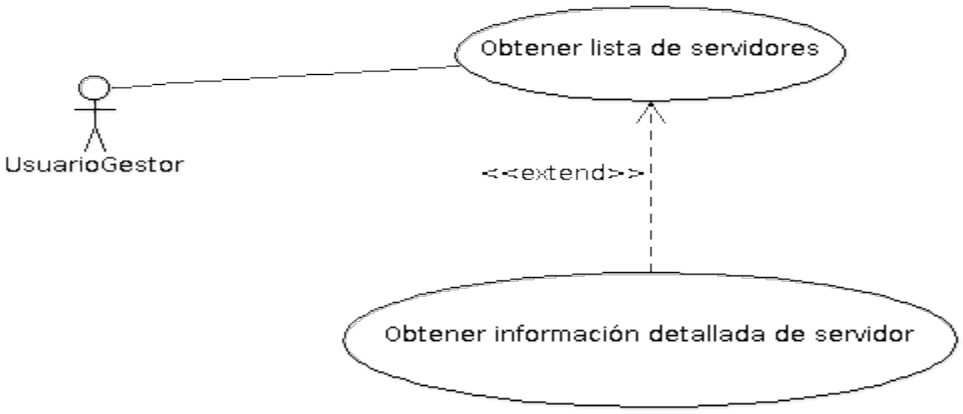
Se ha dividido los diferentes casos de uso en varios paquetes, para no dibujar todos y cada uno de los casos de uso en un solo diagrama y facilitar la comprensión.

A continuación, los diagramas de casos de uso de cada uno de los paquetes que se detallan en el diagrama de caso de uso principal:

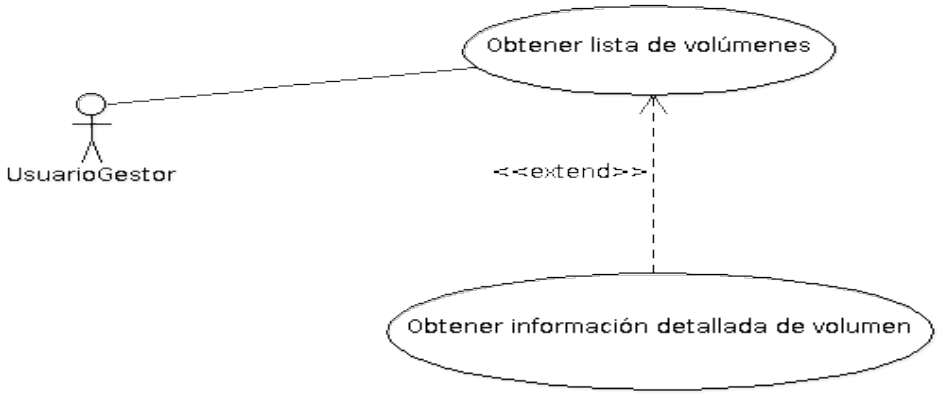
Paquete: Gestión de perfiles de conexión



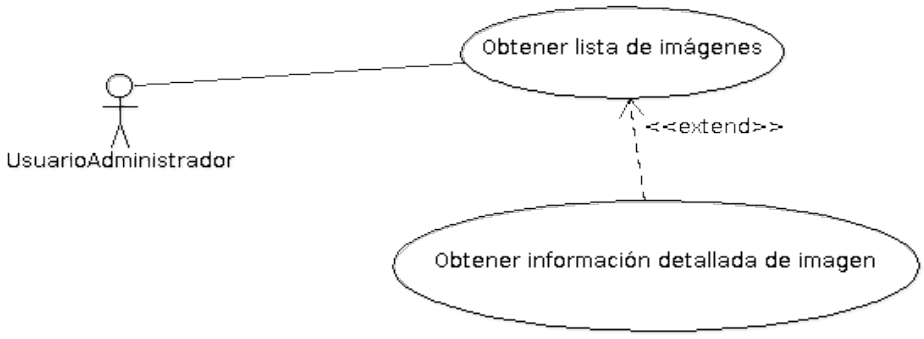
Paquete: Gestión de servidores



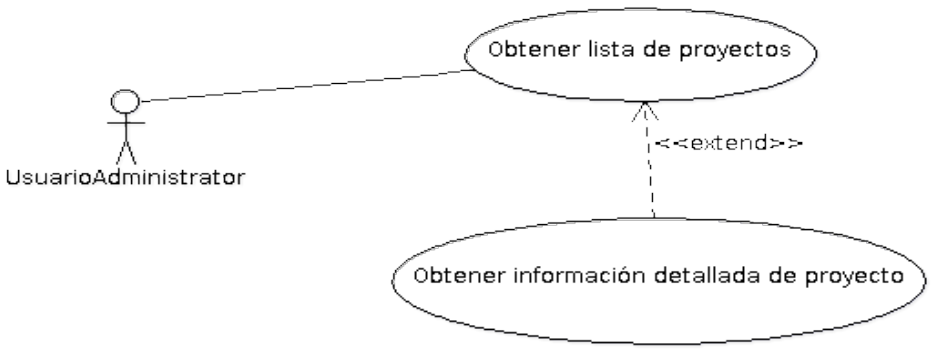
Paquete: Gestión de volúmenes



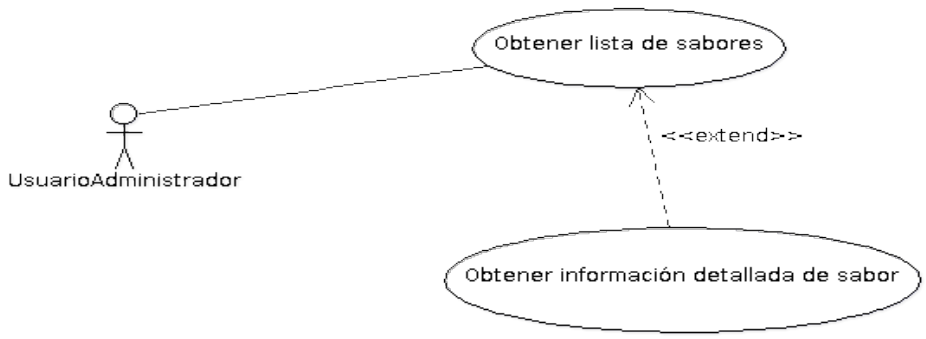
Paquete: Gestión de imágenes



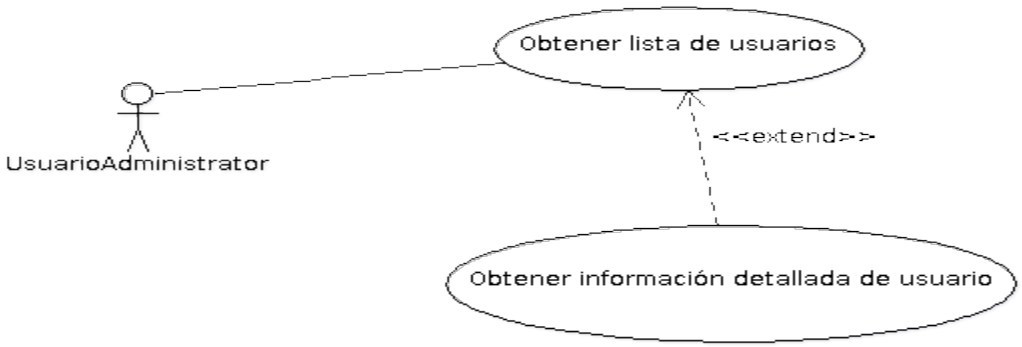
Paquete: Gestión de proyectos



Paquete: Gestión de sabores



Paquete: Gestión de usuarios



Descripción textual de casos de uso

Nombre	Crear perfil de conexión
Autor	Ricardo Carrillo Cruz
Resumen	Crea un perfil de conexión, con una terna usuario/contraseña/proyecto, para posterior utilización como conexión a Openstack
Actor	Usuario
Precondiciones	Ninguna
Postcondiciones	El usuario crea con éxito el perfil de conexión y se muestra disponible en pantalla de conexión
Flujo normal	1. El usuario toca el botón de añadir perfil 2. Se abre la pantalla de añadir perfil 3. El usuario especifica usuario, contraseña y proyecto 4. El usuario guarda el perfil 5. Se vuelve a pantalla de login, mostrando el perfil recién creado
Flujos alternativos	El usuario toca el botón de volver y no guarda el perfil
Inclusiones	Ninguna
Extensiones	Niguna

Nombre	Borrar perfil de conexión
Autor	Ricardo Carrillo Cruz
Resumen	Borra un perfil de conexión existente
Actor	Usuario
Precondiciones	El usuario ha creado un perfil como mínimo
Postcondiciones	El perfil se borra con éxito y deja de mostrarse en la pantalla principal de login
Flujo normal	1. El usuario deja pulsado el perfil a borrar 2. Se activa el botón de borrar en la barra de acciones superior 3. El usuario pulsa el botón de borrar 4. El perfil se borra y ya no aparece en la pantalla principal de login
Flujos alternativos	Ninguno
Inclusiones	Ninguna
Extensiones	Niguna

Nombre	Clonar perfil de conexión

Autor	Ricardo Carrillo Cruz
Resumen	Clona o copia un perfil de conexión existente, para reutilizar los datos de un perfil existente y posterior modificación por parte del usuario
Actor	Usuario
Precondiciones	El usuario ha creado un perfil de conexión como mínimo
Postcondiciones	Se crea un nuevo perfil con los datos de conexión del perfil de origen
Flujo normal	1. El usuario deja pulsado el perfil a clonar 2. Se activa el botón de clonar perfil en la barrra de acciones superior 3. El usuario toca el botón de clonar perfil 4. Se presenta un diálogo preguntando por el nombre del nuevo perfil 5. El usuario toca el botón OK 6. Se clona el perfil y el nuevo perfil aparece en la pantalla principal de login
Flujos alternativos	El usuario toca el botón de volver en la pantalla de diálogo que pide el nombre del nuevo perfil clonado
Inclusiones	Ninguna
Extensiones	Niguna

Nombre	Autenticar en sistema con perfil de conexión
Autor	Ricardo Carrillo Cruz
Resumen	El usuario se autentica en Openstack con los detalles de conexión del perfil seleccionado
Actor	Usuario
Precondiciones	El usuario ha creado un perfil de conexión como mínimo
Postcondiciones	La autenticación se realiza con éxito o se reporta al usuario la razón de la autenticación fallida
Flujo normal	1. El usuario selecciona un perfil de conexión 2. La aplicación se conecta a Openstack con el usuario, contraseña y proyecto del perfil 3. El componente Keystone de Openstack devuelve un token de autenticación 4. La aplicación guarda en una variable el token de autenticación 5. La aplicación carga la pantalla de navegador de recursos de Openstack

Flujos alternativos	El componente Keystone de Openstack devuelve un error durante la autenticación y la aplicación muestra el fallo
Inclusiones	Ninguna
Extensiones	Niguna

Nombre	Obtener lista de servidores
Autor	Ricardo Carrillo Cruz
Resumen	Muestra servidores o instancias de Openstack
Actor	UsuarioGestor
Precondiciones	El usuario se ha autenticado con éxito
Postcondiciones	La pantalla navegador de recursos muestra una lista de todos los servidores o instancias del proyecto al cual el usuario se ha autenticado anteriormente
Flujo normal	1. El usuario toca la pestaña “Servidores” 2. La aplicación conecta con API de Openstack 3. Openstack devuelve lista de resultados 4. La aplicación muestra una lista con todos los resultados
Flujos alternativos	Openstack devuelve un error de conexión o el token de autenticación ha expirado
Inclusiones	Ninguna
Extensiones	Obtener información detallada de servidor

Nombre	Obtener información detallada de servidor
Autor	Ricardo Carrillo Cruz
Resumen	Muestra información detallada de servidor o instancia de Openstack
Actor	UsuarioGestor
Precondiciones	El usuario se ha autenticado con éxito
Postcondiciones	La pantalla navegador de recursos muestra una pantalla lateral con información detallada del servidor seleccionado
Flujo normal	1. El usuario toca la pestaña “Servidores” 2. La aplicación conecta con API de Openstack 3. Openstack devuelve lista de resultados 4. La aplicación muestra una lista con todos los resultados 5. El usuario toca uno de los servidores de la lista

	6. La aplicación muestra información detallada del servidor seleccionado
Flujos alternativos	Openstack devuelve un error de conexión o el token de autenticación ha expirado
Inclusiones	Ninguna
Extensiones	NInguna

Nombre	Obtener lista de volúmenes
Autor	Ricardo Carrillo Cruz
Resumen	Muestra volúmenes de Openstack
Actor	UsuarioGestor
Precondiciones	El usuario se ha autenticado con éxito
Postcondiciones	La pantalla navegador de recursos muestra una lista de todos los volúmenes del proyecto al cual el usuario se ha autenticado anteriormente
Flujo normal	1. El usuario toca la pestaña “Volúmenes” 2. La aplicación conecta con API de Openstack 3. Openstack devuelve lista de resultados 4. La aplicación muestra una lista con todos los resultados
Flujos alternativos	Openstack devuelve un error de conexión o el token de autenticación ha expirado
Inclusiones	Ninguna
Extensiones	Obtener información detallada de volumen

Nombre	Obtener información detallada de volumen
Autor	Ricardo Carrillo Cruz
Resumen	Muestra información detallada de volumen de Openstack
Actor	UsuarioGestor
Precondiciones	El usuario se ha autenticado con éxito
Postcondiciones	La pantalla navegador de recursos muestra una pantalla lateral con información detallada del volumen seleccionado
Flujo normal	1. El usuario toca la pestaña “Volúmenes” 2. La aplicación conecta con API de Openstack 3. Openstack devuelve lista de resultados 4. La aplicación muestra una lista con todos los resultados 5. El usuario toca uno de los volúmenes de la lista

	6. La aplicación muestra información detallada del volumen seleccionado
Flujos alternativos	Openstack devuelve un error de conexión o el token de autenticación ha expirado
Inclusiones	Ninguna
Extensiones	Ninguna

Nombre	Obtener lista de imágenes
Autor	Ricardo Carrillo Cruz
Resumen	Muestra imágenes de Openstack
Actor	UsuarioAdministrador
Precondiciones	El usuario se ha autenticado con éxito
Postcondiciones	La pantalla navegador de recursos muestra una lista de todos las imágenes del proyecto al cual el usuario se ha autenticado anteriormente
Flujo normal	1. El usuario toca la pestaña “Imágenes” 2. La aplicación conecta con API de Openstack 3. Openstack devuelve lista de resultados 4. La aplicación muestra una lista con todos los resultados
Flujos alternativos	Openstack devuelve un error de conexión o el token de autenticación ha expirado
Inclusiones	Ninguna
Extensiones	Obtener información detallada de imagen

Nombre	Obtener información detallada de imagen
Autor	Ricardo Carrillo Cruz
Resumen	Muestra información detallada de imagen de Openstack
Actor	UsuarioAdministrador
Precondiciones	El usuario se ha autenticado con éxito
Postcondiciones	La pantalla navegador de recursos muestra una pantalla lateral con información detallada de la imagen seleccionada
Flujo normal	1. El usuario toca la pestaña “Imágenes” 2. La aplicación conecta con API de Openstack 3. Openstack devuelve lista de resultados 4. La aplicación muestra una lista con todos los resultados 5. El usuario toca una de las imágenes de la lista

	6. La aplicación muestra información detallada de la imagen seleccionada
Flujos alternativos	Openstack devuelve un error de conexión o el token de autenticación ha expirado
Inclusiones	Ninguna
Extensiones	Ninguna

Nombre	Obtener lista de proyectos
Autor	Ricardo Carrillo Cruz
Resumen	Muestra proyectos de Openstack
Actor	UsuarioAdministrador
Precondiciones	El usuario se ha autenticado con éxito
Postcondiciones	La pantalla navegador de recursos muestra una lista de todos los proyectos
Flujo normal	1. El usuario toca la pestaña “Proyectos” 2. La aplicación conecta con API de Openstack 3. Openstack devuelve lista de resultados 4. La aplicación muestra una lista con todos los resultados
Flujos alternativos	Openstack devuelve un error de conexión o el token de autenticación ha expirado
Inclusiones	Ninguna
Extensiones	Obtener información detallada de proyecto

Nombre	Obtener información detallada de proyecto
Autor	Ricardo Carrillo Cruz
Resumen	Muestra información detallada de proyecto de Openstack
Actor	UsuarioAdministrador
Precondiciones	El usuario se ha autenticado con éxito
Postcondiciones	La pantalla navegador de recursos muestra una pantalla lateral con información detallada del proyecto seleccionado
Flujo normal	1. El usuario toca la pestaña “Proyectos” 2. La aplicación conecta con API de Openstack 3. Openstack devuelve lista de resultados 4. La aplicación muestra una lista con todos los resultados 5. El usuario toca uno de los proyectos de la lista 6. La aplicación muestra información detallada del

	proyecto seleccionado
Flujos alternativos	Openstack devuelve un error de conexión o el token de autenticación ha expirado
Inclusiones	Ninguna
Extensiones	Ninguna

Nombre	Obtener lista de sabores
Autor	Ricardo Carrillo Cruz
Resumen	Muestra sabores de Openstack
Actor	UsuarioAdministrador
Precondiciones	El usuario se ha autenticado con éxito
Postcondiciones	La pantalla navegador de recursos muestra una lista de todos los sabores
Flujo normal	1. El usuario toca la pestaña “Sabores” 2. La aplicación conecta con API de Openstack 3. Openstack devuelve lista de sabores 4. La aplicación muestra una lista con todos los resultados
Flujos alternativos	Openstack devuelve un error de conexión o el token de autenticación ha expirado
Inclusiones	Ninguna
Extensiones	Obtener información detallada de sabor

Nombre	Obtener información detallada de sabor
Autor	Ricardo Carrillo Cruz
Resumen	Muestra información detallada de sabor de Openstack
Actor	UsuarioAdministrador
Precondiciones	El usuario se ha autenticado con éxito
Postcondiciones	La pantalla navegador de recursos muestra una pantalla lateral con información detallada del sabor seleccionado
Flujo normal	1. El usuario toca la pestaña “Sabores” 2. La aplicación conecta con API de Openstack 3. Openstack devuelve lista de resultados 4. La aplicación muestra una lista con todos los resultados 5. El usuario toca uno de los sabores de la lista 6. La aplicación muestra información detallada del sabor seleccionado

Flujos alternativos	Openstack devuelve un error de conexión o el token de autenticación ha expirado
Inclusiones	Ninguna
Extensiones	Ninguna

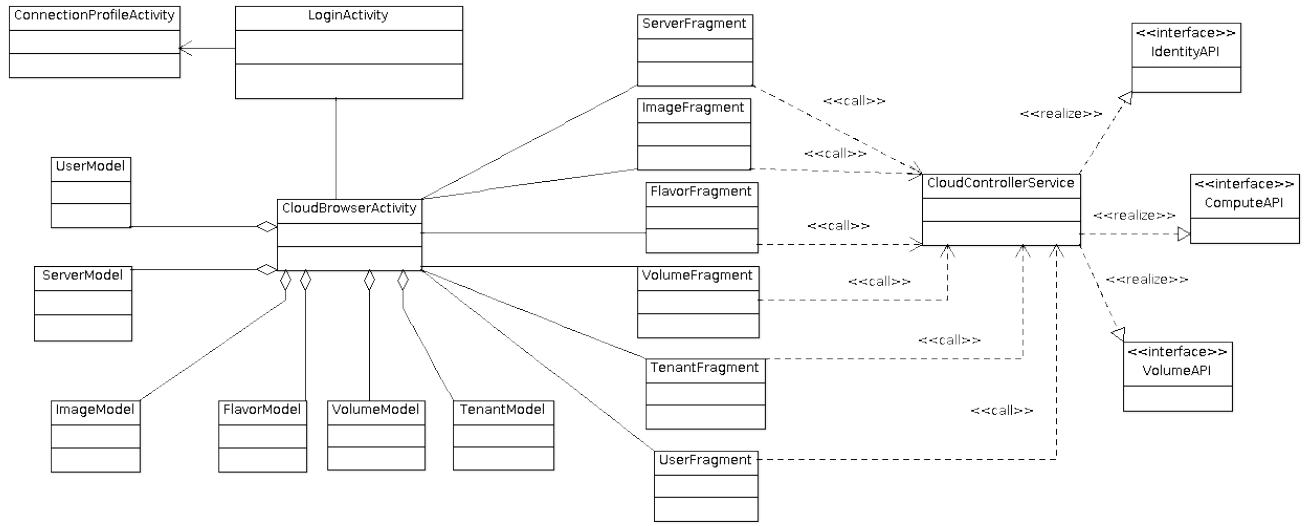
Nombre	Obtener lista de usuarios
Autor	Ricardo Carrillo Cruz
Resumen	Muestra sabores de Openstack
Actor	UsuarioAdministrador
Precondiciones	El usuario se ha autenticado con éxito
Postcondiciones	La pantalla navegador de recursos muestra una lista de todos los usuarios
Flujo normal	1. El usuario toca la pestaña “Usuarios” 2. La aplicación conecta con API de Openstack 3. Openstack devuelve lista de Usuarios 4. La aplicación muestra una lista con todos los resultados
Flujos alternativos	Openstack devuelve un error de conexión o el token de autenticación ha expirado
Inclusiones	Ninguna
Extensiones	Obtener información detallada de usuario

Nombre	Obtener información detallada de usuario
Autor	Ricardo Carrillo Cruz
Resumen	Muestra información detallada de usuario de Openstack
Actor	UsuarioAdministrador
Precondiciones	El usuario se ha autenticado con éxito
Postcondiciones	La pantalla navegador de recursos muestra una pantalla lateral con información detallada del usuario seleccionado
Flujo normal	1. El usuario toca la pestaña “Usuarios” 2. La aplicación conecta con API de Openstack 3. Openstack devuelve lista de resultados 4. La aplicación muestra una lista con todos los resultados 5. El usuario toca uno de los usuarios de la lista 6. La aplicación muestra información detallada del usuario seleccionado
Flujos alternativos	Openstack devuelve un error de conexión o el

	token de autenticación ha expirado
Inclusiones	Ninguna
Extensiones	Ninguna

Diagrama de clases

A continuación, muestro una imagen del diagrama de clases de la aplicación:



La función que tiene cada una de las clases:

LoginActivity
Esta clase muestra una lista de los perfiles de conexión disponibles y controla eventos de usuario sobre los mismos.

ConnectionProfileActivity
Esta clase es la encargada de crear y editar perfiles de conexión. Esta información se guardará en una base de datos o ficheros.

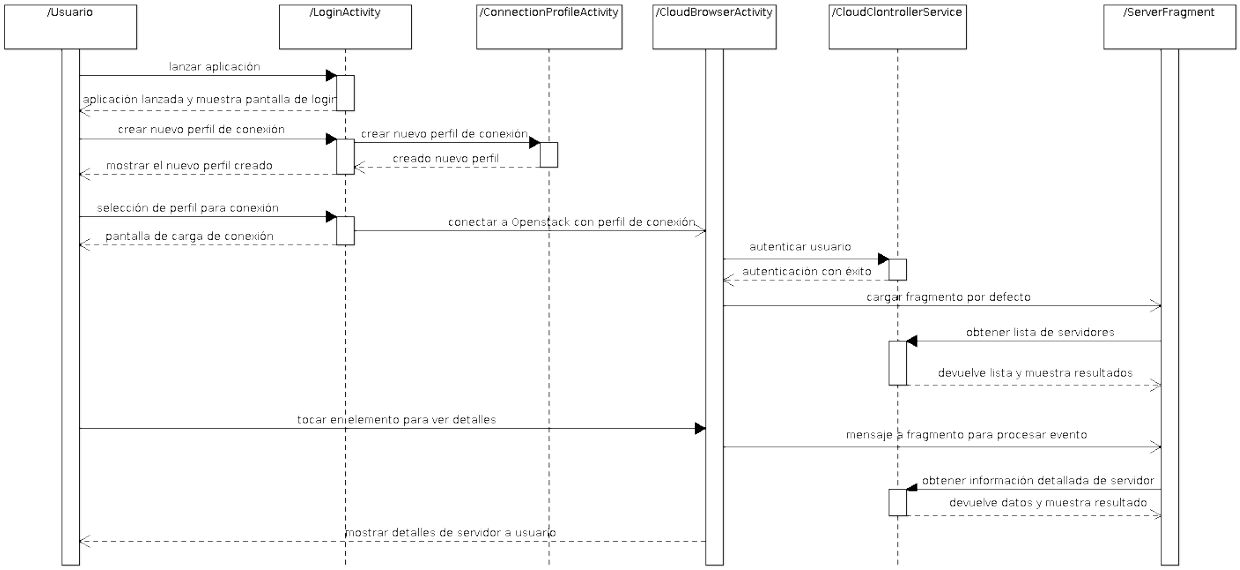
CloudBrowserActivity
Es la la clase principal, la que hace de navegador de recursos de Openstack. Tiene un Fragment para cada tipo de recurso, que manejan los tipos de Openstack correspondientes.

ServerModel, ImageModel, FlavorModel, VolumeModel, TenantModel
Son las clases que hacen de modelo de los recursos de Openstack. Una vez se obtienen de Openstack, se guarda en memoria en arrays según los distintos tipos.

CloudControllerService
Esta clase hace de cliente REST de Openstack, traduciendo las acciones del interfaz de usuario de CloudBrowserActivity a llamadas REST.
Es la clase que se encarga de implementar los distintos APIs de Openstack.

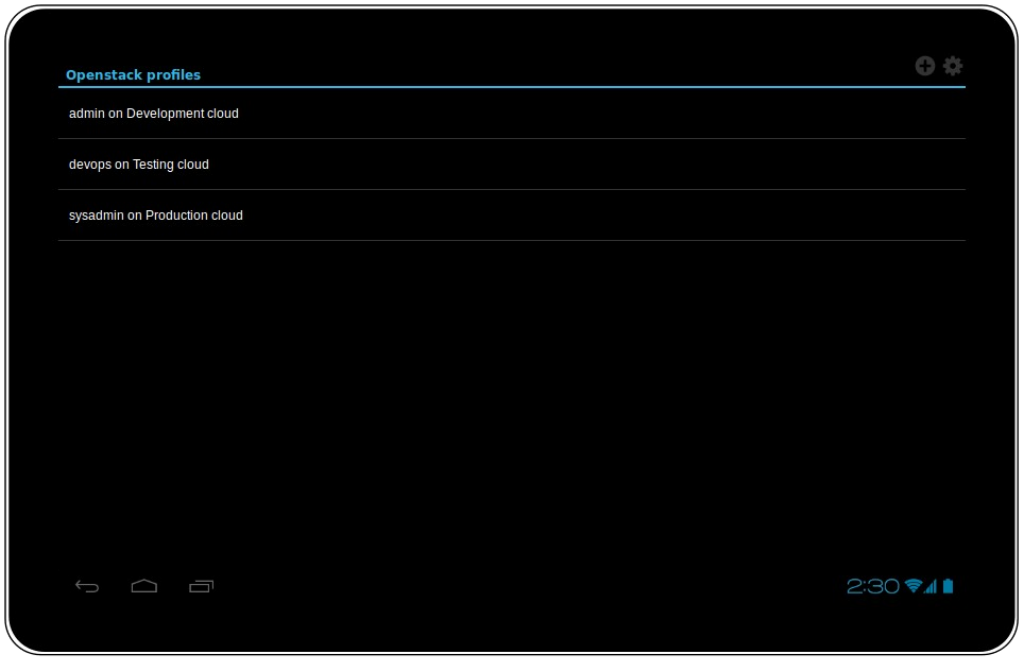
Diagrama de secuencia

A continuación, un diagrama de secuencia que muestra las distintas interacciones entre los componentes de la aplicación y los APIs de Openstack:



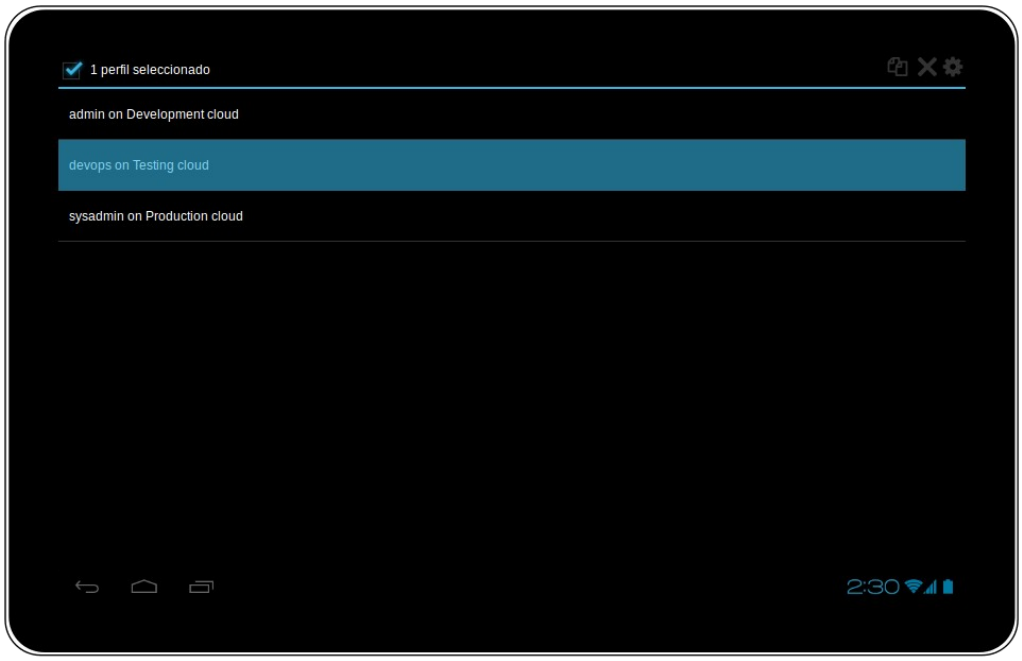
Prototipo de interfaz

A continuación, muestro una serie de pantallas del prototipo de interfaz para la aplicación OpenstackDroid:



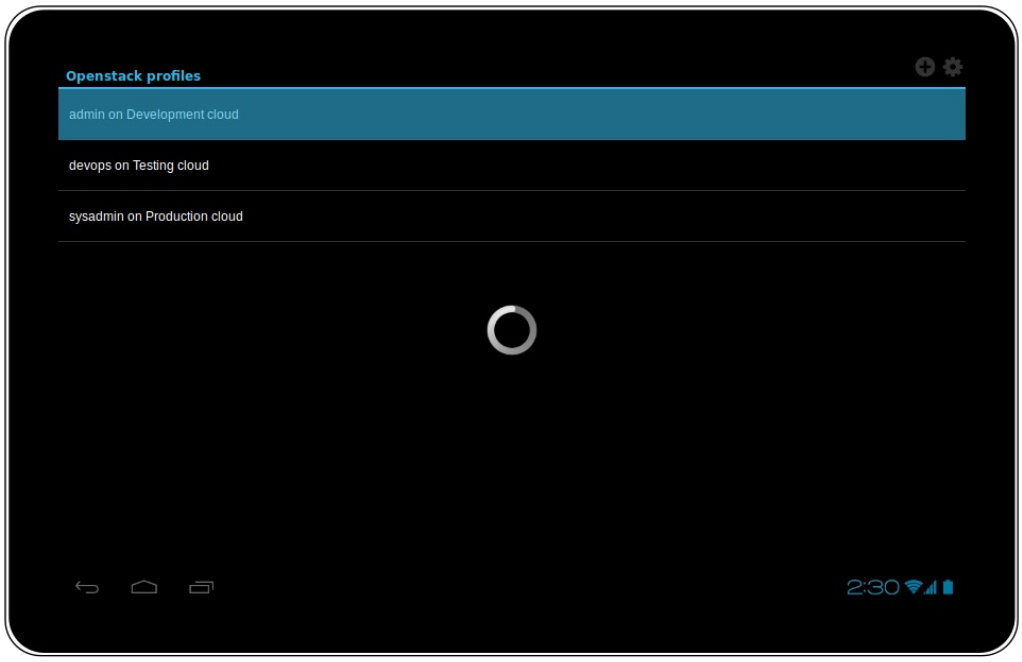
Esta es la pantalla principal, que muestra la lista de perfiles de conexión disponibles para autenticarse con Openstack.
Pueden verse en la parte superior derecha un botón para añadir perfiles y otro para ajustes en general.

Si el usuario quisiera clonar o borrar algún perfil de conexión, tan solo tendría que dejar pulsado el correspondiente perfil de la lista.
Esta acción provoca un cambio en la barra de acciones superior, tal y como se muestra en la siguiente imagen:



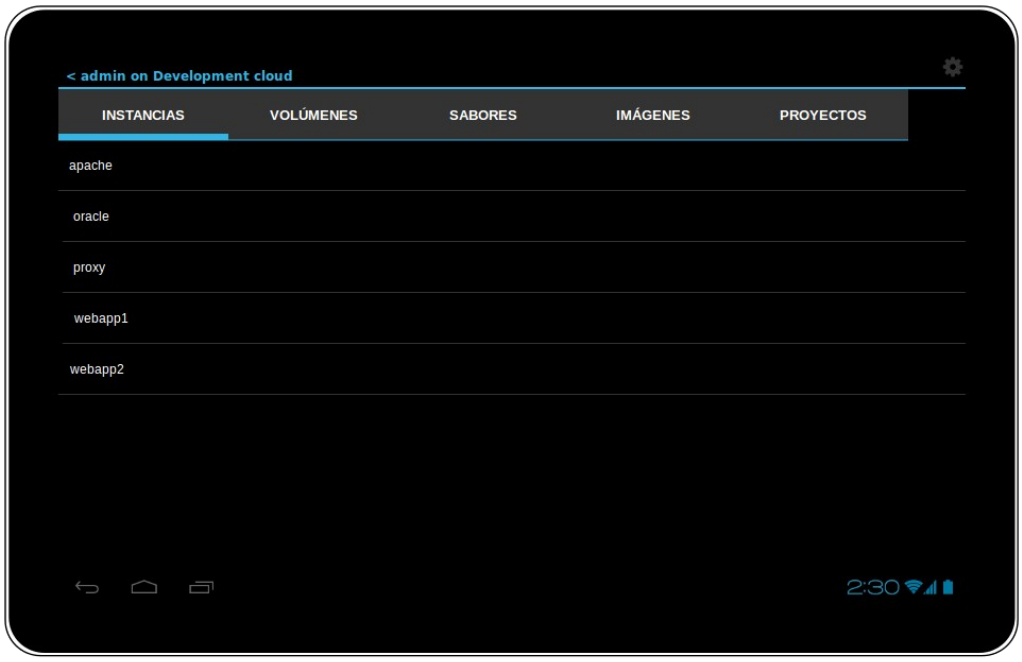
Se puede comprobar en la imagen que han aparecido dos nuevos botones en la barra superior de acciones: clonar y borrar perfil.
Así mismo, el botón de añadir perfil ha desaparecido ya que no tiene sentido en el actual contexto de perfil seleccionado.

Si por el contrario el usuario quisiera autenticarse sin más en Openstack, tan solo tendría que pulsar uno de los perfiles disponibles.
Internamente, esto provoca que la aplicación contacte con el API del componente Keystone de Openstack.
Este momento de conexión, aunque mínimo, se expresa visualmente con un icono de carga que se mueve de forma circular, como se puede ver en la siguiente imagen:



Si el API de Openstack autentica con éxito las credenciales presentadas por la aplicación, devuelve un token de autenticación que la aplicación utilizará para las siguientes interacciones con Openstack, a modo de sesión.

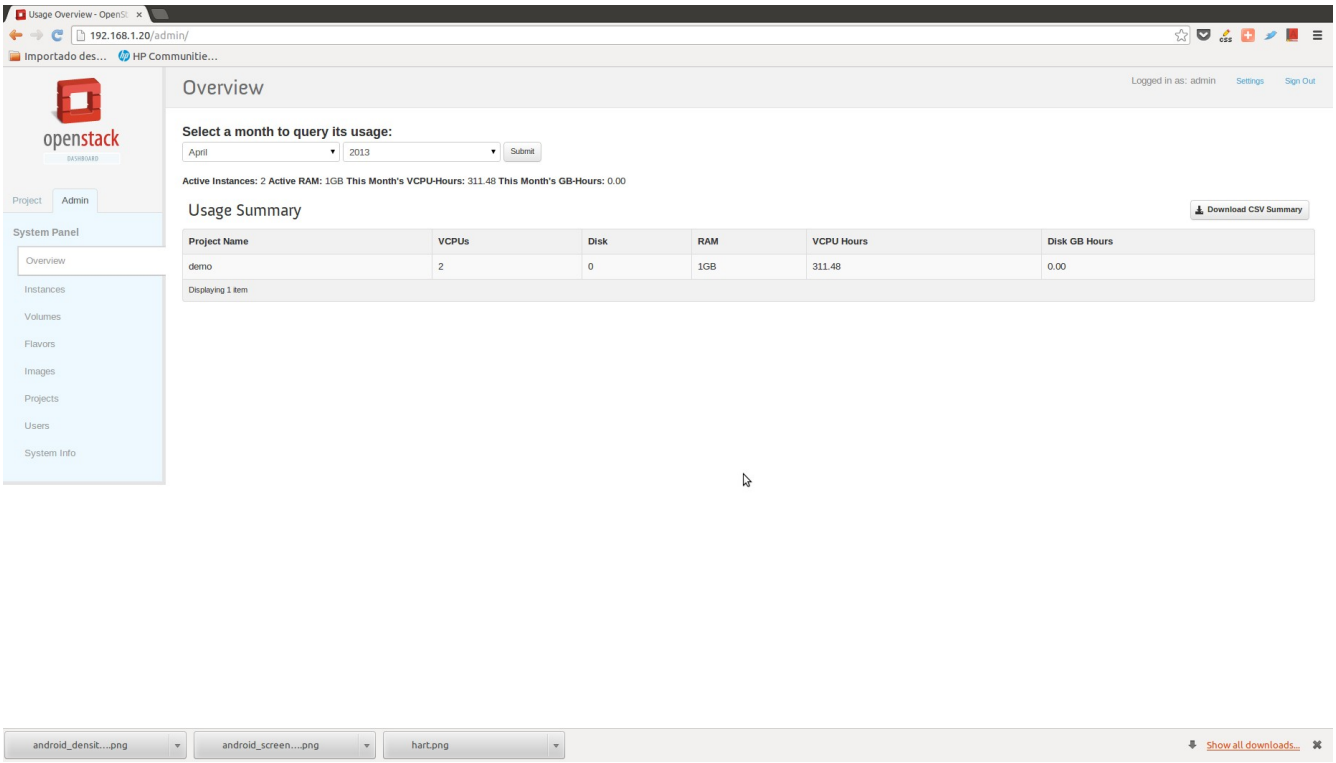
Seguidamente, la pantalla de autenticación de perfiles desaparece y da paso a la pantalla de navegador de recursos. Esto se puede ver en la siguiente imagen:



Se puede comprobar varias cosas en esta pantalla:

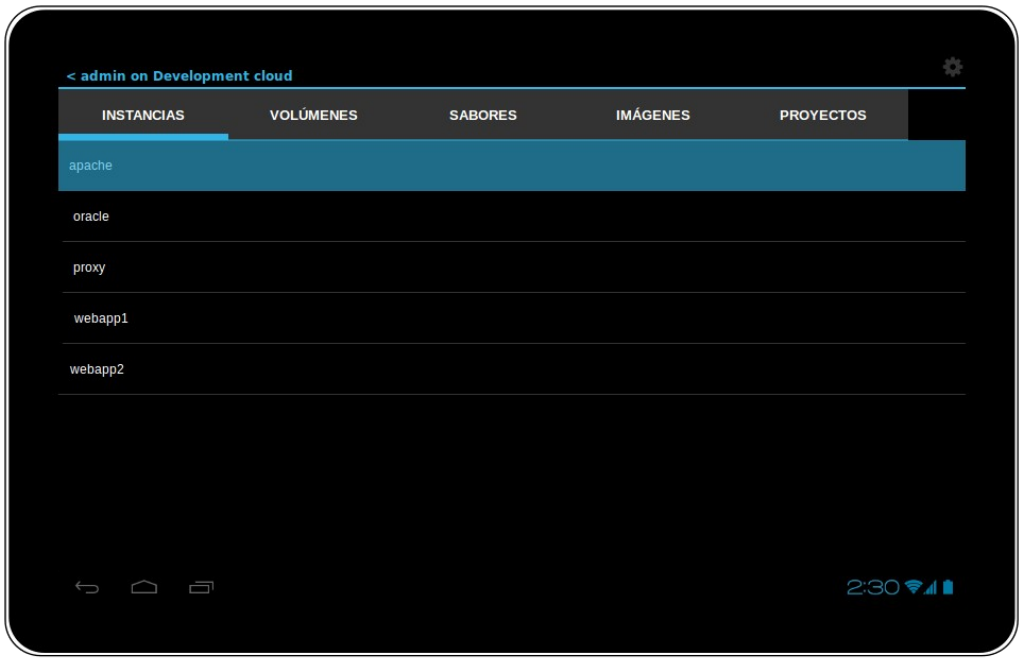
- La barra de acciones muestra en la parte izquierda el nombre del perfil seleccionado. También contiene un botón de atrás para cerrar sesión y volver a la pantalla de inicio
- Debajo de la barra de acciones, los distintos recursos de Openstack aparecen como pestañas. El usuario tan solo debe pulsar la pestaña correspondiente para visualizar los recursos deseados
- El área debajo de las pestañas muestra en forma de lista todos los recursos de Openstack que corresponden a la pestaña seleccionada

Esta interfaz de pestañas es similar a la que se puede ver en el interfaz web Horizon que trae Openstack de serie:

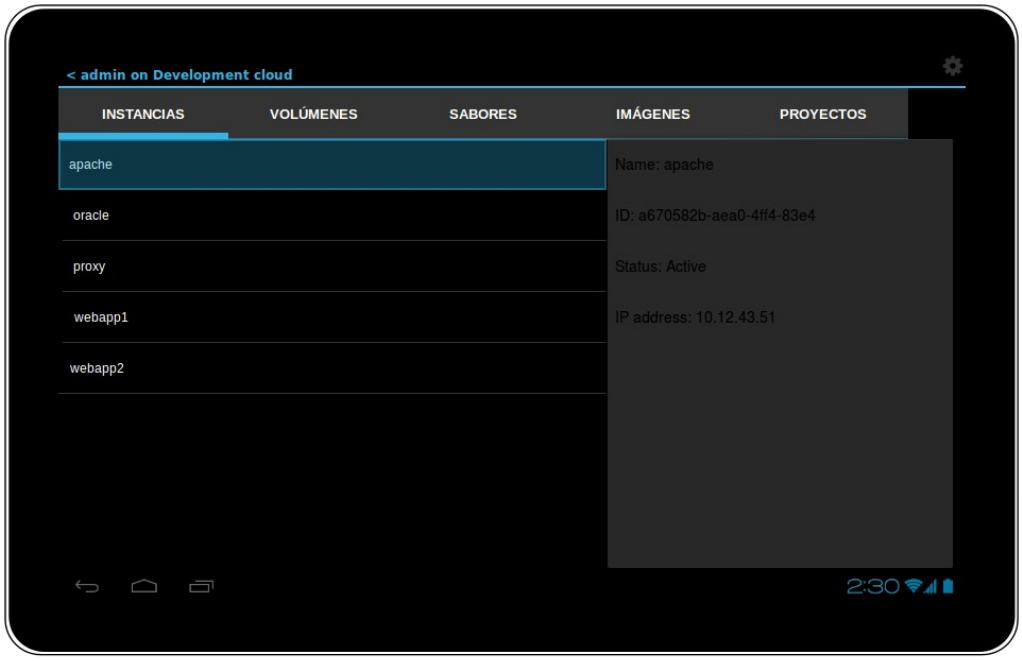


Aquí se puede ver que los distintos recursos o categorías aparecen de forma vertical en la parte izquierda.
Sin embargo esta organización no es óptima para tabletas, ya que tienen un tamaño y resolución menor que los ordenadores personales, y por tanto se ha optado por utilizar un interfaz con pestañas horizontales.

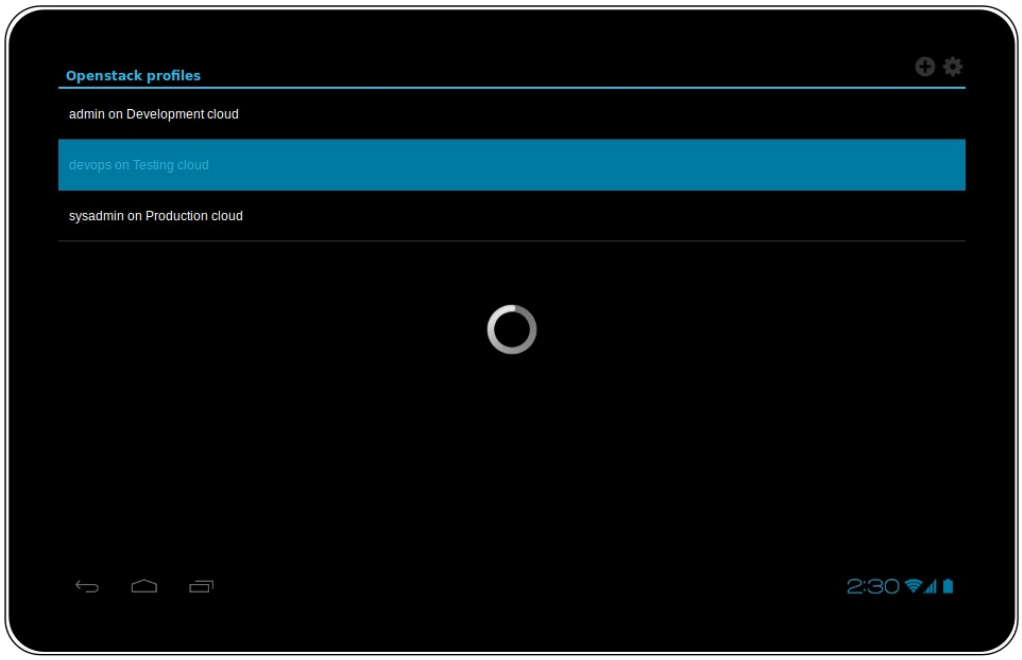
Si el usuario quisiera ver información más detallada sobre cualquiera de los recursos listados, tan solo debe pulsar en cualquiera de ellos:

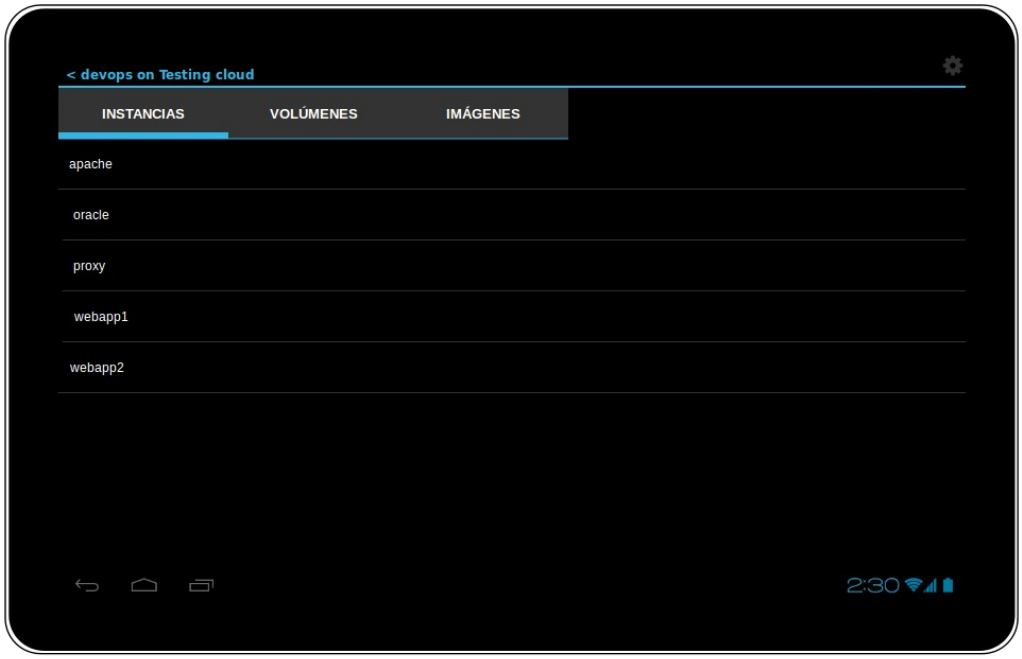


Esto provoca la aparición de una pantalla auxiliar lateral, que muestra información detallada del recurso seleccionado:



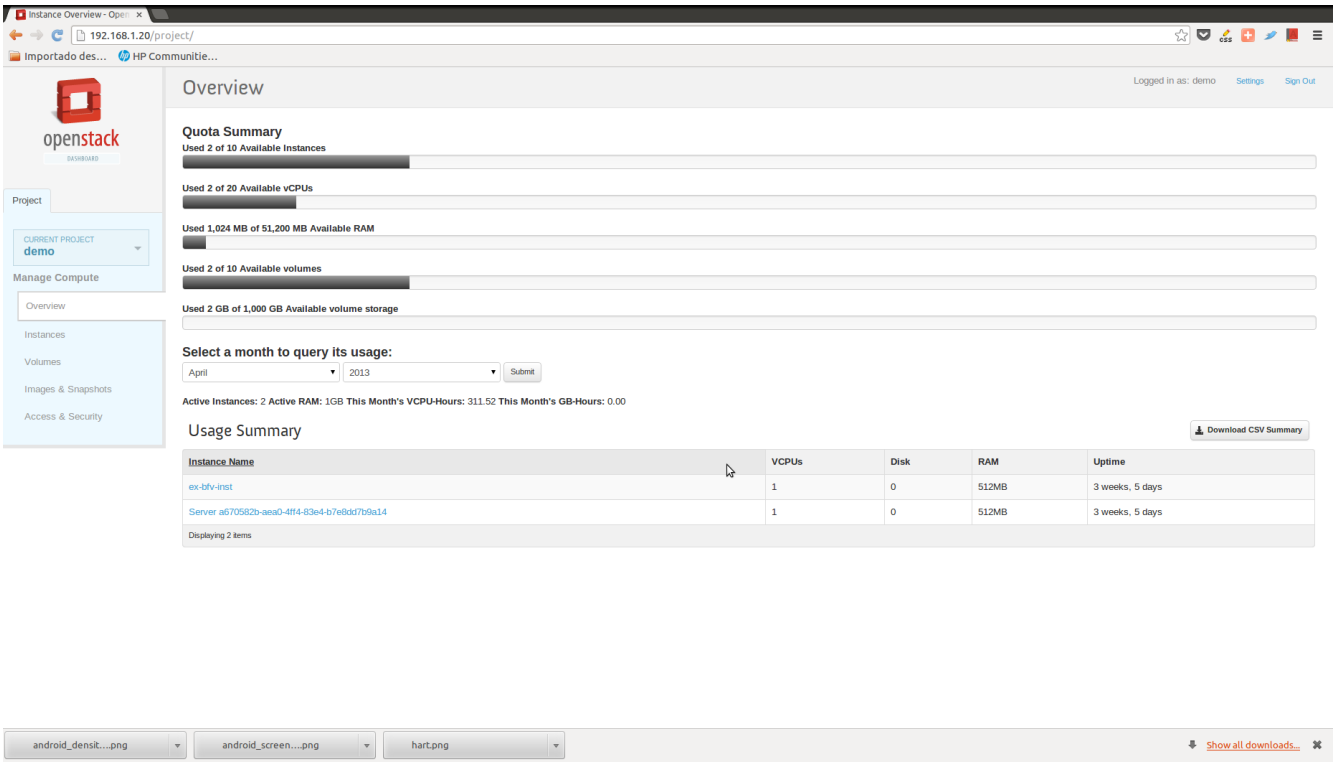
En la secuencia de pantallas anterior, se ha tomado como punto de partida que el usuario se autentica con un perfil de administrador.
Como se comentó en la sección de requisitos no funcionales, los recursos visibles y las acciones disponibles dependen de la autorización del usuario autenticado en cada momento.
Muestro a continuación un ejemplo de autenticación de usuario no administrador:





Se puede comprobar que hay pestañas que no aparecen para este usuario en comparación al ejemplo de autenticación con el usuario administrador.

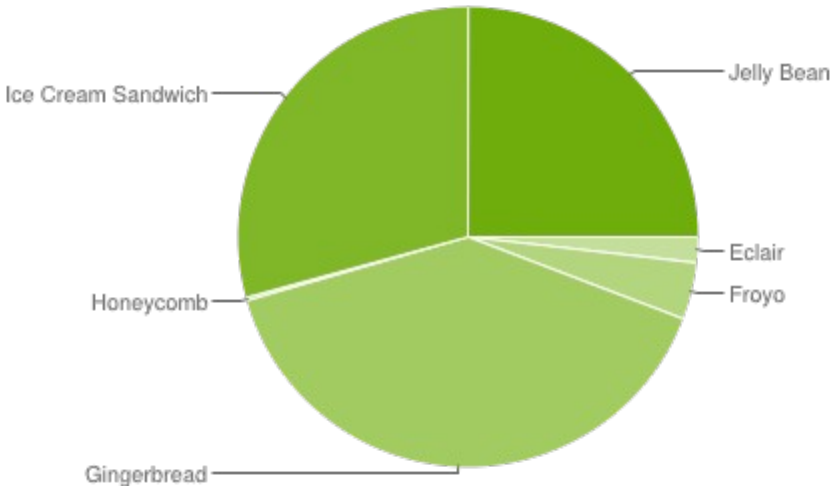
Este comportamiento también es similar al de la aplicación web Horizon, como muestra los recursos mostrados para un usuario no administrador en la siguiente imagen:


Instance Overview - Open
192.168.1.20/project/
Importado des...
HP Communitie...
openstack
DASHBOARD
Project
CURRENT PROJECT
demo
Manage Compute
Overview
Instances
Volumes
Images & Snapshots
Access & Security
Overview
Logged in as: demo
Settings
Sign Out
Quota Summary
Used 2 of 10 Available Instances
Used 2 of 20 Available vCPUs
Used 1,024 MB of 51,200 MB Available RAM
Used 2 of 10 Available volumes
Used 2 GB of 1,000 GB Available volume storage
Select a month to query its usage:
April
2013
Submit
Active Instances: 2 Active RAM: 1GB This Month's VCPU-Hours: 311.52 This Month's GB-Hours: 0.00
Usage Summary
Download CSV Summary
Instance Name
VCPUs
Disk
RAM
Uptime
ex-bfv-inst
1
0
512MB
3 weeks, 5 days
Server a670582b-aea0-4ff4-83e4-b7e8dd7b9a14
1
0
512MB
3 weeks, 5 days
Displaying 2 items
android_densit....png
android_screen....png
hart.png
Show all downloads...

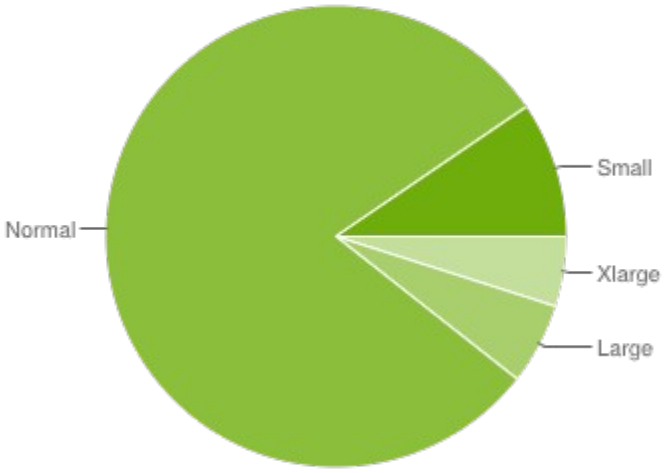
Fragmentación de diferentes versiones de Android

Uno de los problemas más reconocidos en el ecosistema Android es la fragmentación de las diversas versiones.
Google, empresa propietaria y desarrolladora del sistema operativo Android, está constantemente desarrollando nuevas funcionalidades de su sistema operativo. Hasta a día de hoy, ha ido liberando nuevas versiones de Android cada 6 meses más o menos.
El problema es que con nuevas versiones se requieren nuevos requerimientos hardware en los dispositivos, haciendo a veces imposible actualizar el sistema operativo de los dispositovos para estar a la última.
A este hecho también se une los intereses comerciales de los diferentes fabricantes y sus políticas de actualización. Algunos de ellos son reticentes a actualizar de versión sus dispositivos para incrementar las ventas, y hacen coincidir su calendario de lanzamientos de nuevos productos con las fechas de lanzamiento de nuevas versiones de Android.

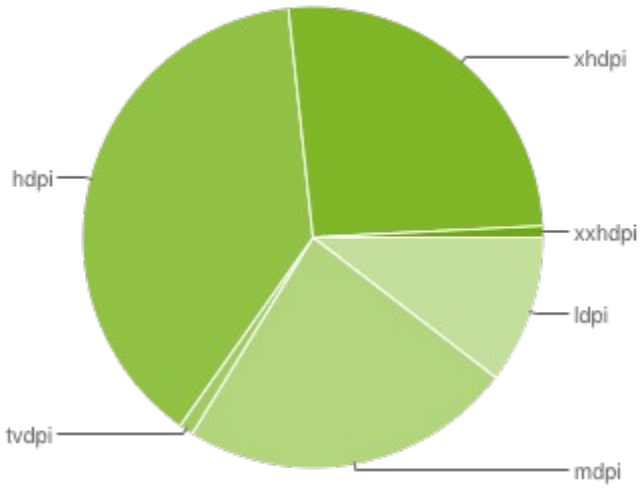
En fecha de elaboración de este proyecto, este gráfico muestra la distribución mundial (o fragmentación) de versiones en dispositivos con sistema operativo Android:



Este gráfico muestra la fragmentación de tamaños de los dispositivos que utilizan Android:



Finalmente, la fragmentación de resoluciones que utilizan los dispositivos:



Rango de dispositivos soportados

La aplicación está diseñada y será testeada para tabletas de cualquier resolución de las que se pueden encontrar habitualmente en el mercado.

La versión mínima es Android 4.0 ICS, ya que en breve será la versión más popular como se ve en los diagramas del apartado anterior y es la primera versión que soporta tanto tabletas como smartphones.

La orientación será en horizontal, para un mayor aprovechamiento del espacio de pantalla.

Si bien no se testea en móviles, el interfaz se implementará con Fragments, con lo cual se abre la puerta a que se pueda utilizar en dispositivos con pantallas más pequeñas como Smartphones.
Esta funcionalidad es en teoría factible con la utilización de la Android Support Library, que añade compatibilidad de Fragments en versiones antiguas:

http://developer.android.com/tools/extras/support-library.html

Futuras mejoras

A continuación, detallo una lista de mejoras que se podrían hacer al proyecto en el futuro después entregar el proyecto:

- Autenticación mediante HTTPS y/o Oauth
- Operaciones de inicio,parada y suspensión de instancias
- Visualización de logs de instancias
- Conexión por SSH a instancias en ejecución, mediante aplicaciones de terceros (ConnectBot o JuiceSSH)
- Operaciones de creación de objetos Openstack (subida de imágenes, creación de sabores, etc)
- Operaciones de destrucción de objetos Openstack (borrado de volúmenes, sabores, etc)
- Filtrar resultados de las listas de recursos de Openstack mediante un campo de búsqueda
- Integración con software de terceros para configuración y manejo de software (Chef o Puppet)
- Soporte de smartphones y otros dispositivos Android

Implementación

Actividad LoginActivity

Esta es la actividad que se muestra cuando se lanza la aplicación.
Se encarga de mostrar la lista de perfiles de conexión, manejar las distintas acciones que el usuario puede realizar sobre éstos perfiles (añadir perfil, editar perfil, clonar perfil, borrar perfil) y autenticar con Openstack cuando el usuario selecciona un perfil de conexión.

La cabecera de la declaración de la clase es:

```
public class LoginActivity extends ListActivity implements Receiver,
LoaderManager.LoaderCallbacks<Cursor>
```

Como podemos ver, la clase hereda de *ListActivity*. Esta clase es un especialización de *Activity* que contiene por defecto un componente *ListView*. Este componente gráfico muestra una lista de elementos de forma vertical, uno encima de otro, y se puede *scrollear* en caso de que la lista de elementos sea mayor al tamaño de la pantalla.

La clase implementa los interfaces *Receiver* y *LoaderManager.LoaderCallbacks<Cursor>*.
El interfaz *Receiver* está definido en la clase propia *CloudControllerResultReceiver*. *LoginActivity* tiene una variable miembro de este tipo, y su utilidad es recibir los mensajes de vuelta del servicio *CloudControllerService* que veremos más adelante.
El interfaz *LoaderManager.LoaderCallbacks<Cursor>* es un interfaz callback que implemento debido a que la clase *LoginActivity* maneja los datos de los perfiles de conexión mediante un *ContentProvider,* y la carga y control de modificación de los mismos en el *ListView* que los muestra con un *Loader*. Si, debido algún cambio en los datos de los perfiles de conexión (creación de nuevos perfiles, cambios de nombre de perfiles, borrado de perfiles) se necesita un cambio de visualización en el *ListView*, se invocan los métodos del interfaz callback *LoaderManager.LoaderCallbacks*.

Cuando se crea la actividad, se invoca el método *onCreate*:

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        final ListView lv = getListView();
        lv.setChoiceMode(ListView.CHOICE_MODE_SINGLE);

        lv.setOnItemLongClickListener(new OnItemLongClickListener() {

            @Override
            public boolean onItemLongClick(AdapterView<?> parent, View view, int
position, long id) {

                  if (mActionMode != null) {
                        return false;
                  }
                  selectedItemId = id;

                  mActionMode =
LoginActivity.this.startActionMode(mActionModeCallback);
```

```
                view.setSelected(true);
                lv.setItemChecked(position, true);

                return true;
            }
        });

        mReceiver = new CloudControllerResultReceiver(new Handler());
        mReceiver.setReceiver(this);

        populateProfiles();
    }
```

En este método se inicializan variables, se define un *listener* para el evento de pulsación prolongada y se llama al método *populateProfiles* para cargar en el *ListView* los perfiles de conexión de la base de datos:

```
      private void populateProfiles() {
          String[] from = new String[]
{ ConnectionProfileTable.COLUMN_PROFILE_NAME };
          int[] to = new int[] { R.id.profile_name };

          getLoaderManager().initLoader(0, null, this);
          adapter = new SimpleCursorAdapter(this, R.layout.profile_list_item,
null, from, to, 0);

          setListAdapter(adapter);
      }
```

El método *populateProfiles* inicializa el *Loader,* crea un *SimpleCursorAdapter* que mapea la columna de nombres de base de datos de perfiles de conexión con el valor mostrado en el *ListView,* para finalmente utilizarlo llamando a *setListAdapter(adapter)*.
En este punto, el *Loader* carga de forma asíncrona todos los perfiles de conexión disponibles en la base de datos y la aplicación queda esperando a eventos de usuario.
Dependiendo de estos eventos, se ejecutarán distintas acciones:

- Si el usuario pulsa un perfil de conexión cualquiera, se invoca el método *onListItemClick.* Se obtiene los datos del perfil de conexión asociado a la entrada pulsada mediante un *Cursor* asociado a nuestro *ContentProvider* de perfiles de conexión. Se crea un *intent* para autenticar con Openstack y obtener un token. Este *intent* será tratado por *CloudControllerService*, y se pasan como parámetros el tipo de operación (GetTokenOperation), el *ResultReceiver* que recibirá la respuesta, usuario, password, y tenant. En definitiva, todo lo necesario para que nuestro servicio pueda comunicarse con Openstack y nos devuelva los resultados
- Si el usuario crea el botón “ADD”, se llama al método *addConnectionProfile,* que simplemente crea un *intent* para lanzar la actividad *ConnectionProfileActivity*
- Si el usuario realiza una pulsación larga, se activa la barra de modo contextual. En este método, se pueden editar, borrar o clonar el perfil que se seleccione con tan solo pulsar los botones “EDIT”, “DELETE”, “CLONE” respectivamente.

Después de que el usuario pulsa un perfil de conexión de la lista de perfiles para autenticarse y se realizan todas las acciones detalladas anteriormente, *LoginActivity* queda a la espera de que *CloudControllerService* le envíe la respuesta.

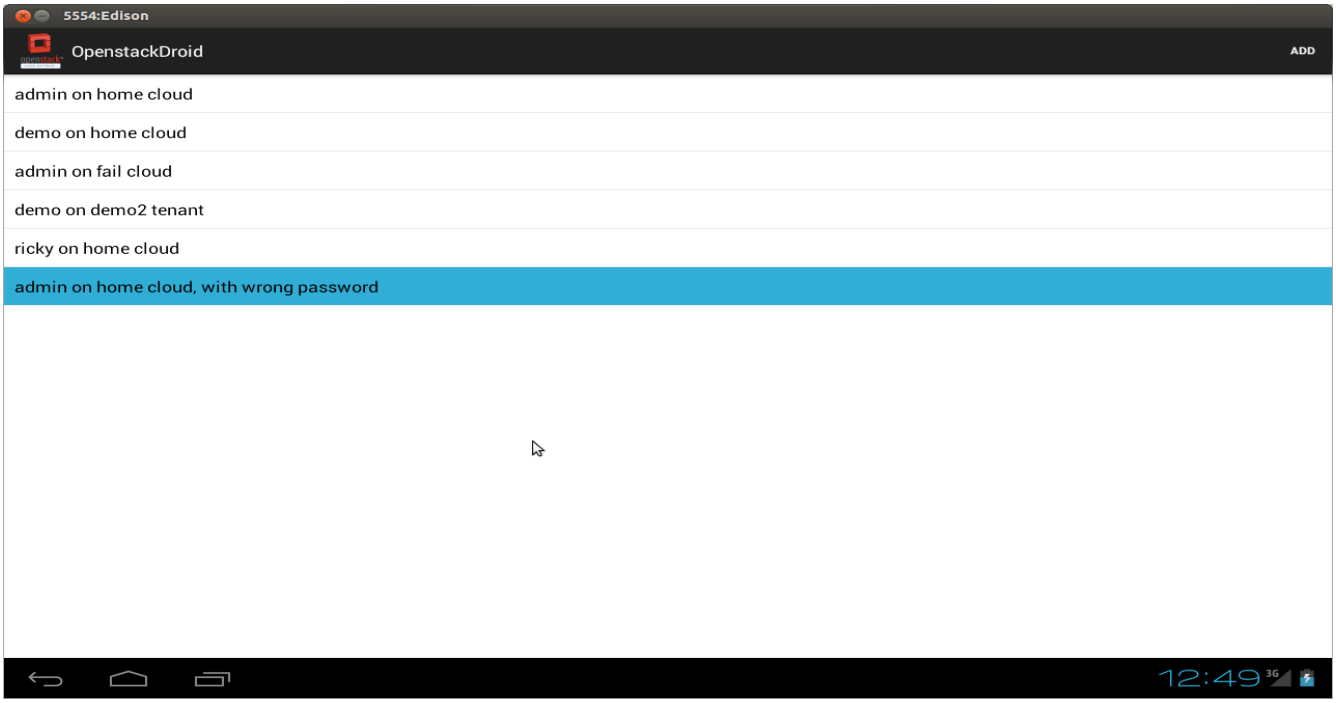
Cuando esto ocurre, se invoca el método *onReceiveResult*:

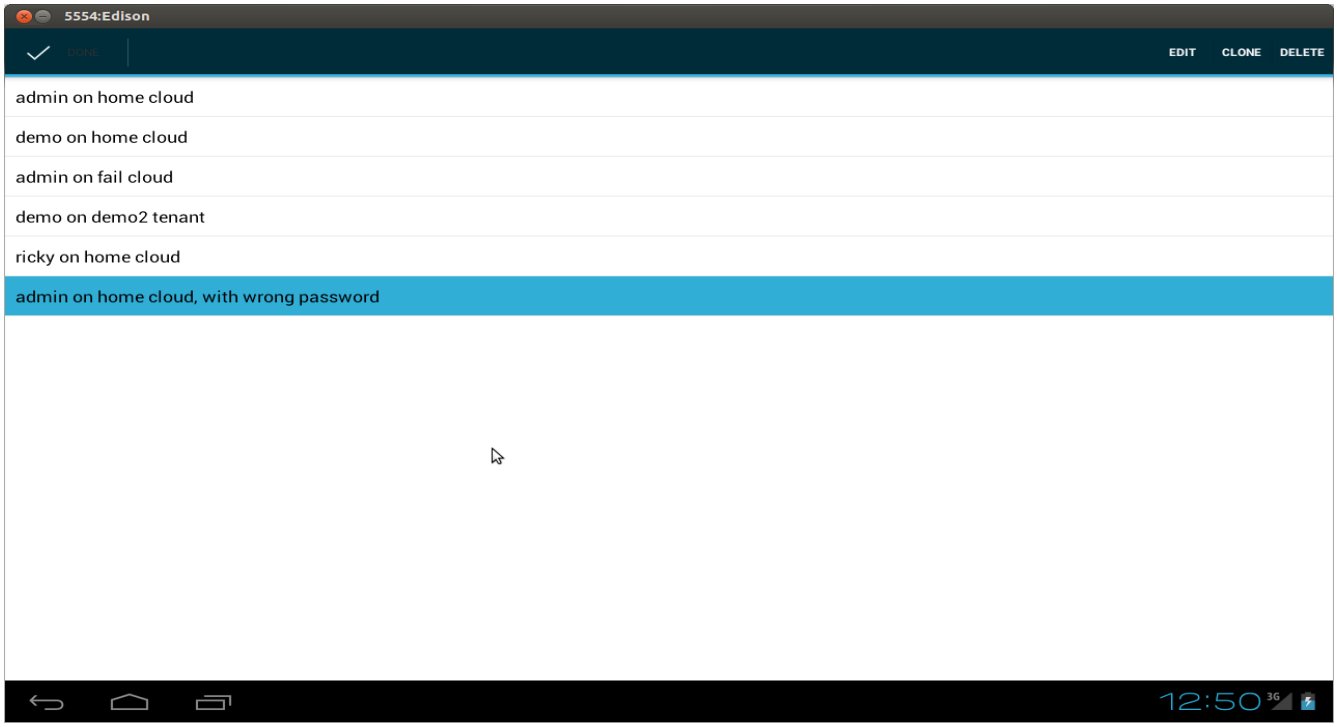
```
public void onReceiveResult(int resultCode, Bundle resultData) {
            progressDialog.dismiss();

            if (resultCode == 200) {
                  Gson gson = new Gson();
                  GetTokenResponse gtr =
gson.fromJson(resultData.getString(CloudControllerService.OPERATION_RESULTS),
GetTokenResponse.class);
                  OpenstackdroidApplication application =
(OpenstackdroidApplication) getApplication();
                  List<ServiceCatalogObject> sc =
gtr.getAccess().getServiceCatalog();

                  application.setAdminUser(false);
                  Iterator<RoleObject> it =
gtr.getAccess().getUser().getRoles().iterator();

                  while (it.hasNext()) {
                        if (it.next().getName().equals("admin"))
                              application.setAdminUser(true);
                  }

                  application.setToken(gtr.getAccess().getToken().getId());
                  application.setTenantId(gtr.getAccess().getToken().getTenant().ge
tId());
                  application.setComputeEndpoint(getEndpointByType(sc,
COMPUTE_ENDPOINT).getPublicURL());
                  application.setVolumeEndpoint(getEndpointByType(sc,
VOLUME_ENDPOINT).getPublicURL());

                  EndpointObject ieo = getEndpointByType(sc, IDENTITY_ENDPOINT);
                  application.setIdentityEndpoint(ieo.getPublicURL());
                  application.setIdentityAdminEndpoint(ieo.getAdminURL());

                  application.setImageEndpoint(getEndpointByType(sc,
IMAGE_ENDPOINT).getPublicURL());

            Intent browserIntent = new Intent(this, CloudBrowserActivity.class);
            startActivity(browserIntent);
            } else {
                  Toast.makeText(this, "Error " + resultCode + ": " +
resultData.getString(CloudControllerService.OPERATION_RESULTS) ,
Toast.LENGTH_LONG).show();
            }
      }
```

En este método, se compara el valor de la variable del método *resultCode* con el valor 200 (en HTTP, 200 es el código definido que se devuelve cuando una petición HTTP se ejecuta correctamente).
Si el resultado es igual a 200, deserializamos el cuerpo de la respuesta de representación JSON a un objeto de tipo *GetTokenResponse* y extraemos de este objeto el token, datos de usuario y la información de los distintos endpoints (URLs de los distintos APIs de Openstack). Se guardan estos datos en *OpenstackdroidApplication* para usarlos posteriormente y se lanza la actividad *CloudBrowserActivity*.
En caso de que el *resultCode* devuelto no sea 200, se muestra una notificación tipo *Toast* que muestre

el código de error contenido en la respuesta.

A continuación, un par de capturas de pantalla de *LoginActivity* mostrando el inicio y el evento de pulsación larga respectivamente:





Actividad ConnectionProfileActivity

Esta actividad se encarga de mostrar en pantalla un formulario con los datos necesarios para crear o editar un perfil de conexión, leer los datos introducidos por el usuario y guardarlos en la base de datos.

Los datos necesarios para un perfil de conexión son:

1. Nombre de perfil
2. URL de conexión a Openstack (se sugiere el formato http://<IP de Openstack Keystone>:5000)
3. Usuario
4. Password
5. ID de tenant (o proyecto, son sinónimos)

Durante la creación de esta actividad, el método *onCreate* guarda en variables privadas los diferentes componentes de texto *EditText* del layout, para posteriormente referenciarlos y obtener los datos introducidos por el usuario:

```
	protected void onCreate(Bundle savedInstanceState) {
		// TODO Auto-generated method stub
		super.onCreate(savedInstanceState);
		setContentView(R.layout.connection_profile_activity);

		mProfileName = (EditText) findViewById(R.id.profile_name_edittext);
		mEndpoint = (EditText) findViewById(R.id.endpoint_edittext);
		mUsername = (EditText) findViewById(R.id.username_edittext);
		mPassword = (EditText) findViewById(R.id.password_edittext);
		mTenantId = (EditText) findViewById(R.id.tenant_id_edittext);

		Bundle extras = getIntent().getExtras();

		if (extras == null) {
			connectionProfileUri = null;
		} else {
			connectionProfileUri =
extras.getParcelable(ConnectionProfileContentProvider.CONTENT_ITEM_TYPE);
			populateForm(connectionProfileUri);
		}
	}
```

También se comprueba que en los extras del *intent* exista una variable *connectionProfileUri.* Si existe, entonces sabemos que el usuario ha pulsado en el botón de editar perfil de conexión, en vez de añadir perfil, y se llama al método *populateForm* pasando el valor de *connectionProfileUri.* Este método obtiene los datos del perfil de conexión a editar del *ContentProvider* y rellena los distintos campos del formulario.

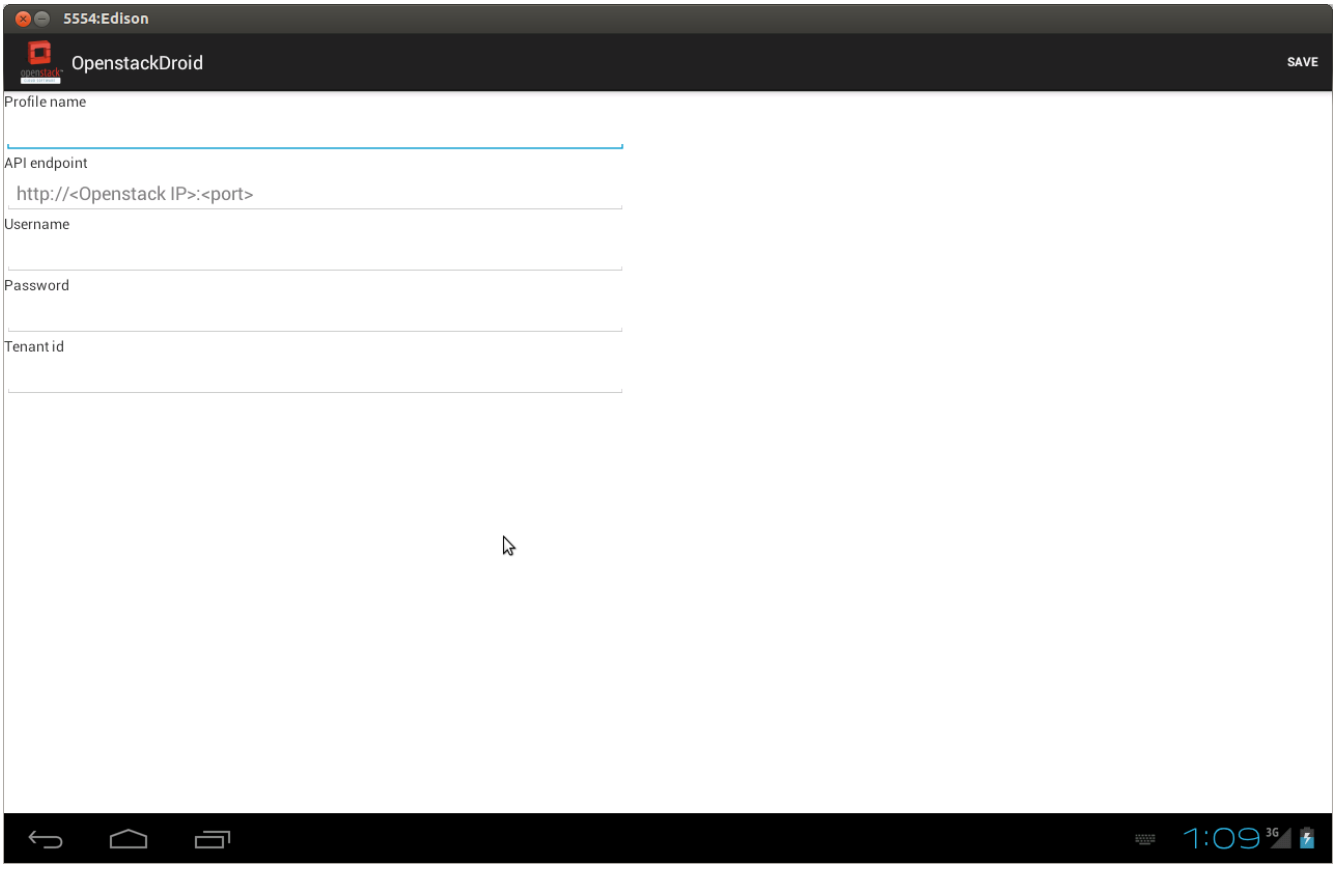
En este punto, el usuario puede introducir los datos en los distintos campos del formulario, y cuando pulsa el botón del menú de opciones “SAVE”, se invoca el método *saveConnectionProfile*:

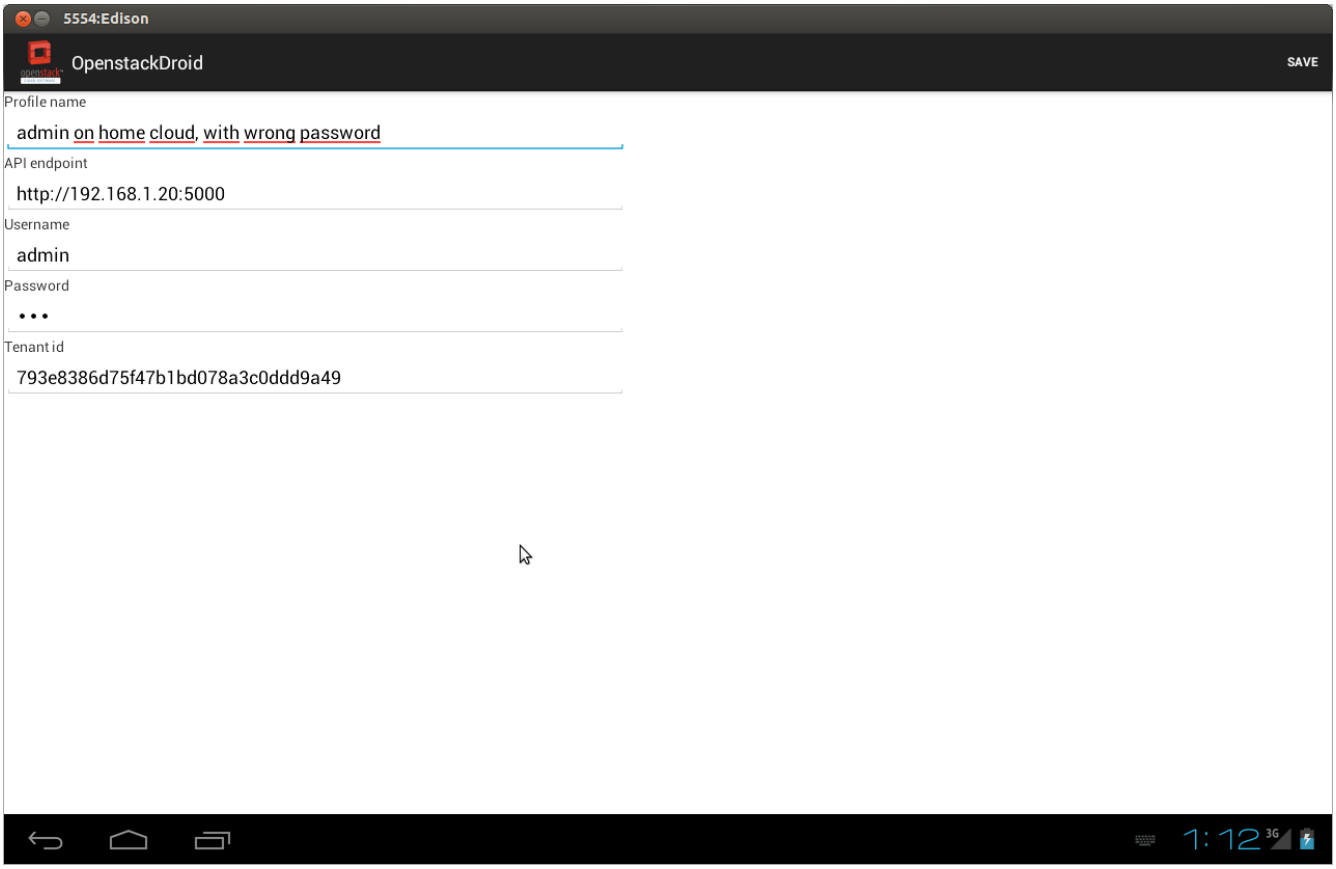
```
	private void saveConnectionProfile() {
		// TODO Auto-generated method stub
		String profileName = (String) mProfileName.getText().toString();
		String endpoint = mEndpoint.getText().toString();
		String username = mUsername.getText().toString();
		String password = mPassword.getText().toString();
```

```
            String tenantId = mTenantId.getText().toString();

            ContentValues values = new ContentValues();
            values.put(ConnectionProfileTable.COLUMN_PROFILE_NAME, profileName);
            values.put(ConnectionProfileTable.COLUMN_ENDPOINT, endpoint);
            values.put(ConnectionProfileTable.COLUMN_USERNAME, username);
            values.put(ConnectionProfileTable.COLUMN_PASSWORD, password);
            values.put(ConnectionProfileTable.COLUMN_TENANT_ID, tenantId);

            if (connectionProfileUri == null) {
                  connectionProfileUri =
getContentResolver().insert(ConnectionProfileContentProvider.CONTENT_URI, values);

            } else {
                  getContentResolver().update(connectionProfileUri, values, null,
null);
            }

            finish();
      }
```

Este método simplemente lee los valores de los distintos componentes de texto *EditText* y los guarda en la base datos utilizando los métodos de nuestro *ContentProvider.*

A continuación, muestro un par de capturas de pantalla de *ConnectionProfileActivity* mostrando las acciones de añadir perfil y editar perfil respectivamente*:*




5554:Edison
OpenstackDroid
SAVE
Profile name
admin on home cloud, with wrong password
API endpoint
http://192.168.1.20:5000
Username
admin
Password
Tenant id
793e8386d75f47b1bd078a3c0ddd9a49
1:12

Aplicación OpenstackdroidApplication

Esta clase guarda diversa información obtenida del perfil de conexión seleccionado por el usuario en *LoginActivity*. Creando esta clase como *Application* nos permite guardar esta información de manera global para otros componentes de la aplicación, y así nos evitamos tener que pasarla mediante *intents*. La información que guarda es:

```
public class OpenstackdroidApplication extends Application {
	private String token;
	private String tenantId;
	public boolean isAdminUser;
	private String computeEndpoint;
	private String volumeEndpoint;
	private String identityEndpoint;
	private String identityAdminEndpoint;
	private String imageEndpoint;
```

Vemos que se guarda el token obtenido durante la autenticación, el identificador del *tenant* y si el usuario que se autentica es administrador o no. Según este valor, la actividad *CloudBrowserActivity* mostrará recursos que atañen solo a usuarios administradores de Openstack.
Finalmente, se guardan también las URLs de los distintos componentes de Openstack (*nova*, *cinder, keystone, glance)* que serán utilizados por la aplicación para ejecutar operaciones.

Actividad CloudBrowserActivity

Esta es la actividad principal de nuestra aplicación, la que muestra la información de los distintos objetos o recursos de Openstack, una vez nos hemos autenticado correctamente.

La cabecera de la declaración de la clase es:

```
public class CloudBrowserActivity extends Activity implements TabListener
```

Como podemos ver, la clase implementa el interfaz *TabListener*.
Esto es debido a que *CloudBrowserActivity* utiliza pestañas o *tabs* para navegar los distintos tipos de recursos de Openstack.
Estas pestañas se muestran en la parte superior de la pantalla.
La actividad contiene como variables miembro listas de elementos de cada uno de los recursos que la aplicación puede mostrar información:

```
    private List<ServerModel> servers;
    private List<VolumeModel> volumes;
    private List<FlavorModel> flavors;
    private List<ImageModel> images;
    public List<UserModel> users;
    public List<TenantModel> tenants;
```

La definición de las clases <Recurso>Model se detallan más adelante.

Así mismo, la actividad tiene variables para referenciar los fragmentos que se encargan de mostrar la interfaz de usuario para cada uno de los recursos y el fragmento que está mostrándose en pantalla en ese momento:

```
    private Fragment mListFragmentAttached;
    private Fragment mServerListFragment;
    private Fragment mServerDetailsFragment;
    private Fragment mVolumeListFragment;
    private Fragment mVolumeDetailsFragment;
    private Fragment mFlavorListFragment;
    private Fragment mFlavorDetailsFragment;
    private Fragment mImageListFragment;
    private Fragment mImageDetailsFragment;
    private Fragment mUserListFragment;
    private Fragment mUserDetailsFragment;
    private Fragment mTenantListFragment;
    private Fragment mTenantDetailsFragment;
```

Como veremos más adelante, <Recurso>ListFragment muestra una lista de elementos de tipo <Recurso> y <Recurso>DetailsFragment muestra información detallada del elemento seleccionado de la lista de <Recurso>ListFragment.
Por defecto, *CloudBrowserActivity* solo muestra el <Recurso>ListFragment correspondiente a la pestaña de recurso seleccionada de forma que ocupe toda la pantalla, y se muestra el <Recurso>DetailsFragment ocupando la mitad de la pantalla a la derecha de <Recurso>ListFragment cuando se selecciona un elemento de la lista.

Esto se consigue con el layout utilizado por *CloudBrowserActivity* (cloud_browser.xml)*:*

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >

    <FrameLayout
        android:id="@+id/items_list"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

    <FrameLayout
        android:id="@+id/item_details"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="0"
        android:background="?android:attr/detailsElementBackground" />
</LinearLayout>
```

La técnica que utilizo para mostrar el fragmento de detalles o no, es cambiar el atributo *android:layout_weight:* si vale 1 se muestra, si vale 0 no se muestra.
Esto se puede ver en los métodos auxiliares *showDetailsLayout* y *hideDetailsLayout*:

```
      public void showDetailsLayout() {
        LayoutParams params = new LinearLayout.LayoutParams(0,
LayoutParams.MATCH_PARENT, 1.0f);
        FrameLayout fl = (FrameLayout) findViewById(R.id.item_details);
        fl.setLayoutParams(params);
        setDetailEnabled(true);
      }

      public void hideDetailsLayout() {
            LayoutParams params = new LinearLayout.LayoutParams(0,
LayoutParams.MATCH_PARENT, 0);
        FrameLayout fl = (FrameLayout) findViewById(R.id.item_details);
        fl.setLayoutParams(params);
        setDetailEnabled(false);
      }
```

La lógica de traducir acciones de usuario a llamadas de Openstack se implementa en los fragmentos <Recurso>ListFragment y <Recurso>DetailsListFragment, como veremos en las próximas secciones.

Fragmentos <Recurso>ListFragment y <RecursoDetailsFragment>

Los fragmentos <Recurso>ListFragment muestran en forma de lista los distintos recursos de Openstack que la aplicación es capaz de mostrar (instancias, volúmenes, imágenes, sabores, usuarios y proyectos). Por su parte, <Recurso>DetailsFragment muestra información detallada del recurso seleccionado en <Recurso>ListFragment.
Los distintos tipos de recursos con sus correspondientes parejas <Recurso>ListFragment/<Recurso>DetailsFragment son:

Instancias (o servidores) → *ServerListFragment/ServerDetailsFragment*
Volúmenes → *VolumeListFragment/VolumeDetailsFragment*
Imágenes → *ImageListFragment/ImageDetailsFragment*
Sabores → *FlavorListFragment/FlavorDetailsFragment*
Usuarios → *UserListFragment/UserDetailsFragment*
Proyectos (o tenants) → *TenantListFragment/TenantDetailsFragment*

Como todos estos fragmentos son similares, la explicación de la implementación la haré sobre *ServerListFragment/ServerDetailsFragment*.

Los fragmentos <Recurso>ListFragment heredan de la clase *CloudBrowserListFragment*, que encapsula métodos y variables comunes a todos ellos:

```
public abstract class CloudBrowserListFragment extends ListFragment implements
Receiver {
      protected String endpoint;
      protected int mCurCheckPosition;
      protected CloudControllerResultReceiver mReceiver;
```

Como vemos, hereda de *ListFragment* (con lo cual, contiene por defecto un *ListView*) e implementa el interfaz *Receiver*.
También contiene la variable *endpoint*, que contiene la URL que apunta al componente que se encarga de manejar el correspondiente recurso. Por ejemplo, *ServerListFragment* tendrá esta variable apuntando al componente que se encarga del manejo de instancias (*nova)*, *VolumeListFragment* apuntará al componente que se encarga de los volúmenes (*cinder)*, etc.
La variable *mReceiver* es una instancia de nuestra clase *CloudControllerResultReceiver*, cuya finalidad será recibir la respuesta de mensajes del servicio *CloudControllerService*.
Finalmente, la variable *mCurCheckPosition* guarda la posición del elemento de la lista seleccionado en cada momento. Este valor se utilizará más tarde por <Recurso>DetailsFragment para mostrar los datos asociados a la entrada seleccionada.

Cuando se crea la actividad padre *CloudBrowserActivity* y se instancia el fragmento, se ejecuta el método *onActivityCreated*:

```
            super.onActivityCreated(savedInstanceState);
            servers = ((CloudBrowserActivity) getActivity()).getServers();
        endpoint = ((OpenstackdroidApplication)
(getActivity().getApplication())).getComputeEndpoint();

        Intent serviceIntent = new Intent(getActivity(),
CloudControllerService.class);
```

```
        serviceIntent.setData(Uri.parse(endpoint));
        serviceIntent.putExtra(CloudControllerService.OPERATION,
CloudControllerService.GET_SERVERS_OPERATION);
        serviceIntent.putExtra(CloudControllerService.TOKEN,
((OpenstackdroidApplication) getActivity().getApplication()).getToken());
        serviceIntent.putExtra(CloudControllerService.TENANT,
((OpenstackdroidApplication) getActivity().getApplication()).getTenantId());
        serviceIntent.putExtra(CloudControllerService.RECEIVER, mReceiver);
        Bundle params = new Bundle();
        serviceIntent.putExtra(CloudControllerService.PARAMS, params);

        getActivity().startService(serviceIntent);

        adapter = new ArrayAdapter<ServerModel>(getActivity(),
android.R.layout.simple_list_item_activated_1, servers);
        setListAdapter(adapter);
        getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);
```

Este método obtiene la lista que modela los servidores de la actividad padre *CloudBrowserActivity*. También, inicializa la variable *endpoint* para que apunte al componente *nova*. Este valor se obtiene de la aplicación *OpenstackdroidApplication*, que contiene valores inicializados por *LoginActivity* y relativos al perfil de conexión utilizado.
Luego, se crea un *intent* para ejecutar la operación de obtener servidores y se inicia el servicio *CloudControllerService* con este *intent*.

Cuando se recibe el resultado de la operación de *CloudControllerService*, se invoca el método *onReceiveResult*:

```
      public void onReceiveResult(int resultCode, Bundle resultData) {
            // TODO Auto-generated method stub
            if (resultCode == 200) {
                  String operation =
resultData.getString(CloudControllerService.OPERATION);

                  if
(operation.equals(CloudControllerService.GET_SERVERS_OPERATION)) {
                        Gson gson = new Gson();
                        GetServersResponse gsr =
gson.fromJson(resultData.getString(CloudControllerService.OPERATION_RESULTS),
GetServersResponse.class);

                        populateItems(gsr);
                  }
            }
      }
```

Si el resultado devuelto es correcto, se rellena la lista del fragmento con los valores contenidos en el *resultData* devuelto.

En caso de que el usuario toque cualquier elemento, se ejecuta el método *onListItemClick:*

```
    public void onListItemClick(ListView l, View v, int position, long id) {
      // TODO Auto-generated method stub
            mCurCheckPosition = position;
```

```
            getListView().setItemChecked(position, true);

            showDetails(position);
    }

      protected void showDetails(int position) {
            // TODO Auto-generated method stub
            ServerDetailsFragment sdf = (ServerDetailsFragment)
((CloudBrowserActivity) getActivity()).getmServerDetailsFragment();

            if (sdf == null || sdf.getShownIndex() != position)
                  sdf = ServerDetailsFragment.newInstance(position);

            FragmentTransaction ft = getFragmentManager().beginTransaction();
        ft.replace(R.id.item_details, sdf);
        ft.commit();

          ((CloudBrowserActivity) getActivity()).showDetailsLayout();
      }
```

En éste método, se guarda en *mCurCheckPosition* la posición del elemento de la lista seleccionado y se marca el elemento como seleccionado.
Luego se llama el método auxiliar *showDetails* pasando la posición. Este método lo que hace es instanciar un fragmento (o recuperar si ya se había creado antes) *ServerDetailsFragment* que muestre la información detallada del elemento seleccionado. En este proceso de instanciación, se invoca al método *onCreateView* de *ServerDetailsFragment*:

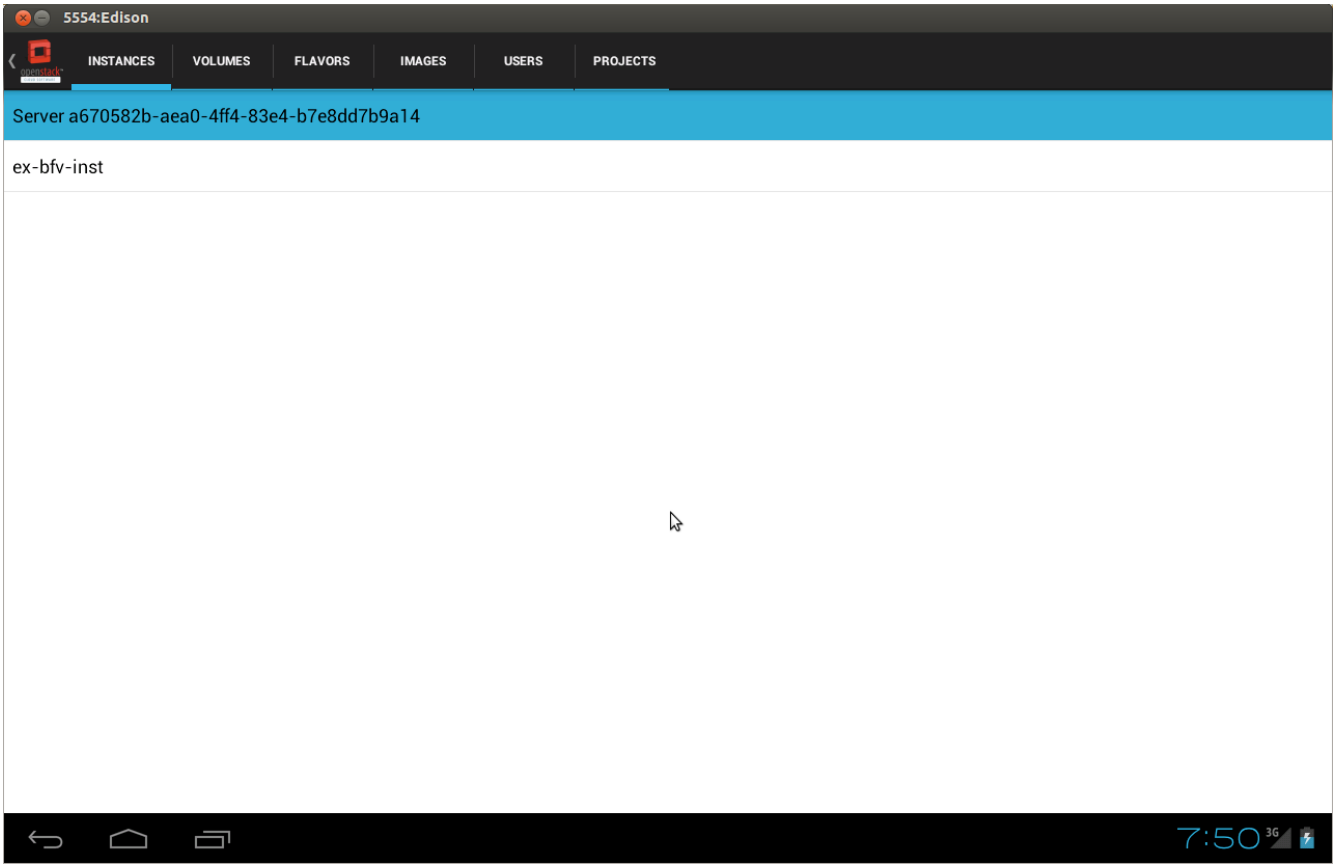
```
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {

        scroller = new ScrollView(getActivity());
        tv = new TextView(getActivity());
        int padding = (int)TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP,
                4, getActivity().getResources().getDisplayMetrics());
        tv.setPadding(padding, padding, padding, padding);
        scroller.addView(tv);
        ServerModel server =
(((CloudBrowserActivity)getActivity()).getServers()).get(getShownIndex());
        StringBuffer sb = new StringBuffer();
        sb.append("ID: " + server.getId() + "\n\n");
        sb.append("Name: " + server.getName() + "\n\n");
        sb.append("Created: " + server.getCreated() + "\n\n");
        sb.append("Updated: " + server.getUpdated() + "\n\n");
        sb.append("Status: " + server.getStatus() + "\n\n");
        sb.append("Image: " + server.getImage() + "\n\n");
        sb.append("Flavor: " + server.getFlavor() + "\n\n");
        sb.append("Private IP addresses:\n\n");

        Iterator<IPAddressModel> it = (Iterator<IPAddressModel>)
server.getPrivateAddresses().iterator();

        while (it.hasNext()) {
            sb.append(it.next().getAddr().toString() + "\n");
        }

        sb.append("\nPublic IP addresses: \n\n");
```

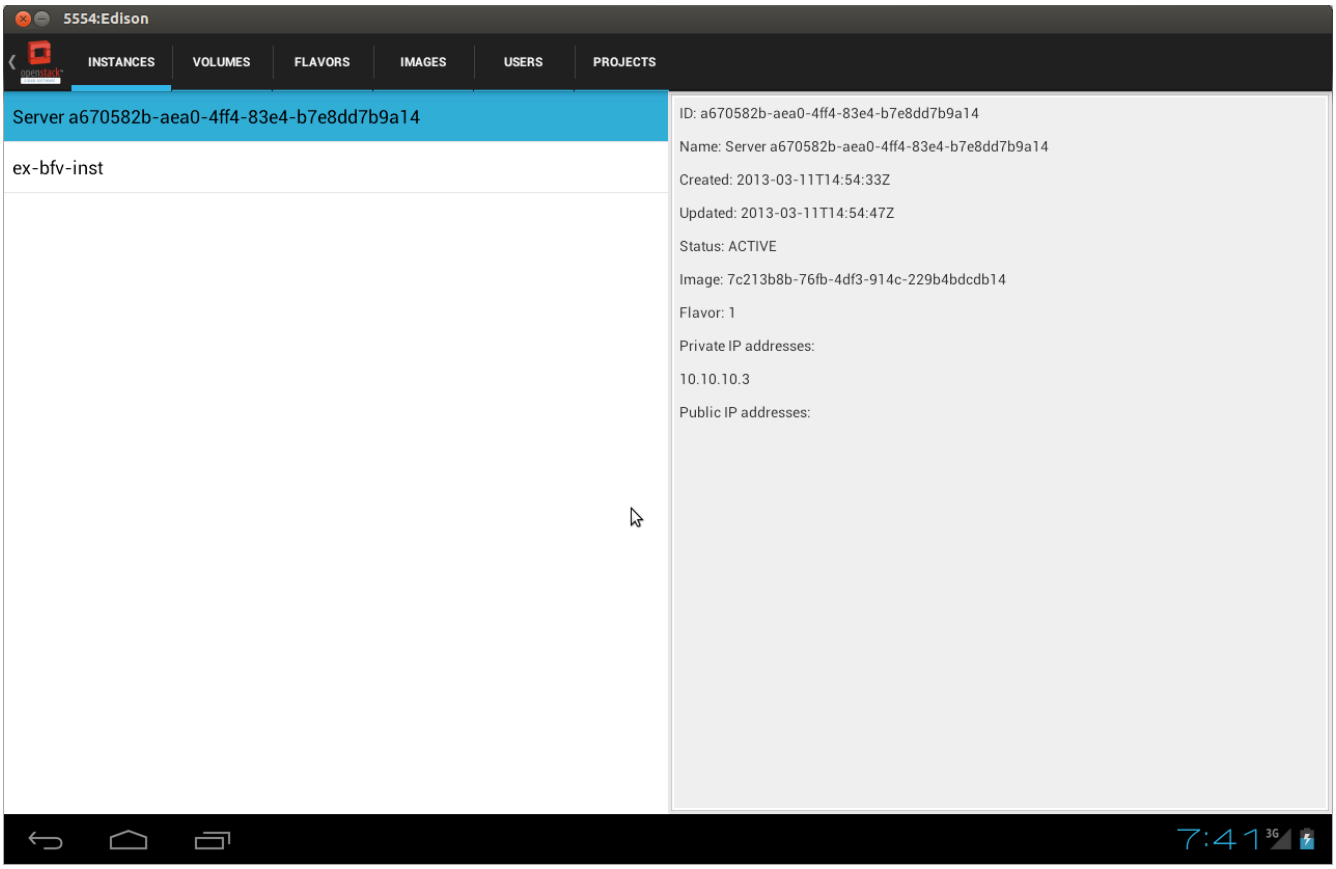
```
        it = (Iterator<IPAddressModel>) server.getPublicAddresses().iterator();

        while (it.hasNext()) {
            sb.append(it.next().getAddr().toString() + "\n");
        }

        tv.setText(sb.toString());
        return scroller;
    }
```

Este método simplemente obtiene la información del modelo correspondiente a la instancia seleccionada y la formatea en un *TextView*.

Finalmente en *ServerListFragment*, se llama al método *showDetailsLayout* de la actividad padre *CloudBrowserActivity*, que hace que se muestre en la parte derecha de la pantalla el *ServerDetailsFragment*.

Como muestra, pego dos capturas de pantalla de la aplicación mostrando la lista de servidores y otra mostrando los detalles de un servidor cualquiera seleccionado:




5554:Edison
INSTANCES
VOLUMES
FLAVORS
IMAGES
USERS
PROJECTS
Server a670582b-aea0-4ff4-83e4-b7e8dd7b9a14
ex-bfv-inst
ID: a670582b-aea0-4ff4-83e4-b7e8dd7b9a14
Name: Server a670582b-aea0-4ff4-83e4-b7e8dd7b9a14
Created: 2013-03-11T14:54:33Z
Updated: 2013-03-11T14:54:47Z
Status: ACTIVE
Image: 7c213b8b-76fb-4df3-914c-229b4bdcdb14
Flavor: 1
Private IP addresses:
10.10.10.3
Public IP addresses:
7:41

Servicio CloudControllerService y clases ApiOperation

La clase *CloudControllerService* es la que se encarga de traducir las acciones de los usuarios a llamadas de APIs de Openstack y devolver los resultados.

La cabecera de la clase es:

```
public class CloudControllerService extends IntentService {
```

Hereda de la clase *IntentService*, que es una especialización de *Service*. Esta clase es capaz de responder asíncronamente a peticiones de servicios por parte de actividades de aplicación. El mecanismo es que una actividad hace una petición llamando al método *startService* , se inicia un *worker thread* en nuestro *IntentService*, responde a la petición y el thread del servicio se termina.
El método que se invoca internamente cuando el servicio recibe es *onHandleIntent*.
Veamos cómo es la implementación del método en *CloudControllerService*:

```
	protected void onHandleIntent(Intent intent) {
		// TODO Auto-generated method stub
		Uri data = intent.getData();
		Bundle extras = intent.getExtras();

		String operation = extras.getString(OPERATION);
		String token = extras.getString(TOKEN);
		String tenantId = extras.getString(TENANT);
		ResultReceiver receiver = extras.getParcelable(RECEIVER);
		Bundle params = extras.getParcelable(PARAMS);

		ApiOperation apiOperation = getOperationInstance(operation);

		HttpRequestBase request = apiOperation.invoke(data, token, tenantId,
params);

		HttpResponse response = null;
		response = executeOperation(request);
		returnResultstoReceiver(operation, response, receiver);
	}
```

Como vemos, se extrae del *intent* diversa información pasada por la actividad que inicia el servicio. En particular, qué operación se necesita ejecutar en Openstack, el token de sesión, el identificador del *tenant*, el *ResultReceiver* para poder devolver los resultados de vuelta a la aplicación y cualquier parámetro que sea necesario para poder ejecutar la operación.

En este código también vemos que se asigna un valor de la función *getOperationInstance* a una variable de tipo *ApiOperation*.
La función *getOperationInstance* lo único que hace es traducir la cadena que representa la operación que se necesita ejecutar a su correspondiente clase, y su código es:

```
apiOperation = (ApiOperation) Class.forName(operation).newInstance();
```

Una vez tenemos la clase *ApiOperation* real que corresponde a la acción del *intent*, se llama a su método *invoke*.

La clase *ApiOperation* es un interfaz que implementan todas las clases que implementan operaciones de las APIs de Openstack, y tan solo tiene un método:

```
public interface ApiOperation {

	public HttpRequestBase invoke(Uri endpoint, String token, String tenantId,
Bundle params);

}
```

En función de la finalidad de cada una de las operaciones, las clases que implementan este interfaz darán la definición final del método *invoke*, veamos por ejemplo la clase *GetServersOperation*:

```
public class GetServersOperation implements ApiOperation {
	private static final String urlTail = "/servers/detail";

	public GetServersOperation() {
		super();
	}

	@Override
	public HttpRequestBase invoke(Uri endpoint, String token, String tenantId,
			Bundle params) {
		// TODO Auto-generated method stub
		HttpGet httpGet = new HttpGet(endpoint.toString()+urlTail);
		httpGet.setHeader("X-Auth-Token", token);

		return httpGet;
	}
```

Como vemos, esta operación implementa un método GET, y lo que hace es devolver una variable *HttpGet* que contiene la URL *http://<dirección IP y puerto de nova>/servers/detail* con cabecera *X-Auth-Token* con el valor del token de sesión.
Esto coincide con lo que dicta la guía de desarrollo del compute API de Openstack para la operación "Listar servidores":

http://docs.openstack.org/api/openstack-compute/2/content/List_Servers-d1e2078.html

Todas las clases que implementan *ApiOperation* y que, por tanto, implementan operaciones de las APIs de Openstack se encuentran en los paquetes *com.rcarrillocruz.android.openstackdroid.operations.** .

De vuelta en *CloudControllerService*, ya tenemos un objeto *HttpRequestBase* que representa la operación HTTP a ejecutar (en el caso del ejemplo *GetServersOperation* es un HTTP GET, pero podría ser un HTTP POST, HTTP DELETE o HTTP PUT. Esto lo indican las guías de desarrollo de APIs de Openstack para cada una de las operaciones).

Finalmente, se envía de vuelta los resultados de la operación HTTP al *ResultReceiver* que recibimos anteriormente del *intent*.

Representaciones JSON de peticiones y respuestas

El formato que se utiliza en la aplicación para hacer peticiones y recibir respuestas de Openstack es JSON.
Openstack también acepta XML, sin embargo he optado por usar JSON ya que es más eficiente y fácil en dispositivos móviles.
Es necesario modelar en clases tanto las peticiones como respuestas JSON, estas clases se encuentran en el módulo *com.rcarrillocruz.android.openstackdroid.json.** .
Veamos como ejemplo la representación JSON de la respuesta de la operación “Listar Servidores” (que en nuestra aplicación corresponde a *GetServersOperation)* según la guía de desarrollo de Openstack:

http://docs.openstack.org/api/openstack-compute/2/content/List_Servers-d1e2078.html

Siguiendo la documentación, creamos la clase *GetServersResponse* de manera que los atributos y la estructura coincidan:

```
public class GetServersResponse {
      List<ServerDetailsObject> servers;
```

La clase puede contener otras clases anidadas que representan a su vez estructuras internas de la respuesta JSON, en este caso *GetServersResponse*:

```
      private String id;
      private String name;
      private IPAddressesObject addresses;
      private String created;
      private String updated;
      private String status;
      private ImageObject image;
      private FlavorObject flavor;
```

Por motivos de espacio, no sigo mostrando las sucesivas clases usadas en *GetServersResponse*, para verlas en detalle se pueden ver en el código fuente que adjunto a esta memoria.

La librería que se utiliza para serializar y deserializar JSON es Google GSON.
El proceso de serializar consiste en traducir un objeto que representa una petición JSON a su correspondiente cadena en formato JSON.
A su vez, deserializar consiste en traducir de una cadena en formato JSON a su objeto de respuesta correspondiente.

Por tanto, antes de enviar las peticiones a Openstack se hacen (si procede) las pertinentes serializaciones de los objetos que representan las peticiones y se deserializan las respuestas recibidas de Openstack en sus objetos correspondientes.

Entidades <Recurso>Model

Estas clases son las que modelan los distintos recursos de Openstack que la aplicación es capaz de mostrar.
Los distintos tipos de recursos con sus correspodientes <Recurso>Model son:

Instancias (o servidores) → *ServerModel*
Volúmenes → *VolumeModel*
Imágenes → *ImageModel*
Sabores → *FlavorModel*
Usuarios → *UserModel*
Proyectos (o tenants) → *TenantModel*

Son simples entidades con los atributos propios del recurso, y se encuentran en el paquete *com.rcarrillocruz.android.openstackdroid*.

Como muestra, pego los atributos de la entidad correspondiente a las instancias, *ServerModel*:

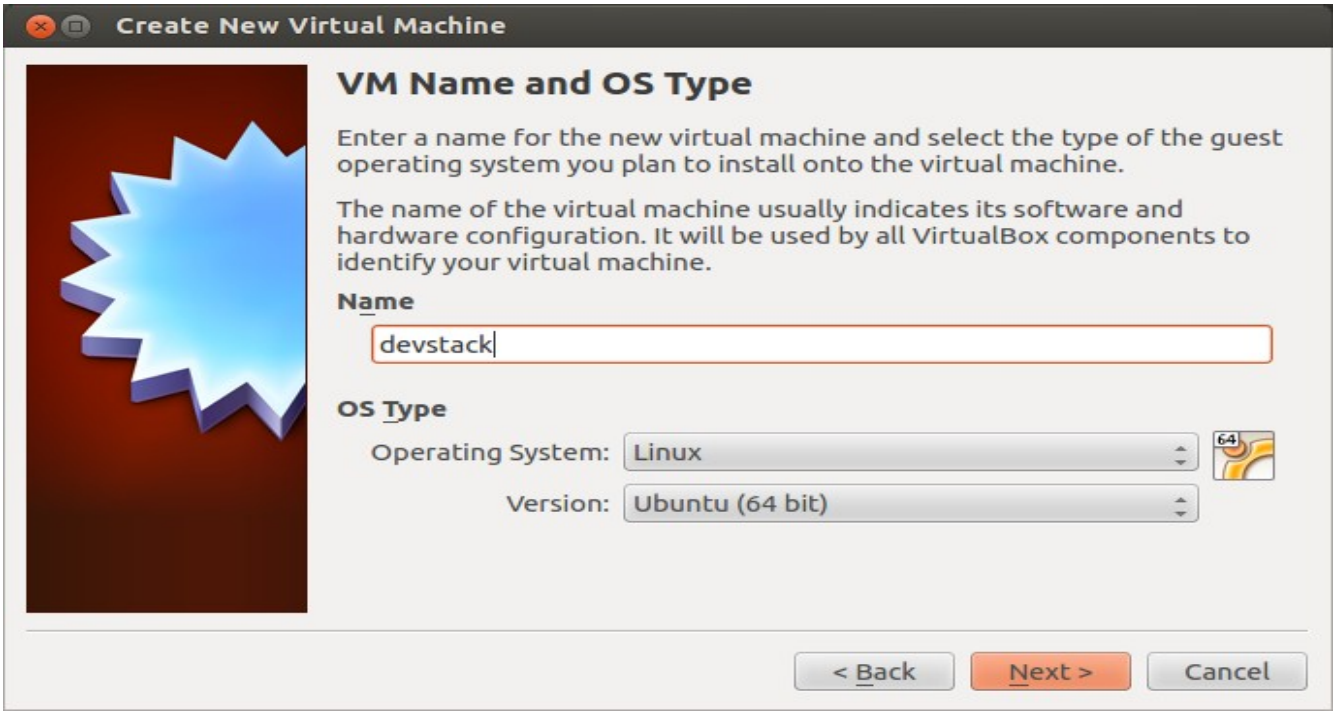
```
public class ServerModel {
	private String id;
	private String name;
	private String status;
	private String created;
	private String updated;
	List<IPAddressModel> privateAddresses;
	List<IPAddressModel> publicAddresses;
	private String image;
	private String flavor;
```

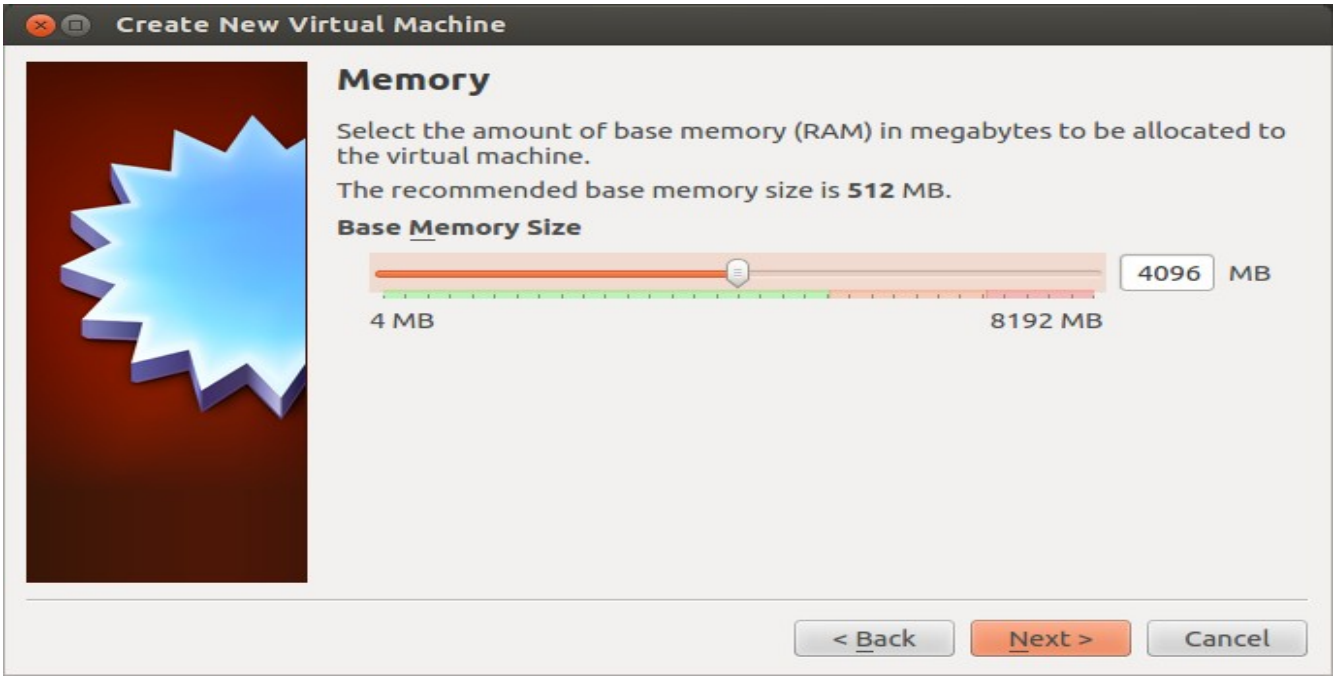
Anexo A: Instrucciones de instalación de Devstack para testear Openstackdroid

Devstack (http://devstack.org/) es un script que simplifica la instalación de Openstack en entornos de desarrollo, ya que es un software que no es sencillo de instalar.
Es por ello que en este apartado se explica como instalar Openstack versión Folsom (usada para la elaboración de este proyecto) en una máquina virtual mediante Devstack y poder probar la aplicación Openstackdroid.

A continuación, los pasos a seguir con capturas de pantalla mostrando el proceso:

1. Instalar el software Virtualbox, siguiendo las instrucciones de la página oficial https://www.virtualbox.org/wiki/Downloads (en mi caso lo instalé en Ubuntu Linux con solo ejecutar *sudo apt-get install virtualbox)*
2. Descargar Ubuntu 12.04 de http://releases.ubuntu.com/precise/
3. Crear una máquina virtual que albergará Openstack. A continuación, unas capturas que detallan los pasos a seguir y parámetros sugeridos:

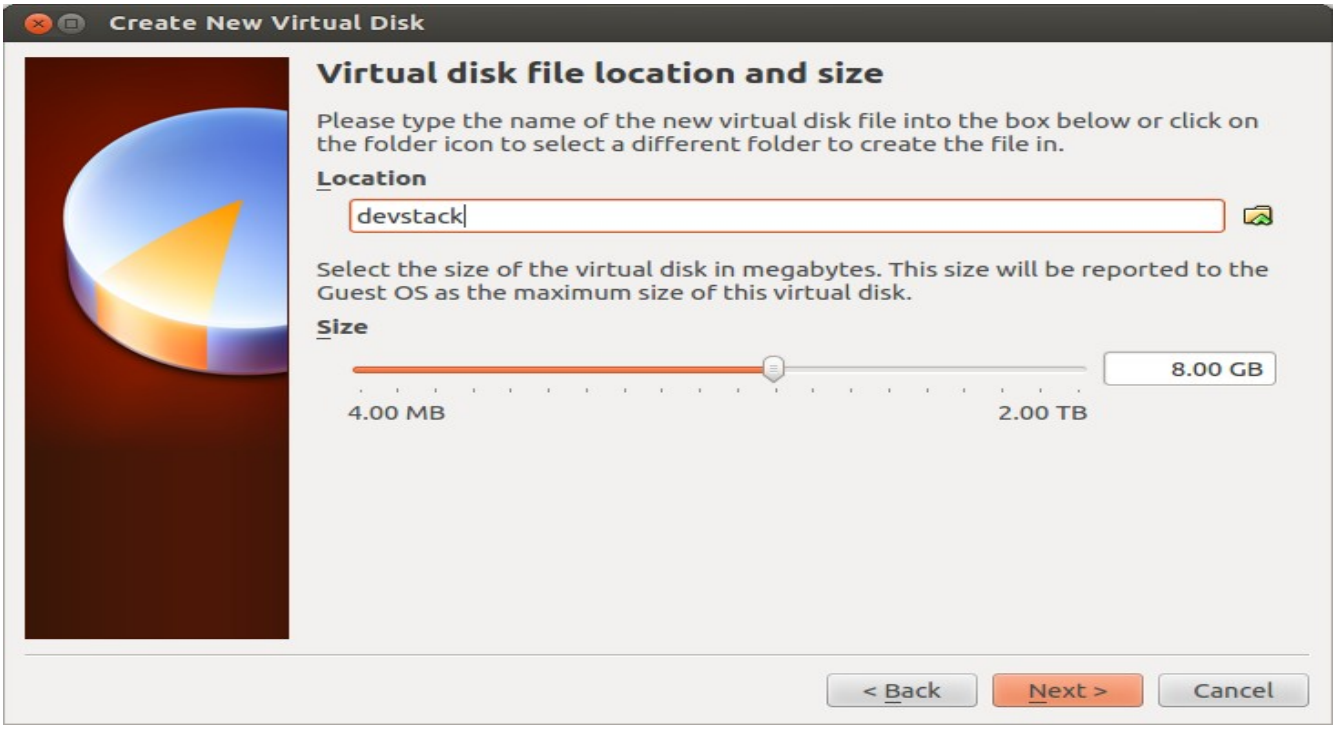


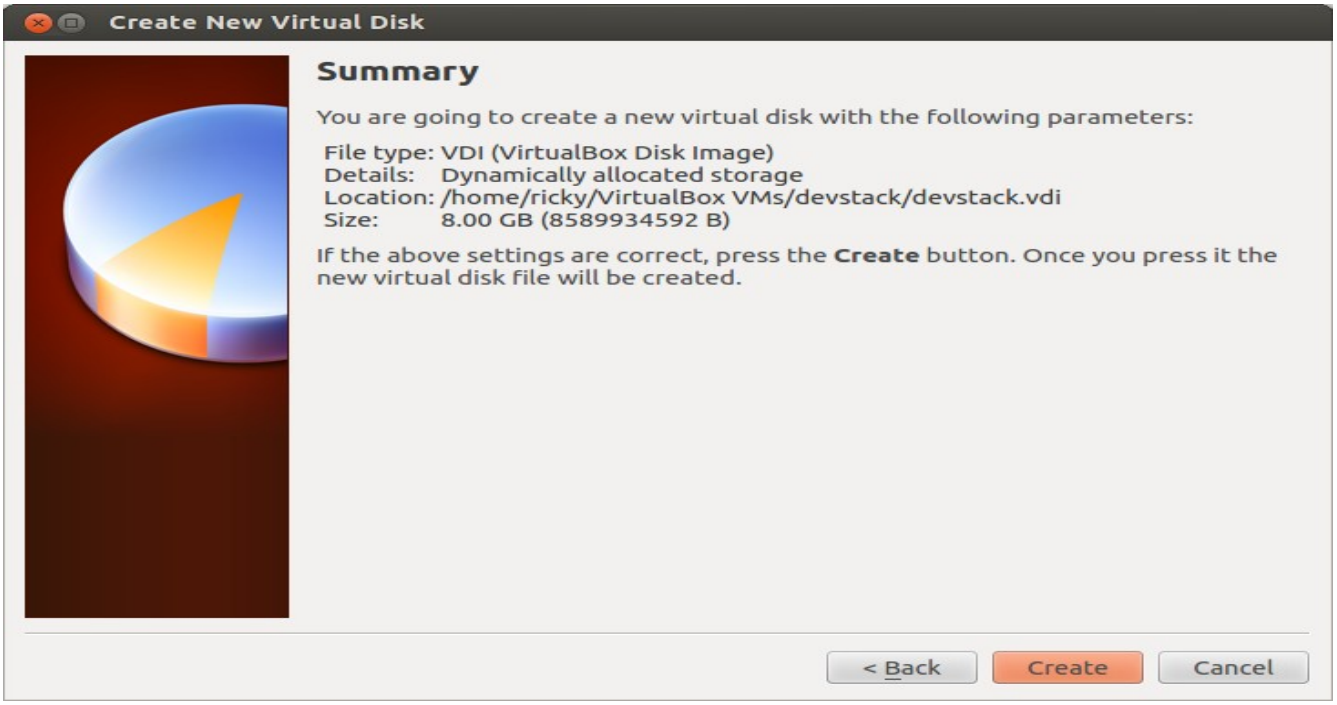

Create New Virtual Machine
Memory
Select the amount of base memory (RAM) in megabytes to be allocated to the virtual machine.
The recommended base memory size is 512 MB.
Base Memory Size
4096
MB
4 MB
8192 MB
< Back
Next >
Cancel


Create New Virtual Machine
Virtual Hard Disk
If you wish you can now add a start-up disk to the new machine. You can either create a new virtual disk or select one from the list or from another location using the folder icon.
If you need a more complex virtual disk setup you can skip this step and make the changes to the machine settings once the machine is created.
The recommended size of the start-up disk is 8.00 GB.
Start-up Disk
Create new hard disk
Use existing hard disk
Empty
< Back
Next >
Cancel

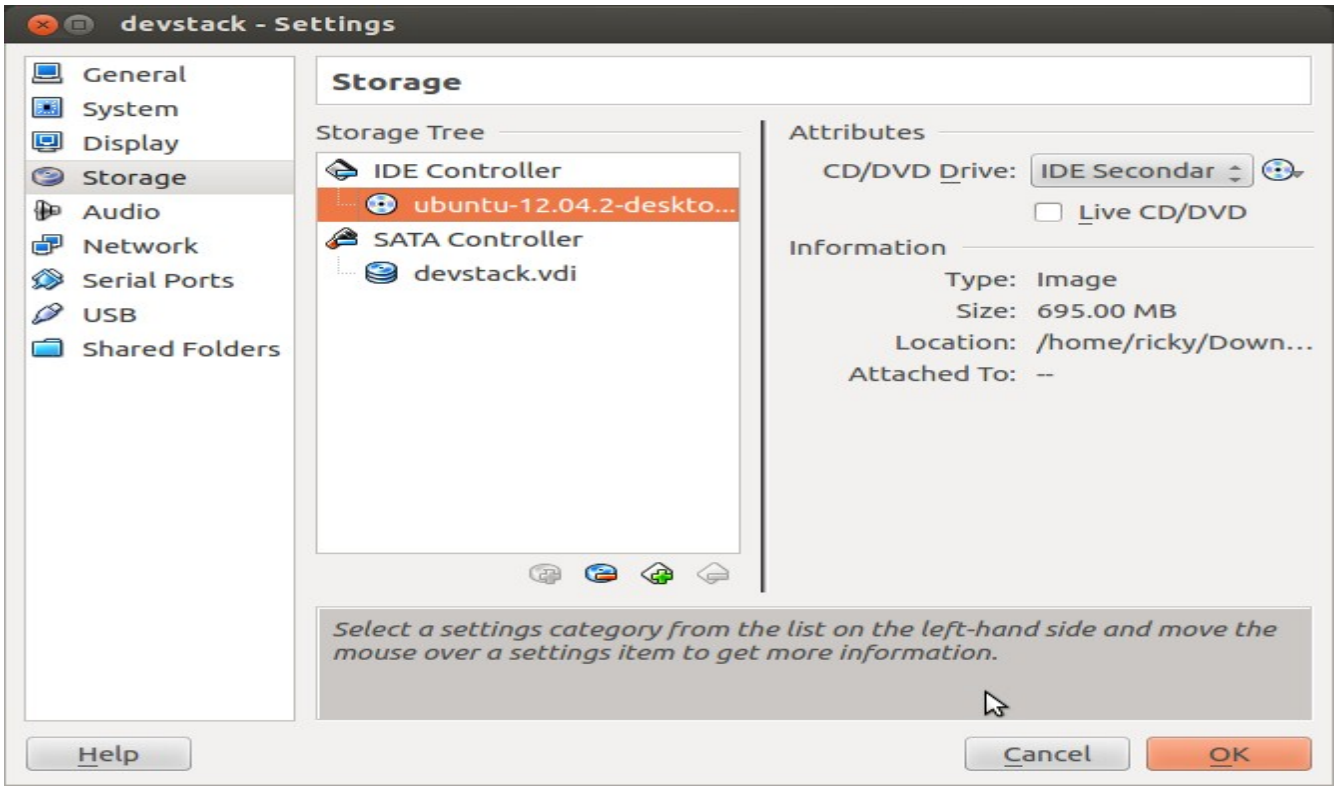

Create New Virtual Disk
Welcome to the virtual disk creation wizard
This wizard will help you to create a new virtual disk for your virtual machine.
Use the Next button to go to the next page of the wizard and the Back button to return to the previous page. You can also press Cancel if you want to cancel the execution of this wizard.
Please choose the type of file that you would like to use for the new virtual disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.
File type
VDI (VirtualBox Disk Image)
VMDK (Virtual Machine Disk)
VHD (Virtual Hard Disk)
HDD (Parallels Hard Disk)
< Back
Next >
Cancel


Create New Virtual Disk
Virtual disk storage details
Please choose whether the new virtual disk file should be allocated as it is used or if it should be created fully allocated.
A dynamically allocated virtual disk file will only use space on your physical hard disk as it fills up, although it will not shrink again automatically when space on it is freed.
A fixed size virtual disk file may take longer to create on some systems but is often faster to use.
Storage details
Dynamically allocated
Fixed size
< Back
Next >
Cancel

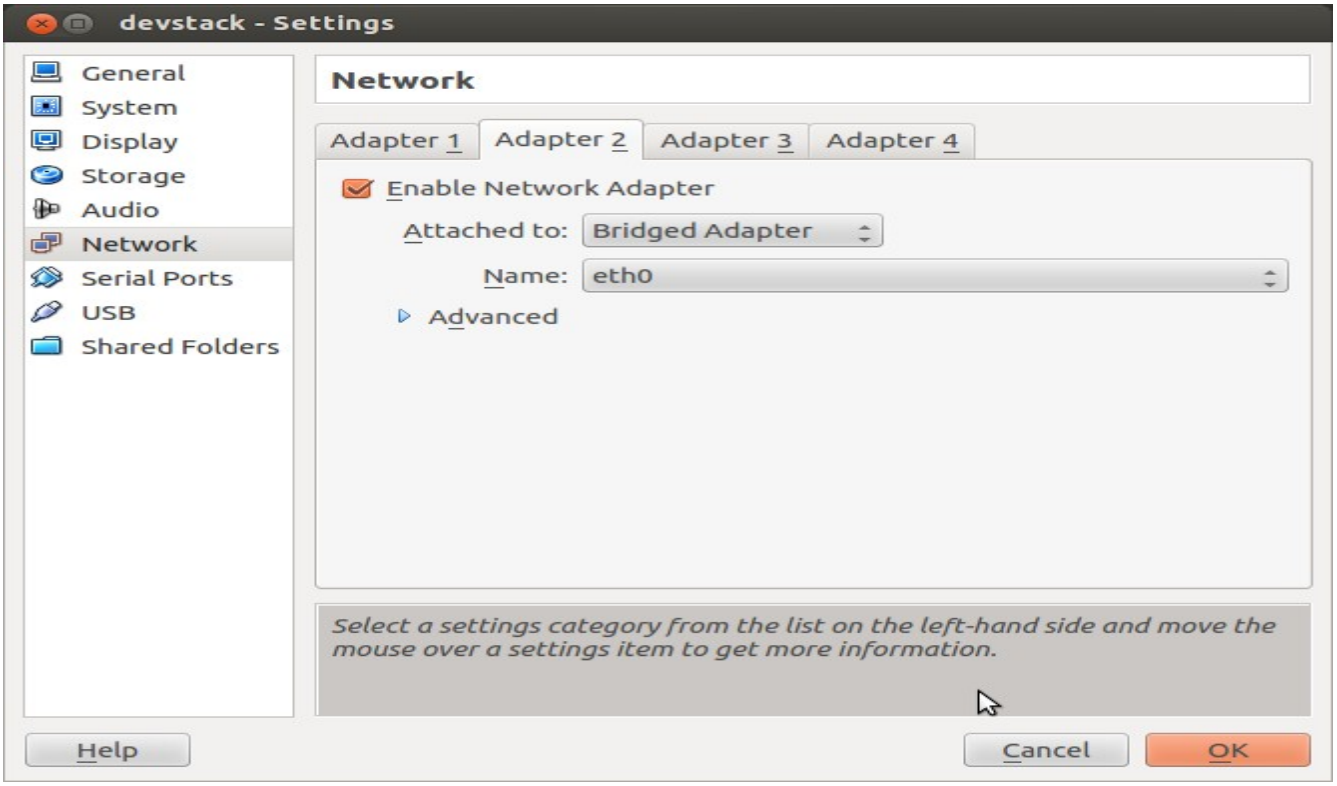

Create New Virtual Disk
Virtual disk file location and size
Please type the name of the new virtual disk file into the box below or click on the folder icon to select a different folder to create the file in.
Location
devstack
Select the size of the virtual disk in megabytes. This size will be reported to the Guest OS as the maximum size of this virtual disk.
Size
8.00 GB
4.00 MB
2.00 TB
< Back
Next >
Cancel


Create New Virtual Disk
Summary
You are going to create a new virtual disk with the following parameters:
File type: VDI (VirtualBox Disk Image)
Details: Dynamically allocated storage
Location: /home/ricky/VirtualBox VMs/devstack/devstack.vdi
Size: 8.00 GB (8589934592 B)
If the above settings are correct, press the Create button. Once you press it the new virtual disk file will be created.
< Back
Create
Cancel

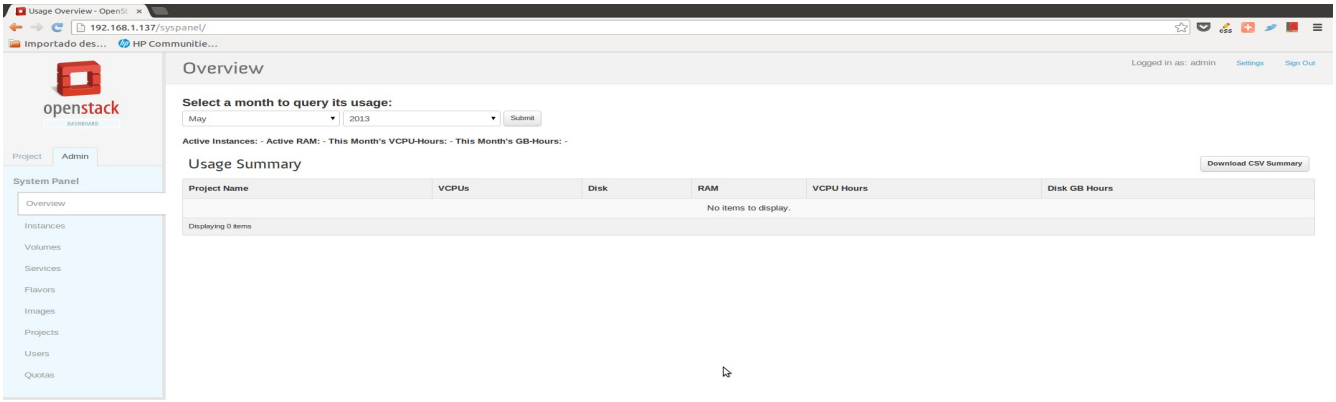
4. Configurar la unidad de CDROM de la máquina virtual creada para que arranque la imagen de Ubuntu que hemos descargado en el paso 2:



5. Añadir un adaptador de red en modo “Bridged” a la máquina virtual:



6. Arrancamos la máquina virtual e instalamos Ubuntu 12.04 (no es necesario cambiar ningún parámetro por defecto)
7. Una vez terminada la instalación, abrimos un terminal e instalamos git con el comando *sudo apt-get install git*
8. Clonamos el repositorio de Openstack Folsom ejecutando el comando *git clone https://github.com/openstack-dev/devstack.git -b stable/folsom devstack/*
9. Cambiamos al directorio creado devstack con *cd devstack*
10. Creamos un fichero con nombre “localrc” con el contenido de la URL https://gist.github.com/everett-toews/3887022
11. Ejecutamos el script con *./devstack.sh*
12. Cambiamos la IP fijada en la BD de Openstack de los componentes para que se use la IP del segundo interfaz de red en modo “bridge” (en mi caso 192.168.1.137) ejecutando los comandos:
    mysql -uroot -pdevstack keystone
    update endpoint set extra = replace(extra, '"publicurl": "http://10.0.2.15', '"publicurl": "http://192.168.1.137') where instr(extra, '"publicurl": "http://10.0.2.15') > 0;
13. Reiniciamos los componentes ejecutando los comandos:
    screen -X -S stack quit
    screen -c stack-screenrc
14. Abrimos un navegador y vamos a la IP del interfaz “bridge” para loguearnos con el usuario “admin” y contraseña “devstack”



Ya tenenemos Openstack funcionando en la máquina virtual.
Se recomienda cambiar al proyecto por defecto “demo” pinchando en la pestaña “Project” y el botón “CURRENT PROJECT” y crear algunos servidores, volúmenes, etc, para tener un conjunto de datos con los cuales testear la aplicación Openstackdroid.

Para conectar a nuestra instalación de Openstack desde la aplicación Openstackdroid, anotar el identificador de *project* del proyecto “demo” pinchando en “Projects” desde la pestaña “Admin” y crear un perfil de conexión en Openstackdroid copiando este valor.
Sugiero que se crean varios perfiles de conexión con distintos usuarios y contraseñas (“admin”/”devstack” y “demo”/”devstack” están por defecto creados) para ver los distintos comportamientos de la aplicación según si el usuario es administrador o no.

Anexo B: Bibliografía

Libros

Marko Gargenta (2011). *Learning Android.* Ed. O'Reilly Media
Zigurd Mednieks, Laird Dornin, G. Blake Meike, Masumi Nakamura (2012). *Programming Android, 2nd Edition.* Ed. O'Reilly Media

Recursos web

Guías de desarrollo oficiales de Android: http://developer.android.com/develop/index.html
Tutoriales de Vogella: http://vogella.com/
Preguntas y respuestas de Stackoverflow: http://stackoverflow.com/