Documento de Arquitectura de Software

Extensibilidad de herramienta de gestión para VMware Cloud Director

Aldo Díaz Betizagasti
Marcelo Sureda

Contenido

Introducción **4**

Vista de Casos de Uso **5**
- Casos de uso identificados 5
  - Programación de secuencias de actividades 5
  - Escalado automático de recursos 5
  - Acciones aplicadas a varias organizaciones 5
  - Eliminación automática de snapshots 5
  - Reporte de recursos por organización 5
- Diagrama de Casos de Uso Críticos 6
- Especificación de Casos de Uso Críticos 7

Vista Lógica **10**
- Estilo Arquitectónico 10
- Diagrama de Componentes del Sistema 12
- Subsistemas y Componentes 13
  - Frontend 13
    - Descripción 13
  - Servicios 14
    - Diagrama de Clase 14
    - Descripción 14
  - Lógica 16
    - Diagrama de Clase 16
    - Descripción 16
  - Entidades 17
    - Diagrama de Clases 17
    - Descripción 17
  - Librería de Soporte 19
    - Diagrama de Componentes 19
    - Descripción 19
  - Persistencia 23
    - Diagrama de Componentes 23
    - Descripción 23
- Diseño de Interfaces 24
  - Wireframes de cada sección 24

Vista de Procesos **28**
- Diagramas de Interacción 28
  - Programación de secuencias de actividades 28
  - Escalado automático de recursos 30
  - Acciones aplicadas a varias organizaciones 31
  - Eliminación automática de snapshots 32

Reporte de recursos por organización 32
Vista de Distribución – Deployment **33**
Vista de Implementación **35**
Referencias **36**

1 Introducción

El framework de extensión para vCloud Director[1] brinda a los proveedores de servicios en la nube la capacidad de extender la API estándar. Los proveedores de servicios pueden registrar un patrón de URL que, al ser invocado, se enruta, a través del Advanced Message Queuing Protocol (AMQP), a un módulo que ejecutará la lógica personalizada y codificada por el usuario. Al proporcionar sus servicios como una extensión de la API del producto, se proporciona un único punto de acceso integrando la interfaz de usuario (con personalizaciones propias o de terceros), para los providers y para sus clientes (tenants). A medida que más clientes adopten DevOps, es fundamental brindar la capacidad de automatizar más tareas dentro de sus plataformas IaaS.

El sistema descrito en este documento se inscribe dentro de este esquema de extensión de API provisto por VMWare, conformando un nuevo framework que busca cubrir los aspectos requeridos para cumplir con las funcionalidades habituales de las extensiones de vCloud Director. De esta forma se extiende la API disponible, simplificando al desarrollador la tarea de implementar la extensión, agregando, además, nuevas funcionalidades dentro del framework que potencian las posibilidades existentes.

El framework está compuesto por un subsistema de frontend para la interfaz de usuario y un subsistema de backend que funciona integrado en el servidor como la extensión de funcionalidades de vCloud Director. Los componentes y módulos dentro de cada subsistema, como así también la interacción entre ellos, se describen en este documento, tomando como base el modelo 4+1[2].

2 Vista de Casos de Uso

En esta sección se detalla la representación de los requisitos del sistema a ser construido en forma de casos de uso. Se describe el comportamiento de estos como también la información necesaria que requiere el sistema por parte del usuario.

2.1 Casos de uso identificados

2.1.1 Programación de secuencias de actividades

Este requisito permite al usuario poder elegir una serie de tareas en un orden arbitrario y poder agregar información de calendario para agendar cuándo estas tareas deben de ser ejecutadas. Luego, el sistema, en el tiempo especificado, ejecuta las tareas de manera asincrónica y según el orden predispuesto.

2.1.2 Escalado automático de recursos

Este requisito tiene como motivación lograr utilizar de una manera más eficiente los recursos del sistema en base a las métricas de uso de las máquinas virtuales activas. Para ello, el usuario define una serie de parámetros que servirán como cotas para que el sistema, a través de las herramientas de monitorización del uso, determine si una máquina virtual está utilizando más recursos de los asignados o viceversa. En estos casos se quitan o asignan recursos, respectivamente.

2.1.3 Acciones aplicadas a varias organizaciones

Este requisito tiene como motivación poder realizar una misma tarea para distintas organizaciones de una manera sencilla y centralizada. Para ello, el usuario sería capaz de elegir una funcionalidad, elegir las organizaciones a las cuales desee aplicar tal tarea y comenzar la ejecución en masa.

2.1.4 Eliminación automática de snapshots

La idea principal de este requisito es poder tener un control automático sobre las snapshots[7] que se crean en el sistema y que con el tiempo se vuelven obsoletas. Las mismas ocupan un considerable espacio en el sistema de almacenamiento y a su vez provocan un aumento en el esfuerzo para mantener el estado actual de la máquina virtual.

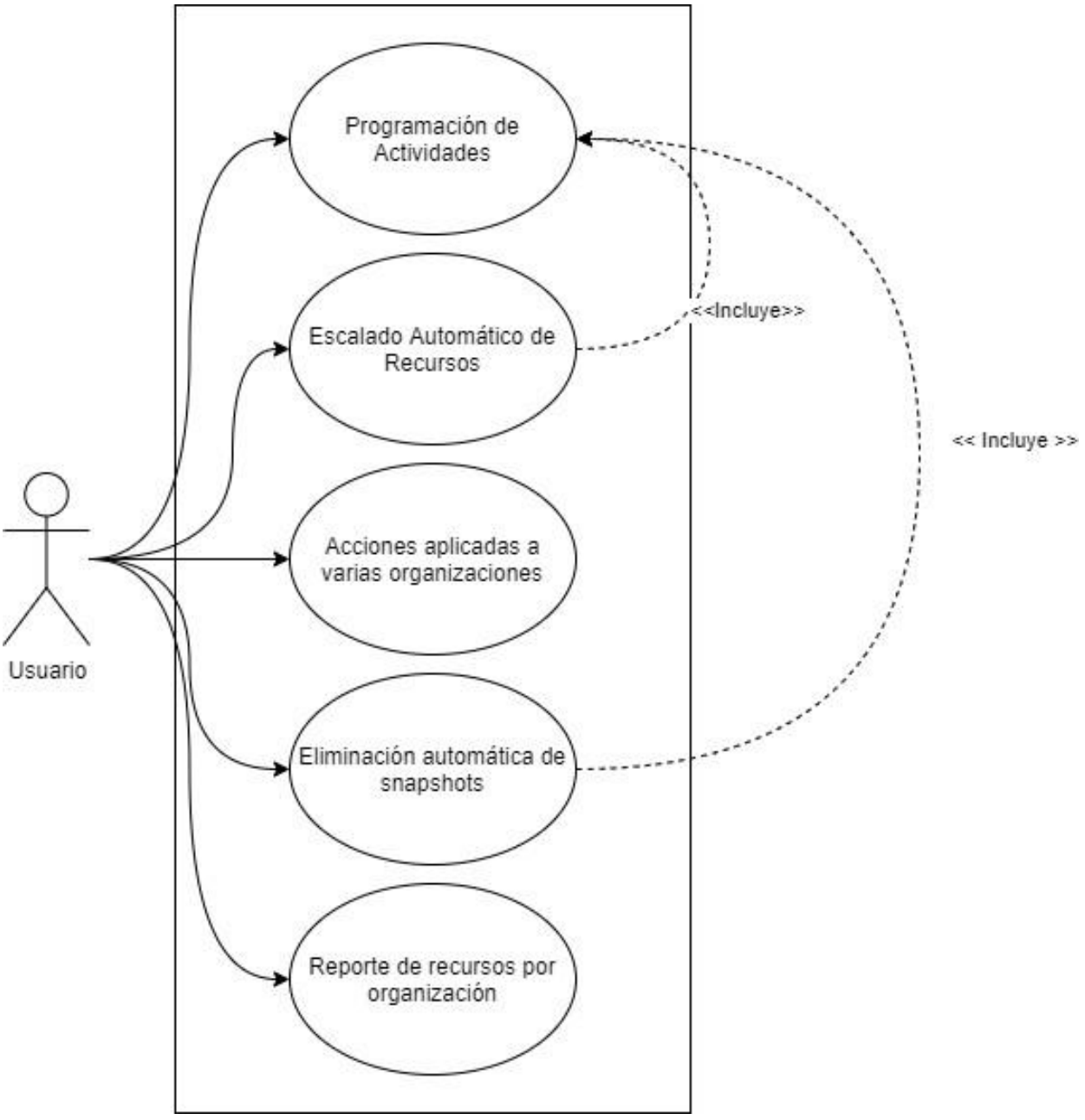
Por lo tanto, el usuario debe de ingresar el identificador de la máquina virtual que se desee monitorear, el período de tiempo en que se debe de chequear la misma y a su vez una cota para la cual las snapshots con un tiempo de vida mayor se consideran obsoletas. El sistema, por otro lado, periódicamente obtiene las snapshots de la máquina virtual especificada, consulta el tiempo de vida y las elimina según corresponda, liberando el espacio que éstas ocupan.

2.1.5 Reporte de recursos por organización

Este requisito tiene como propósito inicial poder obtener en una vista unificada la información de las distintas organizaciones creadas. La principal información que se mostrará de las distintas organizaciones es: aplicaciones virtuales existentes, máquinas virtuales que se encuentren en éstas últimas, redes creadas, direcciones de IP en uso, uso de CPU, memoria, entre otros.
La motivación se centra en poder monitorear y tener un control global sobre la información relevante de infraestructura de las organizaciones existentes en el sistema.

2.2 Diagrama de Casos de Uso Críticos

En esta sección se detalla las relaciones entre los casos de uso descritos. Debido a que los casos de usos fueron específicamente tomados con el objetivo de poder englobar una serie de funcionalidades particulares y distintas del sistema, éstos presentan una notable característica de independencia entre sí.



2.3 Especificación de Casos de Uso Críticos

En esta sección se detallan los flujos de ejecución de los casos de uso paso a paso. A su vez, se añaden las condiciones requeridas para la ejecución de los mismos.

Nombre: Programación de Secuencia de actividades
Objetivo: Delegar una serie de tareas a una actividad programada para su ejecución en momentos especificados.
Actores: Usuario.
Precondiciones: Existen tareas en el sistema.
Descripción: El Usuario selecciona un conjunto de tareas en un orden particular y específica información de calendario. Las tareas serán ejecutadas en el tiempo y orden dados.
Flujo Normal: 1. El usuario selecciona una lista de tareas existentes en el sistema en cierto orden. 2. El usuario selecciona la información de calendario en la que estas tareas serán ejecutadas. 3. El sistema recibe la información para su procesamiento. Se crean todas las entidades de la lista de tareas provista por el usuario. Se utiliza el TaskScheduler para agendar la tarea de programar las actividades en el tiempo especificado. 4. El sistema retorna que la solicitud fue correcta y las tareas son ejecutadas en el período dado.
Flujos Alternativos: 3A. Ocurre un error al instanciar las entidades de las tareas. • El sistema retorna un mensaje de error acorde. • Vuelve al paso 1. 4A. Ocurre un error al instanciar las entidades necesarias para la ejecución de las tareas. • El sistema registra el error en el centro de registro. • Se finaliza la tarea programada. • Fin de caso de uso.
Postcondiciones: Las tareas son registradas en una actividad programada y son ejecutadas en el tiempo indicado.

Nombre: Escalado Automático de recursos.
Objetivo: Proveer de una funcionalidad para realizar un uso más eficiente de los recursos del sistema.
Actores: Usuario.
Precondiciones: Existen máquinas virtuales en el sistema.
Descripción: El Usuario elige una máquina virtual, un recurso a ser monitoreado y un umbral condicionante. El sistema monitorea el recurso indicado de la máquina virtual, y en base al umbral indicado el sistema limita o provee dicho recurso.
Flujo Normal: 1. El Usuario ingresa a la sección en el frontend del caso de uso. 2. El Sistema despliega una lista de máquinas virtuales y la lista de recursos que son posibles monitorear. 3. El Usuario elige una máquina virtual y un recurso a monitorear 4. El Sistema recibe la información para ser procesada. Se crea un plan, se instancia una tarea de escalado de recursos, se incluye la información proporcionada por el usuario. 5. La actividad programada se encargará de ejecutar la tarea en el período seleccionado. La máquina virtual es monitoreada sobre el recurso especificado cada período de tiempo dado. Si el uso de tal recurso supera el umbral provisto, entonces se aumenta la cantidad del recurso y viceversa. 6. El Sistema retorna un mensaje al frontend que la actividad fue agendada.

Flujos Alternativos:
5A. Ocurre un error al escalar el recurso.

- El sistema registra el error en el centro de registro.
- Se finaliza la tarea programada.
- Fin de caso de uso.

Postcondiciones: La máquina virtual seleccionada tiene acceso a más del recurso seleccionado si lo necesita o menos si no lo utiliza.

Nombre: Acciones aplicadas a varias organizaciones.

Objetivo: Proveer de una funcionalidad para poder ejecutar una misma tarea en lote a varias organizaciones.

Actores: Usuario.

Precondiciones: Existen tareas y organizaciones en el sistema.

Descripción: El Usuario selecciona una tarea en particular junto con una colección de organizaciones. El sistema luego ejecuta dicha tarea en las organizaciones.

Flujo Normal:
1. El Usuario ingresa a la sección en el frontend de aplicar acciones a varias organizaciones.
2. El Sistema despliega una lista de tareas disponibles para el caso de uso y una lista de organizaciones.
3. El Usuario elige la tarea a ejecutar y la lista de organizaciones en las cuales quiere ejecutar dicha tarea.
4. El Sistema recibe la información para ser procesada. Instancia un plan, añade en él las tareas seleccionadas por el usuario, y ejecuta el plan.
5. El sistema itera sobre las organizaciones y ejecuta el plan creado.
6. El sistema retorna un mensaje confirmando que las tareas fueron ejecutadas en las organizaciones.

Flujos Alternativos:
2A. La combinación nombre de usuario/contraseña es incorrecta.

- El sistema despliega un mensaje de error acorde.
- Vuelve a 1.

Postcondiciones: La tarea seleccionada es ejecutada en todas las organizaciones indicadas por el usuario.

Nombre: Eliminación automática de snapshots

Objetivo: Proveer de una funcionalidad para monitorear el estado de las snapshots de una máquina virtual y eliminarlas si las mismas fueron creadas posterior a un tiempo indicado por el usuario.

Actores: Usuario.

Precondiciones: Existen máquinas virtuales en el sistema.

Descripción: El Usuario selecciona una máquina virtual, una frecuencia de tiempo y un tiempo fijo determinado para que el sistema pueda discernir si una snapshot es obsoleta o no. En caso de que lo sea, la misma es eliminada. Este proceso es realizado periódicamente según la frecuencia indicada.

Flujo Normal:
1. El Usuario ingresa a la sección del caso de uso en el frontend.
2. El Sistema despliega una lista con las máquinas virtuales del sistema.
3. El Usuario selecciona una máquina virtual y define un período de retención por cual el sistema discernirá si una snapshot es obsoleta o no.
4. El sistema envía la información para ser procesada. Se crea un plan y se instancia la tarea del caso de uso, se incluye la información provista por el usuario.
5. El sistema retorna un mensaje para informar que la tarea fue creada con éxito.
6. La actividad programada ejecuta cada cierto tiempo definido y lee las snapshots de la máquina virtual especificada. Si la misma tiene un tiempo de vida mayor al umbral dado por el usuario, entonces se elimina, sino se mantiene en el sistema.

Flujos Alternativos:
4A. Ocurre un error al instanciar e inicializar la tarea del caso de uso

- El sistema despliega un mensaje de error acorde.
- Vuelve al paso 2.

6A. Ocurre un error en el sistema en el momento de ejecución de la tarea.

- El sistema registra el error en el centro de registro.
- Se finaliza la tarea programada.
- Fin de caso de uso.

Postcondiciones: Las snapshots, de una máquina virtual elegida, que tengan una fecha de creación mayor a la seleccionada son eliminadas del sistema automáticamente.

Nombre: Reporte de recursos de organizaciones.

Objetivo: Proveer de un pantallazo general completo de las entidades que componen a los data-centers de las organizaciones existentes.

Actores: Usuario.

Precondiciones: Existe al menos una organización con al menos un data-center.

Descripción: El Usuario puede visualizar un estado general de todas las organizaciones que pertenecen al sistema en una sola pantalla.

Flujo Normal:
1. El Usuario ingresa a la sección del caso de uso en el frontend.
2. El Sistema despliega toda la información referente a las entidades: Data Centers, vApps, Máquinas virtuales, Redes de las organizaciones registradas y añade información más particular y pertinente de cada una.

Flujos Alternativos:
2A. Ocurre un error al obtener la información de alguna entidad para alguna organización.

- El sistema despliega un mensaje de error acorde.
- Vuelve a 1.

Postcondiciones: La información acerca de los data-centers de las organizaciones registradas en el sistema es desplegada en una sola pantalla.

3 Vista Lógica

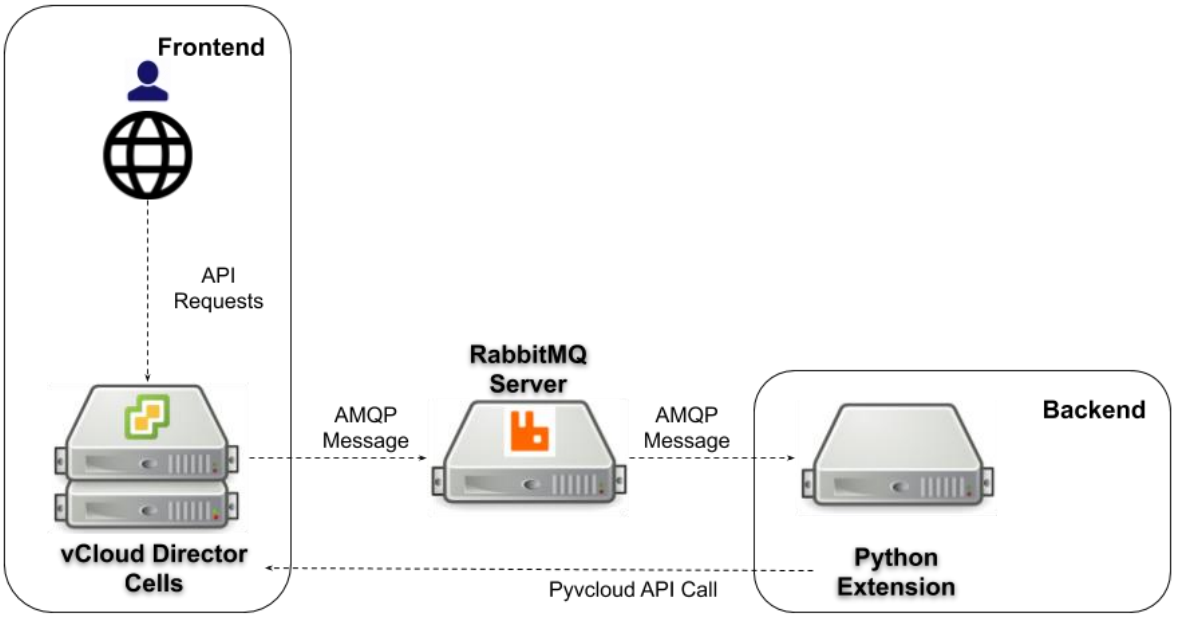
Basados en los Casos de Uso identificados anteriormente se definió abordar el trabajo desde dos diferentes aspectos:

- Aspecto Conceptual: Define, a alto nivel, qué tipo de entidades participan del modelo y cuáles son las relaciones entre ellas.
- Aspecto de Librerías de soporte: Conjunto de componentes que brindan una funcionalidad base al resto del framework.

Este enfoque llevó a identificar un conjunto de subsistemas, componentes y clases que integran la solución. Dichos elementos fueron mutando y su estructura se ha ido ajustando en revisiones del diseño inicial y en sucesivas iteraciones.

3.1 Estilo Arquitectónico

El estilo arquitectónico está determinado por el entorno de la tecnología en la que se implementa la solución. El framework a implementar consiste en una extensión a vCloud Director y, por lo tanto, su despliegue debe ajustarse a la arquitectura definida por la solución de VMware que se presenta en el siguiente diagrama simplificado de distribución.

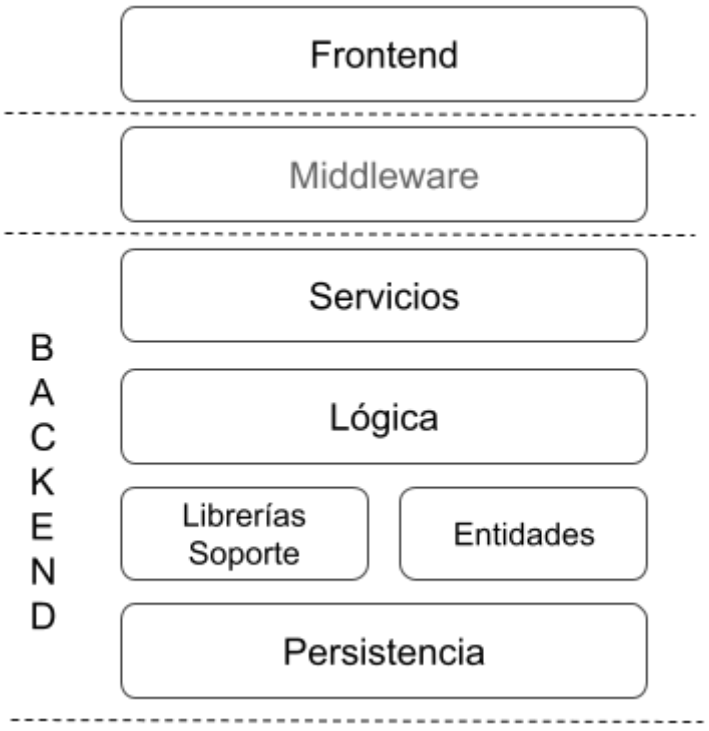


Se detallan a continuación algunas características técnicas de la solución, describiendo cada uno de los principales subsistemas que intervienen:

- Una aplicación de frontend integrada naturalmente con la interfaz web de vCloud Director.
  - Para la integración con la tecnología subyacente la aplicación frontend estará desarrollada en TypeScript.
  - La integración de “look and feel” (estilo) con la interfaz web se consigue haciendo uso del framework Clarity Design.[6]

- Un endpoint con el URL que se debe registrar e integrar a las APIs de vCloud Director.
    - La propia célula de vCloud Director cuando reciba los requerimientos en dicha URL del endpoint desde la aplicación frontend los enviará al servicio de backend.
- Un servidor o “broker” de mensajería RabbitMQ[3] que rutea los mensajes recibidos desde vCloud Director al servicio configurado en la solución.
    - La instancia de RabbitMQ ya debe estar instalada y configurada. Se sabe que el “broker” de mensajería forma parte de la infraestructura requerida de vCloud Director.
    - Hace uso del protocolo AMQP.
- Un servicio de backend que recibe los requerimientos y procesa los mismos.[4] [5]
    - Se integra el backend con las APIs de vCloud Director a través del conector Python disponible con la librería pyvcloud.
    - Cabe notar que el soporte por parte de VMware de conectores para otros lenguajes ha sido discontinuado, y el único que garantiza compatibilidad con las últimas versiones de la API es el de Python).
    - Esto motiva que la tecnología utilizada para el backend sea el lenguaje de programación Python que, además posee una integración con el servidor de mensajería RabbitMQ a través de la librería Pika.

De esa manera, y a partir del diagrama simplificado de distribución, queda definido un estilo arquitectónico en capas. Las capas de frontend y de backend, con sus respectivos componentes, aparecen claramente delimitadas.

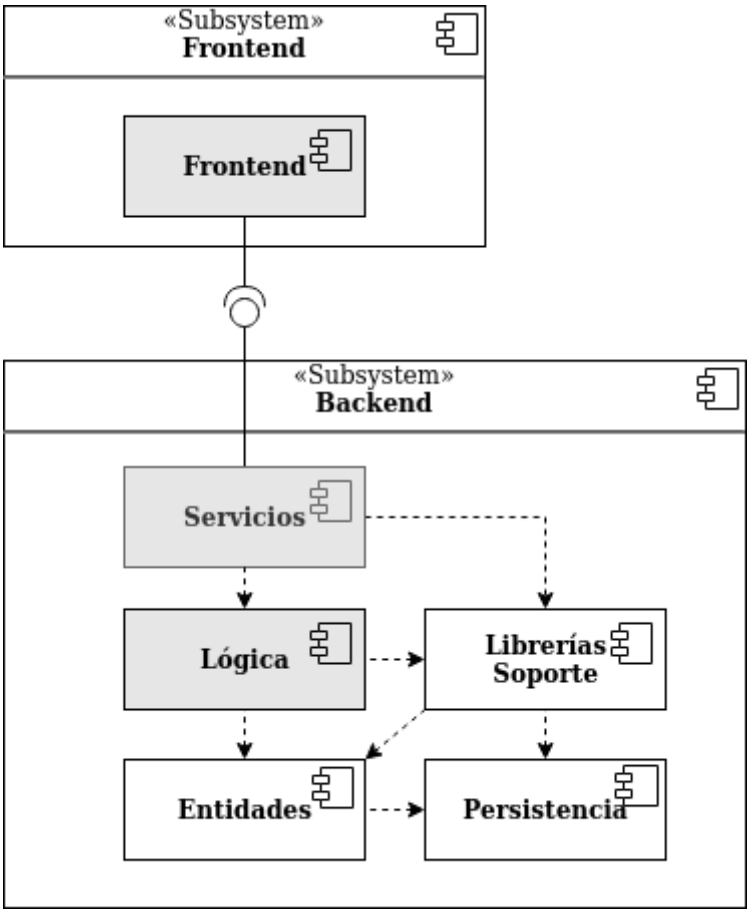


La implementación de la prueba de concepto del Caso de Uso “Agendar Apagado de una VM” permitió identificar y validar los detalles de la arquitectura. Ese modelo se describe en todo detalle en el anexo “Extensión vCloud Director - Solución Técnica”, incluyendo el flujo de los mensajes entre las diferentes capas.

3.2 Diagrama de Componentes del Sistema

Quitando el middleware del diagrama anterior, enfatizando los componentes lógicos, y haciendo foco en los subsistemas que se deben desarrollar para implementar la solución, se obtiene un diagrama de componentes que refleja el estilo arquitectónico, y permite seguir iterando en sucesivos refinamientos sobre la vista lógica.

A continuación, el diagrama con sus capas o subsistemas y componentes que lo integran.



En el diagrama figuran con fondo blanco aquellos componentes que conforman el núcleo (*core*) del framework. Estos componentes aparecen abajo en diagrama formando parte del backend. Son los componentes de Librerías de Soporte, Entidades y Persistencia.

En el diagrama aparecen con fondo gris aquellos componentes que el usuario del framework tiene libertad para personalizar. Estos componentes permiten ajustar la funcionalidad para cumplir con los requerimientos específicos de cada caso de uso. Son los componentes de Lógica y Servicios en el backend y componente de Frontend, en la aplicación o subsistema de frontend.

El subsistema de Frontend y sus componentes se detallan en la sección Frontend.

Dentro de la capa o subsistema de la Aplicación Backend se reflejan los enfoques mencionados anteriormente: el aspecto conceptual en el subsistema llamado Entidades, y el

aspecto de librerías de soporte, en los subsistemas Librerías de Soporte y Persistencia, que brindan soporte a los componentes de Lógica y Servicios.

En la siguiente sección se detallan los diferentes subsistemas entrando en detalle en sus componentes y las interfaces que implementan.

3.3 Subsistemas y Componentes

En esta sección se describen cada uno de los subsistemas identificados anteriormente. El diagrama de componentes muestra en detalle aquellos que corresponden a cada subsistema. En las diferentes secciones se va haciendo zoom en los componentes o packages que constituyen el subsistema. En cada caso se detallan las interfaces que implementa.

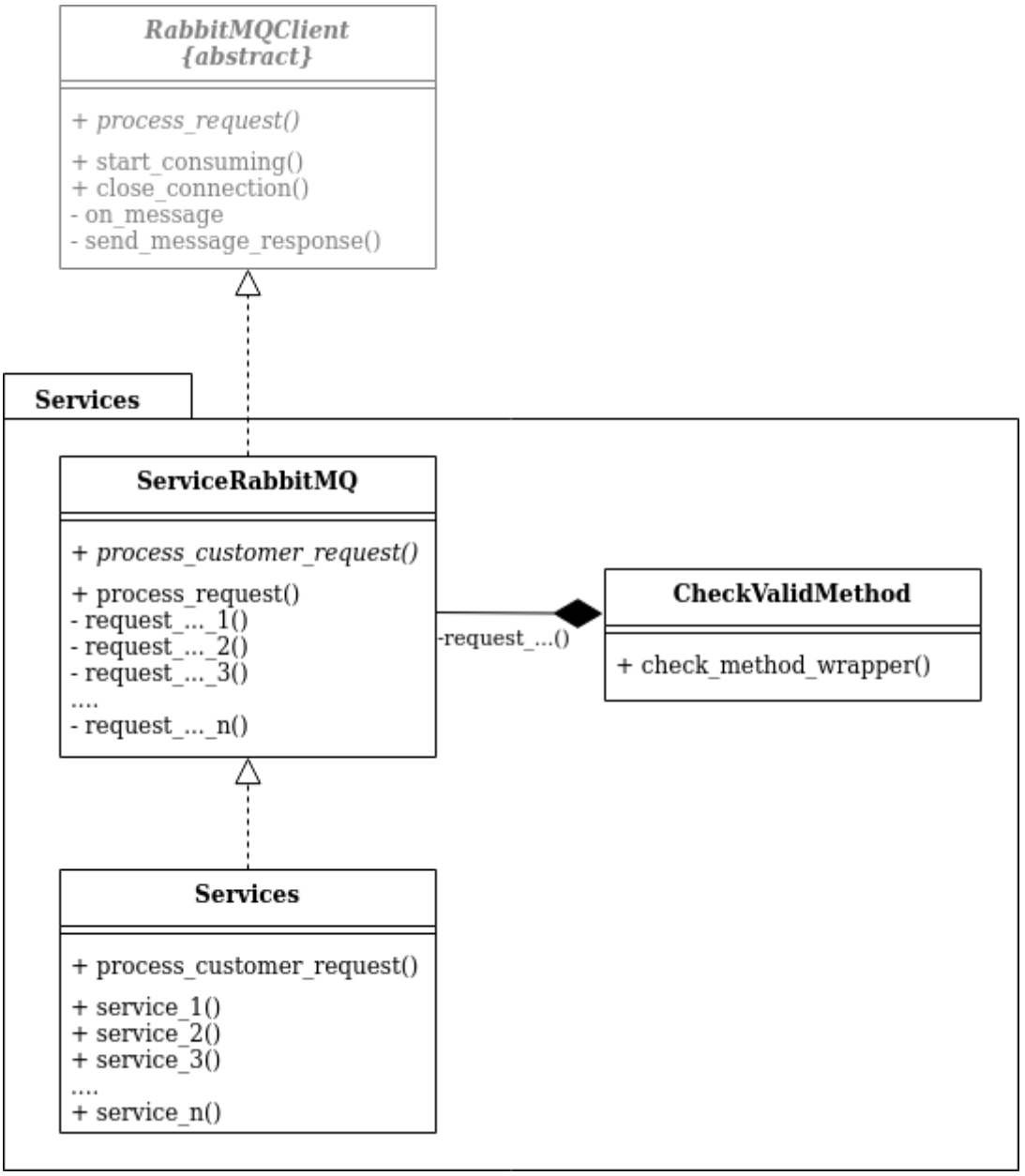
3.3.1 Frontend

3.3.1.1 Descripción

Único componente que corresponde a la capa de frontend. Para cada caso de uso se deberá implementar un componente específico e integrar con la interfaz web de vCloud Director y debe ser implementado por el usuario del framework. Invoca los endpoints que el componente de servicios expone.

3.3.2 Servicios

3.3.2.1 Diagrama de Clase



3.3.2.2 Descripción

Este componente expone los servicios al frontend y delega las operaciones haciendo uso de los otros componentes que dan soporte a la funcionalidad del backend, principalmente el componente de Lógica. El componente de Servicios marca la frontera de la aplicación backend y es la que recibe los requerimientos desde el frontend. Consiste, principalmente, en implementar el método process_request() de la clase abstracta RabbitMQClient. El usuario del framework deberá encargarse de dicha implementación. Ese método realizará los siguientes pasos:

- Atender los requerimientos entrantes del frontend: recibir y validar el mensaje.
- Invocará un método de la clase de lógica de negocios (ver sección Lógica).
- Formatear y devolver la respuesta

En primera instancia, el componente de Servicios debe analizar la URL del mensaje de entrada para determinar cuál es el tipo de servicio que se está invocando. Esa tarea se realiza en el método process_request(). Una vez determinado el tipo de servicio, y obtenidos los parámetros del mensaje de requerimiento recibido, se invoca el método correspondiente para validar la operación HTTP (GET, POST, PUT, etc), y ese método a su vez, debe invocar la lógica de negocios del componente de Lógica.

Para una descripción más detallada de la interacción del componente de Servicios con la clase RabbitMQClient consultar la documentación de dicha clase en la sección de Librerías de Soporte.
Dentro del framework se exponen un conjunto limitado de operaciones que dan soporte a los casos de uso implementados extendiendo el framework. Dichas operaciones son:

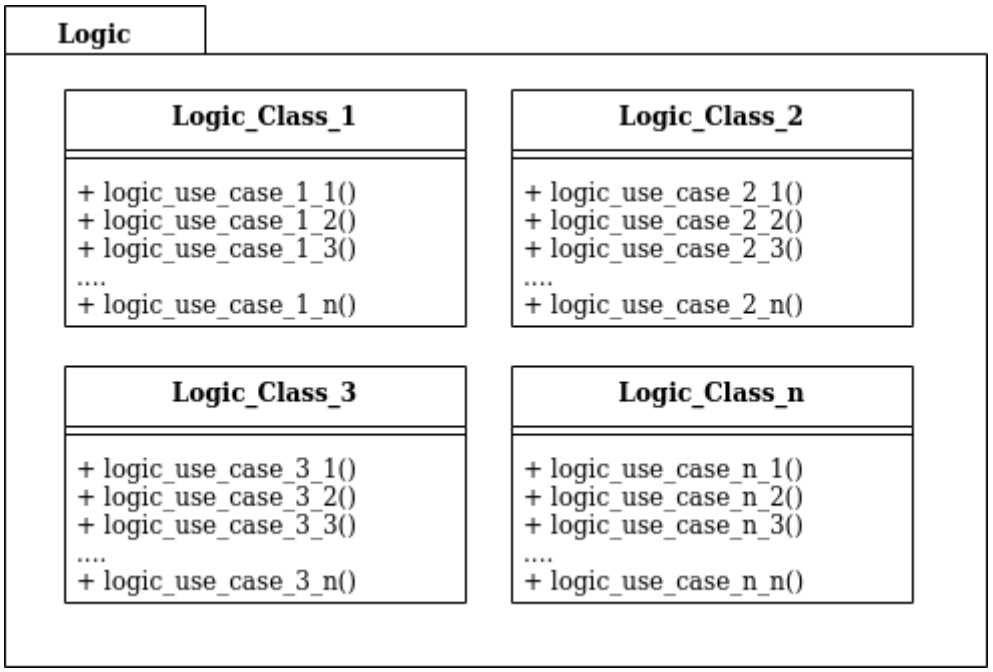
URI	**Método**	**Descripción**
/api/extension/orgs	GET	Listar organizaciones
/api/extension/org/{org-id}/vapps	GET	Listar vApps y VMs de una organización
/api/extension/org/{org-id}/vms	GET	Listar sólo VMs de una organización
/api/extension/vapp/methods	GET	Listar métodos de vApps y VMs
/api/extension/vm/methods	GET	Listar métodos sólo de VMs

En el documento de Manual de Usuario del framework se detallan las operaciones y los endpoints de la capa de Servicios y se brindan las guías generales para extender la clase de Servicios del framework para extender su funcionalidad.
El usuario puede desarrollar un nuevo componente (por ej. CustomerServices) para extender el componente de servicios, personalizando el framework e implementando nuevos casos de uso. De la misma forma que el componente de Servicios hereda y extiende la clase RabbitMQClient, se debe extender el componente de Servicios implementando el método process_customer_request().
En el ejemplo disponible en el repositorio para poder cumplir con los casos de uso identificados , se extiende la clase de Servicios con las siguientes operaciones:

URI	**Method**	**Description**
/api/extension/org/{org-id}/schedule	POST	Programar secuencia de actividades
/api/extension/org/{org-id}/vm/{vm-id}/autoscaling	POST	Escalado automático de recursos
/api/extension/orgs/actions	POST	Aplicar acciones a varias organizaciones
/api/extension/org/{org-id}/vm/{vm-id}/snapshot/clean	POST	Eliminación automática de snapshots
/api/extension/org/{org-id}/resources	GET	Reporte de recursos

3.3.3 Lógica

3.3.3.1 Diagrama de Clase



3.3.3.2 Descripción

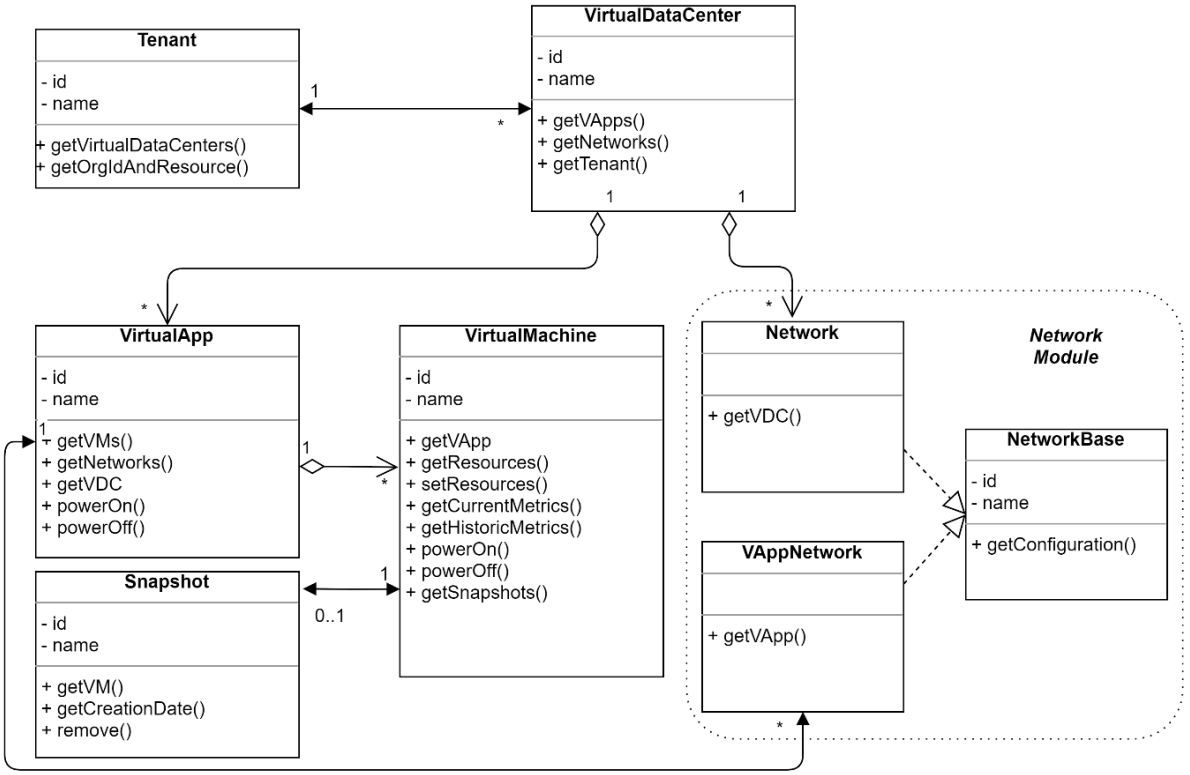
Este componente se debe encargar de implementar la lógica del Caso de Uso. Para ello podrá utilizar cualquiera de los componentes del framework que dan base a la funcionalidad del subsistema de backend: Entidades, Librerías de Soporte (considerando todos sus componentes), y Persistencia. La implementación de este componente, que debe concentrar la lógica de negocios, corre por cuenta del usuario del framework. Para implementarlo, el usuario cuenta con total libertad en cuanto a los módulos internos a definir e implementar internamente, pero al hacer uso de los componentes del framework debe ajustarse a la definición de las interfaces correspondientes.

Dentro del framework la clase de Lógica brinda soporte a las operaciones listadas en la sección de Servicios.

En el documento de Manual de Usuario del framework se brindan las guías generales para desarrollar nuevas extensiones complementando la capa de Servicios e implementando la lógica de nuevos casos de uso.

3.3.4 Entidades

3.3.4.1 Diagrama de Clases



3.3.4.2 Descripción

En el componente Entidades se modela una jerarquía de entidades presentes en vCloud Director necesarias para resolver los problemas planteados por los requerimientos de los Casos de Uso descritos en la vista correspondiente. El diagrama de clases presenta las interfaces de cada una y las relaciones entre ellas.

Las clases del componente *Entidades* implementan todos sus métodos accediendo a las APIs de vCloud Director. No se requiere que los objetos creados instanciando dichas clases mantengan atributos propios para que los métodos devuelvan la información solicitada o ejecuten las acciones requeridas por el usuario. Los únicos atributos que se mantienen dentro de los objetos creados (y que son persistidos dentro de la extensión) son el nombre y el identificador del elemento. Todos los datos del sistema ya están presentes en la célula de vCloud Director y los métodos cumplen su funcionalidad invocando las acciones disponibles en las APIs.

Tenant:
Representa una visión simplificada del elemento *Org* disponible en vCloud Director. Es un punto de acceso que permite al usuario navegar hacia los Data Centers virtuales que ese Tenant tiene disponible.

VirtualDataCenter:
Representa una visión simplificada del elemento *VDC*, que corresponde al Data Center de una organización. Permite al usuario acceder a las vApp y Máquinas Virtuales en dicho Data Center.

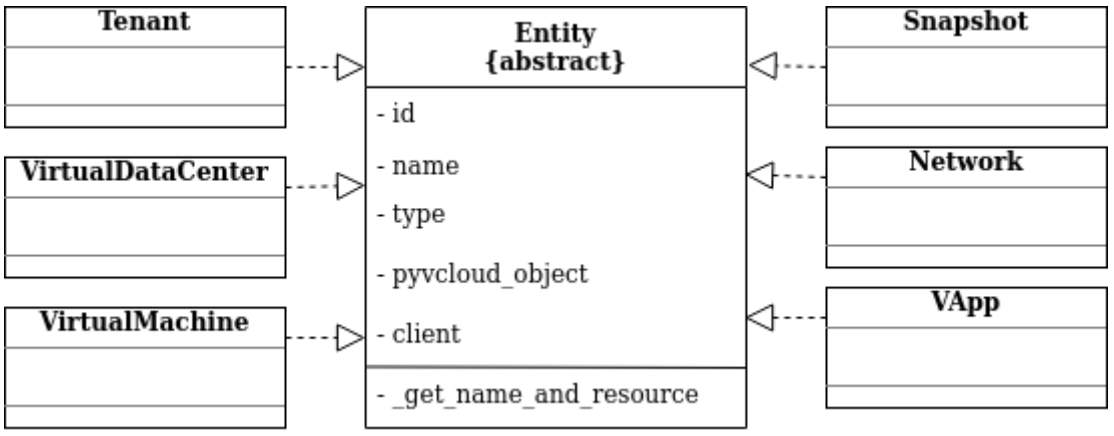
VApp:
Representa una visión simplificada del elemento *VApp* de vCloud Director. Permite al usuario acceder a las máquinas virtuales asociadas, las redes y los snapshots (en caso de que los hubiera).

VirtualMachine:
Representa una visión simplificada del elemento *VM*. Permite al usuario acceder a los recursos asociados a la máquina virtual, y ejecutar acciones sobre las mismas. Lista sus snapshots (en caso de que los hubiera).

Snapshot:
Representa una visión simplificada del elemento *Snapshot*. Brinda información sobre su fecha de creación, la Virtual Machine a la que corresponde, y ofrece un método que permite su eliminación del sistema.

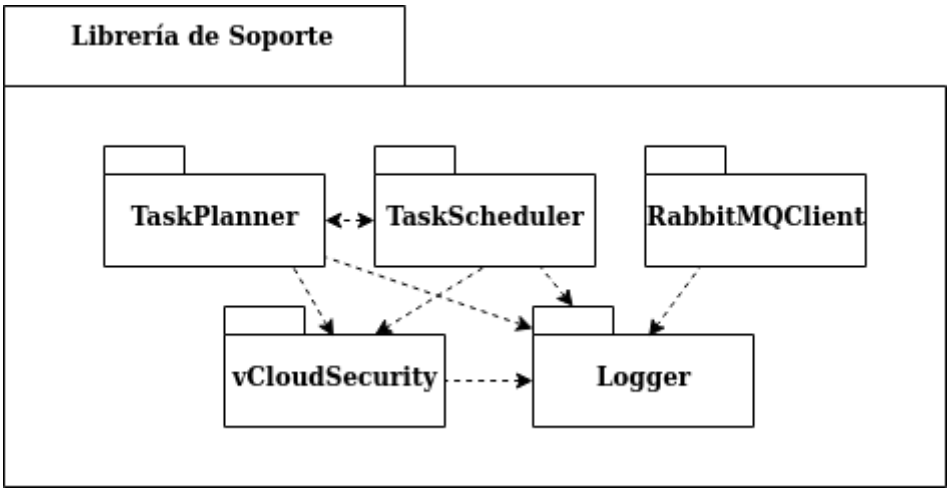
NetworkBase, VAppNetwork y Network:
Representan una visión simplificada del elemento *Network* de vCloud. Las tres clases se agrupan dentro del mismo módulo Network, ya que responden al concepto general de red. La red de la VApp (clase VAppNetwork) se diferencia de la red del Virtual Data Center (Network) porque al método constructor de cada una, le corresponde acceder a diferentes recursos de vCloud Director, y cada una redirige a la entidad contenedora correspondiente. Brinda acceso a las características de la configuración de la red y lista las direcciones IP en uso dentro de la misma.

Entity:
Clase base abstracta. Por simplicidad en el primer diagrama no se representan las relaciones entre la clase abstracta Entity y el resto de las entidades vCloud, que son definidas como sus clases derivadas. Este diseño permite que la clase Task del TaskPlanner pueda relacionarse con todas las entidades de vCloud sin necesidad de conocer detalles de su tipo específico (ver sección Librería de Soporte). En este segundo diagrama se detalla dicha relación, haciendo referencia a las clases ya presentadas sin especificar todos sus métodos y atributos.



3.3.5 Librería de Soporte

3.3.5.1 Diagrama de Componentes



3.3.5.2 Descripción

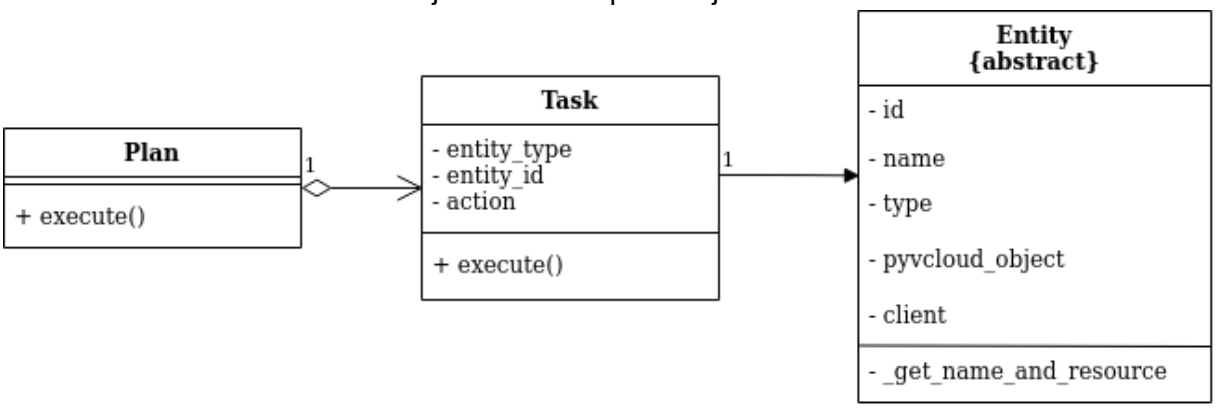
En el aspecto de las librerías se presentan aquellas clases que dan soporte al resto del framework. Constituyen, tanto un núcleo funcional disponible para el resto del sistema, que permite cumplir requerimientos como el agendado de tareas (TaskPlanner y TaskScheduler), como también es encargado de cumplir con algunos requerimientos no funcionales: interoperabilidad (RabbitMQClient), seguridad (vCloudSecurity), y auditabilidad (Logger), entre otros. A continuación se presenta el diagrama con las relaciones entre los diferentes subcomponentes.

Se identificaron algunas grandes áreas de funcionalidades base que dan apoyo a la implementación de los diferentes requerimientos previamente identificados. En esta sección se hace zoom nuevamente en cada uno de los subcomponentes del package Librerías de soporte.

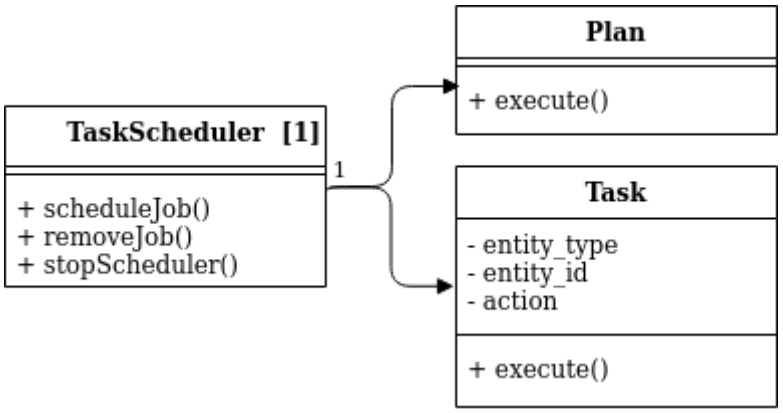
TaskPlanner: Administración de secuencias de tareas (Plan)
Permite lo siguiente:

- Agrupar, en formato de Plan, un conjunto de Tareas o actividades ejecutables
- Establecer un orden de precedencia entre las tareas
- Definir una lógica básica para saber si el plan debe continuar su ejecución.

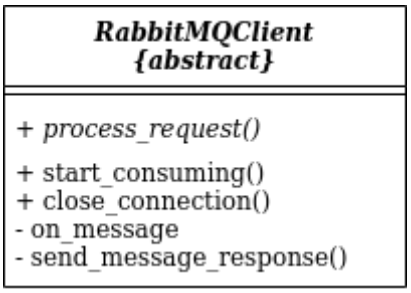
Cada Plan está compuesto de una lista ordenada de Tareas. Cada Tarea incluye su propia acción a ejecutar, parámetros para controlar la acción, y un criterio de éxito. Además, la tarea tiene una relación con el objeto sobre el que se ejecuta la acción.



TaskScheduler: Agendar tareas
Brinda funcionalidad de programación de actividades ejecutables. Es el “scheduler” central del sistema. Permite ejecutar actividades agendadas en forma diaria, semanal, o mensual. Puede agendar objetos de tipo Plan y de tipo Task. TaskScheduler es un singleton: una única instancia de esta clase está disponible al resto del sistema.



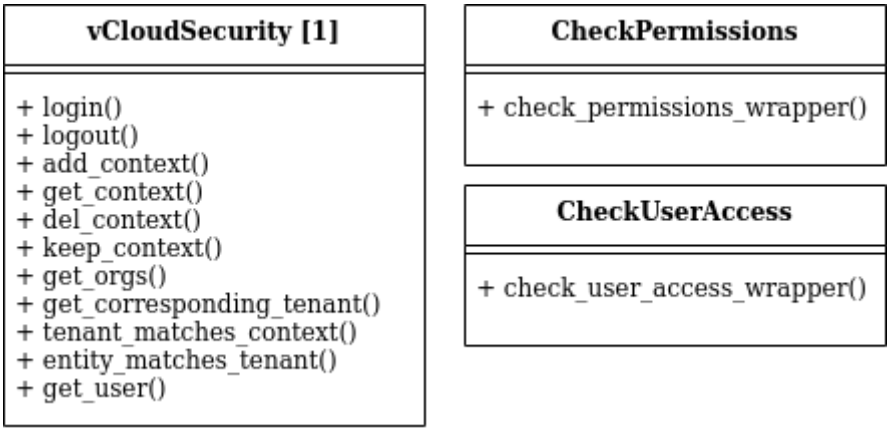
RabbitMQClient: Cliente RabbitMQ
Este es el componente fundamental en la interacción con la extensión del frontend de vCloud Director. La interacción se realiza a través del consumo de mensajes del "broker" de mensajería RabbitMQ. Este módulo encapsula los detalles de la interacción con RabbitMQ para que el usuario del framework no tenga que lidiar con detalles de la conexión, y la lectura y escritura de mensajes al servidor. RabbitMQClient es una clase abstracta con el método virtual process_request() que el usuario debe encargarse de implementar dentro del componente de Servicios, para procesar el requerimiento recibido desde el frontend y devolver la respuesta.



vCloudSecurity: Seguridad, autenticación y validación de permisos vCloud Director
Las tareas que se ejecutan contra el servidor de vCloud Director requieren una autenticación previa que valide las credenciales del servicio. Además de la autenticación, se requiere una validación de permisos para garantizar que los usuarios que envían solicitudes al sistema puedan realizar solamente aquellas acciones para las que poseen permisos. Para ello se provee este módulo dentro de la librería que permite realizar la autenticación, el login en el servidor, la validación de permisos y autorización de acciones dentro del sistema. vCloudSecurity es un singleton: una única instancia de esta clase está disponible al resto del sistema.
Los métodos de la clase vCloudSecurity son los siguientes:

- login: Autenticarse en el sistema.
- logout: Hacer el logout del servidor de vCloud.
- add_context: Agregar contexto del requerimiento a la Base de Datos (interacción con clase de Persistencia)
- get_context: Obtener el contexto del requerimiento desde la Base de Datos (interacción con clase de Persistencia)
- del_context: Eliminar el contexto del requerimiento de la Base de Datos (interacción con clase de Persistencia)
- keep_context: Mantener información del contexto en la Base de Datos más allá de la vida del requerimiento. Básicamente esta función consiste en mantener el contexto para validación de seguridad de tareas agendadas.
- get_orgs: Obtener organizaciones en el sistema. Se devuelve la lista de organizaciones disponibles para el usuario autenticado en el sistema. Solamente los usuarios administradores de sistema (Providers) pueden acceder a todas las organizaciones.
- get_corresponding_tenant: Devuelve información de la organización dueña de la entidad (VM, vApp, Network, etc.)
- tenant_matches_context: Valida que el Tenant de la entidad a la que se le va a aplicar la acción corresponde al Tenant del contexto, es decir, del usuario que solicita la operación.
- entity_matches_tenant: Valida que el Tenant es dueña de la entidad.

- get_user: Devuelve información del usuario que corresponde al contexto del requerimiento autenticado en el sistema.



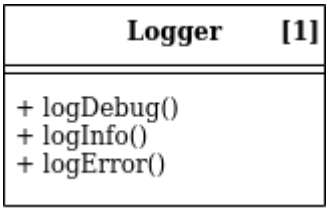
Algunas funcionalidades básicas de autorización son provistas al resto del framework, y al usuario encargado de extenderlo, mediante el patrón de diseño Decorator. El Decorator es un patrón de diseño que permite simplificar el código, agregando comportamiento a objetos externos. En este componente se definen las siguientes clases con este patrón de diseño:

- CheckPermissions: Decorator que valida que el usuario logueado en el sistema tiene asociados los permisos correspondientes para ejecutar la acción requerida.
- CheckUserAccess: Decorator utilizado para validar que el usuario logueado en el sistema tiene acceso a la entidad vApp asociada a la operación que se requiere ejecutar.

Estas clases realizan la validación de los permisos y autorizaciones correspondientes consultando al servidor de vCloud Director, integrando el framework con la implementación y configuración de seguridad ya existentes en el sistema. No hay una implementación de seguridad diferente de la ya existente en vCloud Director.

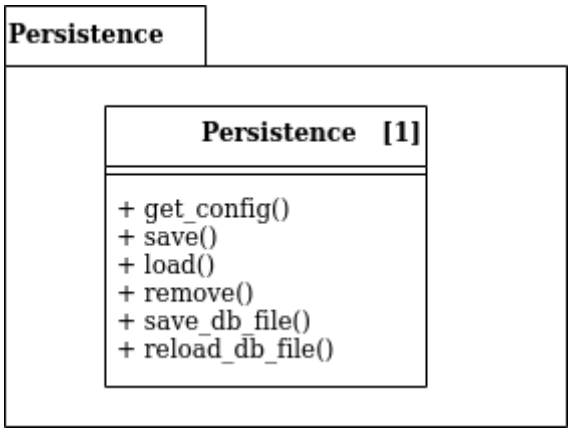
Logger: Registro de eventos

Componente encargado de capturar los eventos que ocurren durante la ejecución de las funcionalidades provistas por el framework. El módulo Logger da consistencia al registro de esos eventos dentro de la librería. Logger es un singleton: una única instancia de esta clase está disponible al resto del sistema.



3.3.6 Persistencia

3.3.6.1 Diagrama de Componentes



3.3.6.2 Descripción

Determinadas funcionalidades requieren una capa de persistencia que debe ser brindada por el sistema. Se deben persistir diferentes entidades para retomar el procesamiento de forma consistente ante el eventual reinicio del servidor. Los objetos que serán persistidos se serializarán en forma de pares clave – valor para su almacenamiento.

El módulo incluye la funcionalidad de lectura del archivo de configuración con parámetros que determinan lo siguiente:

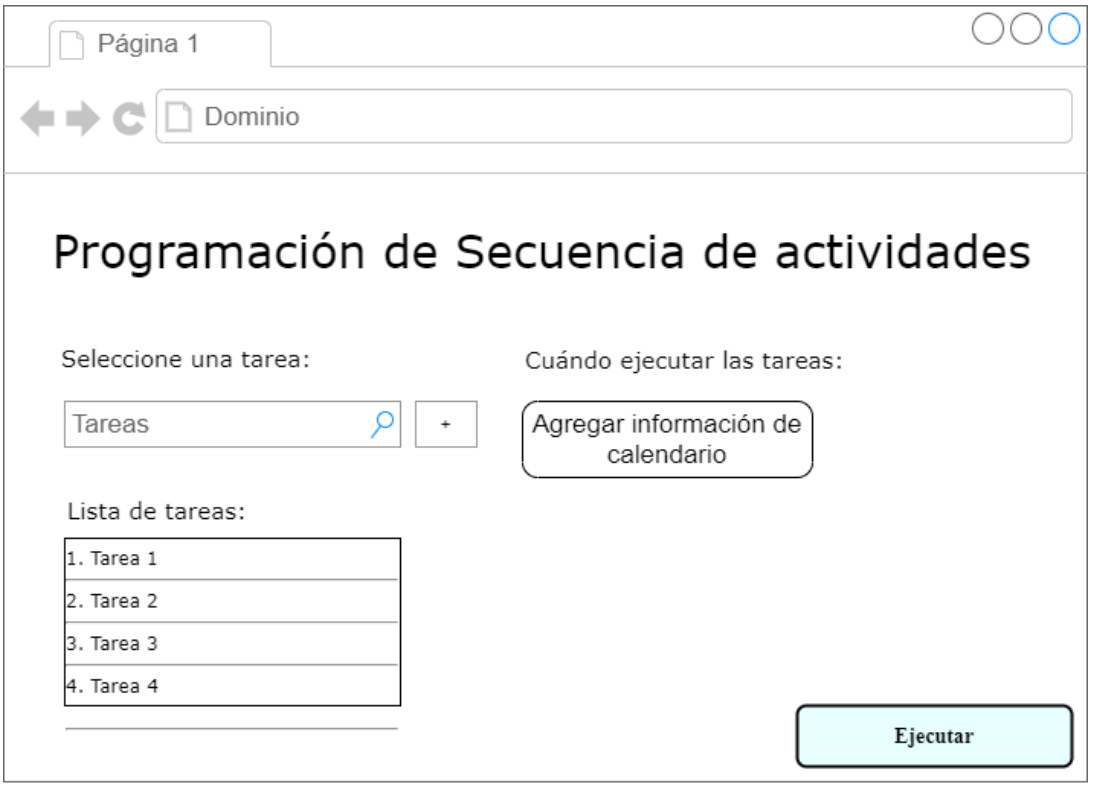
- Ambiente de ejecución: IPs/puerto de servidores.
- Autenticación y acceso: usuarios, passwords.
- Comportamiento del framework: validación SSL, versión API vCloud aceptada, entidades RabbitMQ.

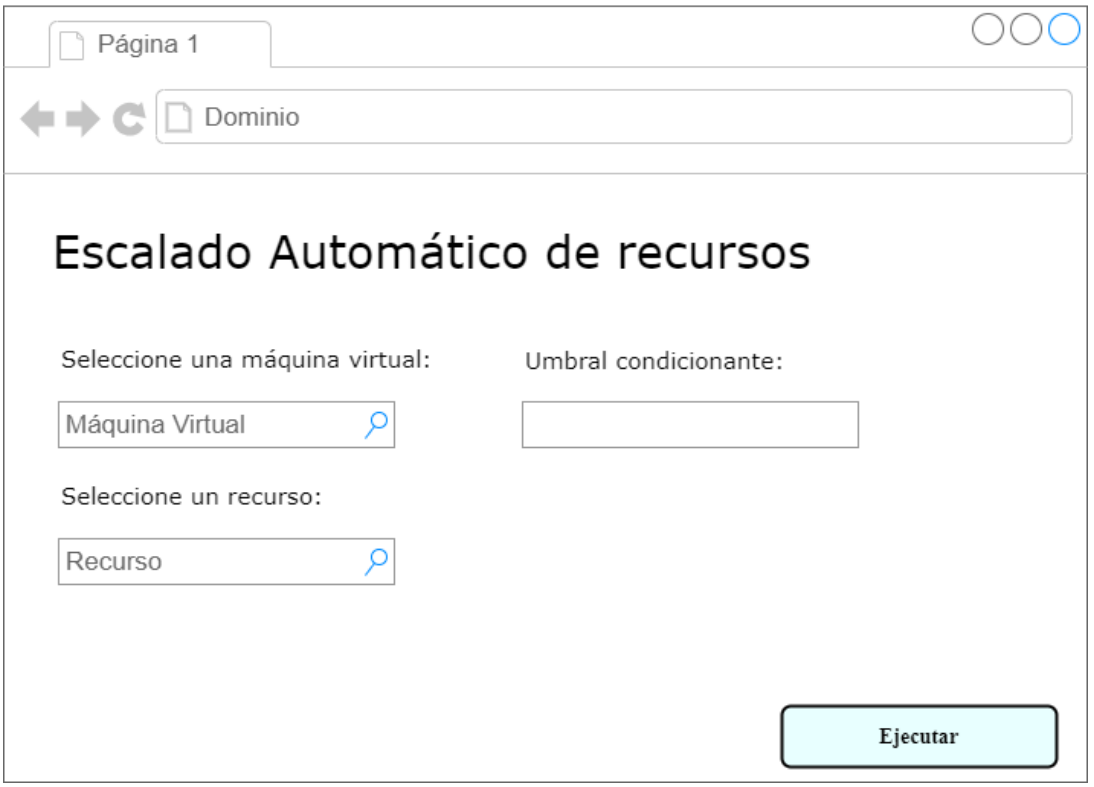
Persistence es un singleton: una única instancia de esta clase está disponible al resto del sistema.

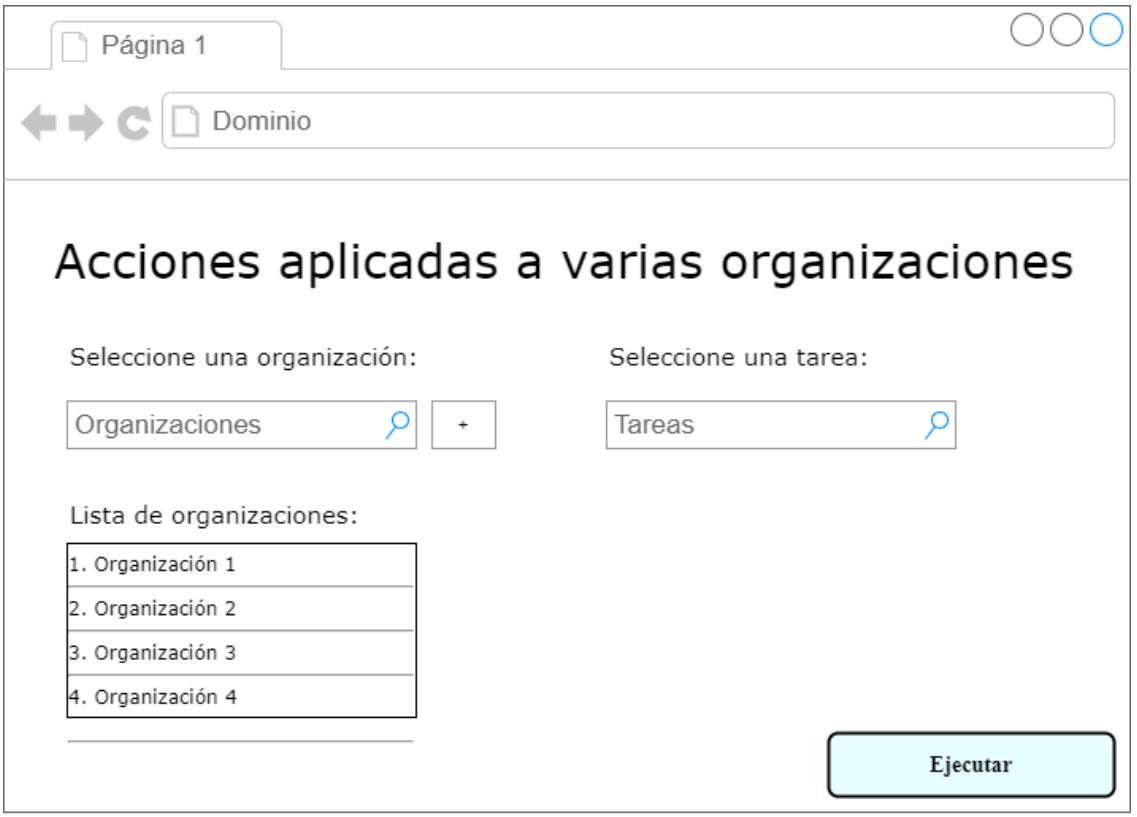
3.4 Diseño de Interfaces

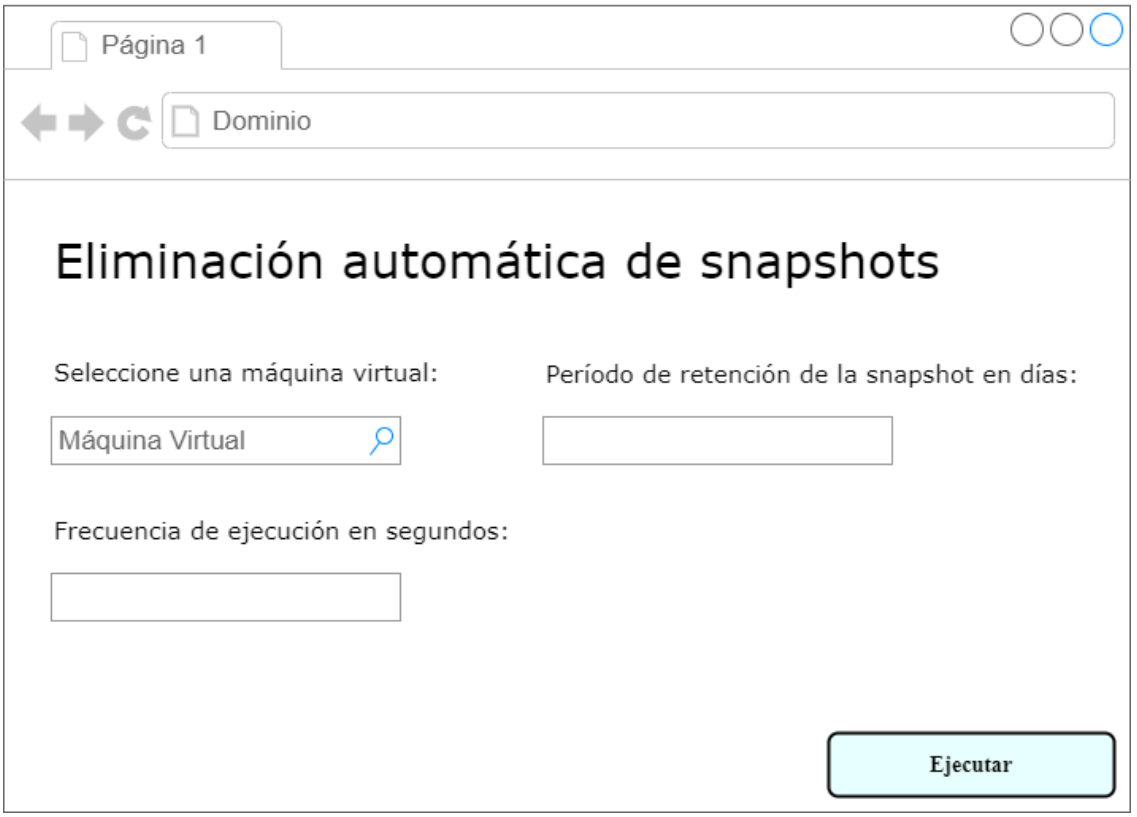
3.4.1 Wireframes de cada sección

En esta sección se detallan los wireframes correspondientes a cada sección de la interfaz de usuario. El criterio de división de las distintas secciones es en base a los casos de uso ya definidos. Se dispondrá de un menú en el lateral izquierdo en el cual se podrá acceder a cada sección de manera global.


Página 1
Dominio
Programación de Secuencia de actividades
Seleccione una tarea:
Tareas
+
Cuándo ejecutar las tareas:
Agregar información de calendario
Lista de tareas:
1. Tarea 1
2. Tarea 2
3. Tarea 3
4. Tarea 4
Ejecutar


Página 1
Dominio
Escalado Automático de recursos
Seleccione una máquina virtual:
Máquina Virtual
Umbral condicionante:
Seleccione un recurso:
Recurso
Ejecutar


Página 1
Dominio
Acciones aplicadas a varias organizaciones
Seleccione una organización:
Organizaciones
+
Seleccione una tarea:
Tareas
Lista de organizaciones:
1. Organización 1
2. Organización 2
3. Organización 3
4. Organización 4
Ejecutar


Página 1
Dominio
Eliminación automática de snapshots
Seleccione una máquina virtual:
Máquina Virtual
Período de retención de la snapshot en días:
Frecuencia de ejecución en segundos:
Ejecutar

Reporte de recursos de organizaciones.

Infraestructura de las organizaciones:

Organización 1:
- Data center virtual
  - vApp 1
    - Máquina Virtual 1
    - Máquina Virtual 2
  - vApp 2
    - Máquina Virtual 3
  - vApp 3
    - Máquina Virtual 4
  - Networks

Organización 2:
- Data center virtual
  - vApp 1
    - Máquina Virtual 1
  - vApp 2
    - Máquina Virtual 2
    - Máquina Virtual 3
  - Networks

4 Vista de Procesos

La vista de procesos trata con aspectos dinámicos del sistema. Representa cómo se comunican entre sí los subsistemas y sus componentes. Se enfoca en el comportamiento del sistema en tiempo de ejecución.

4.1 Diagramas de Interacción

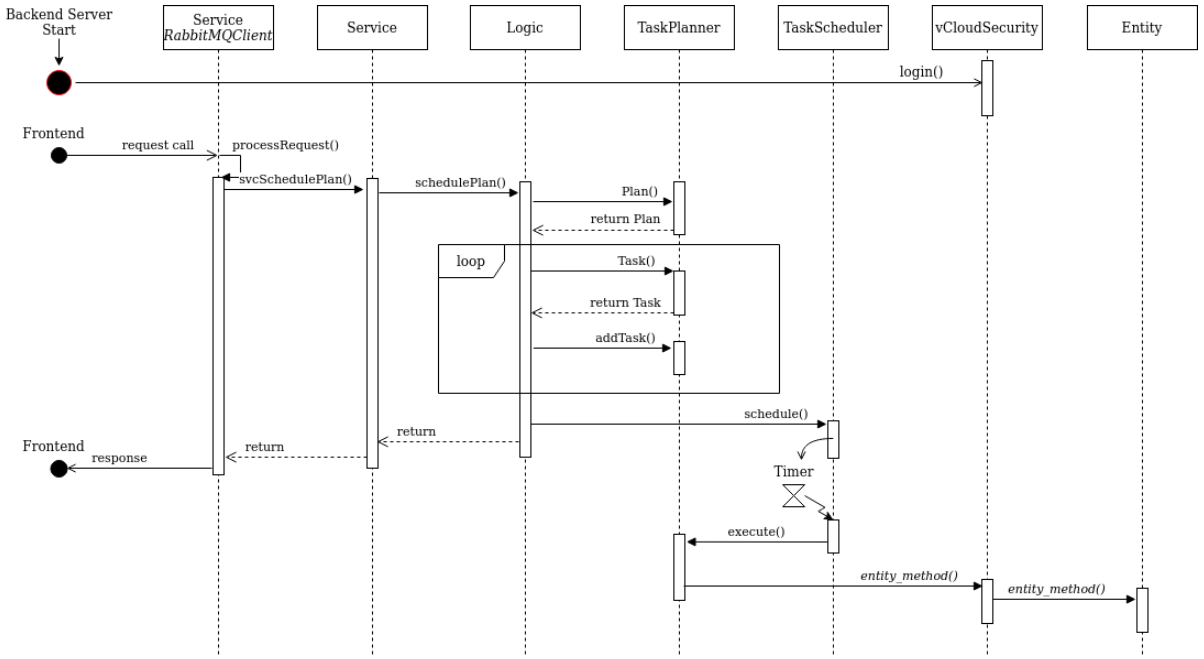
En esta sección se incluyen los diagramas de interacción con escenarios de ejecución exitosa (conocidos como “happy path”) de cada caso de uso crítico señalado.

En aras de la simplicidad, en los diagramas de actividad se omiten las invocaciones a los componentes de persistencia (Persistence) y de registro de eventos (Logger). Se puede asumir que hay un llamado al componente de persistencia en las siguientes instancias:

- En cada creación de un objeto de las clases que corresponden a entidades de vCloud Director, derivadas de la clase abstracta Entity.
- En cada creación de una Tarea o actualización de un Plan en invocaciones al componente TaskPlanner.
- En cada solicitud para agendar tareas en llamados al componente TaskScheduler.
- Al inicio del sistema de backend para recuperar la información de configuración.

Se puede asumir que hay invocación al componente de Logger en cada instancia que ocurre una excepción o error en el procesamiento (por ej. en la autenticación o en el acceso a una determinada entidad de vCloud Director), y también sobre el final de cada ejecución de una tarea en el componente TaskPlanner indicando el resultado de su ejecución.

Programación de secuencias de actividades

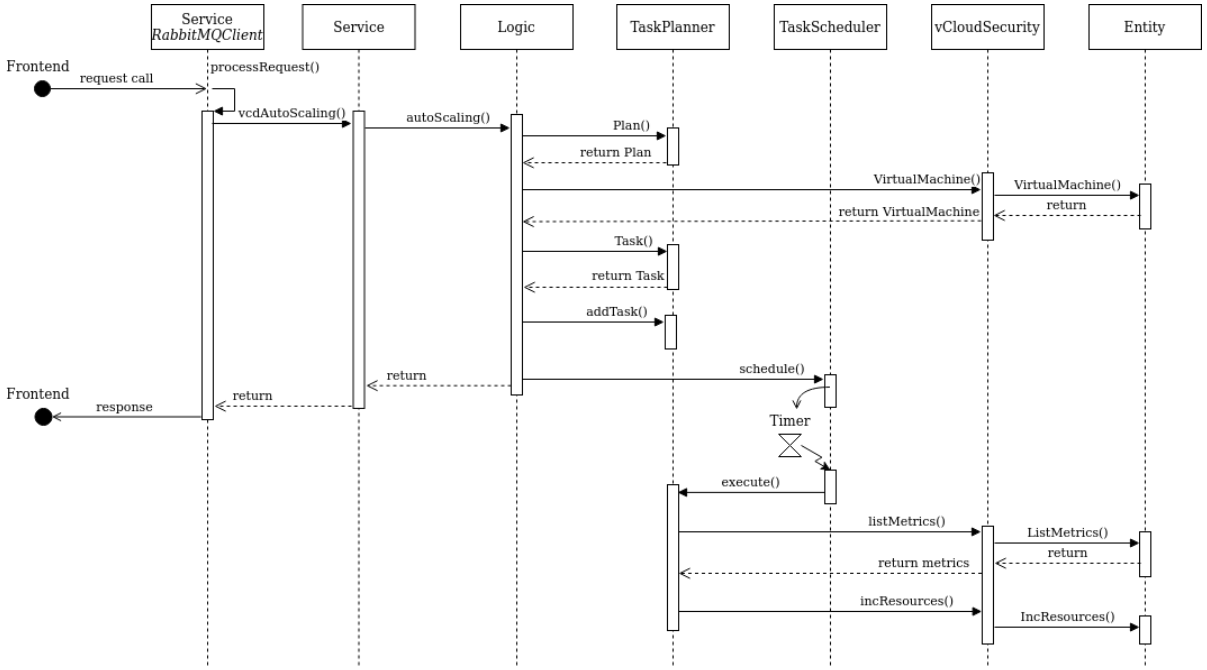


Al comenzar el servicio de Backend la primera invocación que se realiza es al método login() del componente de vCloudSecurity para autenticarse ante el servidor de vCloud Director. Eso se aplica a la lógica de inicialización del sistema de Backend y, por lo tanto, a todos los casos de uso y todos los escenarios.

El diagrama de actividades de este caso de uso presenta la siguiente secuencia:

- Llega el requerimiento del Frontend al componente Service que instancia el método de RabbitMQClient. Se invoca el método process_request() que es capturado por el componente Services.
- Invocación al método schedule_plan() del componente Logic, quien se encarga de ejecutar la lógica del caso de uso.
- Invocación al componente TaskPlanner para crear un nuevo Plan, llamando al constructor de la clase.
- Creación de la tarea con el método del objeto de vCloud instanciado en el paso anterior. Accede al componente TaskPlanner e invoca al constructor de la clase Task indicando el objeto de vCloud, el método y los parámetros requeridos.
- Inserción de la tarea al plan creado invocando el método add_task() del mismo. Se agrega al final de la lista de tareas del plan, manteniendo el orden de ejecución.
- Se repiten en loop los últimos tres pasos con todas las tareas que se indican en el requerimiento de entrada desde el frontend.
- Agenda el plan creado invocando al método schedule() del componente TaskScheduler, indicando el plan de ejecución y días, horas para realizar el agendado de la tarea.
- Se devuelve la respuesta al frontend con el resultado de la operación schedule_plan().
- Cuando vence el temporizador programado se invoca el método execute() del componente TaskPlanner.
- Se invocan los métodos de los objetos creados durante el proceso inicial. En este punto puede haber varias invocaciones a los métodos de las clases del componente Entity. Se crearán instancias de los objetos indicados en el requerimiento inicial (Tenant, VirtualDataCenter, VApp, VM, etc.) mediante invocaciones a los constructores de la clase correspondiente. La acción que se ejecutará en la tarea será un método del objeto instanciado.

Escalado automático de recursos

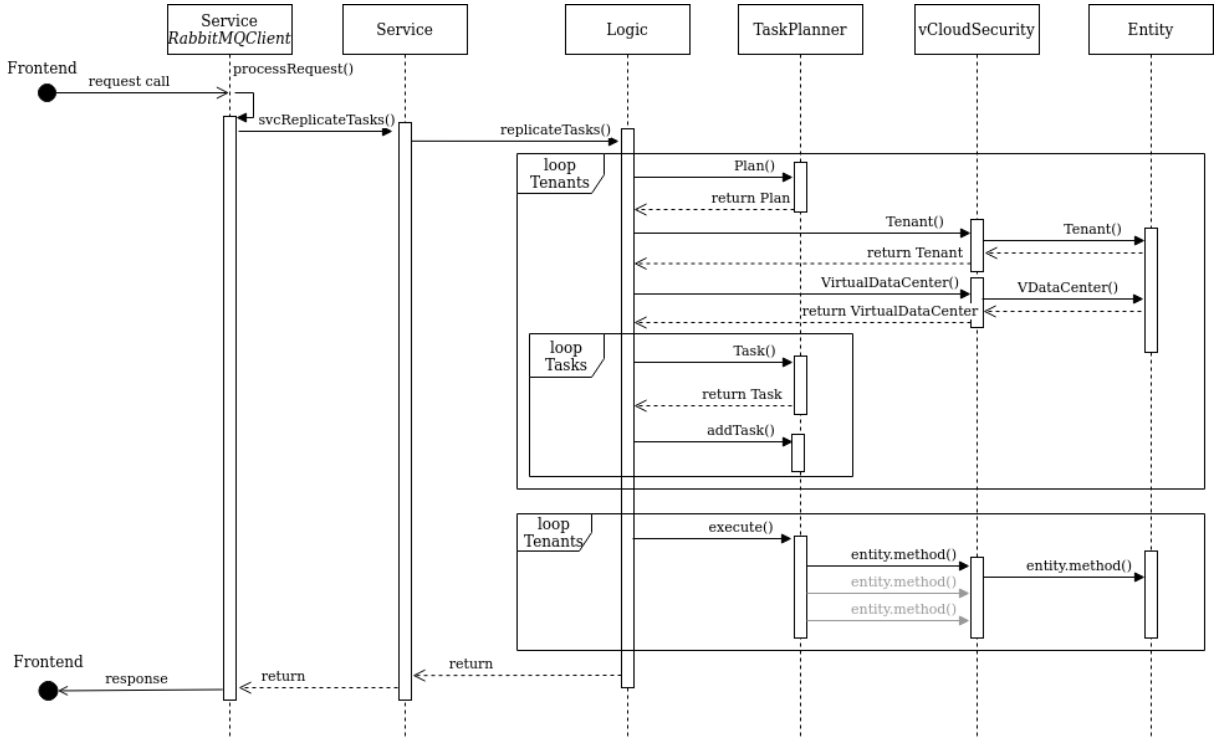


El diagrama de actividad para este caso de uso es prácticamente el mismo que para el caso de uso anterior “Programación de secuencias de actividades”. Para ajustarse a los requerimientos de este caso de uso se modifican los métodos invocados del componente Entity, como también deben modificarse los parámetros de las tareas y los criterios de éxito de estas. No se representan gráficamente estos cambios en el diagrama de actividades.
En este diagrama se considera el escenario en el que:

- Se validan las métricas sobre consumo de recursos de una Virtual Machine.
- Luego de validar las métricas de consumo se determina de acuerdo con los parámetros que el recurso en cuestión (CPU o memoria) debe ser incrementado invocando set_resources().

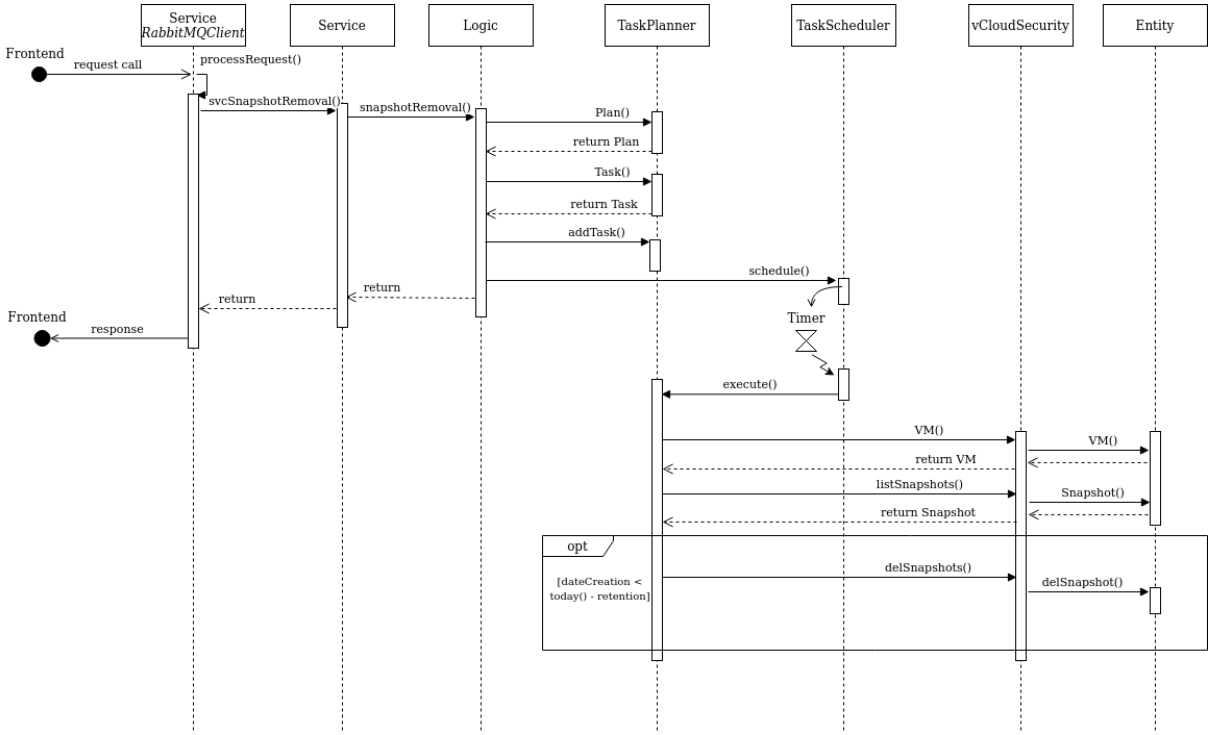
Cabe notar que en este y en los siguientes diagramas de actividad no se representa el login realizado al iniciar el servicio de Backend.

Acciones aplicadas a varias organizaciones

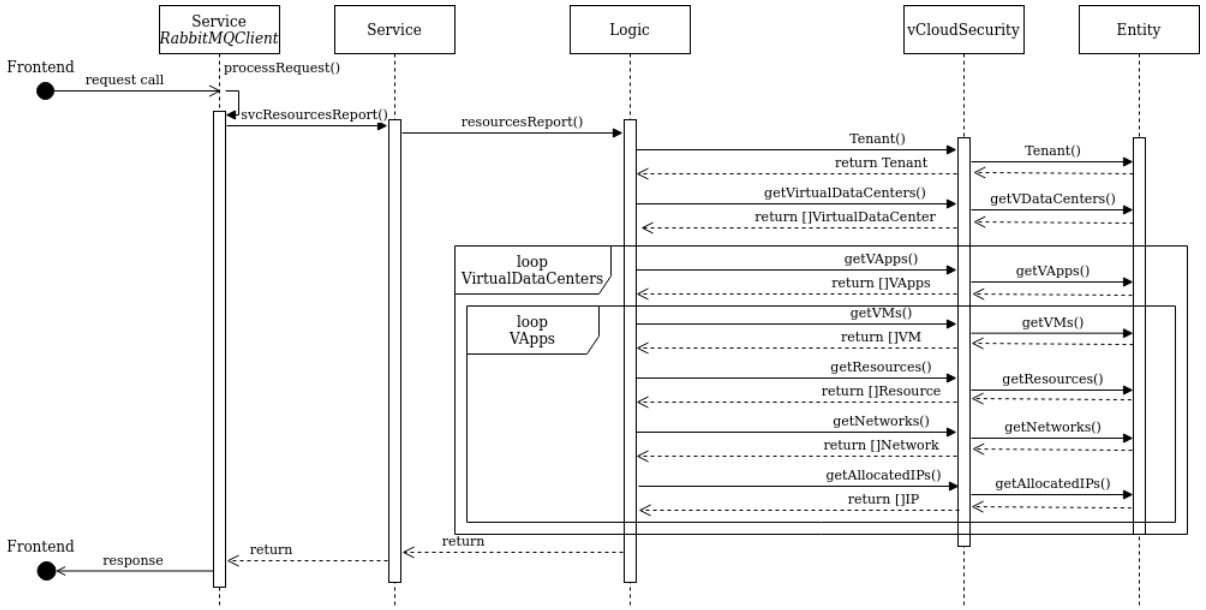


En este caso de uso se crea un plan por cada organización en la que se requiere replicar un conjunto de actividades. Eso se grafica en el primer loop exterior sobre los Tenants. Luego se itera en segunda instancia, sobre el conjunto de tareas que se van a replicar a todas las organizaciones. De esta forma cada plan tendrá asociado un Tenant y un conjunto de tareas para ese tenant. Luego se itera nuevamente sobre cada uno de los Tenants para replicar el mismo conjunto de tareas en todas las organizaciones. En cada iteración de este loop final, pueden ejecutarse una o varias acciones sobre el data-center de cada organización.

Eliminación automática de snapshots

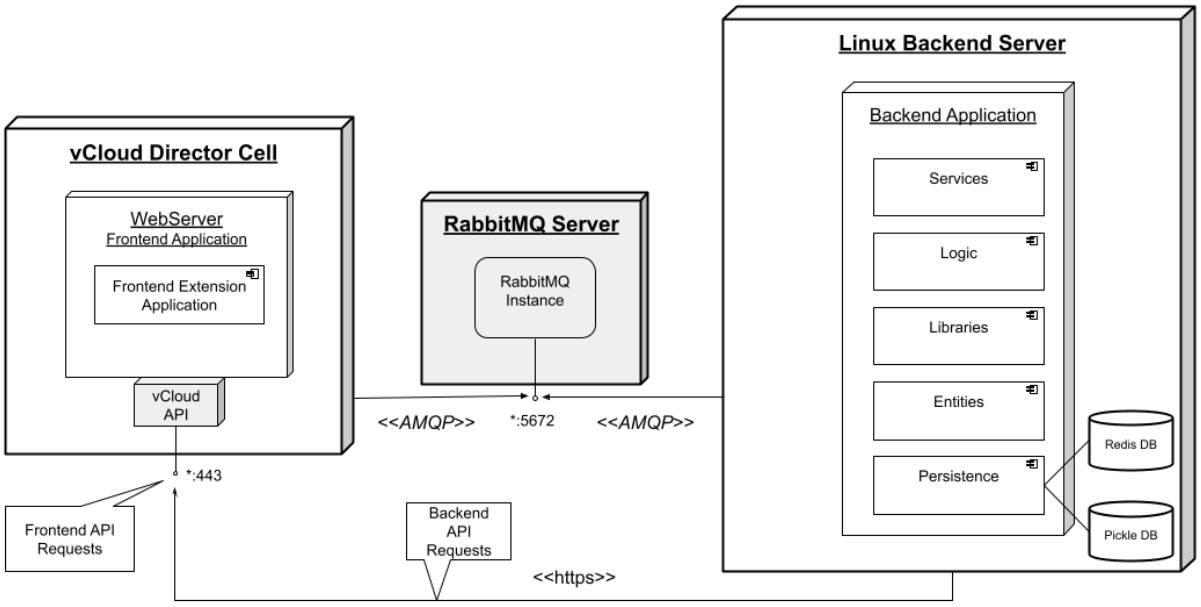


Reporte de recursos por organización



5 Vista de Distribución – Deployment

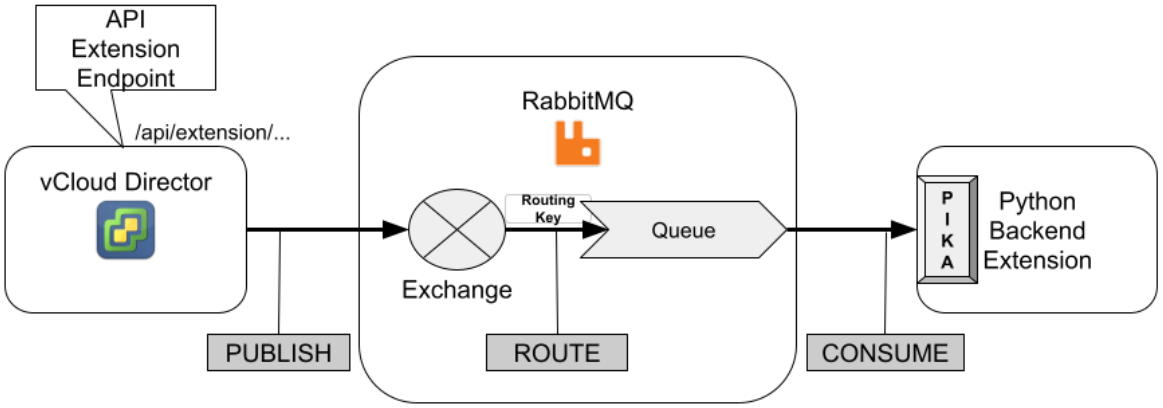
En esta vista se presenta la distribución de los componentes y subsistemas en los diferentes nodos.



Se presenta el servidor de vCloud Director, con la aplicación Frontend. Se debe a que en el despliegue de la solución se publica la aplicación web que extiende la funcionalidad de vCloud dentro del servidor.

El servidor Backend con SO Linux contiene todos los componentes y módulos descritos en detalle en la vista lógica.

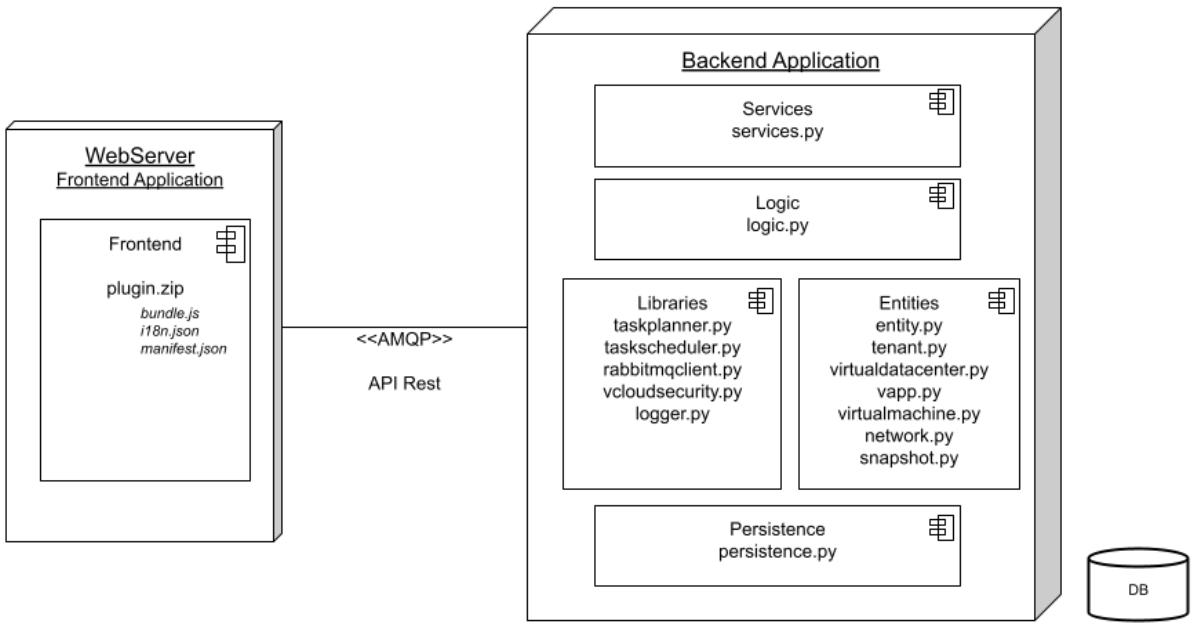
El servidor de RabbitMQ se presenta dentro del diagrama ya que, aunque no es necesario programar componentes que desplegar en ese servidor, forma parte fundamental de la solución. Dentro del mismo es necesario configurar un Exchange y una Routing Key (queue). La URL del endpoint de la extensión se asocian al Exchange y al Routing Key correspondiente para que vCloud Director rutee el requerimiento al Backend. A continuación, se presenta un esquema de la interrelación entre los diferentes subsistemas haciendo foco en el rol de RabbitMQ y sus entidades: Exchange y Routing-Key. A la izquierda la celda de vCloud Director, en el centro RabbitMQ, y a la derecha el servidor del Backend.


API
Extension
Endpoint
/api/extension/...
vCloud Director
RabbitMQ
Routing
Key
Queue
Exchange
P
I
K
A
Python
Backend
Extension
PUBLISH
ROUTE
CONSUME

6 Vista de Implementación

En esta vista se presentan los componentes en tiempo de ejecución que conforman el sistema.

- Aplicación frontend desplegada en el servidor vCloud Director:
  - plugin.zip
    - bundle.js
- Packages y módulos de Python en el backend:
  - Services
    - services.py
  - Logic
    - logic.py
  - Libraries
    - taskplanner.py
    - taskscheduler.py
    - rabbitmqclient.py
    - vcloudsecurity.py
    - logger.py
  - Entities
    - entity.py
    - tenant.py
    - virtualdatacenter.py
    - vapp.py
    - virtualmachine.py
    - network.py
    - snapshot.py
  - Persistence
    - persistence.py



7 Referencias

[1] Documentación de VMware Cloud Director. (2021). Recuperado de https://docs.vmware.com/es/VMware-Cloud-Director/index.html

[2] Kruchten, Philippe. (1995). Architectural Blueprints — The “4+1” View Model of Software Architecture. Recuperado de http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf

[3] Johansson, Lovisa & Vinka, Elin. (2020). The Optimal RabbitMQ Guide. Recuperado de https://www.cloudamqp.com/rabbitmq_ebook.html

[4] Python SDK for VMware vCloud Director. (2018). Recuperado de http://vmware.github.io/pyvcloud/

[5] Dwyer, John. (2018). Extending VMware vCloud® API with vCloud Extensibility Framework. Recuperado de https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/vcat/vmware-vcloud-api-extension-whitepaper.pdf

[6] Clarity Design System. (2021). Recuperado de https://clarity.design/

[7] Copia instantánea de volumen. Recuperado Marzo, 2021, de https://es.wikipedia.org/wiki/Copia_instant%C3%A1nea_de_volumen